EventSync

Cahier de taches détaillées — Conformité specification

Christian | *Nomena* | *Fenohasina* | *Harena*

Stack : Next.js 14 + Tailwind CSS | Express.js | Prisma ORM | PostgreSQL

Entités définies par le sujet

Le sujet définit 5 entités exactes. Tout le schéma Prisma doit les refléter fidèlement.

Entite	Attributs exacts du sujet	Porteur
Evenement	Titre, Description, Date début, Date fin, Lieu, Liste des sessions	Christian
Session	Titre, Description, Heure debut, Heure fin, Salle, Capacité (informative), Liste des intervenants, Liste des questions	Nomena
Salle (Room)	Nom uniquement — sert de critère de filtrage	Nomena
Intervenant (Speaker)	Nom complet, Photo de profil, Biographie, Liens externes, Liste des sessions associées	Fenohasina
Question	Contenu (texte), Nom (optionnel/anonyme), Nombre de upvotes, Session associee, Date/heure de creation	Harena

Rôles définis par le sujet

Role	Access	Capacites exactes selon le sujet
Organisateur (Admin)	Authentification requise	Créer/modifier/supprimer evenements, gérer sessions, assigner intervenants, définir salles et horaires, gérer profils intervenants
Participant	Accès public SANS authentification	Consulter événements, voir planning, identifier sessions live, accéder detail session, poser questions (si live), upvoter questions, ajouter sessions en favoris
Intervenant (Speaker)	Accès public SANS authentification spécifique	Acceder a sa page publique, voir ses sessions, visualiser les questions posées sur ses sessions

⚠ **Aucune authentification pour les participants ni les intervenants — acces libre uniquement**

Regles de gestion strictes du sujet

- Une session appartient a UN SEUL événement
- Une session est associée à UNE SEULE salle
- Une session possède AU MOINS UN intervenant (validation obligatoire en backend ET frontend)
- Une question est associée à une session et ne peut etre creee QUE lorsque la session est live
- Une question peut être anonyme (champ nom optionnel)
- Les upvotes sont cumulés via un compteur numérique — pas de limite dans cette version
- Les favoris sont stockés cote navigateur (localStorage) — pas en base de données
- Pas de modération des questions dans cette version
- Pas de système d'inscription aux sessions — la capacite est purement informative

Architecture et structure du projet

Structure des dossiers

- eventsync-backend/
  - prisma/schema.prisma <- schema unique, tous les membres y contribuent
  - prisma/seed.ts <- donnees de test realistes
  - src/index.ts <- serveur Express + enregistrement des routes
  - src/middleware/auth.ts <- JWT middleware (cree par Christian, partage avec tous)
  - src/utils/isLive.ts <- fonction isLive (creee par Nomena, partagee avec tous)
  - src/routes/auth.routes.ts <- Christian
  - src/routes/events.routes.ts <- Christian
  - src/routes/sessions.routes.ts <- Nomena
  - src/routes/rooms.routes.ts <- Nomena
  - src/routes/speakers.routes.ts <- Fenohasina
  - src/routes/questions.routes.ts <- Harena
- eventsync-frontend/
  - app/page.tsx <- page accueil avec liste des evenements
  - app/events/[id]/page.tsx <- page publique evenement (Christian)
  - app/events/[id]/planning/page.tsx <- planning multi-track (Nomena)
  - app/rooms/[id]/page.tsx <- vue planning par salle (Nomena)
  - app/sessions/[id]/page.tsx <- detail session (Fenohasina)
  - app/speakers/[id]/page.tsx <- page publique intervenant (Fenohasina)
  - app/my-schedule/page.tsx <- itineraire personnel/favoris (Harena)
  - app/admin/login/page.tsx <- connexion admin (Christian)
  - app/admin/events/page.tsx <- gestion evenements (Christian)
  - app/admin/sessions/page.tsx <- gestion sessions (Nomena)
  - app/admin/speakers/page.tsx <- gestion intervenants (Fenohasina)
  - components/LiveBadge.tsx <- badge En direct reutilisable (Nomena)
  - components/QASection.tsx <- section questions/reponses (Harena)
  - components/FavoriteButton.tsx <- bouton favori (Harena)

- lib/api.ts <- fonctions fetch centralisees
- lib/favoritesService.ts <- gestion localStorage favoris (Harena)

Variables d'environnement

Fichier .env (backend) :

```
DATABASE_URL="postgresql://user:password@localhost:5432/eventsync"
JWT_SECRET="cle_secrete_longue_et_aleatoire_minimum_32_caracteres"
PORT=3001
```

Fichier .env.local (frontend) :

```
NEXT_PUBLIC_API_URL="http://localhost:3001/api"
```

A FAIRE EN TOUT PREMIER — Setup du projet

[CONFIG] Initialiser les deux projets et le depot Git

⚠ **Tant que ce setup n'est pas fait, personne ne peut commencer. A faire le premier jour.**

- 1. Créerle dépôt sur GitHub/GitLab et inviter les 4 membres
- 2. Créer et initialiser le backend :

```
mkdir eventsync-backend && cd eventsync-backend
npm init -y
npm install express cors dotenv jsonwebtoken bcryptjs @prisma/client
npm install -D typescript ts-node @types/express @types/node @types/cors prisma
npx tsc --init
npx prisma init  # cree prisma/schema.prisma avec provider postgresql
```

- 3. Modifier prisma/schema.prisma : s'assurer que la premiere ligne est :

```
datasource db { provider = "postgresql" url = env("DATABASE_URL") }
```

- 4. Créer src/index.ts avec le serveur Express de base :

```
import express from 'express'
import cors from 'cors'
import dotenv from 'dotenv'
dotenv.config()
const app = express()
app.use(cors({ origin: process.env.FRONTEND_URL || 'http://localhost:3000' }))
app.use(express.json())
// Les routes seront ajoutees ici par chaque membre
app.listen(Number(process.env.PORT) || 3001, () => {
  console.log('Serveur Express demarre sur le port 3001')
})
```

- 5. Creer le frontend Next.js :

```
npx create-next-app@latest eventsync-frontend \
  --typescript --tailwind --eslint --app --no-src-dir
```

- 6. Créer les fichiers de configuration :
  - -> .env.example (backend) avec les cles vides
  - -> .env.local.example (frontend) avec les cles vides
  - -> .gitignore : node_modules/, .env, .env.local, dist/, .next/
- 7. Creer le README.md avec les commandes pour lancer le projet
- 8. Pousser le tout sur Git, creer les branches christian / nomena / fenohasina / harena
- 9. Envoyer le lien du repo a l'equipe avec les instructions de clonage

CHRISTIAN — Module Evenements et Authentification

📋 **SPEC : Organisateur : creer/modifier/supprimer evenements, authentification securisee par JWT**
📋 **SPEC : Participant : consulter les evenements, voir la liste des sessions, identifier les sessions live**

Phase 1 — Schema Prisma

[CONFIG] Modeles Event et User dans prisma/schema.prisma

- Ouvrir prisma/schema.prisma et ajouter les deux modèles suivants :

```
model User {
  id           Int      @id @default(uuid())
  email        String   @unique
  passwordHash String
  createdAt    DateTime @default(now())
}
```

```
model Event {
  id          Int       @id @default(uuid())
  title       String
  description String
  startDate   DateTime
  endDate     DateTime
  location    String
  sessions    Session[]
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt
}
```

- Dans prisma/seed.ts : créer un compte admin et 2 événements de test :

```
await prisma.user.create({
  data: { email: 'admin@eventsync.com',
          passwordHash: await bcrypt.hash('Admin1234', 10) }
})
// Evenement 1 : dates futures
// Evenement 2 : en cours aujourd'hui (pour tester le live)
```

Phase 2 — API Express

[BACKEND] src/middleware/auth.ts — Middleware JWT (a partager immediatement)

⚠ **Ce fichier doit etre pousse sur Git en PREMIER. Tous les autres membres en dependent.**

- Installer : npm install jsonwebtoken bcryptjs && npm install -D @types/jsonwebtoken @types/bcryptjs
- Creer src/middleware/auth.ts :

```
import jwt from 'jsonwebtoken'
import { Request, Response, NextFunction } from 'express'

export function authMiddleware(req: Request, res: Response, next: NextFunction) {
  const header = req.headers.authorization
  if (!header?.startsWith('Bearer '))
```

```
    return res.status(401).json({ error: 'Token manquant ou invalide' })
  try {
    const payload = jwt.verify(header.split(' ')[1], process.env.JWT_SECRET!)
    ;(req as any).user = payload
    next()
  } catch {
    return res.status(401).json({ error: 'Token expire ou invalide' })
  }
}
```

[BACKEND] Route POST /api/auth/login

- Fichier : src/routes/auth.ts
- Body attendu : { email: string, password: string }
- Etape 1 : chercher l'utilisateur par email

```
const user = await prisma.user.findUnique({ where: { email } })
if (!user) return res.status(401).json({ error: 'Identifiants incorrects' })
```

- Etape 2 : verifier le mot de passe

```
const valid = await bcrypt.compare(password, user.passwordHash)
if (!valid) return res.status(401).json({ error: 'Identifiants incorrects' })
```

- Etape 3 : generer et retourner le token

```
const token = jwt.sign({ userId: user.id }, process.env.JWT_SECRET!, { expiresIn: '24h'
})
res.json({ data: { token } })
```

- Tester sur Postman : POST http://localhost:3001/api/auth/login

[BACKEND] Routes CRUD Evenements — src/routes/events.ts

📋 SPEC : Acces public pour la lecture, authentification admin pour les modifications

- GET /api/events (public — participant : consulter les événements)

```
const events = await prisma.event.findMany({ orderBy: { startDate: 'asc' } })
res.json({ data: events })
```

- GET /api/events/:id (public — participant : voir liste des sessions avec statut live)

```
const event = await prisma.event.findUnique({
  where: { id: Number(req.params.id) },
  include: {
    sessions: {
      include: { room: true, speakers: true },
      orderBy: { startTime: 'asc' }
    }
  }
})
if (!event) return res.status(404).json({ error: 'Evenement introuvable' })
// Ajouter isLive a chaque session (utiliser la fonction de Nomena)
const enriched = {
  ...event,
  sessions: event.sessions.map(s => ({
    ...s,
    isLive: computeIsLive(s.startTime, s.endTime)
  }))
}
res.json({ data: enriched })
```

ℹ *Importer computeIsLive depuis src/utils/isLive.ts (cree par Nomena)*

- POST /api/events (PROTEGE — admin uniquement)
  - -> Utiliser authMiddleware dans la route
  - -> Valider title, description, startDate, endDate, location presents et non vides
  - -> Valider startDate < endDate, sinon status 400 : { error: 'La date de fin doit etre apres la date de debut' }

```
const event = await prisma.event.create({
  data: { title, description, startDate: new Date(startDate),
          endDate: new Date(endDate), location }
})
res.status(201).json({ data: event })
```

- PUT /api/events/:id (PROTEGE)

```
const event = await prisma.event.update({
  where: { id: Number(req.params.id) },
  data: { title, description, startDate: new Date(startDate),
          endDate: new Date(endDate), location }
})
res.json({ data: event })
```

- DELETE /api/events/:id (PROTEGE)

```
await prisma.event.delete({ where: { id: Number(req.params.id) } })
res.json({ message: 'Evenement supprime avec succes' })
```

Phase 3 — Frontend Next.js

[FRONTEND] app/admin/login/page.tsx — Page de connexion admin

- 'use client' en premiere ligne (composant interactif)
- Etats : email, password, errorMsg, loading
- Formulaire Tailwind avec les champs email (type=email) et mot de passe (type=password)
- Fonction handleSubmit :

```
const res = await fetch(`${process.env.NEXT_PUBLIC_API_URL}/auth/login`, {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ email, password })
})
const data = await res.json()
if (res.ok) {
  localStorage.setItem('eventsync_token', data.data.token)
  router.push('/admin/events')
} else {
  setErrorMsg(data.error)
}
```

- Créer un hook ou HOC PrivateRoute : si localStorage.getItem('eventsync_token') est null, rediriger vers /admin/login

[FRONTEND] app/events/[id]/page.tsx — Page publique d'un evenement

📋 **SPEC : Participant : afficher titre, description, dates, liste des sessions avec badge live si en cours**

- Composant serveur Next.js (pas de 'use client')

- Fetch : GET /api/events/:id — la réponse contient déjà isLive sur chaque session

```
const res = await fetch(`${process.env.NEXT_PUBLIC_API_URL}/events/${params.id}`,
  { cache: 'no-store' }) // no-store car le statut live change en temps reel
const { data: event } = await res.json()
if (!event) return notFound()
```

- Afficher : titre (text-3xl font-bold), description, dates formatées, lieu
- Formater les dates en francais :

```
new Date(event.startDate).toLocaleDateString('fr-FR', {
  weekday: 'long', day: 'numeric', month: 'long', year: 'numeric'
})
```

- Liste des sessions : pour chaque session afficher titre, heure debut-fin, salle, intervenants
- Si session.isLive === true : afficher le composant <LiveBadge /> (cree par Nomena)
- Chaque session est un lien cliquable vers /sessions/:id
- Lien vers le planning global : <Link href={`/events/${event.id}/planning`}>Voir le planning complet</Link>

[FRONTEND] app/admin/events/page.tsx — Gestion admin des evenements

📋 **SPEC : Organisateur : creer, modifier, supprimer des événements**

- 'use client' — composant interactif
- Au montage : charger la liste des événements avec le token

```
const token = localStorage.getItem('eventsync_token')
const res = await fetch(`${process.env.NEXT_PUBLIC_API_URL}/events`, {
  headers: { Authorization: `Bearer ${token}` }
})
```

- Tableau des événements : titre, dates, lieu, actions (modifier, supprimer)
- Formulaire création : titre*, description, date debut* (datetime-local), date fin* (datetime-local), lieu*
- Validation côté client : vérifier que tous les champs requis sont remplis et que fin > debut
- Soumission creation : POST /api/events avec header Authorization: Bearer token
- Soumission modification : PUT /api/events/:id
- Suppression : window.confirm puis DELETE /api/events/:id, rafraichir la liste

NOMENA — Module Sessions et Planning

📋 **SPEC : Organisateur : gerer les sessions, assigner intervenants, definir salles et horaires**
📋 **SPEC : Participant : planning multi-track, vue par salle, badge live visible dans toutes les vues**
📋 **SPEC : Regle : une session appartient a un seul evenement, une seule salle, au moins un intervenant**

Phase 1 — Schema Prisma

[CONFIG] Modeles Session et Room dans prisma/schema.prisma

```
model Room {
  id       Int       @id @default(uuid())
  name     String
  sessions Session[]
}
```

```
model Session {
  id          Int        @id @default(uuid())
  eventId     Int
  roomId      Int
  title       String
  description String
  startTime   DateTime
  endTime     DateTime
  capacity    Int        // valeur informative selon le sujet
  event       Event      @relation(fields: [eventId], references: [id], onDelete:
Cascade)
  room        Room       @relation(fields: [roomId], references: [id])
  speakers    Speaker[]  @relation('SessionSpeakers')
  questions   Question[]
  createdAt   DateTime   @default(now())
  updatedAt   DateTime   @updatedAt
}
```

- Seed : creer 3 salles (ex : Salle A, Salle B, Amphitheatre)
- Creer 6 sessions reparties sur differentes salles et horaires
- IMPORTANT pour les tests : une session DOIT avoir startTime = maintenant-30min et endTime = maintenant+30min
- S'assurer que chaque session a au moins un intervenant assigne dans la table de liaison (voir modele de Fenohasina)

Phase 2 — API Express

[BACKEND] src/utils/isLive.ts — Fonction partagee (a pousser immediatement sur Git)

⚠ **Cette fonction est utilisee par Christian (events/:id), Fenohasina (speakers/:id) et Harena (POST /questions). La pousser sur Git des qu'elle est ecrite.**

```
export function computeIsLive(startTime: Date, endTime: Date): boolean {
  const now = new Date()
  return now >= startTime && now <= endTime
}
```

[BACKEND] Routes des salles — src/routes/rooms.ts

📋 **SPEC : Les salles sont un critere de filtrage pour afficher les sessions associees**

- GET /api/rooms (public — liste toutes les salles pour les menus de navigation)

```
const rooms = await prisma.room.findMany({ orderBy: { name: 'asc' } })
res.json({ data: rooms })
```

- GET /api/rooms/:id/sessions (public — vue planning par salle)

📋 **SPEC : Afficher sessions d'une salle : heure, titre, intervenants, badge live si en cours**

```
const sessions = await prisma.session.findMany({
  where: { roomId: Number(req.params.id) },
  include: { speakers: true, room: true },
  orderBy: { startTime: 'asc' }
})
const enriched = sessions.map(s => ({
  ...s, isLive: computeIsLive(s.startTime, s.endTime)
}))
res.json({ data: enriched })
```

- POST /api/rooms (PROTEGE — admin cree les salles)
  - -> Body : { name: string }
  - -> Valider name non vide, sinon status 400

```
const room = await prisma.room.create({ data: { name } })
res.status(201).json({ data: room })
```

- DELETE /api/rooms/:id (PROTEGE)

```
await prisma.room.delete({ where: { id: Number(req.params.id) } })
res.json({ message: 'Salle supprimee' })
```

[BACKEND] Routes des sessions — src/routes/sessions.ts

📋 **SPEC : Regle : une session possede AU MOINS un intervenant — valider en backend**

- GET /api/sessions (public)

```
const sessions = await prisma.session.findMany({
  include: { room: true, speakers: true, event: true },
  orderBy: { startTime: 'asc' }
})
const enriched = sessions.map(s => ({ ...s, isLive: computeIsLive(s.startTime,
s.endTime) }))
res.json({ data: enriched })
```

- GET /api/events/:eventId/sessions (public — planning d'un evenement)

```
const sessions = await prisma.session.findMany({
  where: { eventId: Number(req.params.eventId) },
  include: { room: true, speakers: true },
  orderBy: { startTime: 'asc' }
})
const enriched = sessions.map(s => ({ ...s, isLive: computeIsLive(s.startTime,
s.endTime) }))
res.json({ data: enriched })
```

- GET /api/sessions/:id (public — detail d'une session)

```
const session = await prisma.session.findUnique({
```

```
  where: { id: Number(req.params.id) },
  include: {
    room: true,
    speakers: { include: { links: true } },
    questions: { orderBy: [{ upvotes: 'desc' }, { createdAt: 'asc' }] }
  }
})
if (!session) return res.status(404).json({ error: 'Session introuvable' })
res.json({ data: { ...session, isLive: computeIsLive(session.startTime,
session.endTime) } })
```

- POST /api/sessions (PROTEGE — admin cree une session)

⚠ **REGLE DU SUJET : valider qu'au moins un intervenant est assigne**

  - -> Body : { eventId, roomId, title, description, startTime, endTime, capacity, speakerIds: number[] }
  - -> Validation 1 : tous les champs obligatoires presents
  - -> Validation 2 : startTime < endTime
  - -> Validation 3 : speakerIds doit contenir au moins 1 element

```
if (!speakerIds || speakerIds.length === 0)
  return res.status(400).json({ error: 'Une session doit avoir au moins un intervenant'
})
```

  - -> Creation avec connexion des speakers :

```
const session = await prisma.session.create({
  data: {
    eventId, roomId, title, description, capacity,
    startTime: new Date(startTime),
    endTime: new Date(endTime),
    speakers: { connect: speakerIds.map((id: number) => ({ id })) }
  },
  include: { speakers: true, room: true }
})
res.status(201).json({ data: session })
```

- PUT /api/sessions/:id (PROTEGE — admin modifie)
  - -> Meme validations que le POST (speakerIds >= 1)
  - -> Mettre a jour les speakers : d'abord vider, puis reconnecter

```
const session = await prisma.session.update({
  where: { id: Number(req.params.id) },
  data: {
    title, description, roomId, capacity,
    startTime: new Date(startTime), endTime: new Date(endTime),
    speakers: { set: [], connect: speakerIds.map((id: number) => ({ id })) }
  },
  include: { speakers: true, room: true }
})
res.json({ data: session })
```

- DELETE /api/sessions/:id (PROTEGE)

```
await prisma.session.delete({ where: { id: Number(req.params.id) } })
res.json({ message: 'Session supprimee' })
```

Phase 3 — Frontend Next.js

[FRONTEND] components/LiveBadge.tsx — Badge partage (a pousser immediatement)

📋 **SPEC : Badge visible dans : page evenement, planning global, vue par salle**

```
export function LiveBadge() {
  return (
    <span className='inline-flex items-center gap-1.5 bg-red-500 text-white
      text-xs font-bold px-2.5 py-1 rounded-full'>
      <span className='w-2 h-2 bg-white rounded-full animate-pulse' />
      En direct
    </span>
  )
}
```

- Pousser ce fichier sur Git immediatement pour que Christian et Fenohasina puissent l'utiliser

[FRONTEND] app/events/[id]/planning/page.tsx — Planning multi-track

📋 **SPEC : Grille temporelle : sessions organisees par horaires, affichage simultane par salle, sessions paralleles cote a cote**

📋 **SPEC : Informations : titre, heure, salle, intervenants — badge live si en cours**

📋 **SPEC : Actions : acceder au detail d'une session, ajouter en favori**

- 'use client' car interactif (favoris dynamiques)
- Charger : GET /api/events/:id/sessions et GET /api/rooms
- Construire la liste des tranches horaires (toutes les 30 min entre la 1ere et derniere session)
- Structure de la grille CSS :

```
<div className='grid' style={{ gridTemplateColumns: `80px repeat(${rooms.length}, 1fr)`
}}>
  {/* Colonne heure + une colonne par salle */}
</div>
```

- Positionner chaque session dans la bonne colonne (salle) et ligne (heure) :

```
const startMinutes = s.startTime.getHours() * 60 + s.startTime.getMinutes()
const endMinutes   = s.endTime.getHours()   * 60 + s.endTime.getMinutes()
const gridRowStart = Math.floor((startMinutes - offsetMinutes) / 30) + 2
const gridRowSpan  = Math.ceil((endMinutes - startMinutes) / 30)
```

- Chaque carte de session affiche : titre, heure debut-fin, liste des intervenants, badge LiveBadge si isLive
- Chaque carte contient le bouton <FavoriteButton sessionId={s.id} /> de Harena
- Chaque carte est un lien cliquable vers /sessions/:id

[FRONTEND] app/rooms/[id]/page.tsx — Vue planning par salle

📋 **SPEC : Liste chronologique des sessions pour une salle donnee + badge live si en cours**

📋 **SPEC : Sert de critere de filtrage selon le sujet**

- Composant serveur : fetch GET /api/rooms/:id/sessions et GET /api/rooms
- Navigation en haut : liens vers chaque salle (boucle sur la liste des salles)

```
<nav className='flex gap-2 mb-6'>
  {rooms.map(room => (
    <Link key={room.id} href={`/rooms/${room.id}`}
      className={`px-4 py-2 rounded ${currentRoomId === room.id ? 'bg-blue-700
text-white' : 'border'}`}>
      {room.name}
    </Link>
  ))}
</nav>
```

- Pour chaque session : heure debut — heure fin, titre, noms des intervenants, badge LiveBadge si isLive
- Chaque session est un lien vers /sessions/:id

[FRONTEND] app/admin/sessions/page.tsx — Gestion admin des sessions

📋 **SPEC : Admin : creer sessions, assigner intervenants, definir salles et horaires**

⚠ **Validation obligatoire : au moins 1 intervenant selectionne avant de pouvoir soumettre**

- 'use client' — composant interactif
- Charger au montage : GET /api/events (pour le selecteur), GET /api/rooms, GET /api/speakers (de Fenohasina)
- Formulaire de creation/modification :
  - -> Selecteur evenement (dropdown avec les evenements disponibles)
  - -> Selecteur salle (dropdown avec les salles disponibles)
  - -> Champ titre (obligatoire)
  - -> Textarea description
  - -> Input datetime-local pour heure debut et heure fin
  - -> Input number pour capacite (affichage informatif uniquement)
  - -> Liste de checkboxes pour les intervenants (GET /api/speakers)
- Validation cote client avant soumission :

```
if (selectedSpeakerIds.length === 0) {
  setError('Vous devez selectionner au moins un intervenant')
  return
}
if (new Date(endTime) <= new Date(startTime)) {
  setError('L heure de fin doit etre apres l heure de debut')
  return
}
```

FENOHASINA — Module Intervenants et Detail Session

📋 **SPEC : Organisateur : gerer les profils des intervenants**
📋 **SPEC : Participant : page publique intervenant avec photo, bio, liens, sessions**
📋 **SPEC : Intervenant : acceder a sa page publique, voir ses sessions, voir les questions posees sur ses sessions**

Phase 1 — Schema Prisma

[CONFIG] Modeles Speaker et SpeakerLink dans prisma/schema.prisma

```
model Speaker {
  id        Int           @id @default(uuid())
  fullName  String
  photoUrl  String?
  bio       String?
  links     SpeakerLink[]
  sessions  Session[]     @relation('SessionSpeakers')
  createdAt DateTime      @default(now())
  updatedAt DateTime      @updatedAt
}
```

```
model SpeakerLink {
  id        Int     @id @default(uuid())
  speakerId Int
  platform  String  // ex: 'twitter', 'linkedin', 'website', 'github'
  url       String
  speaker   Speaker @relation(fields: [speakerId], references: [id], onDelete: Cascade)
}
```

- Seed : creer 4 intervenants, chacun avec photo, bio et au moins 1 lien externe
- Assigner ces intervenants aux sessions creees par Nomena dans le seed

Phase 2 — API Express

[BACKEND] Routes des intervenants — src/routes/speakers.ts

- GET /api/speakers (public — liste pour les selecteurs admin et l'affichage)

```
const speakers = await prisma.speaker.findMany({
  include: { links: true },
  orderBy: { fullName: 'asc' }
})
res.json({ data: speakers })
```

- GET /api/speakers/:id (public — page publique de l'intervenant)

📋 **SPEC : Intervenant : acceder a sa page, voir ses sessions ET les questions posees sur ses sessions**

```
const speaker = await prisma.speaker.findUnique({
  where: { id: Number(req.params.id) },
  include: {
    links: true,
    sessions: {
```

```
      include: {
        room: true,
        questions: { orderBy: [{ upvotes: 'desc' }, { createdAt: 'asc' }] }
      },
      orderBy: { startTime: 'asc' }
    }
  }
})
if (!speaker) return res.status(404).json({ error: 'Intervenant introuvable' })
// Ajouter isLive et enrichir les questions (authorName null -> 'Anonyme')
const enriched = {
  ...speaker,
  sessions: speaker.sessions.map(s => ({
    ...s,
    isLive: computeIsLive(s.startTime, s.endTime),
    questions: s.questions.map(q => ({
      ...q, authorName: q.authorName ?? 'Anonyme'
    }))
  }))
}
res.json({ data: enriched })
```

- POST /api/speakers (PROTEGE — admin cree un intervenant)
  - -> Body : { fullName, photoUrl?, bio?, links: [{ platform, url }] }
  - -> Valider que fullName est present et non vide

```
const speaker = await prisma.speaker.create({
  data: {
    fullName, photoUrl, bio,
    links: { create: links ?? [] }
  },
  include: { links: true }
})
res.status(201).json({ data: speaker })
```

- PUT /api/speakers/:id (PROTEGE)
  - -> Supprimer les anciens liens puis recreer les nouveaux :

```
await prisma.speakerLink.deleteMany({ where: { speakerId: Number(req.params.id) } })
const speaker = await prisma.speaker.update({
  where: { id: Number(req.params.id) },
  data: { fullName, photoUrl, bio, links: { create: links ?? [] } },
  include: { links: true }
})
res.json({ data: speaker })
```

- DELETE /api/speakers/:id (PROTEGE)

```
await prisma.speaker.delete({ where: { id: Number(req.params.id) } })
// Les SpeakerLink sont supprimes automatiquement (onDelete: Cascade)
res.json({ message: 'Intervenant supprime' })
```

Phase 3 — Frontend Next.js

[FRONTEND] app/speakers/[id]/page.tsx — Page publique de l'intervenant

📋 **SPEC : Contenu : photo de profil, nom, biographie, liens externes, liste des sessions associees**

📋 **SPEC : Intervenant peut aussi voir les questions posees sur ses sessions**

- Composant serveur : fetch GET /api/speakers/:id
- Section profil (en haut) :

```
<div className='flex gap-6 items-start'>
  {speaker.photoUrl && (
    <img src={speaker.photoUrl} alt={speaker.fullName}
      className='w-32 h-32 rounded-full object-cover shrink-0' />
  )}
  <div>
    <h1 className='text-3xl font-bold'>{speaker.fullName}</h1>
    {speaker.bio && <p className='text-gray-600 mt-3'>{speaker.bio}</p>}
    <div className='flex gap-3 mt-4'>
      {speaker.links.map(link => (
        <a key={link.id} href={link.url} target='_blank' rel='noopener noreferrer'
          className='text-blue-600 underline capitalize'>{link.platform}</a>
      ))}
    </div>
  </div>
</div>
```

- Section sessions de l'intervenant :

```
<h2 className='text-xl font-semibold mt-10 mb-4'>Ses sessions</h2>
{speaker.sessions.map(session => (
  <div key={session.id} className='border rounded-lg p-4 mb-3'>
    <div className='flex items-center gap-3'>
      {session.isLive && <LiveBadge />}
      <Link href={`/sessions/${session.id}`} className='font-semibold hover:underline'>
        {session.title}
      </Link>
    </div>
    <p className='text-sm text-gray-500 mt-1'>
      {formatTime(session.startTime)} — {formatTime(session.endTime)} |
{session.room.name}
    </p>
    {/* Questions de cette session visibles par l'intervenant */}
    {session.questions.length > 0 && (
      <details className='mt-3'>
        <summary className='text-sm text-gray-500 cursor-pointer'>
          {session.questions.length} question(s) posee(s)
        </summary>
        {session.questions.map(q => (
          <div key={q.id} className='mt-2 pl-3 border-l text-sm'>
            <p>{q.content}</p>
            <p className='text-gray-400'>{q.authorName} — {q.upvotes} upvote(s)</p>
          </div>
        ))}
      </details>
    )}
  </div>
))}
```

[FRONTEND] app/sessions/[id]/page.tsx — Page detail d'une session

📋 **SPEC : Contenu : titre, description, horaires, salle, capacite (informative), intervenants avec lien, section questions**

- Composant serveur pour la partie statique + QASection client pour les questions

- Fetch : GET /api/sessions/:id (cache: 'no-store' car isLive change en temps reel)
- Afficher en haut :
  - -> Titre (text-3xl font-bold) + badge LiveBadge si isLive === true
  - -> Salle : {session.room.name}
  - -> Horaires : {formatTime(session.startTime)} — {formatTime(session.endTime)}
  - -> Capacite : Capacite : {session.capacity} personnes (mention informative)
  - -> Description
- Section Intervenants :

```
<h2 className='text-lg font-semibold mt-8 mb-4'>Intervenants</h2>
<div className='flex flex-wrap gap-4'>
  {session.speakers.map(sp => (
    <Link key={sp.id} href={`/speakers/${sp.id}`}
      className='flex items-center gap-2 hover:underline'>
      {sp.photoUrl && <img src={sp.photoUrl} className='w-10 h-10 rounded-full
object-cover' />}
      <span className='font-medium'>{sp.fullName}</span>
    </Link>
  ))}
</div>
```

- Bouton favori (composant de Harena) :

```
<FavoriteButton sessionId={session.id} />
```

- Section Q&R (composant de Harena) :

```
<QASection sessionId={session.id} isLive={session.isLive} />
```

i *Coordonner avec Harena : si QASection n'est pas encore pret, utiliser un placeholder*

[FRONTEND] app/admin/speakers/page.tsx — Gestion admin des intervenants

📋 **SPEC : Organisateur : gerer les profils des intervenants**

- 'use client' — composant interactif
- Liste des intervenants : photo miniature, nom, nombre de sessions, boutons modifier/supprimer
- Formulaire avec gestion dynamique des liens :
  - -> Etat links : tableau de { platform: string, url: string }
  - -> Bouton 'Ajouter un lien' : setLinks([...links, { platform: '', url: '' }])
  - -> Bouton X sur chaque lien : setLinks(links.filter((_, i) => i !== index))
  - -> Apercu de la photo si photoUrl valide

```
{photoUrl && <img src={photoUrl} className='w-20 h-20 rounded-full object-cover' />}
```

HARENA — Module Questions, Favoris et Setup

📋 **SPEC : Participant : poser questions (si live), upvoter, ajouter sessions en favoris, consulter itineraire personnel**
📋 **SPEC : Regle : question soumise UNIQUEMENT si session live — validation backend ET frontend**
📋 **SPEC : Regle : upvotes non limites dans cette version**
📋 **SPEC : Regle : favoris stockes cote navigateur (localStorage), pas en base de donnees**

Phase 1 — Schema Prisma

[CONFIG] Modele Question dans prisma/schema.prisma

```
model Question {
  id         Int      @id @default(autoincrement())
  sessionId  Int
  content    String
  authorName String?  // null = anonyme selon le sujet
  upvotes    Int      @default(0)
  session    Session  @relation(fields: [sessionId], references: [id], onDelete:
Cascade)
  createdAt  DateTime @default(now())
  @@index([sessionId])
}
```

- Seed : creer 5 questions de test sur la session live de Nomena
  - -> 2 questions anonymes (authorName: null)
  - -> 3 questions avec nom
  - -> Varier les upvotes (ex: 0, 3, 7, 12, 1) pour tester le tri

Phase 2 — API Express

[BACKEND] Routes Questions et Upvotes — src/routes/questions.ts

- GET /api/sessions/:sessionId/questions (public — accessible a tous)

📋 **SPEC : Tri par nombre de upvotes decroissant**

```
const questions = await prisma.question.findMany({
  where: { sessionId: Number(req.params.sessionId) },
  orderBy: [{ upvotes: 'desc' }, { createdAt: 'asc' }]
})
// Remplacer authorName null par 'Anonyme' dans la reponse
const enriched = questions.map(q => ({
  ...q, authorName: q.authorName ?? 'Anonyme'
}))
res.json({ data: enriched })
```

- POST /api/questions (public — participant pose une question)

⚠ **REGLE CRITIQUE : valider que la session est live COTE SERVEUR avant d'inserer**

  - -> Body : { sessionId: number, content: string, authorName?: string }
  - -> Validation 1 : content non vide

```
if (!content || content.trim() === '')
```

```
  return res.status(400).json({ error: 'La question ne peut pas etre vide' })
```

- -> Validation 2 : recuperer la session et verifier qu'elle est live

```
const session = await prisma.session.findUnique({
  where: { id: Number(sessionId) }
})
if (!session) return res.status(404).json({ error: 'Session introuvable' })
if (!computeIsLive(session.startTime, session.endTime))
  return res.status(403).json({
    error: 'Les questions ne peuvent etre posees que pendant une session en cours'
  })
```

- -> Insertion en base :

```
const question = await prisma.question.create({
  data: {
    sessionId: Number(sessionId),
    content: content.trim(),
    authorName: authorName?.trim() || null  // null si vide = anonyme
  }
})
// Retourner avec authorName affiche
res.status(201).json({
  data: { ...question, authorName: question.authorName ?? 'Anonyme' }
})
```

- POST /api/questions/:id/upvote (public — upvote non limite selon le sujet)

📋 **SPEC : Les upvotes sont cumules via un compteur numerique, non limites dans cette version**

```
const updated = await prisma.question.update({
  where: { id: Number(req.params.id) },
  data: { upvotes: { increment: 1 } }
})
res.json({ data: { ...updated, authorName: updated.authorName ?? 'Anonyme' } })
```

Phase 3 — Frontend Next.js

[FRONTEND] components/QASection.tsx — Section Q&R (composant cle)

📋 **SPEC : Visible UNIQUEMENT si session en cours (isLive === true)**

📋 **SPEC : Soumission : champ texte obligatoire + nom optionnel (anonyme possible)**

📋 **SPEC : Liste triee par upvotes decroissants**

📋 **SPEC : Upvote : chaque participant peut upvoter, pas de limite**

- 'use client' — composant interactif
- Props : { sessionId: number, isLive: boolean }
- Si isLive === false : afficher message et ne rien d'autre

```
<div className='py-8 text-center'>
  <p className='text-gray-400 italic'>
    Les questions seront disponibles lors de la session
  </p>
</div>
```

- Si isLive === true — Etats : questions[], content, authorName, submitError, isSubmitting
- Chargement des questions au montage (useEffect) :

```
useEffect(() => {
  fetch(`${process.env.NEXT_PUBLIC_API_URL}/sessions/${sessionId}/questions`)
    .then(r => r.json())
    .then(d => setQuestions(d.data))
```

```
}, [sessionId])
```

- Formulaire de soumission :

```
<div className='bg-gray-50 rounded-lg p-4 mb-6'>
  <h3 className='font-semibold mb-3'>Poser une question</h3>
  <textarea
    value={content}
    onChange={e => setContent(e.target.value)}
    placeholder='Ecrivez votre question...'
    className='w-full border rounded-lg p-3 resize-none text-sm'
    rows={3} />
  {submitError && <p className='text-red-500 text-xs mt-1'>{submitError}</p>}
  <input
    value={authorName}
    onChange={e => setAuthorName(e.target.value)}
    placeholder='Votre nom (laisser vide pour rester anonyme)'
    className='w-full border rounded-lg px-3 py-2 text-sm mt-2' />
  <button onClick={handleSubmit} disabled={isSubmitting}
    className='mt-3 bg-blue-600 hover:bg-blue-700 text-white px-4 py-2
      rounded-lg text-sm disabled:opacity-50'>
    {isSubmitting ? 'Envoi...' : 'Envoyer la question'}
  </button>
</div>
```

- Fonction handleSubmit :

```
async function handleSubmit() {
  if (!content.trim()) {
    setSubmitError('Veuillez ecrire votre question')
    return
  }
  setSubmitError('')
  setIsSubmitting(true)
  const res = await fetch(`${process.env.NEXT_PUBLIC_API_URL}/questions`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ sessionId, content: content.trim(), authorName:
authorName.trim() || undefined })
  })
  const data = await res.json()
  if (res.ok) {
    setContent('')
    setAuthorName('')
    // Ajouter la nouvelle question en tete de liste
    setQuestions(prev => [data.data, ...prev])
  } else {
    setSubmitError(data.error)
  }
  setIsSubmitting(false)
}
```

- Liste des questions (triee par upvotes) :

```
<div className='space-y-3'>
  {questions.length === 0 && (
    <p className='text-gray-400 text-sm text-center py-4'>
      Aucune question pour le moment. Soyez le premier !</p>
  )}
  {questions.map(q => (
    <div key={q.id} className='flex items-start gap-4 border rounded-lg p-3'>
      <div className='flex-1'>
        <p className='text-sm'>{q.content}</p>
```

```
        <p className='text-xs text-gray-400 mt-1'>
          {q.authorName}  •  {new Date(q.createdAt).toLocaleTimeString('fr-FR')}
        </p>
      </div>
      <button onClick={() => handleUpvote(q.id)}
        className='flex flex-col items-center text-blue-600 hover:text-blue-800
shrink-0'>
        <span className='text-lg leading-none'>▲</span>
        <span className='text-sm font-bold'>{q.upvotes}</span>
      </button>
    </div>
  ))}
</div>
```

- Fonction handleUpvote : appeler l'API et mettre a jour le compteur localement

```
async function handleUpvote(questionId: number) {
  const res = await
fetch(`${process.env.NEXT_PUBLIC_API_URL}/questions/${questionId}/upvote`, {
    method: 'POST'
  })
  if (res.ok) {
    const { data } = await res.json()
    // Mettre a jour le compteur + retrier par upvotes
    setQuestions(prev =>
      [...prev.map(q => q.id === questionId ? data : q)]
        .sort((a, b) => b.upvotes - a.upvotes || new Date(a.createdAt).getTime() - new
Date(b.createdAt).getTime())
    )
  }
}
```

ℹ *Pousser ce composant sur Git des qu'il fonctionne — Fenohasina en a besoin pour la page session*

[FRONTEND] lib/favoritesService.ts — Service de favoris (localStorage)

📋 **SPEC : Favoris stockes cote navigateur — pas en base de donnees**

```
const KEY = 'eventsync_favorites'

export function getFavoriteIds(): number[] {
  if (typeof window === 'undefined') return []  // protection Next.js SSR
  try { return JSON.parse(localStorage.getItem(KEY) || '[]') }
  catch { return [] }
}

export function addFavorite(sessionId: number): void {
  const ids = getFavoriteIds()
  if (!ids.includes(sessionId))
    localStorage.setItem(KEY, JSON.stringify([...ids, sessionId]))
}

export function removeFavorite(sessionId: number): void {
  localStorage.setItem(KEY, JSON.stringify(getFavoriteIds().filter(id => id !==
sessionId)))
}

export function isFavorite(sessionId: number): boolean {
  return getFavoriteIds().includes(sessionId)
```

```
}
```

[FRONTEND] components/FavoriteButton.tsx — Bouton favori reutilisable

📋 **SPEC : Ajouter/retirer une session des favoris — utilise dans le planning et la page session**

- 'use client' — composant interactif
- Props : { sessionId: number }

```
'use client'
import { useState } from 'react'
import { addFavorite, removeFavorite, isFavorite } from '@/lib/favoritesService'

export function FavoriteButton({ sessionId }: { sessionId: number }) {
  const [fav, setFav] = useState(() => isFavorite(sessionId))
  function toggle() {
    if (fav) removeFavorite(sessionId)
    else addFavorite(sessionId)
    setFav(!fav)
  }
  return (
    <button onClick={toggle}
      className='flex items-center gap-2 text-sm border rounded-lg px-3 py-1.5
        hover:bg-gray-50 transition'>
      <span className='text-yellow-400 text-base'>{fav ? '★' : '☆'}</span>
      <span>{fav ? 'Retirer des favoris' : 'Ajouter aux favoris'}</span>
    </button>
  )
}
```

[FRONTEND] app/my-schedule/page.tsx — Itineraire personnel (favoris)

📋 **SPEC : Participant : consulter la liste des sessions favorites**

📋 **SPEC : Ajouter et retirer des sessions de l'itineraire personnel**

- 'use client' — besoin de localStorage
- Au montage : lire les IDs puis charger les details de chaque session en parallele

```
const [sessions, setSessions] = useState<any[]>([])
useEffect(() => {
  const ids = getFavoriteIds()
  if (ids.length === 0) return
  Promise.all(
    ids.map(id =>
      fetch(`${process.env.NEXT_PUBLIC_API_URL}/sessions/${id}`)
        .then(r => r.json()).then(d => d.data)
    )
  ).then(results => {
    // Trier par heure de debut
    results.sort((a, b) => new Date(a.startTime).getTime() - new
Date(b.startTime).getTime())
    setSessions(results)
  })
}, [])
```

- Affichage si aucun favori :

```
<div className='text-center py-20'>
  <p className='text-gray-400 text-lg'>Votre planning est vide.</p>
```

```
  <Link href='/' className='text-blue-600 underline mt-3 inline-block'>
    Parcourir les evenements
  </Link>
</div>
```

- Affichage des sessions favorites : titre, salle, horaires, badge live si applicable
- Bouton 'Retirer' sur chaque session :

```
function removeFromSchedule(id: number) {
  removeFavorite(id)
  setSessions(prev => prev.filter(s => s.id !== id))
}
```

Recapitulatif — Conformite avec le sujet

Verification des fonctionnalites du sujet

Fonctionnalite du sujet	Module	Porteur	Conforme
4.1 Consultation evenement : titre, desc, dates, sessions, badge live	Page /events/:id	Christian	Oui
4.2 Planning multi-track : grille horaire, salles, sessions paralleles	Page /events/:id/planning	Nomena	Oui
4.2 Planning : titre, heure, salle, intervenants affichés	Planning	Nomena	Oui
4.2 Planning : acceder detail session + ajouter favori	Planning	Nomena + Harena	Oui
4.3 Badge Live dans page evenement	Page evenement	Christian	Oui
4.3 Badge Live dans planning global	Planning	Nomena	Oui
4.3 Badge Live dans vue par salle	Vue salle	Nomena	Oui
4.4 Detail session : titre, desc, horaires, salle, capacite, intervenants, questions	Page session	Fenohasina	Oui
4.5 Q&R visible uniquement si session live	QASection	Harena	Oui
4.5 Q&R : champ texte obligatoire + nom optionnel (anonyme)	QASection	Harena	Oui
4.5 Q&R : tri par upvotes decroissant	QASection + API	Harena	Oui
4.5 Upvote : increment compteur, non limite	QASection + API	Harena	Oui
4.6 Page intervenant : photo, nom, bio, liens, sessions	Page /speakers/:id	Fenohasina	Oui
4.6 Intervenant voit les questions sur ses sessions	Page /speakers/:id	Fenohasina	Oui
4.7 Vue par salle : liste chronologique + badge live	Page /rooms/:id	Nomena	Oui
4.8 Favoris : ajouter, retirer, consulter — localStorage	FavoriteButton + my-schedule	Harena	Oui
5. Session >= 1 intervenant (validation backend + frontend)	POST/PUT sessions	Nomena	Oui
5. Question creee uniquement si session live (validation backend)	POST questions	Harena	Oui
5. Question peut etre anonyme	QASection + API	Harena	Oui

6. Auth uniquement pour admin	Middleware JWT	Christian	Oui
6. Acces public participants sans auth	Toutes pages pub.	Tous	Oui
6. Pas de moderation questions	Aucune moderation	N/A	Oui
6. Capacite non contraignante (informative)	Schema + affichage	Nomena + Fenohasina	Oui
6. Pas d'inscription aux sessions	Aucune inscription	N/A	Oui

Dependances entre membres

Qui attend	Quoi	De qui	Quand
Tous	Setup repo + structure	Christian	Avant Phase 1
Christian, Fenohasina, Harena	src/middleware/auth.ts	Christian	Debut Phase 2
Christian, Fenohasina, Harena	src/utils/isLive.ts	Nomena	Debut Phase 2
Fenohasina (page session)	QASection.tsx	Harena	Phase 3
Fenohasina + Nomena (planning)	FavoriteButton.tsx	Harena	Phase 3
Nomena (admin sessions)	GET /api/speakers	Fenohasina	Phase 2
Christian (page evenement)	isLive dans sessions	Nomena	Phase 2
Tous (migration BDD)	Schema Prisma complet	Tous ensemble	Fin Phase 1

Je vous invite à bien lire attentivement et respecter le règlement imposer dans la documentation concernant Git & Github ci dessous :

eventsync_git_reglement

Merci 😂