Application de gestion de Festival de Jeux de Société - Spécification complète

1. Contexte et objectif

Problème

Gérer le workflow complet de réservation de tables par les éditeurs pour un festival de jeux de société : du premier contact à la facturation finale.

Solution

Application web interne pour les organisateurs permettant de :

- Gérer plusieurs festivals (basculement entre festivals)
- Suivre les réservations avec workflow en 9 étapes
- Attribuer des tables et placer des jeux
- Calculer automatiquement les prix et remises
- Tracer tous les contacts et actions

Important

- **Outil interne uniquement** : pas de réservation en ligne pour les éditeurs
- Toutes les interactions éditeurs/organisateurs sont **externes** (téléphone, email) puis saisies manuellement
- Petite partie publique pour que les visiteurs voient les infos du festival

---

2. Architecture et utilisateurs

Système de comptes

- Utilisateurs non connectés **demandent** un compte
- Un **admin valide ou refuse** la demande
- Emails peuvent être bloqués (anti-spam)

Hiérarchie des rôles (4 niveaux)

Rôle	Droits
Bénévole	Lecture seule
Organisateur	+ Gérer éditeurs, jeux, réservations, attribution tables
Super-organisateur	+ Créer festivals, zones tarifaires, zones du plan
Admin	+ Gérer comptes utilisateurs

Hiérarchique : chaque niveau hérite des droits inférieurs

---

3. Modèle de données

3.1 Tables principales

Utilisateur

Comptes organisateurs/admins.

```
sql

- id (PK)
- email (UNIQUE, NOT NULL)
- password_hash (NOT NULL)
- role ENUM('benevole', 'organisateur', 'super_organisateur', 'admin')
- statut ENUM('en_attente', 'valide', 'refuse')
- date_demande
- valide_par (FK → Utilisateur)
- email_bloque (BOOLEAN)
```

Festival

Événement festival.

```
sql

- id (PK)
- nom (UNIQUE, NOT NULL)
- date_creation
```

- Un festival = "festival courant" par défaut (dernier créé)
- L'application travaille toujours sur le festival courant

ZoneTarifaire

Tarif applicable à des tables.

```
sql

- id (PK)
- festival_id (FK → Festival)
- nom (NOT NULL)
- nombre_tables_total (INT)
- prix_table (DECIMAL)
- prix_m2 (DECIMAL, défaut: prix_table / 4.5)
- UNIQUE(festival_id, nom)
```

- **Réutilisable** : plusieurs zones du plan peuvent utiliser la même zone tarifaire
- Ne peut être supprimée si des tables l'utilisent

ZoneDuPlan

Zone géographique physique du festival.

```sql
- id (PK)
- festival_id (FK → Festival)
- nom (NOT NULL)
- nombre_tables (INT)
- zone_tarifaire_id (FK → ZoneTarifaire)
- UNIQUE(festival_id, nom)
```

- Représente l'emplacement réel (ex: "Hall Principal", "Salle Annexe")
- Peut contenir des tables avec différentes zones tarifaires

Table_Jeu

Table physique sur le festival.

```sql
- id (PK)
- zone_du_plan_id (FK → ZoneDuPlan)
- zone_tarifaire_id (FK → ZoneTarifaire)
- capacite_jeux (INT, défaut: 2)
- nb_jeux_actuels (INT, défaut: 0, calculé auto)
- statut ENUM('libre', 'reserve', 'plein', 'hors_service')
```

- Un seul format de table ; grouper plusieurs tables pour faire "grande table"
- Statut "plein" automatique quand capacite_jeux atteinte

Editeur

Maison d'édition.

```sql
- id (PK)
- nom (UNIQUE, NOT NULL)
```

- **Partagé entre tous les festivals** (pas de duplication)

Personne

Contacts, auteurs (entité polyvalente).

```sql
- id (PK)
- nom (NOT NULL)
- prenom (NOT NULL)
- telephone (UNIQUE, NOT NULL)
- email
- fonction
```

- Telephone unique = identifiant naturel
- Peut être contact d'éditeur ET auteur de jeux

Jeu

Jeu de société.

```sql
- id (PK)
- nom (NOT NULL)
- nb_joueurs_min, nb_joueurs_max
- duree_minutes
- age_min, age_max
- description (TEXT)
- lien_regles
```

- **Partagé entre tous les festivals**

Reservation

Réservation éditeur × festival.

```sql
- id (PK)
- editeur_id (FK → Editeur)
- festival_id (FK → Festival)
- statut_workflow ENUM(voir section Workflow)
- editeur_presente_jeux (BOOLEAN)
- remise_pourcentage (DECIMAL)
- remise_montant (DECIMAL)
- prix_total (DECIMAL, calculé auto)
- prix_final (DECIMAL, calculé auto)
- commentaires_paiement (TEXT)
- paiement_relance (BOOLEAN)
- date_facture, date_paiement
- created_by, updated_by (FK → Utilisateur)
- UNIQUE(editeur_id, festival_id)
```

JeuFestival

Jeu dans un festival spécifique.

```sql
- id (PK)
- jeu_id (FK → Jeu)
- reservation_id (FK → Reservation)
- festival_id (FK → Festival)
- dans_liste_demandee (BOOLEAN)
- dans_liste_obtenue (BOOLEAN)
- jeux_recu (BOOLEAN)
```

- Permet de tracker l'état par festival

ContactEditeur

Historique des contacts.

```sql
- id (PK)
- editeur_id (FK → Editeur)
- festival_id (FK → Festival)
- utilisateur_id (FK → Utilisateur)
- date_contact
- notes (TEXT)
```

AuditLog

Journal d'audit pour actions critiques.

```sql
- id (PK)
- utilisateur_id (FK → Utilisateur)
- action (VARCHAR)
- entite_type (VARCHAR)
- entite_id (INT)
- date_action
- details (TEXT)
```

3.2 Tables de liaison

EditeurContact

```sql
- editeur_id (FK → Editeur)
- personne_id (FK → Personne)
- PRIMARY KEY(editeur_id, personne_id)
```

JeuEditeur (co-édition possible)

```sql
- jeu_id (FK → Jeu)
- editeur_id (FK → Editeur)
- PRIMARY KEY(jeu_id, editeur_id)
```

JeuAuteur

```sql
- jeu_id (FK → Jeu)
- personne_id (FK → Personne)
- PRIMARY KEY(jeu_id, personne_id)
```

ReservationTable

```sql
- reservation_id (FK → Reservation)
- table_id (FK → Table_Jeu)
- date_attribution
- attribue_par (FK → Utilisateur)
- PRIMARY KEY(reservation_id, table_id)
```

JeuFestivalTable

```sql
- jeu_festival_id (FK → JeuFestival)
- table_id (FK → Table_Jeu)
- PRIMARY KEY(jeu_festival_id, table_id)
```

- Un jeu peut être sur plusieurs tables
- Plusieurs jeux peuvent partager une table

3.3 Triggers automatiques

trg_update_table_jeux_count

Déclenché : INSERT sur JeuFestivalTable Action : Met à jour `nb_jeux_actuels`, marque table "plein" si capacité atteinte

trg_recalcul_prix_reservation

Déclenché : INSERT sur ReservationTable Action : Recalcule `prix_total` (somme des prix des tables) et `prix_final` (avec remises)

---

4. Workflow de réservation

9 états possibles

1. **pas_contacte** : État initial, jamais contacté
2. **contact_pris** : Au moins un contact effectué
3. **discussion_en_cours** : Échanges sans décision finale
4. **sera_absent** : Refus explicite de l'éditeur
5. **considere_absent** : Pas de réponse malgré relances
6. **present** : Présence confirmée, réservation faite
7. **facture** : Facture émise
8. **paiement_recu** : Paiement reçu et encaissé
9. **paiement_en_retard** : Facture émise, paiement non reçu

Process complet

```
pas_contacte → contact_pris → discussion_en_cours
  ↓ (refus)          ↓ (pas de réponse)     ↓ (acceptation)
sera_absent       considere_absent        present
                                ↓
                            facture
                                ↓
                  paiement_recu ou paiement_en_retard
```

---

5. Règles métier clés

Réservations

- **Option B** : Éditeur demande "X tables en zone Y", tables spécifiques attribuées plus tard (sauf si demande spécifique)
- Calcul prix : `prix_total = SOMME(prix des tables attribuées)`
- Remises : `prix_final = prix_total × (1 - remise_pct/100) - remise_montant`
- Un éditeur = 1 réservation max par festival

Tables

- Créées manuellement par super-organisateur
- Peuvent avoir zone_tarifaire différente de leur zone_du_plan
- Marquées "plein" automatiquement par trigger

Jeux

- Un jeu peut être sur plusieurs tables
- Plusieurs jeux peuvent partager une table (si capacité suffit)
- Nombre de tables allouées = COUNT(JeuFestivalTable)

Workflow

- Pas de vérification stricte des zones (confiance organisateurs)
- Toutes modifications possibles à tout moment
- Dates cohérentes : date_facture avant date_paiement

---

6. Cas d'usage essentiels

Use Case 1 : Créer un festival

1. Super-orga saisit nom (unique)
2. Définit zones tarifaires (min 1) : nom, nb tables, prix
3. Crée zones du plan (min 1) : nom, nb tables, zone tarifaire
4. Système crée tables physiques automatiquement
5. Festival devient "festival courant"

Use Case 2 : Gérer une réservation

Phase 1 - Contact

```sql
INSERT INTO ContactEditeur (editeur_id, festival_id, utilisateur_id, notes)
VALUES (1, 1, 2, 'Premier contact, intéressé');

UPDATE Reservation SET statut_workflow = 'contact_pris' WHERE id = 1;
```

Phase 2 - Confirmation

```sql
UPDATE Reservation
SET statut_workflow = 'present', editeur_presente_jeux = TRUE
WHERE id = 1;
```

Phase 3 - Attribution tables

```sql
INSERT INTO ReservationTable (reservation_id, table_id, attribue_par)
VALUES (1, 5, 2), (1, 6, 2), (1, 7, 2);
-- Trigger calcule automatiquement prix_total et prix_final
```

Phase 4 - Ajout jeux

```sql
INSERT INTO JeuFestival (jeu_id, reservation_id, festival_id, dans_liste_demandee)
VALUES (10, 1, 1, TRUE);

-- Confirmation éditeur
UPDATE JeuFestival SET dans_liste_obtenue = TRUE WHERE id = 1;

-- Réception physique
UPDATE JeuFestival SET jeux_recu = TRUE WHERE id = 1;
```

Phase 5 - Placement jeux

```sql
INSERT INTO JeuFestivalTable (jeu_festival_id, table_id)
VALUES (1, 5), (1, 6);
-- Trigger met à jour nb_jeux_actuels automatiquement
```

Phase 6 - Facturation/Paiement

```sql
UPDATE Reservation
SET statut_workflow = 'facture', date_facture = NOW()
WHERE id = 1;

UPDATE Reservation
SET statut_workflow = 'paiement_recu', date_paiement = NOW()
WHERE id = 1;
```

---

7. Requêtes SQL utiles

Dashboard festival

```sql
SELECT
  f.nom,
  COUNT(DISTINCT e.id) AS nb_editeurs,
  COUNT(DISTINCT CASE WHEN r.statut_workflow = 'present' THEN e.id END) AS presents,
  COUNT(DISTINCT t.id) AS nb_tables_total,
  COUNT(DISTINCT CASE WHEN t.statut = 'libre' THEN t.id END) AS libres,
  SUM(r.prix_final) AS revenus
FROM Festival f
CROSS JOIN Editeur e
LEFT JOIN Reservation r ON r.festival_id = f.id AND r.editeur_id = e.id
LEFT JOIN ZoneDuPlan zp ON zp.festival_id = f.id
LEFT JOIN Table_Jeu t ON t.zone_du_plan_id = zp.id
WHERE f.id = ?
GROUP BY f.id, f.nom;
```

Liste éditeurs avec statut réservation

```sql
SELECT
  e.nom,
  r.statut_workflow,
  MAX(ce.date_contact) AS dernier_contact,
  COUNT(DISTINCT rt.table_id) AS nb_tables,
  r.prix_final
FROM Editeur e
LEFT JOIN Reservation r ON r.editeur_id = e.id AND r.festival_id = ?
LEFT JOIN ReservationTable rt ON rt.reservation_id = r.id
LEFT JOIN ContactEditeur ce ON ce.editeur_id = e.id AND ce.festival_id = ?
GROUP BY e.id, e.nom, r.statut_workflow, r.prix_final
ORDER BY r.statut_workflow, e.nom;
```

Jeux par zone du plan

```sql
SELECT
  zp.nom AS zone,
  j.nom AS jeu,
  e.nom AS editeur,
  GROUP_CONCAT(DISTINCT p.nom) AS auteurs,
  COUNT(DISTINCT jft.table_id) AS nb_tables
FROM ZoneDuPlan zp
JOIN Table_Jeu t ON t.zone_du_plan_id = zp.id
JOIN JeuFestivalTable jft ON jft.table_id = t.id
JOIN JeuFestival jf ON jf.id = jft.jeu_festival_id
JOIN Jeu j ON j.id = jf.jeu_id
JOIN Reservation r ON r.id = jf.reservation_id
JOIN Editeur e ON e.id = r.editeur_id
LEFT JOIN JeuAuteur ja ON ja.jeu_id = j.id
LEFT JOIN Personne p ON p.id = ja.personne_id
WHERE zp.festival_id = ?
GROUP BY zp.id, zp.nom, j.id, j.nom, e.nom;
```

Factures impayées

```sql
SELECT
  e.nom,
  r.prix_final,
  r.date_facture,
  DATEDIFF(NOW(), r.date_facture) AS jours_retard,
  r.paiement_relance
FROM Reservation r
JOIN Editeur e ON e.id = r.editeur_id
WHERE r.festival_id = ?
  AND r.statut_workflow IN ('facture', 'paiement_en_retard')
ORDER BY r.date_facture;
```

---

8. Points techniques importants

Sécurité

- Mots de passe hashés (bcrypt)
- Vérification du rôle avant chaque action
- Protection CSRF, XSS, injection SQL
- Sessions sécurisées, HTTPS obligatoire

Performance

- Index sur tous les FK et champs de recherche
- Vues pour requêtes complexes répétitives
- Triggers pour calculs automatiques
- Pagination si > 50 items

Validation

- Contraintes BD : NOT NULL, UNIQUE, FK, CHECK
- Validation serveur obligatoire
- Messages d'erreur clairs

Transactions

Utiliser pour opérations multi-tables :

```sql
START TRANSACTION;
INSERT INTO Festival ...;
INSERT INTO ZoneTarifaire ...;
INSERT INTO ZoneDuPlan ...;
COMMIT; -- ou ROLLBACK si erreur
```

---

9. Décisions de conception clés

Pourquoi une table Personne unique ?

Mutualisation : une personne peut être contact d'éditeur ET auteur de jeux.

Pourquoi Option B pour attribution tables ?

Plus réaliste : éditeurs ne connaissent pas les numéros de tables. Plus flexible pour organisateurs.

Pourquoi 2 champs remise séparés ?

Anticiper tous les cas : remise % OU montant OU les deux.

Pourquoi JeuFestival ?

Séparer le jeu générique (réutilisable) de sa présence dans un festival spécifique (avec état).

Pourquoi pas de vérification stricte ?

Confiance organisateurs. Simplification. Flexibilité en cas de réorganisation.

Pourquoi éditeurs/jeux partagés ?

Évite duplication. Facilite usage multi-festivals. Historique automatique.

10. Checklist implémentation

Base de données

- [ ] Toutes tables créées avec contraintes
- [ ] Index sur FK et champs de recherche
- [ ] Triggers créés et testés
- [ ] Vues créées
- [ ] Utilisateur admin par défaut

Fonctionnalités core

- [ ] Authentification + gestion rôles
- [ ] CRUD festivals, zones, tables
- [ ] CRUD éditeurs, personnes, jeux
- [ ] Workflow réservation complet (9 états)
- [ ] Attribution tables + calcul auto prix
- [ ] Placement jeux + update auto compteurs
- [ ] Historique contacts
- [ ] Audit log actions critiques

Interface

- [ ] Responsive (mobile/desktop)
- [ ] Indicateur festival courant
- [ ] Formulaires avec validation
- [ ] Messages succès/erreur
- [ ] Tri, filtres, recherche
- [ ] Codes couleur statuts

Sécurité

- [ ] Hashage mots de passe
- [ ] Protection CSRF/XSS/SQL injection
- [ ] Vérification rôles sur toutes actions
- [ ] HTTPS
- [ ] Rate limiting

Tests

- [ ] Tests triggers (calculs, compteurs)
- [ ] Tests workflow complet
- [ ] Tests droits par rôle
- [ ] Tests cas limites (capacité table, remises, etc.)

---

11. Résumé technique

Architecture : Application web avec 14 tables principales + 5 liaisons

Particularités :

- Éditeurs/jeux partagés entre festivals (unicité globale)
- Tables physiques identifiées individuellement
- Workflow 9 états avec traçabilité complète
- Calculs automatiques (triggers)
- Hiérarchie 4 niveaux de droits

Base de données : MySQL 5.7+ / MariaDB 10.2+ / PostgreSQL 9.6+

Prêt pour : Implémentation immédiate par équipe dev web

Annexe : Script création rapide

```sql
-- Ordre de création (respecter les dépendances FK)
CREATE TABLE Utilisateur (...);
CREATE TABLE Festival (...);
CREATE TABLE ZoneTarifaire (...);
CREATE TABLE ZoneDuPlan (...);
CREATE TABLE Table_Jeu (...);
CREATE TABLE Editeur (...);
CREATE TABLE Personne (...);
CREATE TABLE EditeurContact (...);
CREATE TABLE Jeu (...);
CREATE TABLE JeuEditeur (...);
CREATE TABLE JeuAuteur (...);
CREATE TABLE Reservation (...);
CREATE TABLE ReservationTable (...);
CREATE TABLE JeuFestival (...);
CREATE TABLE JeuFestivalTable (...);
CREATE TABLE ContactEditeur (...);
CREATE TABLE AuditLog (...);

-- Triggers
CREATE TRIGGER trg_update_table_jeux_count ...;
CREATE TRIGGER trg_recalcul_prix_reservation ...;

-- Vues
CREATE VIEW v_tables_disponibles_par_zone ...;
CREATE VIEW v_reservations_festival ...;
CREATE VIEW v_jeux_par_zone_plan ...;

-- Admin par défaut
INSERT INTO Utilisateur (email, password_hash, role, statut)
VALUES ('admin@festival.local', '$2y$10$...', 'admin', 'valide');
```

Version : 1.0 (Concise)
Pages : ~20
Complet pour : Développement, implémentation, compréhension système