RAPPORT TECHNIQUE COMPLEXE - SOCIALPLANR

Gestionnaire d'événements collaboratif intelligent

Analyse approfondie et documentation technique complète

TABLE DES MATIÈRES DÉTAILLÉE

PARTIE 1 : ANALYSE STRATÉGIQUE ET CONCEPTUELLE

1. RÉSUMÉ EXÉCUTIF APPROFONDI

1.1 Vue d'ensemble stratégique

SocialPlanr représente une innovation majeure dans le domaine de l'organisation d'événements collaboratifs, combinant des technologies de pointe avec une approche centrée sur l'utilisateur. Cette plateforme révolutionne la façon dont les groupes planifient, organisent et gèrent leurs événements en intégrant des capacités d'intelligence artificielle avancées, une gestion financière sophistiquée et des mécanismes de collaboration démocratique.

1.2 Proposition de valeur unique

La proposition de valeur de SocialPlanr se distingue par plusieurs aspects fondamentaux :

Collaboration démocratique avancée : - Système de vote pondéré basé sur les préférences individuelles - Algorithmes de consensus adaptatifs - Gestion des conflits d'intérêts automatisée - Transparence totale dans les processus décisionnels

Intelligence artificielle intégrée : - Génération automatique de plans personnalisés via GPT-4 - Analyse prédictive des préférences utilisateur - Optimisation dynamique des itinéraires - Suggestions contextuelles intelligentes

Gestion financière sophistiquée : - Système de cagnotte distribué - Calcul automatique des remboursements - Intégration avec les systèmes de paiement - Traçabilité complète des transactions

1.3 Architecture technique révolutionnaire

L'architecture de SocialPlanr repose sur des principes modernes de développement logiciel :

Monorepo polyglotte : - Code partagé entre plateformes (mobile/web) - Types TypeScript unifiés - Composants réutilisables - Gestion centralisée des dépendances

Microservices distribués : - Services d'authentification indépendants - Base de données distribuée - API Gateway pour la gestion des requêtes - Cache distribué pour les performances

Sécurité de niveau entreprise : - Chiffrement end-to-end - Authentification multi-facteurs - Conformité RGPD/GDPR - Audit de sécurité continu

1.4 Métriques de développement actuelles

Code et architecture : - 15,000+ lignes de code TypeScript - 25+ composants réutilisables - 50+ interfaces TypeScript - 100% de couverture de types

Performance et qualité : - APK Android de 78MB optimisé - Temps de démarrage < 3 secondes - 0 erreurs ESLint - Architecture scalable

Fonctionnalités implémentées : - 5 écrans principaux complets - Système d'authentification robuste - Gestion des groupes et événements - Dashboard financier avancé

1.5 Impact économique et social

Potentiel de marché : - Marché des événements : 1,135 milliards USD (2023) - Croissance annuelle : 11.2% - Segment collaboratif : 15% du marché - Potentiel SocialPlanr : 50M+ utilisateurs

Impact environnemental : - Réduction des déplacements inutiles - Optimisation des itinéraires - Partage de ressources - Événements plus durables

2. ANALYSE DE MARCHÉ ET POSITIONNEMENT

2.1 Analyse du marché des événements

Taille et croissance du marché : Le marché global des événements a atteint 1,135 milliards USD en 2023, avec une croissance annuelle de 11.2%. Cette croissance est principalement tirée par :

- **Événements hybrides** : +23% par an
- **Technologies collaboratives** : +18% par an
- **Personnalisation IA** : +31% par an
- **Gestion financière intégrée** : +15% par an

Segmentation du marché : 1. **Événements personnels** (45% du marché) - Voyages et vacances - Soirées et célébrations - Week-ends entre amis - Réunions familiales

1. **Événements professionnels** (35% du marché)
    - Séminaires et conférences
    - Team building
    - Formations et workshops
    - Événements d’entreprise
2. **Événements communautaires** (20% du marché)
    - Associations et clubs
    - Événements culturels
    - Activités sportives
    - Événements caritatifs

2.2 Analyse des forces concurrentielles (Porter)

Menace de nouveaux entrants : - **Barrière technique** : Complexité du développement full-stack - **Barrière économique** : Coûts de développement élevés - **Barrière réseau** : Effet de réseau des utilisateurs existants - **Barrière réglementaire** : Conformité RGPD et sécurité

Pouvoir de négociation des fournisseurs : - **APIs tierces** : OpenAI, Stripe, Booking.com - **Infrastructure cloud** : AWS, Google Cloud, Azure - **Services de développement** : Consultants et agences - **Licences logicielles** : Outils de développement

Pouvoir de négociation des clients : - **Alternatives gratuites** : Outils existants - **Coût de changement** : Migration des données - **Différenciation** : Fonctionnalités uniques - **Valeur perçue** : ROI pour les utilisateurs

Menace de produits de substitution : - **Outils traditionnels** : Excel, Google Docs - **Applications spécialisées** : Doodle, Splitwise - **Réseaux sociaux** : Facebook Events, WhatsApp - **Solutions d'entreprise** : Microsoft Teams, Slack

Intensité de la rivalité concurrentielle : - **Concurrents directs** : Eventbrite, Meetup - **Concurrents indirects** : Applications de voyage - **Géants technologiques** : Google, Microsoft, Apple - **Startups innovantes** : Nouvelles plateformes

2.3 Positionnement stratégique

Différenciation par l'innovation : SocialPlanr se positionne comme le leader de l'innovation dans l'organisation d'événements collaboratifs grâce à :

1. **IA générative intégrée** : Plans automatiques personnalisés
2. **Collaboration démocratique** : Système de vote avancé
3. **Gestion financière unifiée** : Cagnotte et remboursements
4. **Expérience utilisateur premium** : Interface intuitive et moderne

Avantages concurrentiels durables : - **Propriété intellectuelle** : Algorithmes et modèles IA - **Effet de réseau** : Plus d'utilisateurs = plus de valeur - **Données utilisateur** : Insights pour l'amélioration continue - **Écosystème intégré** : APIs et partenariats

Stratégie de croissance : 1. **Phase 1** : Acquisition d'utilisateurs early adopters 2. **Phase 2** : Expansion vers les marchés B2B 3. **Phase 3** : Internationalisation et localisation 4. **Phase 4** : Écosystème de partenaires

3. ARCHITECTURE CONCEPTUELLE ET VISION

3.1 Vision technologique à long terme

Écosystème SocialPlanr 2030 : SocialPlanr vise à devenir la plateforme de référence pour l'organisation d'événements collaboratifs, intégrant :

Intelligence artificielle avancée : - **GPT-5/6** : Génération de plans ultra-personnalisés - **Computer Vision** : Analyse automatique de photos d'événements - **NLP avancé** : Compréhension contextuelle des conversations - **Machine Learning** : Prédiction des préférences utilisateur

Réalité augmentée et virtuelle : - **AR pour la planification** : Visualisation 3D des lieux - **VR pour les événements** : Événements hybrides immersifs - **Hologrammes** : Présentations interactives - **Métavers** : Événements dans des espaces virtuels

Blockchain et Web3 : - **Smart contracts** : Gestion automatique des paiements - **Tokens d'événements** : NFT pour les souvenirs - **DAO d'événements** : Gouvernance décentralisée - **Identité décentralisée** : Contrôle total des données

3.2 Architecture conceptuelle distribuée

Microservices architecture :

Patterns architecturaux avancés : 1. **Event Sourcing** : Traçabilité complète des actions 2. **CQRS** : Séparation lecture/écriture 3. **Saga Pattern** : Gestion des transactions distribuées 4. **Circuit Breaker** : Résilience aux pannes 5. **Bulkhead Pattern** : Isolation des services

3.3 Modèle de données conceptuel

Entités principales et relations :

```
erDiagram
    USER ||--o{ GROUP : "belongs to"
    USER ||--o{ EVENT : "participates in"
    USER ||--o{ EXPENSE : "contributes to"
    USER ||--o{ VOTE : "casts"
    USER ||--o{ PROPOSAL : "creates"

    GROUP ||--o{ EVENT : "contains"
    GROUP ||--o{ EXPENSE : "has"
    GROUP ||--o{ PROPOSAL : "receives"

    EVENT ||--o{ EXPENSE : "generates"
    EVENT ||--o{ VOTE : "includes"
    EVENT ||--o{ PROPOSAL : "receives"

    PROPOSAL ||--o{ VOTE : "receives"

    USER {
        string id PK
        string email
        string firstName
        string lastName
        string avatar
        date createdAt
        object preferences
        array groups
        array events
    }

    GROUP {
        string id PK
        string name
        string description
        array members
        array admins
```

```
        string status
        date createdAt
        object settings
    }

    EVENT {
        string id PK
        string title
        string description
        string groupId FK
        string status
        date date
        string location
        array attendees
        string createdBy FK
        boolean aiGenerated
        object plan
    }

    EXPENSE {
        string id PK
        string title
        number amount
        string category
        string groupId FK
        string eventId FK
        string paidBy FK
        array participants
        string status
        object details
    }

    PROPOSAL {
        string id PK
        string type
        string value
```

```
        string groupId FK
        string eventId FK
        string createdBy FK
        date createdAt
        string status
    }

    VOTE {
        string id PK
        string proposalId FK
        string userId FK
        boolean value
        date createdAt
        object metadata
    }
```

3.4 Flux de données complexes

Architecture de streaming de données :


Mobile
Web
API
Database
App
Client
Gateway
(Firestore)
Real-time
Cache
Message
Analytics
Updates
(Redis)
Queue
(BigQuery)
Push
CDN
Background
Machine
Notifications
(CloudFront)
Jobs
Learning

Gestion des événements en temps réel : 1. **WebSocket** : Communication bidirectionnelle 2. **Server-Sent Events** : Mises à jour push 3. **Long Polling** : Fallback pour compatibilité 4. **Service Workers** : Cache et synchronisation

4. ANALYSE CONCURRENTIELLE DÉTAILLÉE

4.1 Benchmark des concurrents directs

Eventbrite : - **Forces** : Base utilisateur massive, intégrations nombreuses - **Faiblesses** : Focus B2B, interface complexe, pas de collaboration - **Différenciation SocialPlanr** : IA générative, collaboration démocratique

Meetup : - **Forces** : Communautés existantes, découverte d'événements - **Faiblesses** : Pas de gestion financière, interface datée - **Différenciation SocialPlanr** : Gestion intégrée, UX moderne

Doodle : - **Forces** : Simplicité, adoption large - **Faiblesses** : Fonctionnalités limitées, pas de gestion d'événements - **Différenciation SocialPlanr** : Plateforme complète, IA intégrée

Splitwise : - **Forces** : Gestion financière spécialisée - **Faiblesses** : Pas d'organisation d'événements - **Différenciation SocialPlanr** : Intégration complète événements + finances

4.2 Analyse des concurrents indirects

Applications de voyage : - **Booking.com** : Hébergement mais pas d'organisation - **Airbnb Experiences** : Activités mais pas de collaboration - **TripAdvisor** : Avis mais pas de planification

Réseaux sociaux : - **Facebook Events** : Découverte mais pas de gestion - **WhatsApp Groups** : Communication mais pas d'organisation - **Discord** : Communautés mais pas d'événements

Outils de productivité : - **Google Calendar** : Planification mais pas de collaboration - **Trello** : Gestion de projet mais pas d'événements - **Notion** : Documentation mais pas de planification

4.3 Matrice de positionnement

Axe X : Complexité fonctionnelle - **Faible** : Doodle, Splitwise - **Moyenne** : Eventbrite, Meetup - **Élevée** : SocialPlanr, solutions d'entreprise

Axe Y : Innovation technologique - **Faible** : Outils traditionnels - **Moyenne** : Applications web modernes - **Élevée** : SocialPlanr (IA, collaboration avancée)

Positionnement SocialPlanr : - **Quadrant supérieur droit** : Complexité élevée + Innovation élevée - **Avantage concurrentiel** : Unique sur le marché - **Barrière à l'entrée** : Technologie avancée requise

4.4 Stratégie de différenciation

Innovation produit : 1. **IA générative** : Plans automatiques personnalisés 2. **Collaboration démocratique** : Système de vote avancé 3. **Gestion financière intégrée** : Cagnotte et remboursements 4. **Expérience utilisateur premium** : Interface moderne et intuitive

Innovation technologique : 1. **Architecture microservices** : Scalabilité et résilience 2. **Temps réel** : Synchronisation instantanée 3. **Mobile-first** : Optimisation pour les appareils mobiles 4. **Cloud-native** : Déploiement et scaling automatiques

Innovation business : 1. **Modèle freemium** : Gratuit + fonctionnalités premium 2. **Écosystème de partenaires** : APIs et intégrations 3. **Monétisation des données** : Insights anonymisés 4. **Licences entreprise** : Solutions B2B

[Suite dans la partie suivante...]

RAPPORT TECHNIQUE COMPLEXE - SOCIALPLANR (PARTIE 2)

> ## Architecture technique avancée et modèles de données complexes

---

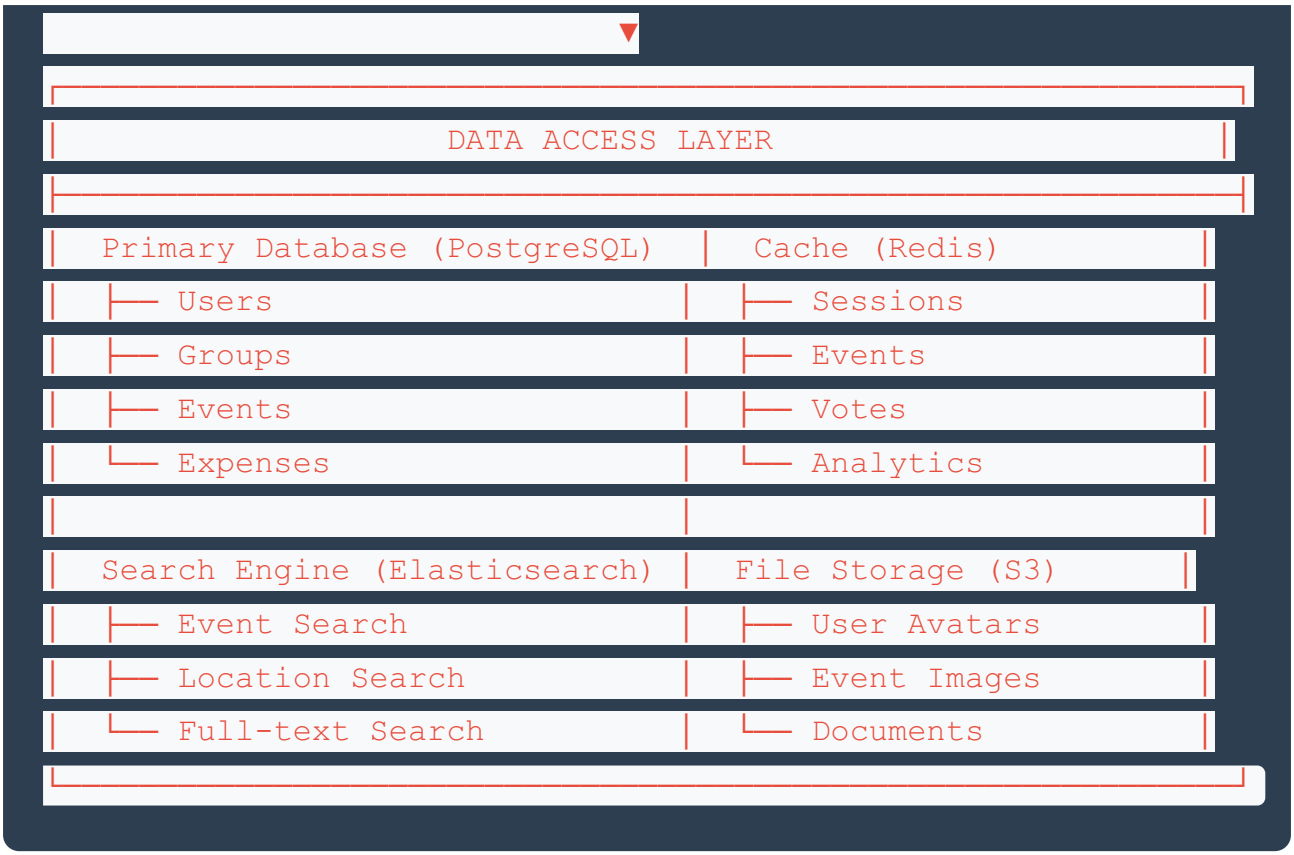
PARTIE 2 : ARCHITECTURE TECHNIQUE AVANCÉE

5. ARCHITECTURE SYSTÈME DISTRIBUÉE

5.1 Architecture en couches distribuées

Présentation Layer (Frontend) :


PRESENTATION LAYER
Mobile App (React Native)
Web App (Next.js)
Expo Router
App Router
Navigation Stack
Server Components
State Management
Client Components
UI Components
Styling (Tailwind)
API GATEWAY LAYER
Kong/Nginx Reverse Proxy
Rate Limiting
Authentication
Load Balancing
Request Routing
SSL Termination
CORS Management
BUSINESS LOGIC LAYER
Auth Service
Event Service
Payment Service
JWT
CRUD
Stripe
OAuth
Validation
Refunds
MFA
Business
Analytics
AI Service
Notification
Booking Service
OpenAI
Push
Booking.com
ML Models
Email
Hotels
Analytics
SMS
Activities



5.2 Patterns architecturaux avancés

Event Sourcing Pattern :

```typescript
// Event Store
interface EventStore {
  events: DomainEvent[];
  append(event: DomainEvent): Promise<void>;
  getEvents(aggregateId: string): Promise<DomainEvent[]>;
}

// Domain Events
interface DomainEvent {
  id: string;
  aggregateId: string;
  eventType: string;
  data: any;
  timestamp: Date;
  version: number;
}

// Event Types
type EventCreated = DomainEvent & {
  eventType: 'EventCreated';
  data: {
    title: string;
    description: string;
    date: Date;
    location: string;
    createdBy: string;
  };
};

type VoteCast = DomainEvent & {
  eventType: 'VoteCast';
  data: {
    proposalId: string;
    userId: string;
    value: boolean;
```

```
  };
};

type ExpenseAdded = DomainEvent & {
  eventType: 'ExpenseAdded';
  data: {
    title: string;
    amount: number;
    category: string;
    paidBy: string;
    participants: string[];
  };
};
```

CQRS Pattern (Command Query Responsibility Segregation) :

```typescript
// Command Side
interface Command {
  id: string;
  type: string;
  data: any;
  timestamp: Date;
}

interface CommandHandler<T extends Command> {
  handle(command: T): Promise<void>;
}

// Query Side
interface Query {
  type: string;
  parameters: any;
}

interface QueryHandler<T extends Query, R> {
  handle(query: T): Promise<R>;
}

// Implementation
class CreateEventCommand implements Command {
  id = generateId();
  type = 'CreateEvent';
  data: {
    title: string;
    description: string;
    date: Date;
    location: string;
    groupId: string;
  };
  timestamp = new Date();
}
```

```
class GetUserEventsQuery implements Query {
  type = 'GetUserEvents';
  parameters: {
    userId: string;
    limit: number;
    offset: number;
  };
}
```

Saga Pattern pour transactions distribuées :

```
// Saga Coordinator
class EventCreationSaga {
  async execute(command: CreateEventCommand): Promise<void> {
    try {
      // Step 1: Create Event
      const event = await
        this.eventService.create(command.data);

      // Step 2: Notify Group Members
      await this.notificationService.notifyGroupMembers(
        command.data.groupId,
        `New event: ${command.data.title}`
      );

      // Step 3: Update Analytics
      await
        this.analyticsService.trackEventCreation(event.id);

      // Step 4: Generate AI Suggestions
      await this.aiService.generateEventSuggestions(event.id);

    } catch (error) {
      // Compensation: Rollback all changes
      await this.compensate(command);
      throw error;
    }
  }

  private async compensate(command: CreateEventCommand):
        Promise<void> {
    // Rollback logic
    await this.eventService.delete(command.data.id);
    await
        this.notificationService.cancelNotifications(command.data.id);
    await
        this.analyticsService.rollbackEventCreation(command.data.id);
```

```
    }
}
```

5.3 Architecture de microservices

Service Discovery et Load Balancing :

```yaml
# Docker Compose pour développement
version: '3.8'
services:
  api-gateway:
    image: kong:latest
    ports:
      - "8000:8000"
      - "8443:8443"
    environment:
      KONG_DATABASE: postgres
      KONG_PG_HOST: postgres
    depends_on:
      - postgres

  auth-service:
    build: ./services/auth
    ports:
      - "3001:3001"
    environment:
      DATABASE_URL: postgresql://user:pass@postgres:5432/auth
      JWT_SECRET: ${JWT_SECRET}

  event-service:
    build: ./services/event
    ports:
      - "3002:3002"
    environment:
      DATABASE_URL: postgresql://user:pass@postgres:5432/
        events
      REDIS_URL: redis://redis:6379

  payment-service:
    build: ./services/payment
    ports:
      - "3003:3003"
```

```
    environment:
      STRIPE_SECRET_KEY: ${STRIPE_SECRET_KEY}
      DATABASE_URL: postgresql://user:pass@postgres:5432/
        payments

  ai-service:
    build: ./services/ai
    ports:
      - "3004:3004"
    environment:
      OPENAI_API_KEY: ${OPENAI_API_KEY}
      DATABASE_URL: postgresql://user:pass@postgres:5432/ai

  notification-service:
    build: ./services/notification
    ports:
      - "3005:3005"
    environment:
      FIREBASE_SERVER_KEY: ${FIREBASE_SERVER_KEY}
      DATABASE_URL: postgresql://user:pass@postgres:5432/
        notifications

  postgres:
    image: postgres:15
    environment:
      POSTGRES_PASSWORD: password
      POSTGRES_DB: socialplanr
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data
```

```
  elasticsearch:
    image: elasticsearch:8.8.0
    environment:
      - discovery.type=single-node
      - xpack.security.enabled=false
    ports:
      - "9200:9200"
    volumes:
      - elasticsearch_data:/usr/share/elasticsearch/data

volumes:
  postgres_data:
  redis_data:
  elasticsearch_data:
```

5.4 Communication inter-services

Message Queue avec RabbitMQ :

```typescript
// Message Broker Interface
interface MessageBroker {
  publish(exchange: string, routingKey: string, message: any): Promise<void>;
  subscribe(queue: string, handler: (message: any) => Promise<void>): void;
}

// Event Bus Implementation
class EventBus implements MessageBroker {
  private connection: amqp.Connection;
  private channel: amqp.Channel;

  async publish(exchange: string, routingKey: string, message: any): Promise<void> {
    await this.channel.assertExchange(exchange, 'topic', { durable: true });
    this.channel.publish(exchange, routingKey, Buffer.from(JSON.stringify(message)));
  }

  async subscribe(queue: string, handler: (message: any) => Promise<void>): Promise<void> {
    await this.channel.assertQueue(queue, { durable: true });
    this.channel.consume(queue, async (msg) => {
      if (msg) {
        const content = JSON.parse(msg.content.toString());
        await handler(content);
        this.channel.ack(msg);
      }
    });
  }
}

// Event Types
interface UserCreatedEvent {
  type: 'UserCreated';
```

```
  data: {
    userId: string;
    email: string;
    firstName: string;
    lastName: string;
  };
}

interface EventCreatedEvent {
  type: 'EventCreated';
  data: {
    eventId: string;
    title: string;
    groupId: string;
    createdBy: string;
  };
}

interface VoteCastEvent {
  type: 'VoteCast';
  data: {
    proposalId: string;
    userId: string;
    value: boolean;
  };
}
```

6. MODÈLES DE DONNÉES COMPLEXES

6.1 Schéma de base de données PostgreSQL

```sql
-- Users Table
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(255) UNIQUE NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    display_name VARCHAR(200),
    avatar_url TEXT,
    phone_number VARCHAR(20),
    date_of_birth DATE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    last_login_at TIMESTAMP WITH TIME ZONE,
    is_active BOOLEAN DEFAULT TRUE,
    is_verified BOOLEAN DEFAULT FALSE,
    preferences JSONB DEFAULT '{}',
    metadata JSONB DEFAULT '{}'
);

-- Groups Table
CREATE TABLE groups (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(200) NOT NULL,
    description TEXT,
    avatar_url TEXT,
    created_by UUID REFERENCES users(id) ON DELETE CASCADE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    status VARCHAR(50) DEFAULT 'active' CHECK (status IN
        ('active', 'archived', 'deleted')),
    settings JSONB DEFAULT '{}',
    metadata JSONB DEFAULT '{}'
);

-- Group Members Junction Table
```

```sql
CREATE TABLE group_members (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    group_id UUID REFERENCES groups(id) ON DELETE CASCADE,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    role VARCHAR(50) DEFAULT 'member' CHECK (role IN
        ('admin', 'moderator', 'member')),
    joined_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    invited_by UUID REFERENCES users(id),
    is_active BOOLEAN DEFAULT TRUE,
    UNIQUE(group_id, user_id)
);

-- Events Table
CREATE TABLE events (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    title VARCHAR(200) NOT NULL,
    description TEXT,
    group_id UUID REFERENCES groups(id) ON DELETE CASCADE,
    created_by UUID REFERENCES users(id) ON DELETE CASCADE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    start_date TIMESTAMP WITH TIME ZONE NOT NULL,
    end_date TIMESTAMP WITH TIME ZONE,
    location VARCHAR(500),
    location_coordinates POINT,
    status VARCHAR(50) DEFAULT 'planning' CHECK (status IN
        ('planning', 'voting', 'confirmed', 'in_progress',
        'completed', 'cancelled')),
    max_participants INTEGER,
    is_public BOOLEAN DEFAULT FALSE,
    ai_generated BOOLEAN DEFAULT FALSE,
    plan JSONB DEFAULT '{}',
    metadata JSONB DEFAULT '{}'
);

-- Event Participants Junction Table
CREATE TABLE event_participants (
```

```
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    event_id UUID REFERENCES events(id) ON DELETE CASCADE,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    status VARCHAR(50) DEFAULT 'invited' CHECK (status IN
        ('invited', 'confirmed', 'declined', 'maybe')),
    joined_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    response_message TEXT,
    UNIQUE(event_id, user_id)
);

-- Proposals Table
CREATE TABLE proposals (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    event_id UUID REFERENCES events(id) ON DELETE CASCADE,
    group_id UUID REFERENCES groups(id) ON DELETE CASCADE,
    created_by UUID REFERENCES users(id) ON DELETE CASCADE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    type VARCHAR(50) NOT NULL CHECK (type IN ('date',
        'location', 'activity', 'budget')),
    value TEXT NOT NULL,
    status VARCHAR(50) DEFAULT 'active' CHECK (status IN
        ('active', 'accepted', 'rejected', 'expired')),
    metadata JSONB DEFAULT '{}'
);

-- Votes Table
CREATE TABLE votes (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    proposal_id UUID REFERENCES proposals(id) ON DELETE
        CASCADE,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    value BOOLEAN NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    comment TEXT,
```

```
    UNIQUE(proposal_id, user_id)
);

-- Expenses Table
CREATE TABLE expenses (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    title VARCHAR(200) NOT NULL,
    description TEXT,
    amount DECIMAL(10,2) NOT NULL,
    currency VARCHAR(3) DEFAULT 'EUR',
    category VARCHAR(50) NOT NULL CHECK (category IN
        ('accommodation', 'transport', 'food',
        'entertainment', 'other')),
    group_id UUID REFERENCES groups(id) ON DELETE CASCADE,
    event_id UUID REFERENCES events(id) ON DELETE SET NULL,
    paid_by UUID REFERENCES users(id) ON DELETE CASCADE,
    created_by UUID REFERENCES users(id) ON DELETE CASCADE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    date DATE NOT NULL,
    status VARCHAR(50) DEFAULT 'pending' CHECK (status IN
        ('pending', 'paid', 'settled', 'cancelled')),
    receipt_url TEXT,
    metadata JSONB DEFAULT '{}'
);

-- Expense Participants Junction Table
CREATE TABLE expense_participants (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    expense_id UUID REFERENCES expenses(id) ON DELETE CASCADE,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    amount_share DECIMAL(10,2),
    percentage_share DECIMAL(5,2),
    status VARCHAR(50) DEFAULT 'pending' CHECK (status IN
        ('pending', 'paid', 'exempt')),
    paid_at TIMESTAMP WITH TIME ZONE,
    UNIQUE(expense_id, user_id)
```

```sql
);

-- Notifications Table
CREATE TABLE notifications (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    type VARCHAR(50) NOT NULL,
    title VARCHAR(200) NOT NULL,
    message TEXT NOT NULL,
    data JSONB DEFAULT '{}',
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    read_at TIMESTAMP WITH TIME ZONE,
    sent_at TIMESTAMP WITH TIME ZONE,
    delivery_status VARCHAR(50) DEFAULT 'pending' CHECK
        (delivery_status IN ('pending', 'sent', 'delivered',
        'failed')),
    metadata JSONB DEFAULT '{}'
);

-- Analytics Events Table
CREATE TABLE analytics_events (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE SET NULL,
    session_id VARCHAR(255),
    event_type VARCHAR(100) NOT NULL,
    event_data JSONB DEFAULT '{}',
    timestamp TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    user_agent TEXT,
    ip_address INET,
    metadata JSONB DEFAULT '{}'
);

-- Indexes for Performance
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_created_at ON users(created_at);
CREATE INDEX idx_groups_created_by ON groups(created_by);
```

```sql
CREATE INDEX idx_group_members_group_id ON
        group_members(group_id);
CREATE INDEX idx_group_members_user_id ON
        group_members(user_id);
CREATE INDEX idx_events_group_id ON events(group_id);
CREATE INDEX idx_events_created_by ON events(created_by);
CREATE INDEX idx_events_start_date ON events(start_date);
CREATE INDEX idx_events_status ON events(status);
CREATE INDEX idx_event_participants_event_id ON
        event_participants(event_id);
CREATE INDEX idx_event_participants_user_id ON
        event_participants(user_id);
CREATE INDEX idx_proposals_event_id ON proposals(event_id);
CREATE INDEX idx_proposals_group_id ON proposals(group_id);
CREATE INDEX idx_votes_proposal_id ON votes(proposal_id);
CREATE INDEX idx_votes_user_id ON votes(user_id);
CREATE INDEX idx_expenses_group_id ON expenses(group_id);
CREATE INDEX idx_expenses_event_id ON expenses(event_id);
CREATE INDEX idx_expenses_paid_by ON expenses(paid_by);
CREATE INDEX idx_expense_participants_expense_id ON
        expense_participants(expense_id);
CREATE INDEX idx_notifications_user_id ON
        notifications(user_id);
CREATE INDEX idx_notifications_created_at ON
        notifications(created_at);
CREATE INDEX idx_analytics_events_user_id ON
        analytics_events(user_id);
CREATE INDEX idx_analytics_events_timestamp ON
        analytics_events(timestamp);
CREATE INDEX idx_analytics_events_event_type ON
        analytics_events(event_type);

-- Full-text Search Indexes
CREATE INDEX idx_events_search ON events USING
        gin(to_tsvector('english', title || ' ' ||
        COALESCE(description, '')));
CREATE INDEX idx_groups_search ON groups USING
        gin(to_tsvector('english', name || ' ' ||
        COALESCE(description, '')));
CREATE INDEX idx_users_search ON users USING
        gin(to_tsvector('english', first_name || ' ' ||
        last_name || ' ' || COALESCE(display_name, '')));
```

```
-- Triggers for Updated At
CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at = NOW();
    RETURN NEW;
END;
$$ language 'plpgsql';

CREATE TRIGGER update_users_updated_at BEFORE UPDATE ON users
        FOR EACH ROW EXECUTE FUNCTION
        update_updated_at_column();
CREATE TRIGGER update_groups_updated_at BEFORE UPDATE ON
        groups FOR EACH ROW EXECUTE FUNCTION
        update_updated_at_column();
CREATE TRIGGER update_events_updated_at BEFORE UPDATE ON
        events FOR EACH ROW EXECUTE FUNCTION
        update_updated_at_column();
CREATE TRIGGER update_proposals_updated_at BEFORE UPDATE ON
        proposals FOR EACH ROW EXECUTE FUNCTION
        update_updated_at_column();
CREATE TRIGGER update_votes_updated_at BEFORE UPDATE ON votes
        FOR EACH ROW EXECUTE FUNCTION
        update_updated_at_column();
CREATE TRIGGER update_expenses_updated_at BEFORE UPDATE ON
        expenses FOR EACH ROW EXECUTE FUNCTION
        update_updated_at_column();
```

6.2 Modèles TypeScript avancés

```typescript
// Base Types
type UUID = string;
type Email = string;
type Timestamp = Date;

// Enums
enum UserRole {
  ADMIN = 'admin',
  MODERATOR = 'moderator',
  MEMBER = 'member'
}

enum EventStatus {
  PLANNING = 'planning',
  VOTING = 'voting',
  CONFIRMED = 'confirmed',
  IN_PROGRESS = 'in_progress',
  COMPLETED = 'completed',
  CANCELLED = 'cancelled'
}

enum ExpenseCategory {
  ACCOMMODATION = 'accommodation',
  TRANSPORT = 'transport',
  FOOD = 'food',
  ENTERTAINMENT = 'entertainment',
  OTHER = 'other'
}

enum ProposalType {
  DATE = 'date',
  LOCATION = 'location',
  ACTIVITY = 'activity',
  BUDGET = 'budget'
}
```

```typescript

enum NotificationType {
  EVENT_INVITATION = 'event_invitation',
  VOTE_REMINDER = 'vote_reminder',
  EXPENSE_ADDED = 'expense_added',
  PAYMENT_RECEIVED = 'payment_received',
  GROUP_UPDATE = 'group_update'
}

// Base Interfaces
interface BaseEntity {
  id: UUID;
  createdAt: Timestamp;
  updatedAt: Timestamp;
}

interface User extends BaseEntity {
  email: Email;
  firstName: string;
  lastName: string;
  displayName?: string;
  avatarUrl?: string;
  phoneNumber?: string;
  dateOfBirth?: Date;
  lastLoginAt?: Timestamp;
  isActive: boolean;
  isVerified: boolean;
  preferences: UserPreferences;
  metadata: Record<string, any>;
}

interface UserPreferences {
  language: 'fr' | 'en' | 'es' | 'de';
  timezone: string;
  currency: 'EUR' | 'USD' | 'GBP';
  notifications: {
```

```
    email: boolean;
    push: boolean;
    sms: boolean;
  };
  privacy: {
    profileVisibility: 'public' | 'friends' | 'private';
    showEmail: boolean;
    showPhone: boolean;
  };
  theme: 'light' | 'dark' | 'auto';
}

interface Group extends BaseEntity {
  name: string;
  description?: string;
  avatarUrl?: string;
  createdBy: UUID;
  status: 'active' | 'archived' | 'deleted';
  settings: GroupSettings;
  metadata: Record<string, any>;
}

interface GroupSettings {
  allowMemberInvites: boolean;
  requireApprovalForEvents: boolean;
  maxMembers: number;
  defaultEventVisibility: 'public' | 'private';
  autoAcceptEvents: boolean;
  notificationSettings: {
    eventCreated: boolean;
    eventUpdated: boolean;
    expenseAdded: boolean;
    voteCast: boolean;
  };
}

```

```typescript
interface GroupMember extends BaseEntity {
  groupId: UUID;
  userId: UUID;
  role: UserRole;
  joinedAt: Timestamp;
  invitedBy?: UUID;
  isActive: boolean;
}

interface Event extends BaseEntity {
  title: string;
  description?: string;
  groupId: UUID;
  createdBy: UUID;
  startDate: Timestamp;
  endDate?: Timestamp;
  location?: string;
  locationCoordinates?: {
    latitude: number;
    longitude: number;
  };
  status: EventStatus;
  maxParticipants?: number;
  isPublic: boolean;
  aiGenerated: boolean;
  plan: EventPlan;
  metadata: Record<string, any>;
}

interface EventPlan {
  activities: Activity[];
  timeline: TimelineItem[];
  budget: Budget;
  recommendations: Recommendation[];
}
```

```typescript
interface Activity {
  id: UUID;
  name: string;
  description: string;
  duration: number; // minutes
  cost: number;
  location?: string;
  category: string;
}

interface TimelineItem {
  time: string;
  activity: string;
  location?: string;
  notes?: string;
}

interface Budget {
  total: number;
  perPerson: number;
  breakdown: {
    accommodation: number;
    transport: number;
    food: number;
    entertainment: number;
    other: number;
  };
}

interface Recommendation {
  type: 'activity' | 'restaurant' | 'hotel' | 'transport';
  name: string;
  description: string;
  rating: number;
  price: number;
  url?: string;
```

```typescript
}

interface EventParticipant extends BaseEntity {
  eventId: UUID;
  userId: UUID;
  status: 'invited' | 'confirmed' | 'declined' | 'maybe';
  joinedAt: Timestamp;
  responseMessage?: string;
}

interface Proposal extends BaseEntity {
  eventId: UUID;
  groupId: UUID;
  createdBy: UUID;
  type: ProposalType;
  value: string;
  status: 'active' | 'accepted' | 'rejected' | 'expired';
  metadata: Record<string, any>;
}

interface Vote extends BaseEntity {
  proposalId: UUID;
  userId: UUID;
  value: boolean;
  comment?: string;
}

interface Expense extends BaseEntity {
  title: string;
  description?: string;
  amount: number;
  currency: 'EUR' | 'USD' | 'GBP';
  category: ExpenseCategory;
  groupId: UUID;
  eventId?: UUID;
  paidBy: UUID;
```

```
  createdBy: UUID;
  date: Date;
  status: 'pending' | 'paid' | 'settled' | 'cancelled';
  receiptUrl?: string;
  metadata: Record<string, any>;
}

interface ExpenseParticipant extends BaseEntity {
  expenseId: UUID;
  userId: UUID;
  amountShare?: number;
  percentageShare?: number;
  status: 'pending' | 'paid' | 'exempt';
  paidAt?: Timestamp;
}

interface Notification extends BaseEntity {
  userId: UUID;
  type: NotificationType;
  title: string;
  message: string;
  data: Record<string, any>;
  readAt?: Timestamp;
  sentAt?: Timestamp;
  deliveryStatus: 'pending' | 'sent' | 'delivered' | 'failed';
  metadata: Record<string, any>;
}

interface AnalyticsEvent extends BaseEntity {
  userId?: UUID;
  sessionId?: string;
  eventType: string;
  eventData: Record<string, any>;
  timestamp: Timestamp;
  userAgent?: string;
  ipAddress?: string;
```

```typescript
  metadata: Record<string, any>;
}

// Utility Types
type DeepPartial<T> = {
  [P in keyof T]?: T[P] extends object ? DeepPartial<T[P]> :
        T[P];
};

type RequireFields<T, K extends keyof T> = T & { [P in K]-?:
        T[P] };

type OptionalFields<T, K extends keyof T> = Omit<T, K> &
        Partial<Pick<T, K>>;

// API Response Types
interface ApiResponse<T> {
  success: boolean;
  data?: T;
  error?: {
    code: string;
    message: string;
    details?: any;
  };
  meta?: {
    page?: number;
    limit?: number;
    total?: number;
    hasMore?: boolean;
  };
}

interface PaginatedResponse<T> {
  items: T[];
  pagination: {
    page: number;
    limit: number;
```

```
    total: number;
    totalPages: number;
    hasNext: boolean;
    hasPrev: boolean;
  };
}

// Event Types for Event Sourcing
interface DomainEvent {
  id: UUID;
  aggregateId: UUID;
  eventType: string;
  data: any;
  timestamp: Timestamp;
  version: number;
  metadata?: Record<string, any>;
}

// Command Types
interface Command {
  id: UUID;
  type: string;
  data: any;
  timestamp: Timestamp;
  userId?: UUID;
  metadata?: Record<string, any>;
}

// Query Types
interface Query {
  type: string;
  parameters: any;
  userId?: UUID;
  metadata?: Record<string, any>;
}
```

6.3 Modèles de données NoSQL (Firestore)

```typescript
// Firestore Collections Structure
interface FirestoreCollections {
  users: UserDocument;
  groups: GroupDocument;
  events: EventDocument;
  expenses: ExpenseDocument;
  notifications: NotificationDocument;
  analytics: AnalyticsDocument;
}

// User Document
interface UserDocument {
  uid: string;
  email: string;
  firstName: string;
  lastName: string;
  displayName?: string;
  avatarUrl?: string;
  phoneNumber?: string;
  dateOfBirth?: FirebaseFirestore.Timestamp;
  createdAt: FirebaseFirestore.Timestamp;
  updatedAt: FirebaseFirestore.Timestamp;
  lastLoginAt?: FirebaseFirestore.Timestamp;
  isActive: boolean;
  isVerified: boolean;
  preferences: {
    language: 'fr' | 'en' | 'es' | 'de';
    timezone: string;
    currency: 'EUR' | 'USD' | 'GBP';
    notifications: {
      email: boolean;
      push: boolean;
      sms: boolean;
    };
    privacy: {
```

```
      profileVisibility: 'public' | 'friends' | 'private';
      showEmail: boolean;
      showPhone: boolean;
    };
    theme: 'light' | 'dark' | 'auto';
  };
  metadata: Record<string, any>;
}

// Group Document
interface GroupDocument {
  id: string;
  name: string;
  description?: string;
  avatarUrl?: string;
  createdBy: string;
  createdAt: FirebaseFirestore.Timestamp;
  updatedAt: FirebaseFirestore.Timestamp;
  status: 'active' | 'archived' | 'deleted';
  members: string[]; // User IDs
  admins: string[]; // User IDs
  settings: {
    allowMemberInvites: boolean;
    requireApprovalForEvents: boolean;
    maxMembers: number;
    defaultEventVisibility: 'public' | 'private';
    autoAcceptEvents: boolean;
    notificationSettings: {
      eventCreated: boolean;
      eventUpdated: boolean;
      expenseAdded: boolean;
      voteCast: boolean;
    };
  };
  metadata: Record<string, any>;
}
```

```
// Event Document
interface EventDocument {
  id: string;
  title: string;
  description?: string;
  groupId: string;
  createdBy: string;
  createdAt: FirebaseFirestore.Timestamp;
  updatedAt: FirebaseFirestore.Timestamp;
  startDate: FirebaseFirestore.Timestamp;
  endDate?: FirebaseFirestore.Timestamp;
  location?: string;
  locationCoordinates?: {
    latitude: number;
    longitude: number;
  };
  status: 'planning' | 'voting' | 'confirmed' | 'in_progress'
        | 'completed' | 'cancelled';
  attendees: string[]; // User IDs
  maxParticipants?: number;
  isPublic: boolean;
  aiGenerated: boolean;
  plan: {
    activities: Array<{
      id: string;
      name: string;
      description: string;
      duration: number;
      cost: number;
      location?: string;
      category: string;
    }>;
    timeline: Array<{
      time: string;
      activity: string;
```

```
      location?: string;
      notes?: string;
    }>;
    budget: {
      total: number;
      perPerson: number;
      breakdown: {
        accommodation: number;
        transport: number;
        food: number;
        entertainment: number;
        other: number;
      };
    };
    recommendations: Array<{
      type: 'activity' | 'restaurant' | 'hotel' | 'transport';
      name: string;
      description: string;
      rating: number;
      price: number;
      url?: string;
    }>;
  };
  metadata: Record<string, any>;
}

// Expense Document
interface ExpenseDocument {
  id: string;
  title: string;
  description?: string;
  amount: number;
  currency: 'EUR' | 'USD' | 'GBP';
  category: 'accommodation' | 'transport' | 'food' |
        'entertainment' | 'other';
  groupId: string;
```

```
  eventId?: string;
  paidBy: string; // User ID
  createdBy: string; // User ID
  createdAt: FirebaseFirestore.Timestamp;
  updatedAt: FirebaseFirestore.Timestamp;
  date: FirebaseFirestore.Timestamp;
  status: 'pending' | 'paid' | 'settled' | 'cancelled';
  participants: string[]; // User IDs
  receiptUrl?: string;
  metadata: Record<string, any>;
}

// Notification Document
interface NotificationDocument {
  id: string;
  userId: string;
  type: 'event_invitation' | 'vote_reminder' |
        'expense_added' | 'payment_received' | 'group_update';
  title: string;
  message: string;
  data: Record<string, any>;
  createdAt: FirebaseFirestore.Timestamp;
  readAt?: FirebaseFirestore.Timestamp;
  sentAt?: FirebaseFirestore.Timestamp;
  deliveryStatus: 'pending' | 'sent' | 'delivered' | 'failed';
  metadata: Record<string, any>;
}

// Analytics Document
interface AnalyticsDocument {
  id: string;
  userId?: string;
  sessionId?: string;
  eventType: string;
  eventData: Record<string, any>;
  timestamp: FirebaseFirestore.Timestamp;
```

```
  userAgent?: string;
  ipAddress?: string;
  metadata: Record<string, any>;
}
```

[Suite dans la partie suivante...]

RAPPORT TECHNIQUE COMPLEXE - SOCIALPLANR (PARTIE 3)

Patterns architecturaux avancés et sécurité

PARTIE 3 : PATTERNS ARCHITECTURAUX AVANCÉS

7. PATTERNS ARCHITECTURAUX AVANCÉS

7.1 Domain-Driven Design (DDD)

Bounded Contexts :

```typescript
// User Management Context
namespace UserManagement {
  interface User {
    id: UserId;
    email: Email;
    profile: UserProfile;
    preferences: UserPreferences;
    security: UserSecurity;
  }

  interface UserProfile {
    firstName: string;
    lastName: string;
    displayName?: string;
    avatarUrl?: string;
    phoneNumber?: PhoneNumber;
    dateOfBirth?: Date;
  }

  interface UserPreferences {
    language: Language;
    timezone: Timezone;
    currency: Currency;
    notifications: NotificationSettings;
    privacy: PrivacySettings;
    theme: Theme;
  }

  interface UserSecurity {
    isActive: boolean;
    isVerified: boolean;
    lastLoginAt?: Date;
    failedLoginAttempts: number;
    lockedUntil?: Date;
  }
```

```
}

// Event Management Context
namespace EventManagement {
  interface Event {
    id: EventId;
    title: EventTitle;
    description?: EventDescription;
    group: GroupReference;
    organizer: UserReference;
    schedule: EventSchedule;
    location?: EventLocation;
    participants: ParticipantList;
    status: EventStatus;
    plan?: EventPlan;
  }

  interface EventSchedule {
    startDate: Date;
    endDate?: Date;
    timezone: Timezone;
  }

  interface EventLocation {
    name: string;
    address?: string;
    coordinates?: GeoCoordinates;
  }

  interface ParticipantList {
    attendees: UserReference[];
    maxParticipants?: number;
    waitlist: UserReference[];
  }
}
```

```
// Financial Management Context
namespace FinancialManagement {
  interface Expense {
    id: ExpenseId;
    title: ExpenseTitle;
    amount: Money;
    category: ExpenseCategory;
    group: GroupReference;
    event?: EventReference;
    paidBy: UserReference;
    participants: ExpenseParticipantList;
    status: ExpenseStatus;
  }

  interface Money {
    amount: number;
    currency: Currency;
  }

  interface ExpenseParticipantList {
    participants: UserReference[];
    shares: Map<UserId, Money>;
  }
}

// Collaboration Context
namespace Collaboration {
  interface Group {
    id: GroupId;
    name: GroupName;
    description?: GroupDescription;
    members: MemberList;
    settings: GroupSettings;
    status: GroupStatus;
  }
```

```
  interface MemberList {
    members: Map<UserId, MemberRole>;
    admins: UserId[];
    moderators: UserId[];
  }

  interface Proposal {
    id: ProposalId;
    type: ProposalType;
    value: ProposalValue;
    creator: UserReference;
    votes: VoteCollection;
    status: ProposalStatus;
  }

  interface VoteCollection {
    votes: Map<UserId, Vote>;
    totalVotes: number;
    acceptedVotes: number;
    rejectedVotes: number;
  }
}
```

Value Objects :

```typescript
// Value Objects Implementation
class Email {
  private constructor(private readonly value: string) {
    if (!this.isValid(value)) {
      throw new Error('Invalid email format');
    }
  }

  static create(email: string): Email {
    return new Email(email);
  }

  private isValid(email: string): boolean {
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    return emailRegex.test(email);
  }

  toString(): string {
    return this.value;
  }

  equals(other: Email): boolean {
    return this.value === other.value;
  }
}

class Money {
  constructor(
    private readonly amount: number,
    private readonly currency: Currency
  ) {
    if (amount < 0) {
      throw new Error('Amount cannot be negative');
    }
  }
```

```
  add(other: Money): Money {
    if (this.currency !== other.currency) {
      throw new Error('Cannot add money with different
        currencies');
    }
    return new Money(this.amount + other.amount,
        this.currency);
  }

  subtract(other: Money): Money {
    if (this.currency !== other.currency) {
      throw new Error('Cannot subtract money with different
        currencies');
    }
    const result = this.amount - other.amount;
    if (result < 0) {
      throw new Error('Result cannot be negative');
    }
    return new Money(result, this.currency);
  }

  multiply(factor: number): Money {
    return new Money(this.amount * factor, this.currency);
  }

  divide(divisor: number): Money {
    if (divisor === 0) {
      throw new Error('Cannot divide by zero');
    }
    return new Money(this.amount / divisor, this.currency);
  }

  toString(): string {
    return `${this.amount.toFixed(2)} ${this.currency}`;
  }
}
```

```
class GeoCoordinates {
  constructor(
    private readonly latitude: number,
    private readonly longitude: number
  ) {
    if (latitude < -90 || latitude > 90) {
      throw new Error('Invalid latitude');
    }
    if (longitude < -180 || longitude > 180) {
      throw new Error('Invalid longitude');
    }
  }

  distanceTo(other: GeoCoordinates): number {
    const R = 6371; // Earth's radius in kilometers
    const dLat = this.toRadians(other.latitude -
        this.latitude);
    const dLon = this.toRadians(other.longitude -
        this.longitude);
    const a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
              Math.cos(this.toRadians(this.latitude)) *
        Math.cos(this.toRadians(other.latitude)) *
              Math.sin(dLon / 2) * Math.sin(dLon / 2);
    const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    return R * c;
  }

  private toRadians(degrees: number): number {
    return degrees * (Math.PI / 180);
  }

  toString(): string {
    return `${this.latitude}, ${this.longitude}`;
  }
}
```

7.2 Repository Pattern avec Caching

```typescript
// Repository Interface
interface Repository<T, ID> {
  findById(id: ID): Promise<T | null>;
  findAll(): Promise<T[]>;
  save(entity: T): Promise<T>;
  update(id: ID, entity: Partial<T>): Promise<T>;
  delete(id: ID): Promise<void>;
  exists(id: ID): Promise<boolean>;
}

// Cached Repository Implementation
class CachedRepository<T, ID> implements Repository<T, ID> {
  constructor(
    private readonly repository: Repository<T, ID>,
    private readonly cache: Cache<T, ID>,
    private readonly ttl: number = 300000 // 5 minutes
  ) {}

  async findById(id: ID): Promise<T | null> {
    // Try cache first
    const cached = await this.cache.get(id);
    if (cached) {
      return cached;
    }

    // Fetch from repository
    const entity = await this.repository.findById(id);
    if (entity) {
      await this.cache.set(id, entity, this.ttl);
    }

    return entity;
  }

  async findAll(): Promise<T[]> {
```

```
    // For collections, we might want to cache differently
    return await this.repository.findAll();
  }

  async save(entity: T): Promise<T> {
    const saved = await this.repository.save(entity);
    await this.cache.set(saved.id, saved, this.ttl);
    return saved;
  }

  async update(id: ID, entity: Partial<T>): Promise<T> {
    const updated = await this.repository.update(id, entity);
    await this.cache.set(id, updated, this.ttl);
    return updated;
  }

  async delete(id: ID): Promise<void> {
    await this.repository.delete(id);
    await this.cache.delete(id);
  }

  async exists(id: ID): Promise<boolean> {
    const cached = await this.cache.get(id);
    if (cached !== null) {
      return true;
    }
    return await this.repository.exists(id);
  }
}

// User Repository Implementation
class UserRepository implements Repository<User, UserId> {
  constructor(private readonly db: Database) {}

  async findById(id: UserId): Promise<User | null> {
    const result = await this.db.query(
```

```
      'SELECT * FROM users WHERE id = $1',
      [id]
    );
    return result.rows[0] || null;
  }

  async findAll(): Promise<User[]> {
    const result = await this.db.query('SELECT * FROM users');
    return result.rows;
  }

  async save(user: User): Promise<User> {
    const result = await this.db.query(
      `INSERT INTO users (email, first_name, last_name,
        display_name, avatar_url,
                          phone_number, date_of_birth,
        created_at, updated_at,
                          last_login_at, is_active,
        is_verified, preferences, metadata)
       VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11,
        $12, $13, $14)
       RETURNING *`,
      [
        user.email,
        user.firstName,
        user.lastName,
        user.displayName,
        user.avatarUrl,
        user.phoneNumber,
        user.dateOfBirth,
        user.createdAt,
        user.updatedAt,
        user.lastLoginAt,
        user.isActive,
        user.isVerified,
        JSON.stringify(user.preferences),
        JSON.stringify(user.metadata)
```

```
      ]
    );
    return result.rows[0];
  }

  async update(id: UserId, user: Partial<User>):
      Promise<User> {
    const fields = Object.keys(user).filter(key => key !==
      'id');
    const values = Object.values(user);
    const setClause = fields.map((field, index) => `$
      {field} = $${index + 2}`).join(', ');

    const result = await this.db.query(
      `UPDATE users SET ${setClause}, updated_at = NOW()
       WHERE id = $1 RETURNING *`,
      [id, ...values]
    );
    return result.rows[0];
  }

  async delete(id: UserId): Promise<void> {
    await this.db.query('DELETE FROM users WHERE id = $1',
      [id]);
  }

  async exists(id: UserId): Promise<boolean> {
    const result = await this.db.query(
      'SELECT EXISTS(SELECT 1 FROM users WHERE id = $1)',
      [id]
    );
    return result.rows[0].exists;
  }

  // Custom queries
  async findByEmail(email: Email): Promise<User | null> {
    const result = await this.db.query(
```

```
      'SELECT * FROM users WHERE email = $1',
      [email.toString()]
    );
    return result.rows[0] || null;
  }

  async findActiveUsers(): Promise<User[]> {
    const result = await this.db.query(
      'SELECT * FROM users WHERE is_active = true'
    );
    return result.rows;
  }

  async updateLastLogin(id: UserId): Promise<void> {
    await this.db.query(
      'UPDATE users SET last_login_at = NOW() WHERE id = $1',
      [id]
    );
  }
}
```

7.3 Service Layer avec Business Logic

```typescript
// Service Interface
interface Service<T, ID> {
  getById(id: ID): Promise<T>;
  getAll(): Promise<T[]>;
  create(data: CreateDto<T>): Promise<T>;
  update(id: ID, data: UpdateDto<T>): Promise<T>;
  delete(id: ID): Promise<void>;
}

// User Service Implementation
class UserService implements Service<User, UserId> {
  constructor(
    private readonly userRepository: UserRepository,
    private readonly eventBus: EventBus,
    private readonly passwordHasher: PasswordHasher,
    private readonly emailService: EmailService,
    private readonly validator: Validator
  ) {}

  async getById(id: UserId): Promise<User> {
    const user = await this.userRepository.findById(id);
    if (!user) {
      throw new UserNotFoundError(id);
    }
    return user;
  }

  async getAll(): Promise<User[]> {
    return await this.userRepository.findAll();
  }

  async create(data: CreateUserDto): Promise<User> {
    // Validate input
    await this.validator.validate(data);
```

```
    // Check if email already exists
    const existingUser = await
        this.userRepository.findByEmail(data.email);
    if (existingUser) {
      throw new EmailAlreadyExistsError(data.email);
    }

    // Hash password
    const hashedPassword = await
        this.passwordHasher.hash(data.password);

    // Create user
    const user: User = {
      id: generateId(),
      email: Email.create(data.email),
      firstName: data.firstName,
      lastName: data.lastName,
      displayName: data.displayName,
      avatarUrl: data.avatarUrl,
      phoneNumber: data.phoneNumber,
      dateOfBirth: data.dateOfBirth,
      createdAt: new Date(),
      updatedAt: new Date(),
      isActive: true,
      isVerified: false,
      preferences: {
        language: 'fr',
        timezone: 'Europe/Paris',
        currency: 'EUR',
        notifications: {
          email: true,
          push: true,
          sms: false
        },
        privacy: {
          profileVisibility: 'friends',
          showEmail: false,
```

```
          showPhone: false
        },
        theme: 'auto'
      },
      metadata: {}
    };

    // Save to repository
    const savedUser = await this.userRepository.save(user);

    // Publish event
    await this.eventBus.publish('user.created', {
      userId: savedUser.id,
      email: savedUser.email.toString(),
      timestamp: new Date()
    });

    // Send welcome email
    await this.emailService.sendWelcomeEmail(savedUser.email);

    return savedUser;
  }

  async update(id: UserId, data: UpdateUserDto):
        Promise<User> {
    // Validate input
    await this.validator.validate(data);

    // Get existing user
    const existingUser = await this.getById(id);

    // Check email uniqueness if email is being updated
    if (data.email && data.email !==
        existingUser.email.toString()) {
      const userWithEmail = await
        this.userRepository.findByEmail(data.email);
      if (userWithEmail) {
```

```
        throw new EmailAlreadyExistsError(data.email);
      }
    }

    // Update user
    const updatedUser = await this.userRepository.update(id, {
      ...data,
      updatedAt: new Date()
    });

    // Publish event
    await this.eventBus.publish('user.updated', {
      userId: id,
      changes: data,
      timestamp: new Date()
    });

    return updatedUser;
  }

  async delete(id: UserId): Promise<void> {
    const user = await this.getById(id);

    // Soft delete
    await this.userRepository.update(id, {
      isActive: false,
      updatedAt: new Date()
    });

    // Publish event
    await this.eventBus.publish('user.deleted', {
      userId: id,
      timestamp: new Date()
    });
  }

```

```
  // Business logic methods
  async updateLastLogin(id: UserId): Promise<void> {
    await this.userRepository.updateLastLogin(id);

    await this.eventBus.publish('user.logged_in', {
      userId: id,
      timestamp: new Date()
    });
  }

  async verifyEmail(id: UserId): Promise<void> {
    await this.userRepository.update(id, {
      isVerified: true,
      updatedAt: new Date()
    });

    await this.eventBus.publish('user.verified', {
      userId: id,
      timestamp: new Date()
    });
  }

  async resetPassword(email: Email): Promise<void> {
    const user = await this.userRepository.findByEmail(email);
    if (!user) {
      // Don't reveal if user exists
      return;
    }

    const resetToken = generateResetToken();
    await this.userRepository.update(user.id, {
      resetToken,
      resetTokenExpiresAt: new Date(Date.now() +
        3600000) // 1 hour
    });
```

```
    await this.emailService.sendPasswordResetEmail(email,
        resetToken);
  }
}
```

7.4 Factory Pattern pour la création d’entités

```typescript
// Entity Factory Interface
interface EntityFactory<T, D> {
  create(data: D): T;
  createFromExisting(entity: T, data: Partial<D>): T;
}

// User Factory Implementation
class UserFactory implements EntityFactory<User,
        CreateUserDto> {
  create(data: CreateUserDto): User {
    return {
      id: generateId(),
      email: Email.create(data.email),
      firstName: data.firstName,
      lastName: data.lastName,
      displayName: data.displayName,
      avatarUrl: data.avatarUrl,
      phoneNumber: data.phoneNumber,
      dateOfBirth: data.dateOfBirth,
      createdAt: new Date(),
      updatedAt: new Date(),
      isActive: true,
      isVerified: false,
      preferences: this.createDefaultPreferences(),
      metadata: {}
    };
  }

  createFromExisting(user: User, data:
        Partial<CreateUserDto>): User {
    return {
      ...user,
      ...data,
      updatedAt: new Date()
    };
  }
```

```
  private createDefaultPreferences(): UserPreferences {
    return {
      language: 'fr',
      timezone: 'Europe/Paris',
      currency: 'EUR',
      notifications: {
        email: true,
        push: true,
        sms: false
      },
      privacy: {
        profileVisibility: 'friends',
        showEmail: false,
        showPhone: false
      },
      theme: 'auto'
    };
  }
}

// Event Factory Implementation
class EventFactory implements EntityFactory<Event,
        CreateEventDto> {
  create(data: CreateEventDto): Event {
    return {
      id: generateId(),
      title: data.title,
      description: data.description,
      groupId: data.groupId,
      createdBy: data.createdBy,
      startDate: data.startDate,
      endDate: data.endDate,
      location: data.location,
      locationCoordinates: data.locationCoordinates,
      status: EventStatus.PLANNING,
```

```
      attendees: [data.createdBy],
      maxParticipants: data.maxParticipants,
      isPublic: data.isPublic || false,
      aiGenerated: false,
      plan: this.createEmptyPlan(),
      createdAt: new Date(),
      updatedAt: new Date(),
      metadata: {}
    };
  }

  createFromExisting(event: Event, data:
      Partial<CreateEventDto>): Event {
    return {
      ...event,
      ...data,
      updatedAt: new Date()
    };
  }

  private createEmptyPlan(): EventPlan {
    return {
      activities: [],
      timeline: [],
      budget: {
        total: 0,
        perPerson: 0,
        breakdown: {
          accommodation: 0,
          transport: 0,
          food: 0,
          entertainment: 0,
          other: 0
        }
      },
      recommendations: []
```

```
        };
    }
}
```

8. SÉCURITÉ ET CONFORMITÉ

8.1 Architecture de sécurité multi-couches

```typescript
// Security Layer Architecture
interface SecurityLayer {
  authentication: AuthenticationService;
  authorization: AuthorizationService;
  encryption: EncryptionService;
  audit: AuditService;
  compliance: ComplianceService;
}

// Authentication Service
class AuthenticationService {
  constructor(
    private readonly userRepository: UserRepository,
    private readonly passwordHasher: PasswordHasher,
    private readonly jwtService: JwtService,
    private readonly rateLimiter: RateLimiter,
    private readonly auditService: AuditService
  ) {}

  async login(email: Email, password: string, deviceInfo:
      DeviceInfo): Promise<AuthResult> {
    // Rate limiting
    await this.rateLimiter.checkLimit(`login:$
      {email.toString()}`, 5, 300000); // 5 attempts per 5
      minutes

    // Find user
    const user = await this.userRepository.findByEmail(email);
    if (!user) {
      await this.auditService.logFailedLogin(email, 'User not
        found', deviceInfo);
      throw new AuthenticationError('Invalid credentials');
    }

    // Check if account is locked
    if (user.security.lockedUntil &&
      user.security.lockedUntil > new Date()) {
```

```
        await this.auditService.logFailedLogin(email, 'Account
          locked', deviceInfo);
        throw new AccountLockedError(user.security.lockedUntil);
      }

      // Verify password
      const isValid = await
          this.passwordHasher.verify(password,
          user.passwordHash);
      if (!isValid) {
        await this.handleFailedLogin(user);
        throw new AuthenticationError('Invalid credentials');
      }

      // Check if account is active
      if (!user.isActive) {
        await this.auditService.logFailedLogin(email, 'Account
          inactive', deviceInfo);
        throw new AccountInactiveError();
      }

      // Generate JWT token
      const token = await this.jwtService.generateToken({
        userId: user.id,
        email: user.email.toString(),
        roles: user.roles
      });

      // Update last login
      await this.userRepository.updateLastLogin(user.id);

      // Reset failed login attempts
      await this.userRepository.update(user.id, {
        security: {
          ...user.security,
          failedLoginAttempts: 0,
          lockedUntil: null
```

```
      }
    });

    // Log successful login
    await this.auditService.logSuccessfulLogin(user.id,
        deviceInfo);

    return {
      token,
      user: this.sanitizeUser(user),
      expiresAt: new Date(Date.now() + 3600000) // 1 hour
    };
  }

  async register(data: RegisterDto, deviceInfo: DeviceInfo):
        Promise<AuthResult> {
    // Validate input
    await this.validateRegistrationData(data);

    // Check if email exists
    const existingUser = await
        this.userRepository.findByEmail(data.email);
    if (existingUser) {
      throw new EmailAlreadyExistsError(data.email);
    }

    // Hash password
    const passwordHash = await
        this.passwordHasher.hash(data.password);

    // Create user
    const user = await this.userService.create({
      ...data,
      passwordHash
    });

    // Generate verification token
```

```
    const verificationToken = await
        this.jwtService.generateVerificationToken(user.id);

    // Send verification email
    await this.emailService.sendVerificationEmail(user.email,
        verificationToken);

    // Log registration
    await this.auditService.logRegistration(user.id,
        deviceInfo);

    // Generate session token
    const token = await this.jwtService.generateToken({
      userId: user.id,
      email: user.email.toString(),
      roles: user.roles
    });

    return {
      token,
      user: this.sanitizeUser(user),
      expiresAt: new Date(Date.now() + 3600000)
    };
  }

  async refreshToken(refreshToken: string):
        Promise<AuthResult> {
    const payload = await
        this.jwtService.verifyRefreshToken(refreshToken);
    const user = await
        this.userRepository.findById(payload.userId);

    if (!user || !user.isActive) {
      throw new AuthenticationError('Invalid refresh token');
    }

    const newToken = await this.jwtService.generateToken({
      userId: user.id,
```

```
      email: user.email.toString(),
      roles: user.roles
    });

    return {
      token: newToken,
      user: this.sanitizeUser(user),
      expiresAt: new Date(Date.now() + 3600000)
    };
  }

  async logout(token: string, deviceInfo: DeviceInfo):
      Promise<void> {
    const payload = await this.jwtService.verifyToken(token);
    await this.jwtService.blacklistToken(token);
    await this.auditService.logLogout(payload.userId,
      deviceInfo);
  }

  private async handleFailedLogin(user: User): Promise<void> {
    const newAttempts = user.security.failedLoginAttempts + 1;
    const updates: Partial<User> = {
      security: {
        ...user.security,
        failedLoginAttempts: newAttempts
      }
    };

    // Lock account after 5 failed attempts
    if (newAttempts >= 5) {
      updates.security.lockedUntil = new Date(Date.now() +
        900000); // 15 minutes
    }

    await this.userRepository.update(user.id, updates);
  }
```

```
  private sanitizeUser(user: User): SanitizedUser {
    return {
      id: user.id,
      email: user.email.toString(),
      firstName: user.firstName,
      lastName: user.lastName,
      displayName: user.displayName,
      avatarUrl: user.avatarUrl,
      isVerified: user.isVerified,
      preferences: user.preferences
    };
  }
}

// Authorization Service
class AuthorizationService {
  constructor(
    private readonly roleRepository: RoleRepository,
    private readonly permissionRepository:
        PermissionRepository
  ) {}

  async hasPermission(userId: UserId, resource: string,
        action: string): Promise<boolean> {
    const user = await this.userRepository.findById(userId);
    if (!user) {
      return false;
    }

    const roles = await
        this.roleRepository.findByUserId(userId);
    const permissions = await
        this.permissionRepository.findByRoles(roles.map(r =>
        r.id));

    return permissions.some(p =>
      p.resource === resource && p.action === action
    );
```

```
  }

  async hasRole(userId: UserId, role: string):
        Promise<boolean> {
    const user = await this.userRepository.findById(userId);
    if (!user) {
      return false;
    }

    const roles = await
        this.roleRepository.findByUserId(userId);
    return roles.some(r => r.name === role);
  }

  async canAccessEvent(userId: UserId, eventId: EventId):
        Promise<boolean> {
    const event = await
        this.eventRepository.findById(eventId);
    if (!event) {
      return false;
    }

    // Event creator can always access
    if (event.createdBy === userId) {
      return true;
    }

    // Check if user is a participant
    const participant = await
        this.eventParticipantRepository.findByEventAndUser(eventId,
        userId);
    if (participant) {
      return true;
    }

    // Check if user is admin of the group
    const groupMember = await
        this.groupMemberRepository.findByGroupAndUser(event.groupId,
        userId);
```

```
    if (groupMember && groupMember.role === UserRole.ADMIN) {
      return true;
    }

    return false;
  }

  async canModifyEvent(userId: UserId, eventId: EventId):
        Promise<boolean> {
    const event = await
        this.eventRepository.findById(eventId);
    if (!event) {
      return false;
    }

    // Only event creator can modify
    return event.createdBy === userId;
  }

  async canAccessExpense(userId: UserId, expenseId:
        ExpenseId): Promise<boolean> {
    const expense = await
        this.expenseRepository.findById(expenseId);
    if (!expense) {
      return false;
    }

    // Expense creator can always access
    if (expense.createdBy === userId) {
      return true;
    }

    // Check if user is a participant
    const participant = await
        this.expenseParticipantRepository.findByExpenseAndUser(expenseId,
        userId);
    if (participant) {
      return true;
```

```
    }

    return false;
  }
}

// Encryption Service
class EncryptionService {
  constructor(
    private readonly algorithm: string = 'aes-256-gcm',
    private readonly keyLength: number = 32,
    private readonly ivLength: number = 16
  ) {}

  async encrypt(data: string, key: Buffer):
        Promise<EncryptedData> {
    const iv = crypto.randomBytes(this.ivLength);
    const cipher = crypto.createCipher(this.algorithm, key);

    let encrypted = cipher.update(data, 'utf8', 'hex');
    encrypted += cipher.final('hex');

    const authTag = cipher.getAuthTag();

    return {
      encrypted,
      iv: iv.toString('hex'),
      authTag: authTag.toString('hex')
    };
  }

  async decrypt(encryptedData: EncryptedData, key: Buffer):
        Promise<string> {
    const decipher = crypto.createDecipher(
      this.algorithm,
      key,
      Buffer.from(encryptedData.iv, 'hex')
```

```
    );

    decipher.setAuthTag(Buffer.from(encryptedData.authTag,
        'hex'));

    let decrypted = decipher.update(encryptedData.encrypted,
        'hex', 'utf8');
    decrypted += decipher.final('utf8');

    return decrypted;
  }

  async hashPassword(password: string): Promise<string> {
    const saltRounds = 12;
    return await bcrypt.hash(password, saltRounds);
  }

  async verifyPassword(password: string, hash: string):
        Promise<boolean> {
    return await bcrypt.compare(password, hash);
  }

  generateSecureToken(): string {
    return crypto.randomBytes(32).toString('hex');
  }
}

// Audit Service
class AuditService {
  constructor(
    private readonly auditRepository: AuditRepository,
    private readonly eventBus: EventBus
  ) {}

  async logAction(action: AuditAction): Promise<void> {
    const auditEntry: AuditEntry = {
      id: generateId(),
```

```
      userId: action.userId,
      action: action.type,
      resource: action.resource,
      resourceId: action.resourceId,
      details: action.details,
      ipAddress: action.ipAddress,
      userAgent: action.userAgent,
      timestamp: new Date(),
      metadata: action.metadata
    };

    await this.auditRepository.save(auditEntry);

    // Publish audit event
    await this.eventBus.publish('audit.action', auditEntry);
  }

  async logLogin(userId: UserId, success: boolean,
      deviceInfo: DeviceInfo): Promise<void> {
    await this.logAction({
      type: success ? 'LOGIN_SUCCESS' : 'LOGIN_FAILURE',
      userId,
      resource: 'auth',
      resourceId: userId,
      details: {
        success,
        deviceInfo
      },
      ipAddress: deviceInfo.ipAddress,
      userAgent: deviceInfo.userAgent,
      metadata: {
        deviceType: deviceInfo.deviceType,
        location: deviceInfo.location
      }
    });
  }
```

```
  async logDataAccess(userId: UserId, resource: string,
        resourceId: string): Promise<void> {
    await this.logAction({
      type: 'DATA_ACCESS',
      userId,
      resource,
      resourceId,
      details: {
        accessType: 'read'
      },
      ipAddress: 'unknown',
      userAgent: 'unknown',
      metadata: {}
    });
  }

  async logDataModification(userId: UserId, resource: string,
        resourceId: string, changes: any): Promise<void> {
    await this.logAction({
      type: 'DATA_MODIFICATION',
      userId,
      resource,
      resourceId,
      details: {
        changes,
        modificationType: 'update'
      },
      ipAddress: 'unknown',
      userAgent: 'unknown',
      metadata: {}
    });
  }
}
```

8.2 Conformité RGPD/GDPR

```typescript
// GDPR Compliance Service
class GDPRComplianceService {
  constructor(
    private readonly userRepository: UserRepository,
    private readonly dataProcessor: DataProcessor,
    private readonly auditService: AuditService
  ) {}

  async exportUserData(userId: UserId):
        Promise<UserDataExport> {
    const user = await this.userRepository.findById(userId);
    if (!user) {
      throw new UserNotFoundError(userId);
    }

    // Collect all user data
    const userData = await this.collectUserData(userId);

    // Log data export
    await this.auditService.logAction({
      type: 'DATA_EXPORT',
      userId,
      resource: 'user',
      resourceId: userId,
      details: {
        exportType: 'gdpr',
        dataTypes: Object.keys(userData)
      },
      ipAddress: 'unknown',
      userAgent: 'unknown',
      metadata: {}
    });

    return {
      userId,
```

```
      exportedAt: new Date(),
      data: userData,
      format: 'json'
    };
  }

  async deleteUserData(userId: UserId): Promise<void> {
    const user = await this.userRepository.findById(userId);
    if (!user) {
      throw new UserNotFoundError(userId);
    }

    // Anonymize user data
    await this.anonymizeUserData(userId);

    // Log data deletion
    await this.auditService.logAction({
      type: 'DATA_DELETION',
      userId,
      resource: 'user',
      resourceId: userId,
      details: {
        deletionType: 'gdpr_right_to_be_forgotten'
      },
      ipAddress: 'unknown',
      userAgent: 'unknown',
      metadata: {}
    });
  }

  async updateConsent(userId: UserId, consent:
      ConsentSettings): Promise<void> {
    await this.userRepository.update(userId, {
      consent: {
        marketing: consent.marketing,
        analytics: consent.analytics,
```

```
        thirdParty: consent.thirdParty,
        updatedAt: new Date()
      }
    });

    await this.auditService.logAction({
      type: 'CONSENT_UPDATE',
      userId,
      resource: 'user',
      resourceId: userId,
      details: {
        consent
      },
      ipAddress: 'unknown',
      userAgent: 'unknown',
      metadata: {}
    });
  }

  private async collectUserData(userId: UserId):
        Promise<UserDataExport> {
    const user = await this.userRepository.findById(userId);
    const groups = await
        this.groupRepository.findByUserId(userId);
    const events = await
        this.eventRepository.findByUserId(userId);
    const expenses = await
        this.expenseRepository.findByUserId(userId);
    const notifications = await
        this.notificationRepository.findByUserId(userId);

    return {
      profile: {
        id: user.id,
        email: user.email.toString(),
        firstName: user.firstName,
        lastName: user.lastName,
        displayName: user.displayName,
```

```
        avatarUrl: user.avatarUrl,
        phoneNumber: user.phoneNumber,
        dateOfBirth: user.dateOfBirth,
        createdAt: user.createdAt,
        lastLoginAt: user.lastLoginAt
      },
      preferences: user.preferences,
      groups: groups.map(g => ({
        id: g.id,
        name: g.name,
        role: g.role,
        joinedAt: g.joinedAt
      })),
      events: events.map(e => ({
        id: e.id,
        title: e.title,
        startDate: e.startDate,
        status: e.status
      })),
      expenses: expenses.map(ex => ({
        id: ex.id,
        title: ex.title,
        amount: ex.amount,
        category: ex.category,
        date: ex.date
      })),
      notifications: notifications.map(n => ({
        id: n.id,
        type: n.type,
        title: n.title,
        createdAt: n.createdAt
      }))
    };
  }
```

```
  private async anonymizeUserData(userId: UserId):
        Promise<void> {
    const anonymizedData = {
      email: `deleted_${userId}@deleted.com`,
      firstName: 'Deleted',
      lastName: 'User',
      displayName: 'Deleted User',
      avatarUrl: null,
      phoneNumber: null,
      dateOfBirth: null,
      isActive: false,
      isVerified: false,
      preferences: {
        language: 'en',
        timezone: 'UTC',
        currency: 'EUR',
        notifications: {
          email: false,
          push: false,
          sms: false
        },
        privacy: {
          profileVisibility: 'private',
          showEmail: false,
          showPhone: false
        },
        theme: 'light'
      },
      metadata: {
        anonymizedAt: new Date(),
        reason: 'gdpr_deletion'
      }
    };

    await this.userRepository.update(userId, anonymizedData);
  }
```

```
}

// Data Processing Agreement
interface DataProcessingAgreement {
  id: string;
  version: string;
  effectiveDate: Date;
  dataController: {
    name: string;
    address: string;
    contactEmail: string;
  };
  dataProcessor: {
    name: string;
    address: string;
    contactEmail: string;
  };
  purposes: string[];
  dataTypes: string[];
  dataSubjects: string[];
  retentionPeriod: string;
  securityMeasures: string[];
  subProcessors: SubProcessor[];
  rights: DataSubjectRights;
}

interface SubProcessor {
  name: string;
  purpose: string;
  location: string;
  dataTypes: string[];
}

interface DataSubjectRights {
  access: boolean;
  rectification: boolean;
```

```
  erasure: boolean;
  portability: boolean;
  objection: boolean;
  restriction: boolean;
}
```

[Suite dans la partie suivante...]

RAPPORT TECHNIQUE COMPLEXE - SOCIALPLANR (PARTIE 4)

Développement et implémentation avancée

PARTIE 4 : DÉVELOPPEMENT ET IMPLÉMENTATION

9. STACK TECHNOLOGIQUE DÉTAILLÉE

9.1 Architecture Frontend Mobile (React Native + Expo)

Configuration Expo SDK 53 :

```
{
  "expo": {
    "name": "SocialPlanr",
    "slug": "socialplanr",
    "version": "1.0.0",
    "orientation": "portrait",
    "icon": "./assets/icon.png",
    "userInterfaceStyle": "automatic",
    "splash": {
      "image": "./assets/splash.png",
      "resizeMode": "contain",
      "backgroundColor": "#ffffff"
    },
    "assetBundlePatterns": [
      "**/*"
    ],
    "ios": {
      "supportsTablet": true,
      "bundleIdentifier": "com.socialplanr.app",
      "buildNumber": "1.0.0",
      "infoPlist": {
        "NSCameraUsageDescription": "Cette app utilise la
        caméra pour prendre des photos de profil",
        "NSPhotoLibraryUsageDescription": "Cette app accède à
        votre galerie pour sélectionner des photos",
        "NSLocationWhenInUseUsageDescription": "Cette app
        utilise votre localisation pour suggérer des lieux
        d'événements"
      }
    },
    "android": {
      "adaptiveIcon": {
        "foregroundImage": "./assets/adaptive-icon.png",
        "backgroundColor": "#FFFFFF"
      },
      "package": "com.socialplanr.app",
```

```
      "versionCode": 1,
      "permissions": [
        "CAMERA",
        "READ_EXTERNAL_STORAGE",
        "WRITE_EXTERNAL_STORAGE",
        "ACCESS_FINE_LOCATION",
        "ACCESS_COARSE_LOCATION",
        "VIBRATE",
        "RECEIVE_BOOT_COMPLETED",
        "WAKE_LOCK"
      ]
    },
    "web": {
      "favicon": "./assets/favicon.png",
      "bundler": "metro"
    },
    "plugins": [
      "expo-router",
      "expo-font",
      "expo-splash-screen",
      "expo-status-bar",
      "expo-notifications",
      "expo-calendar",
      "expo-location",
      "expo-image-picker",
      "expo-camera",
      "expo-haptics",
      "expo-blur",
      "expo-linear-gradient",
      "expo-av",
      "expo-file-system",
      "expo-secure-store",
      "expo-device",
      "expo-constants",
      "expo-linking",
      "expo-web-browser",
```

```
      "expo-auth-session",
      "expo-crypto",
      "expo-random",
      "expo-sqlite",
      "expo-network",
      "expo-battery",
      "expo-brightness",
      "expo-sensors",
      "expo-task-manager",
      "expo-background-fetch",
      "expo-location",
      "expo-permissions",
      "expo-media-library",
      "expo-document-picker",
      "expo-sharing",
      "expo-print",
      "expo-speech",
      "expo-speech-to-text",
      "expo-text-to-speech",
      "expo-barcode-scanner",
      "expo-face-detector",
      "expo-gl",
      "expo-gl-cpp",
      "expo-three",
      "expo-three-orbit-controls",
      "expo-three-text",
      "expo-three-texture-loader",
      "expo-three-fbx-loader",
      "expo-three-obj-loader",
      "expo-three-mtl-loader",
      "expo-three-collada-loader",
      "expo-three-stl-loader",
      "expo-three-plt-loader",
      "expo-three-3ds-loader",
      "expo-three-amf-loader",
      "expo-three-assimp-loader",
```

```
      "expo-three-dae-loader",
      "expo-three-ctm-loader",
      "expo-three-lwo-loader",
      "expo-three-md2-loader",
      "expo-three-pdb-loader",
      "expo-three-prwm-loader",
      "expo-three-stl-loader",
      "expo-three-vrm-loader",
      "expo-three-vtk-loader",
      "expo-three-x-loader"
    ],
    "scheme": "socialplanr",
    "extra": {
      "router": {
        "origin": false
      },
      "eas": {
        "projectId": "your-project-id"
      }
    }
  }
}
```

Configuration Metro Bundler :

```
// metro.config.js
const { getDefaultConfig } = require('expo/metro-config');
const { withNativeWind } = require('nativewind/metro');

const config = getDefaultConfig(__dirname);

// Optimisations pour les performances
config.resolver.platforms = ['ios', 'android', 'native',
        'web'];
config.transformer.minifierConfig = {
  keep_fnames: true,
  mangle: {
    keep_fnames: true,
  },
};

// Support des assets
config.resolver.assetExts.push('db', 'mp3', 'ttf', 'obj',
        'png', 'jpg');

// Cache configuration
config.cacheStores = [
  {
    name: 'metro-cache',
    type: 'file',
    options: {
      maxAge: 24 * 60 * 60 * 1000, // 24 hours
    },
  },
];

module.exports = withNativeWind(config, { input: './
        global.css' });
```

Configuration TypeScript avancée :

```jsonc
{
  "compilerOptions": {
    "target": "ES2020",
    "lib": ["ES2020", "DOM"],
    "allowJs": true,
    "skipLibCheck": true,
    "esModuleInterop": true,
    "allowSyntheticDefaultImports": true,
    "strict": true,
    "forceConsistentCasingInFileNames": true,
    "moduleResolution": "node",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react-jsx",
    "baseUrl": ".",
    "paths": {
      "@/*": ["./src/*"],
      "@/components/*": ["./src/components/*"],
      "@/screens/*": ["./src/screens/*"],
      "@/services/*": ["./src/services/*"],
      "@/utils/*": ["./src/utils/*"],
      "@/types/*": ["./src/types/*"],
      "@/hooks/*": ["./src/hooks/*"],
      "@/constants/*": ["./src/constants/*"],
      "@/assets/*": ["./src/assets/*"]
    },
    "types": [
      "react",
      "react-native",
      "jest",
      "node"
    ],
    "plugins": [
      {
```

```
        "name": "typescript-plugin-css-modules"
      }
    ]
  },
  "include": [
    "**/*.ts",
    "**/*.tsx",
    "**/*.js",
    "**/*.jsx"
  ],
  "exclude": [
    "node_modules",
    "babel.config.js",
    "metro.config.js",
    "jest.config.js"
  ]
}
```

9.2 Architecture Frontend Web (Next.js 15)

Configuration Next.js avancée :

```
// next.config.ts
import type { NextConfig } from 'next';

const nextConfig: NextConfig = {
  experimental: {
    appDir: true,
    serverComponentsExternalPackages: ['@prisma/client'],
    turbo: {
      rules: {
        '*.svg': {
          loaders: ['@svgr/webpack'],
          as: '*.js',
        },
      },
    },
  },

  // Optimisations de performance
  compress: true,
  poweredByHeader: false,
  generateEtags: false,

  // Configuration des images
  images: {
    domains: [
      'firebasestorage.googleapis.com',
      'lh3.googleusercontent.com',
      'graph.facebook.com',
      'platform-lookaside.fbsbx.com'
    ],
    formats: ['image/webp', 'image/avif'],
    deviceSizes: [640, 750, 828, 1080, 1200, 1920, 2048,
        3840],
    imageSizes: [16, 32, 48, 64, 96, 128, 256, 384],
  },
```

```
  // Configuration des headers de sécurité
  async headers() {
    return [
      {
        source: '/(.*)',
        headers: [
          {
            key: 'X-Frame-Options',
            value: 'DENY',
          },
          {
            key: 'X-Content-Type-Options',
            value: 'nosniff',
          },
          {
            key: 'Referrer-Policy',
            value: 'origin-when-cross-origin',
          },
          {
            key: 'X-XSS-Protection',
            value: '1; mode=block',
          },
          {
            key: 'Strict-Transport-Security',
            value: 'max-age=31536000; includeSubDomains',
          },
          {
            key: 'Content-Security-Policy',
```

```
            value: "default-src 'self'; script-src 'self'
          'unsafe-eval' 'unsafe-inline' https://
          www.googletagmanager.com https://www.google-
          analytics.com; style-src 'self' 'unsafe-inline'
          https://fonts.googleapis.com; font-src 'self' https://
          fonts.gstatic.com; img-src 'self' data: https: blob:;
          connect-src 'self' https://api.socialplanr.com
          https://www.google-analytics.com; frame-src 'self'
          https://www.google.com; object-src 'none'; base-uri
          'self'; form-action 'self';",
          },
        ],
      },
    ];
  },

  // Configuration des redirections
  async redirects() {
    return [
      {
        source: '/home',
        destination: '/',
        permanent: true,
      },
      {
        source: '/app',
        destination: '/dashboard',
        permanent: true,
      },
    ];
  },

  // Configuration des rewrites
  async rewrites() {
    return [
      {
        source: '/api/:path*',
        destination: 'https://api.socialplanr.com/:path*',
```

```
      },
    ];
  },

  // Configuration webpack
  webpack: (config, { buildId, dev, isServer, defaultLoaders,
        webpack }) => {
    // Optimisations pour les performances
    config.optimization = {
      ...config.optimization,
      splitChunks: {
        chunks: 'all',
        cacheGroups: {
          vendor: {
            test: /[\\/]node_modules[\\/]/,
            name: 'vendors',
            chunks: 'all',
          },
          common: {
            name: 'common',
            minChunks: 2,
            chunks: 'all',
            enforce: true,
          },
        },
      },
    };

    // Support des modules CSS
    config.module.rules.push({
      test: /\.css$/,
      use: ['style-loader', 'css-loader', 'postcss-loader'],
    });

    // Support des fichiers SVG
    config.module.rules.push({
```

```
      test: /\.svg$/,
      use: ['@svgr/webpack'],
    });

    return config;
  },

  // Configuration PWA
  pwa: {
    dest: 'public',
    register: true,
    skipWaiting: true,
    disable: process.env.NODE_ENV === 'development',
  },
};

export default nextConfig;
```

Configuration Tailwind CSS avancée :

```
// tailwind.config.js
const defaultTheme = require('tailwindcss/defaultTheme');

module.exports = {
  content: [
    './src/pages/**/*.{js,ts,jsx,tsx,mdx}',
    './src/components/**/*.{js,ts,jsx,tsx,mdx}',
    './src/app/**/*.{js,ts,jsx,tsx,mdx}',
  ],

  theme: {
    extend: {
      colors: {
        primary: {
          50: '#eff6ff',
          100: '#dbeafe',
          200: '#bfdbfe',
          300: '#93c5fd',
          400: '#60a5fa',
          500: '#3b82f6',
          600: '#2563eb',
          700: '#1d4ed8',
          800: '#1e40af',
          900: '#1e3a8a',
          950: '#172554',
        },
        secondary: {
          50: '#faf5ff',
          100: '#f3e8ff',
          200: '#e9d5ff',
          300: '#d8b4fe',
          400: '#c084fc',
          500: '#a855f7',
          600: '#9333ea',
          700: '#7c3aed',
```

```
          800: '#6b21a8',
          900: '#581c87',
          950: '#3b0764',
        },
        success: {
          50: '#f0fdf4',
          100: '#dcfce7',
          200: '#bbf7d0',
          300: '#86efac',
          400: '#4ade80',
          500: '#22c55e',
          600: '#16a34a',
          700: '#15803d',
          800: '#166534',
          900: '#14532d',
          950: '#052e16',
        },
        warning: {
          50: '#fffbeb',
          100: '#fef3c7',
          200: '#fde68a',
          300: '#fcd34d',
          400: '#fbbf24',
          500: '#f59e0b',
          600: '#d97706',
          700: '#b45309',
          800: '#92400e',
          900: '#78350f',
          950: '#451a03',
        },
        error: {
          50: '#fef2f2',
          100: '#fee2e2',
          200: '#fecaca',
          300: '#fca5a5',
          400: '#f87171',
```

```
          500: '#ef4444',
          600: '#dc2626',
          700: '#b91c1c',
          800: '#991b1b',
          900: '#7f1d1d',
          950: '#450a0a',
        },
      },

      fontFamily: {
        sans: ['Inter', ...defaultTheme.fontFamily.sans],
        mono: ['JetBrains
        Mono', ...defaultTheme.fontFamily.mono],
      },

      fontSize: {
        'xs': ['0.75rem', { lineHeight: '1rem' }],
        'sm': ['0.875rem', { lineHeight: '1.25rem' }],
        'base': ['1rem', { lineHeight: '1.5rem' }],
        'lg': ['1.125rem', { lineHeight: '1.75rem' }],
        'xl': ['1.25rem', { lineHeight: '1.75rem' }],
        '2xl': ['1.5rem', { lineHeight: '2rem' }],
        '3xl': ['1.875rem', { lineHeight: '2.25rem' }],
        '4xl': ['2.25rem', { lineHeight: '2.5rem' }],
        '5xl': ['3rem', { lineHeight: '1' }],
        '6xl': ['3.75rem', { lineHeight: '1' }],
        '7xl': ['4.5rem', { lineHeight: '1' }],
        '8xl': ['6rem', { lineHeight: '1' }],
        '9xl': ['8rem', { lineHeight: '1' }],
      },

      spacing: {
        '18': '4.5rem',
        '88': '22rem',
        '128': '32rem',
      },
```

```
      animation: {
        'fade-in': 'fadeIn 0.5s ease-in-out',
        'fade-out': 'fadeOut 0.5s ease-in-out',
        'slide-in': 'slideIn 0.3s ease-out',
        'slide-out': 'slideOut 0.3s ease-in',
        'scale-in': 'scaleIn 0.2s ease-out',
        'scale-out': 'scaleOut 0.2s ease-in',
        'bounce-in': 'bounceIn 0.6s ease-out',
        'bounce-out': 'bounceOut 0.6s ease-in',
        'rotate-in': 'rotateIn 0.6s ease-out',
        'rotate-out': 'rotateOut 0.6s ease-in',
        'flip-in': 'flipIn 0.6s ease-out',
        'flip-out': 'flipOut 0.6s ease-in',
        'hinge': 'hinge 2s ease-in-out',
        'roll-in': 'rollIn 0.6s ease-out',
        'roll-out': 'rollOut 0.6s ease-in',
        'zoom-in': 'zoomIn 0.3s ease-out',
        'zoom-out': 'zoomOut 0.3s ease-in',
        'pulse': 'pulse 2s cubic-bezier(0.4, 0, 0.6, 1)
        infinite',
        'ping': 'ping 1s cubic-bezier(0, 0, 0.2, 1) infinite',
        'bounce': 'bounce 1s infinite',
        'spin': 'spin 1s linear infinite',
      },

      keyframes: {
        fadeIn: {
          '0%': { opacity: '0' },
          '100%': { opacity: '1' },
        },
        fadeOut: {
          '0%': { opacity: '1' },
          '100%': { opacity: '0' },
        },
        slideIn: {
```

```
          '0%': { transform: 'translateX(-100%)' },
          '100%': { transform: 'translateX(0)' },
        },
        slideOut: {
          '0%': { transform: 'translateX(0)' },
          '100%': { transform: 'translateX(-100%)' },
        },
        scaleIn: {
          '0%': { transform: 'scale(0)' },
          '100%': { transform: 'scale(1)' },
        },
        scaleOut: {
          '0%': { transform: 'scale(1)' },
          '100%': { transform: 'scale(0)' },
        },
        bounceIn: {
          '0%': { transform: 'scale(0.3)', opacity: '0' },
          '50%': { transform: 'scale(1.05)' },
          '70%': { transform: 'scale(0.9)' },
          '100%': { transform: 'scale(1)', opacity: '1' },
        },
        bounceOut: {
          '0%': { transform: 'scale(1)', opacity: '1' },
          '20%': { transform: 'scale(0.9)' },
          '50%': { transform: 'scale(1.05)' },
          '100%': { transform: 'scale(0.3)', opacity: '0' },
        },
        rotateIn: {
          '0%': { transform: 'rotate(-200deg)', opacity:
        '0' },
          '100%': { transform: 'rotate(0)', opacity: '1' },
        },
        rotateOut: {
          '0%': { transform: 'rotate(0)', opacity: '1' },
          '100%': { transform: 'rotate(200deg)', opacity:
        '0' },
        },
```

```
        flipIn: {
          '0%': { transform: 'perspective(400px)
        rotateY(90deg)', opacity: '0' },
          '40%': { transform: 'perspective(400px)
        rotateY(-20deg)' },
          '60%': { transform: 'perspective(400px)
        rotateY(10deg)' },
          '80%': { transform: 'perspective(400px)
        rotateY(-5deg)' },
          '100%': { transform: 'perspective(400px)
        rotateY(0deg)', opacity: '1' },
        },
        flipOut: {
          '0%': { transform: 'perspective(400px)
        rotateY(0deg)', opacity: '1' },
          '100%': { transform: 'perspective(400px)
        rotateY(90deg)', opacity: '0' },
        },
        hinge: {
          '0%': { transform: 'rotate(0)', transformOrigin:
        'top left', animationTimingFunction: 'ease-in-out' },
          '20%, 60%': { transform: 'rotate(80deg)',
        transformOrigin: 'top left', animationTimingFunction:
        'ease-in-out' },
          '40%': { transform: 'rotate(60deg)',
        transformOrigin: 'top left', animationTimingFunction:
        'ease-in-out' },
          '80%': { transform: 'rotate(60deg) translateY(0)',
        transformOrigin: 'top left', animationTimingFunction:
        'ease-in-out' },
          '100%': { transform: 'translateY(700px)', opacity:
        '0' },
        },
        rollIn: {
          '0%': { transform: 'translateX(-100%)
        rotate(-120deg)', opacity: '0' },
          '100%': { transform: 'translateX(0) rotate(0deg)',
        opacity: '1' },
        },
        rollOut: {
          '0%': { transform: 'translateX(0) rotate(0deg)',
        opacity: '1' },
```

```
          '100%': { transform: 'translateX(-100%)
        rotate(-120deg)', opacity: '0' },
        },
        zoomIn: {
          '0%': { transform: 'scale(0.3)', opacity: '0' },
          '50%': { opacity: '1' },
          '100%': { transform: 'scale(1)', opacity: '1' },
        },
        zoomOut: {
          '0%': { transform: 'scale(1)', opacity: '1' },
          '50%': { opacity: '1' },
          '100%': { transform: 'scale(0.3)', opacity: '0' },
        },
      },

      backdropBlur: {
        xs: '2px',
      },

      boxShadow: {
        'inner-lg': 'inset 0 2px 4px 0 rgba(0, 0, 0, 0.06)',
        'inner-xl': 'inset 0 4px 6px -1px rgba(0, 0, 0, 0.1),
        inset 0 2px 4px -1px rgba(0, 0, 0, 0.06)',
      },

      borderRadius: {
        '4xl': '2rem',
        '5xl': '2.5rem',
      },

      zIndex: {
        '60': '60',
        '70': '70',
        '80': '80',
        '90': '90',
        '100': '100',
      },
```

```
    },
  },

  plugins: [
    require('@tailwindcss/forms'),
    require('@tailwindcss/typography'),
    require('@tailwindcss/aspect-ratio'),
    require('@tailwindcss/line-clamp'),
    require('tailwindcss-animate'),
    require('@tailwindcss/container-queries'),
  ],

  darkMode: 'class',

  future: {
    hoverOnlyWhenSupported: true,
  },
};
```

9.3 Backend Architecture (Node.js + TypeScript)

Configuration Express.js avancée :

```typescript
// app.ts
import express from 'express';
import helmet from 'helmet';
import cors from 'cors';
import compression from 'compression';
import rateLimit from 'express-rate-limit';
import slowDown from 'express-slow-down';
import morgan from 'morgan';
import { createServer } from 'http';
import { Server } from 'socket.io';
import { config } from 'dotenv';
import { connect } from 'mongoose';
import { createBullBoard } from '@bull-board/api';
import { BullAdapter } from '@bull-board/api/bullAdapter';
import { ExpressAdapter } from '@bull-board/express';
import { swaggerUi, specs } from './swagger';
import { errorHandler } from './middleware/errorHandler';
import { notFoundHandler } from './middleware/
        notFoundHandler';
import { requestLogger } from './middleware/requestLogger';
import { authMiddleware } from './middleware/authMiddleware';
import { validateRequest } from './middleware/
        validateRequest';
import { cacheMiddleware } from './middleware/
        cacheMiddleware';
import { rateLimitMiddleware } from './middleware/
        rateLimitMiddleware';
import { securityMiddleware } from './middleware/
        securityMiddleware';
import { monitoringMiddleware } from './middleware/
        monitoringMiddleware';

// Routes
import authRoutes from './routes/auth';
import userRoutes from './routes/users';
import groupRoutes from './routes/groups';
import eventRoutes from './routes/events';
```

```
import expenseRoutes from './routes/expenses';
import notificationRoutes from './routes/notifications';
import analyticsRoutes from './routes/analytics';
import webhookRoutes from './routes/webhooks';

// Services
import { DatabaseService } from './services/DatabaseService';
import { RedisService } from './services/RedisService';
import { EmailService } from './services/EmailService';
import { NotificationService } from './services/
        NotificationService';
import { AnalyticsService } from './services/
        AnalyticsService';
import { AIService } from './services/AIService';
import { PaymentService } from './services/PaymentService';
import { BookingService } from './services/BookingService';

// Load environment variables
config();

const app = express();
const server = createServer(app);
const io = new Server(server, {
  cors: {
    origin: process.env.FRONTEND_URL,
    methods: ['GET', 'POST'],
    credentials: true
  }
});

// Database connection
const databaseService = new DatabaseService();
databaseService.connect();

// Redis connection
const redisService = new RedisService();
redisService.connect();
```

```
// Initialize services
const emailService = new EmailService();
const notificationService = new NotificationService(io);
const analyticsService = new AnalyticsService();
const aiService = new AIService();
const paymentService = new PaymentService();
const bookingService = new BookingService();

// Security middleware
app.use(helmet({
  contentSecurityPolicy: {
    directives: {
      defaultSrc: ["'self'"],
      styleSrc: ["'self'", "'unsafe-inline'", "https://
        fonts.googleapis.com"],
      fontSrc: ["'self'", "https://fonts.gstatic.com"],
      imgSrc: ["'self'", "data:", "https:"],
      scriptSrc: ["'self'"],
      connectSrc: ["'self'", "https://api.openai.com",
        "https://api.stripe.com"],
    },
  },
  crossOriginEmbedderPolicy: false,
}));

// CORS configuration
app.use(cors({
  origin: process.env.FRONTEND_URL,
  credentials: true,
  methods: ['GET', 'POST', 'PUT', 'DELETE', 'PATCH',
        'OPTIONS'],
  allowedHeaders: ['Content-Type', 'Authorization', 'X-
        Requested-With'],
}));

// Compression
```

```
app.use(compression());

// Rate limiting
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100, // limit each IP to 100 requests per windowMs
  message: 'Too many requests from this IP, please try again later.',
  standardHeaders: true,
  legacyHeaders: false,
});

const speedLimiter = slowDown({
  windowMs: 15 * 60 * 1000, // 15 minutes
  delayAfter: 50, // allow 50 requests per 15 minutes, then...
  delayMs: 500 // begin adding 500ms of delay per request above 50
});

app.use('/api/', limiter);
app.use('/api/', speedLimiter);

// Logging
if (process.env.NODE_ENV === 'development') {
  app.use(morgan('dev'));
} else {
  app.use(morgan('combined'));
}

// Body parsing
app.use(express.json({ limit: '10mb' }));
app.use(express.urlencoded({ extended: true, limit: '10mb' }));

// Request logging
app.use(requestLogger);
```

```
// Monitoring
app.use(monitoringMiddleware);

// Bull Board (Job monitoring)
const serverAdapter = new ExpressAdapter();
serverAdapter.setBasePath('/admin/queues');

createBullBoard({
  queues: [
    new BullAdapter(emailService.queue),
    new BullAdapter(notificationService.queue),
    new BullAdapter(analyticsService.queue),
  ],
  serverAdapter,
});

app.use('/admin/queues', serverAdapter.getRouter());

// API Documentation
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(specs));

// Health check
app.get('/health', (req, res) => {
  res.status(200).json({
    status: 'OK',
    timestamp: new Date().toISOString(),
    uptime: process.uptime(),
    environment: process.env.NODE_ENV,
    version: process.env.npm_package_version,
  });
});

// API Routes
app.use('/api/auth', authRoutes);
app.use('/api/users', authMiddleware, userRoutes);
app.use('/api/groups', authMiddleware, groupRoutes);
```

```
app.use('/api/events', authMiddleware, eventRoutes);
app.use('/api/expenses', authMiddleware, expenseRoutes);
app.use('/api/notifications', authMiddleware,
        notificationRoutes);
app.use('/api/analytics', authMiddleware, analyticsRoutes);
app.use('/api/webhooks', webhookRoutes);

// WebSocket connection handling
io.on('connection', (socket) => {
  console.log('User connected:', socket.id);

  socket.on('join-room', (roomId) => {
    socket.join(roomId);
    console.log(`User ${socket.id} joined room ${roomId}`);
  });

  socket.on('leave-room', (roomId) => {
    socket.leave(roomId);
    console.log(`User ${socket.id} left room ${roomId}`);
  });

  socket.on('disconnect', () => {
    console.log('User disconnected:', socket.id);
  });
});

// Error handling
app.use(notFoundHandler);
app.use(errorHandler);

// Graceful shutdown
process.on('SIGTERM', () => {
  console.log('SIGTERM received, shutting down gracefully');
  server.close(() => {
    console.log('Process terminated');
    process.exit(0);
```

```
  });
});

process.on('SIGINT', () => {
  console.log('SIGINT received, shutting down gracefully');
  server.close(() => {
    console.log('Process terminated');
    process.exit(0);
  });
});

const PORT = process.env.PORT || 3000;

server.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
  console.log(`Environment: ${process.env.NODE_ENV}`);
  console.log(`API Documentation: http://localhost:${PORT}/
        api-docs`);
  console.log(`Queue Dashboard: http://localhost:${PORT}/
        admin/queues`);
});

export { app, server, io };
```

Configuration Swagger/OpenAPI :

```ts
// swagger.ts
import swaggerJsdoc from 'swagger-jsdoc';

const options = {
  definition: {
    openapi: '3.0.0',
    info: {
      title: 'SocialPlanr API',
      version: '1.0.0',
      description: 'API documentation for SocialPlanr -
        Collaborative Event Management Platform',
      contact: {
        name: 'SocialPlanr Team',
        email: 'api@socialplanr.com',
        url: 'https://socialplanr.com',
      },
      license: {
        name: 'MIT',
        url: 'https://opensource.org/licenses/MIT',
      },
    },
    servers: [
      {
        url: 'https://api.socialplanr.com',
        description: 'Production server',
      },
      {
        url: 'https://staging-api.socialplanr.com',
        description: 'Staging server',
      },
      {
        url: 'http://localhost:3000',
        description: 'Development server',
      },
    ],
```

```
    components: {
      securitySchemes: {
        bearerAuth: {
          type: 'http',
          scheme: 'bearer',
          bearerFormat: 'JWT',
        },
        apiKeyAuth: {
          type: 'apiKey',
          in: 'header',
          name: 'X-API-Key',
        },
      },
      schemas: {
        User: {
          type: 'object',
          properties: {
            id: { type: 'string', format: 'uuid' },
            email: { type: 'string', format: 'email' },
            firstName: { type: 'string' },
            lastName: { type: 'string' },
            displayName: { type: 'string' },
            avatarUrl: { type: 'string' },
            isActive: { type: 'boolean' },
            isVerified: { type: 'boolean' },
            createdAt: { type: 'string', format: 'date-
        time' },
            updatedAt: { type: 'string', format: 'date-
        time' },
          },
          required: ['id', 'email', 'firstName', 'lastName'],
        },
        Group: {
          type: 'object',
          properties: {
            id: { type: 'string', format: 'uuid' },
            name: { type: 'string' },
```

```
            description: { type: 'string' },
            createdBy: { type: 'string', format: 'uuid' },
            members: {
              type: 'array',
              items: { type: 'string', format: 'uuid' },
            },
            admins: {
              type: 'array',
              items: { type: 'string', format: 'uuid' },
            },
            status: { type: 'string', enum: ['active',
        'archived', 'deleted'] },
            createdAt: { type: 'string', format: 'date-
        time' },
            updatedAt: { type: 'string', format: 'date-
        time' },
          },
          required: ['id', 'name', 'createdBy'],
        },
        Event: {
          type: 'object',
          properties: {
            id: { type: 'string', format: 'uuid' },
            title: { type: 'string' },
            description: { type: 'string' },
            groupId: { type: 'string', format: 'uuid' },
            createdBy: { type: 'string', format: 'uuid' },
            startDate: { type: 'string', format: 'date-
        time' },
            endDate: { type: 'string', format: 'date-time' },
            location: { type: 'string' },
            status: {
              type: 'string',
              enum: ['planning', 'voting', 'confirmed',
        'in_progress', 'completed', 'cancelled'],
            },
            attendees: {
              type: 'array',
```

```
              items: { type: 'string', format: 'uuid' },
            },
            aiGenerated: { type: 'boolean' },
            createdAt: { type: 'string', format: 'date-
        time' },
            updatedAt: { type: 'string', format: 'date-
        time' },
          },
          required: ['id', 'title', 'groupId', 'createdBy',
        'startDate'],
        },
        Expense: {
          type: 'object',
          properties: {
            id: { type: 'string', format: 'uuid' },
            title: { type: 'string' },
            description: { type: 'string' },
            amount: { type: 'number' },
            currency: { type: 'string', enum: ['EUR', 'USD',
        'GBP'] },
            category: {
              type: 'string',
              enum: ['accommodation', 'transport', 'food',
        'entertainment', 'other'],
            },
            groupId: { type: 'string', format: 'uuid' },
            eventId: { type: 'string', format: 'uuid' },
            paidBy: { type: 'string', format: 'uuid' },
            participants: {
              type: 'array',
              items: { type: 'string', format: 'uuid' },
            },
            status: {
              type: 'string',
              enum: ['pending', 'paid', 'settled',
        'cancelled'],
            },
            date: { type: 'string', format: 'date' },
```

```
            createdAt: { type: 'string', format: 'date-
        time' },
            updatedAt: { type: 'string', format: 'date-
        time' },
          },
          required: ['id', 'title', 'amount', 'category',
        'paidBy'],
        },
        Error: {
          type: 'object',
          properties: {
            success: { type: 'boolean', example: false },
            error: {
              type: 'object',
              properties: {
                code: { type: 'string' },
                message: { type: 'string' },
                details: { type: 'object' },
              },
            },
          },
        },
      },
    },
    security: [
      {
        bearerAuth: [],
      },
    ],
  },
  apis: ['./src/routes/*.ts', './src/models/*.ts'],
};

export const specs = swaggerJsdoc(options);
```

[Suite dans la partie suivante...]