



RAPPORT DE PROJET ECF

Concepteur Développeur d'Applications

Titre RNCP Niveau 6

INNOV'EVENTS MANAGER

Application de gestion événementielle

Présenté par : Johann KOUAKOU

Année : 2025 – 2026

SOMMAIRE

I – Personal Presentation (English)	2
II – Présentation du projet.....	2
III – Cahier des charges	2
A - Les objectifs.....	3
B - Cibles	3
C - Exigences fonctionnelles	4
1 - Fonctionnalités Front Office (Partie publique)	4
1.1 Menu de navigation	4
1.2 Page d'accueil	4
1.3 Page Événements	4
1.4 Page Avis.....	4
1.5 Page Contact.....	4
1.6 Page Demande de devis	5
1.7 Page Mentions légales.....	5
2 - Back Office : Espace Client.....	5

2.1 Dashboard client	5
2.2 Gestion des devis	5
2.3 Gestion du profil.....	6
3 - Back Office : Espace Employé.....	6
3.1 Consultation	6
3.2 Contribution.....	6
4 - Back Office : Espace Administrateur	6
4.1 Dashboard administrateur.....	6
4.2 Gestion des prospects	6
4.3 Gestion des clients	7
4.4 Gestion des événements	7
4.5 Gestion commerciale (Devis).....	7
4.6 Gestion des comptes	7
4.7 Journalisation.....	7
5 - Confidentialité et sécurité	7
5.1 Protection des données	7
5.2 Journalisation des actions (NoSQL)	7

6 - Authentification	8
6.1 Création de compte	8
6.2 Connexion.....	8
6.3 Mot de passe oublié	8
D - Exigences et choix techniques	8
1 - Exigences techniques	8
1.1 Compatibilité navigateurs	8
1.2 Responsive Design.....	9
1.3 Accessibilité.....	9
1.4 Sécurité	9
2 - Choix technologiques	9
E - Définition du MVP (Minimum Viable Product)	10
IV – Méthodologie et organisation	11
A – La méthode en V.....	12
B – Le choix de la méthode Agile	12
C – Outil de gestion de projet : Trello	13
D – Gestion de version : Git et GitHub.....	14

E – Environnement de développement	16
F – Récapitulatif des outils collaboratifs	17
V – Conception de l'interface graphique	17
A – Zoning	18
B – Wireframe.....	21
C – Charte graphique	24
1 – Les couleurs	24
2 – La typographie	25
3 – Le logo.....	26
D – Maquettage	26
VI – Conception de la base de données	29
A – Dictionnaire des données	30
B - Dépendances fonctionnelles	30
C – Modèle Conceptuel de Données (MCD)	31
D – Modèle Logique de Données (MLD).....	33
E – Modèle Physique de Données (MPD).....	34
F – Jeu de données de test.....	36

G – Journalisation (Logs) – MongoDB	38
VII – Conception de l'application.....	38
A – Diagramme de cas d'utilisation (Use Case).....	39
B- Diagramme de séquence	40
VIII- Conception Multicouche : MVC.....	41
A – Le Modèle	42
B – La Vue	43
C – Le Contrôleur.....	45
D – Communication entre nos 3 composants	47
E – L'architecture 3 tiers	48
IX – Sécurité de l'application.....	51
A – Faille XSS (Cross-Site Scripting).....	52
B – Injection SQL	52
C – Authentification et gestion des sessions	53
D – Gestion des autorisations (RBAC).....	54
E – Protection CORS (Cross-Origin Resource Sharing)	55
F – Journalisation et traçabilité	56

G – Protection des données sensibles	57
H – Validation des données	57
X – Politique de test	57
A – Les tests unitaires	58
1 - Tests de validation du mot de passe	59
2 - Tests des calculs de montants TVA	60
B – Les tests d'intégration (API)	62
1 - Tests de l'inscription (POST /api/auth/register)	62
2 - Tests de la connexion (POST /api/auth/login)	63
3 - Tests du profil utilisateur (GET /api/auth/me)	64
C – Les tests End-to-End (E2E).....	66
D – Exécution et synthèse.....	68
XI – Veille Technologique.....	70
A – Sources de veille en cybersécurité.....	70
1 - ANSSI (Agence Nationale de la Sécurité des Systèmes d'Information)	70
2 - OWASP (Open Web Application Security Project)	71
B – Documentation officielle des technologies utilisées	72

1 - React.js.....	72
2 - Node.js et Express.js.....	72
3- PostgreSQL.....	73
4 - MongoDB et Mongoose	73
5 - Jest et Playwright	73
6 - Tailwind CSS	73
C – Communautés et forums d'entraide	74
1 - Stack Overflow	74
2 - GitHub	74
3 - Dev.to et Medium.....	75
D – Outils de veille utilisés.....	75
E – Apports de la veille pour Innov'Events	75
XII - Déploiement et Documentation	76
A - Architecture de déploiement	76
1 - Services déployés.....	77
B - Conteneurisation avec Docker.....	77
1 - Dockerfile Backend.....	77

2 - Dockerfile Frontend	78
3 - Docker Compose - Environnement complet.....	79
3.1 Schéma des conteneurs Docker	80
C - Intégration Continue et Déploiement Continu (CI/CD)	80
1 - Pipeline GitHub Actions.....	80
1-1 Étapes du pipeline CI/CD	81
D - Hébergement sur Render	82
1- Configuration	82
2 - Configuration du Frontend (Static Site)	83
3 - Configuration PostgreSQL (Render)	84
E - Services externes.....	84
1 - MongoDB Atlas (Logs)	85
2 - Cloudinary (Stockage d'images)	85
F - Flux de déploiement complet.....	86
XIII - Difficultés Rencontrées	86
A - Introduction	87
B - Les images qui disparaissaient	87

C - Les images bloquées par le navigateur	87
D - Gérer deux bases de données.....	87
E - Le pipeline CI/CD qui plante	87
F - Le site lent au premier accès.....	88
XIV - Conclusion	88
1 - Bilan du projet	88
2 - Ce qui fonctionne bien	89
3 - Axes d'amélioration	89
4 - Évolutions futures envisagées.....	89
4.1 Court terme :.....	89
4.2 Moyen terme :.....	89
4.3 Long terme :	89
5 - Mot de la fin.....	90

I – Personal Presentation (English)

Hello everyone, my name is Johann KOUAKOU, I am 26 years old, and I am currently completing my web developer training at Studi. During my internship at Innov'Events, an event management company based in Paris, I developed a full-stack web application to help them manage their prospects, clients, events, and quotes more efficiently.

II – Présentation du projet

Innov'Events Manager est une application web permettant à une entreprise d'événementiel de gérer efficacement ses prospects, clients, événements et devis, remplaçant ainsi les fichiers Excel par une solution moderne, sécurisée et accessible depuis n'importe quel navigateur.

III – Cahier des charges

A - Les objectifs

L'application Innov'Events a pour objectif principal de centraliser et professionnaliser la gestion de l'agence événementielle Innov'Events, fondée par Chloé et José. Actuellement, l'équipe utilise des outils fragmentés (fichiers Excel, documents Word, Google Calendar) qui génèrent des erreurs, des pertes de données et un manque de professionnalisme. Un incident récent a failli coûter un contrat majeur à cause d'une version obsolète du fichier client.

Les objectifs du développement sont les suivants :

- **Fiabilité** : Créer une source unique de vérité pour toutes les données (clients, événements, devis)
- **Productivité** : Automatiser les tâches répétitives (génération de PDF, envoi d'emails)
- **Collaboration** : Permettre à l'équipe de partager notes et informations en temps réel
- **Image professionnelle** : Garantir des documents standardisés et sans erreur
- **Traçabilité** : Journaliser toutes les actions sensibles pour un suivi rigoureux
- **Croissance** : Supporter l'expansion de l'entreprise avec un outil évolutif

B - Cibles

L'application vise plusieurs profils d'utilisateurs avec des besoins et des droits distincts

Profil	Description	Besoins principaux
Visiteur	Personne non connectée	Découvrir l'agence, demander un devis
Client	Utilisateur avec compte	Gérer ses devis, suivre ses événements, évaluer
Employé	Membre de l'équipe	Consulter projets, ajouter notes, gérer tâches
Employé	Chloé (gérante)	Gestion complète : clients, devis, événements

La cible principale est l'équipe interne d'Innov'Events (8 personnes), mais l'application doit également offrir une interface client fluide pour renforcer l'image professionnelle de l'agence.

C - Exigences fonctionnelles

Cette section détaille les fonctionnalités attendues de l'application, organisées par espace utilisateur.

1 - Fonctionnalités Front Office (Partie publique)

1.1 Menu de navigation

Un menu de navigation doit être visible sur toutes les pages et contenir les liens suivants :

- Accueil
- Se connecter / Se déconnecter
- Événements
- Avis
- Contact
- Demande de devis

1.2 Page d'accueil

La page d'accueil est la vitrine de l'agence. Elle doit contenir :

- Présentation de l'entreprise (texte court et professionnel)
- Images illustratives (événements réalisés)

- Bouton visible "Demander un Devis" redirigeant vers le formulaire

1.3 Page Événements

Cette page affiche tous les événements réalisés (avec accord client, statut différent de "brouillon"). Les informations affichées incluent une image et les détails de l'événement (sans le prix).

Fonctionnalités de filtrage :

- Par plage de dates
- Par type d'événement
- Par thème

1.4 Page Avis

Cette page affiche les avis clients validés par les employés. Seuls les avis approuvés sont visibles publiquement.

1.5 Page Contact

Un formulaire permettant aux visiteurs et utilisateurs de contacter l'agence :

- Nom d'utilisateur (facultatif)
- Titre de la demande (obligatoire)
- Adresse email (obligatoire si non connecté)
- Description du message (obligatoire)

1.6 Page Demande de devis

Formulaire de prise de contact pour les prospects. Tous les champs sont obligatoires :

- Nom de l'entreprise
- Nom et prénom du contact
- Email et téléphone
- Lieu de l'événement
- Type d'événement (Séminaire, Conférence, Soirée d'entreprise, Autre)
- Date souhaitée
- Nombre de participants estimé
- Description du besoin

À la soumission, les données sont enregistrées dans la table "prospects" avec le statut "à contacter". Un email est envoyé à contact@innovevents.com et un message de remerciement est affiché.

1.7 Page Mentions légales

Page accessible depuis le pied de page contenant les informations obligatoires :

- Éditeur : Raison sociale, forme juridique, adresse, capital social
- Contact : Email et téléphone
- Responsable de la publication
- Hébergeur : Nom, adresse, téléphone

2 - Back Office : Espace Client

L'espace client est accessible après connexion. Il permet au client de gérer ses interactions avec l'agence.

2.1 Dashboard client

- Widget "Prochains événements" : Affiche les 3 événements à venir
- Liste des devis avec leur statut
- Accès rapide à la modification du profil

2.2 Gestion des devis

Le client peut pour chaque devis reçu :

- Accepter : Le statut passe en "accepté", email envoyé à Innov'Events
- Demander une modification : Saisie d'un motif, statut "modification"
- Refuser : Le statut passe en "refusé"
- 2.3 Suivi des événements
- Visualiser l'évolution de ses demandes
- Évaluer un événement une fois terminé (avis)

2.3 Gestion du profil

- Modifier ses informations personnelles
- Demander la suppression de son compte et de ses données (RGPD)

3 - Back Office : Espace Employé

- L'employé dispose de droits de consultation et de contribution sans pouvoir modifier les éléments structurels.

3.1 Consultation

- Clients : Liste, recherche et fiches détaillées
- Événements : Liste et détails (dates, lieu, statut)

3.2 Contribution

- Notes : Ajouter, modifier, supprimer des notes sur les événements
- Tâches : Gérer le statut de ses tâches assignées (à faire → en cours → terminé)

- Avis : Valider ou refuser les avis soumis par les clients

4 - Back Office : Espace Administrateur

L'administrateur (Chloé) a accès à toutes les fonctionnalités. Son compte est créé par défaut (non créable depuis l'application).

4.1 Dashboard administrateur

- Widget "Prochains événements" : 3 événements les plus proches
- Widget "Notes récentes" : 5 dernières notes ajoutées
- Widget "Indicateurs clés" : Clients actifs, devis en attente, événements en brouillon

4.2 Gestion des prospects

- Voir la liste des prospects demandes de devis
- Convertir un prospect en client
- Passer une demande en "échoué" avec envoi d'email explicatif

4.3 Gestion des clients

- Créer, modifier, supprimer des clients
- Consulter l'historique des événements d'un client

4.4 Gestion des événements

- Créer un événement avec : nom, dates, lieu, statut, visibilité
- Ajouter une image de l'événement
- Modifier le statut : brouillon → accepté → en cours → terminé / annulé
- Ajouter des notes et des tâches

4.5 Gestion commerciale (Devis)

- Associer des prestations à un événement (libellé + montant HT)
- Calcul automatique TVA et TTC
- Générer le devis en PDF
- Envoyer le devis par email au client
- Visualiser les devis acceptés pour passer à l'étape suivante

4.6 Gestion des comptes

- Créer, suspendre, supprimer des comptes employés
- Créer, suspendre, supprimer des comptes clients

4.7 Journalisation

- Accès aux logs de toutes les actions sensibles

5 - Confidentialité et sécurité

5.1 Protection des données

- Les données personnelles des utilisateurs sont protégées (lecture/écriture contrôlées)
- Possibilité de supprimer un compte sur demande
- Les contenus "privés" ne sont visibles que par leur créateur et les admins autorisés

5.2 Journalisation des actions (NoSQL)

Chaque action sensible génère une entrée de log avec :

- Horodatage (date/heure)
- Type d'action (ex: CREATION_CLIENT, CONNEXION_REUSSIE)
- ID de l'utilisateur concerné • Détails contextuels (JSON)

Actions journalisées :

- Connexion réussie / échouée (avec IP)
- CRUD clients (création, modification, suppression)
- CRUD événements et changements de statut
- Génération de devis PDF

Note : La collecte d'adresses IP est soumise au RGPD. L'application doit informer les utilisateurs et obtenir leur consentement.

6 - Authentification

6.1 Crédit de compte

Champs requis :

- Email (login)
- Mot de passe : 8 caractères minimum, 1 majuscule, 1 minuscule, 1 chiffre, 1 caractère spécial
- Prénom, nom, nom d'utilisateur

Un email de confirmation est envoyé à l'utilisateur après création du compte.

6.2 Connexion

- Authentification par email + mot de passe
- Redirection vers l'action souhaitée après connexion

6.3 Mot de passe oublié

- Génération automatique d'un mot de passe temporaire
- Envoi par email
- Obligation de le modifier à la prochaine connexion

D - Exigences et choix techniques

1 - Exigences techniques

1.1 Compatibilité navigateurs

L'application doit fonctionner sur les navigateurs principaux :

- Google Chrome (dernières versions)
- Mozilla Firefox (dernières versions)
- Microsoft Edge (dernières versions)



Figure 1 : Les navigateurs compatibles

1.2 Responsive Design

L'application doit être responsive et s'adapter à tous les types d'écrans (desktop, tablette, smartphone). L'accent est mis sur l'expérience mobile pour les clients qui consultent depuis leur téléphone.

1.3 Accessibilité

L'application doit respecter les principes du RGAA (Référentiel Général d'Amélioration de l'Accessibilité) :

- Navigation au clavier
- Structure HTML sémantique
- Alternatives textuelles pour les images

1.4 Sécurité

- Hashage des mots de passe (bcrypt)
- Protection CSRF
- Validation des entrées côté serveur
- Authentification par JWT (JSON Web Token)
- HTTPS obligatoire en production

2 - Choix technologiques

Composant	Technologie	Justification
Backend	Node.js / Express	Performance, écosystème riche, JavaScript unifié
Base SQL	PostgreSQL	Robuste, relationnelle, open source
Base NoSQL	MongoDB	Flexible pour les logs, documents JSON
Frontend	HTML/CSS/JS ou React	Selon complexité et temps disponible
CSS Framework	Bootstrap / Tailwind	Responsive facilité, composants prêts
Containerisation	Docker	Environnement reproductible
Versioning	Git + GitHub	Collaboration, historique, branches
Authentification	JWT	Stateless, sécurisé, scalable
Email	Nodemailer	Envoi d'emails depuis Node.js
PDF	PDFKit / Puppeteer	Génération de devis PDF

E - Définition du MVP (Minimum Viable Product)

Dans le but d'atteindre le public cible dans un délai optimum, une première version "MVP" de l'application sera développée. Cette version doit offrir les fonctionnalités essentielles permettant de démontrer le flux principal du métier.

Fonctionnalités incluses dans le MVP :

Priorité 1 (Indispensable)

Fonctionnalité	Description
Authentification	Inscription, connexion, mot de passe oublié
Page d'accueil	Présentation + bouton devis
Page Contact	Formulaire de contact
Mentions légales	Informations juridiques
Demande de devis	Formulaire complet → table prospects
Espace Admin : Prospects	Liste, conversion en client
Espace Admin : Clients	CRUD complet
Espace Admin : Devis	Création, prestations, génération PDF

Priorité 2 (Important)

Fonctionnalité	Description
Espace Client	Voir devis, accepter/refuser, profil
Page Événements	Liste avec filtres
Espace Admin : Événements	CRUD événements
Envoi d'emails	Confirmation, devis, notifications

Priorité 3 (Secondaire)

Fonctionnalité	Description
Espace Employé	Consultation, notes, tâches

Page Avis	Affichage + validation
Journalisation	Logs des actions (MongoDB)
Dashboard avancé	Widgets, indicateurs

IV – Méthodologie et organisation

Dans le cadre de ce projet, j'ai fait le choix de l'utilisation de la méthode Agile. Auparavant on plébiscitait la méthode en V qui met l'accent sur la séquentialité et la planification rigoureuse des activités.

A – La méthode en V

La méthode en V est une approche traditionnelle de gestion de projet qui se décompose en deux phases principales :

- **Phase descendante** : Analyse des besoins → Spécifications → Conception → Implémentation
- **Phase ascendante** : Tests unitaires → Tests d'intégration → Validation → Déploiement

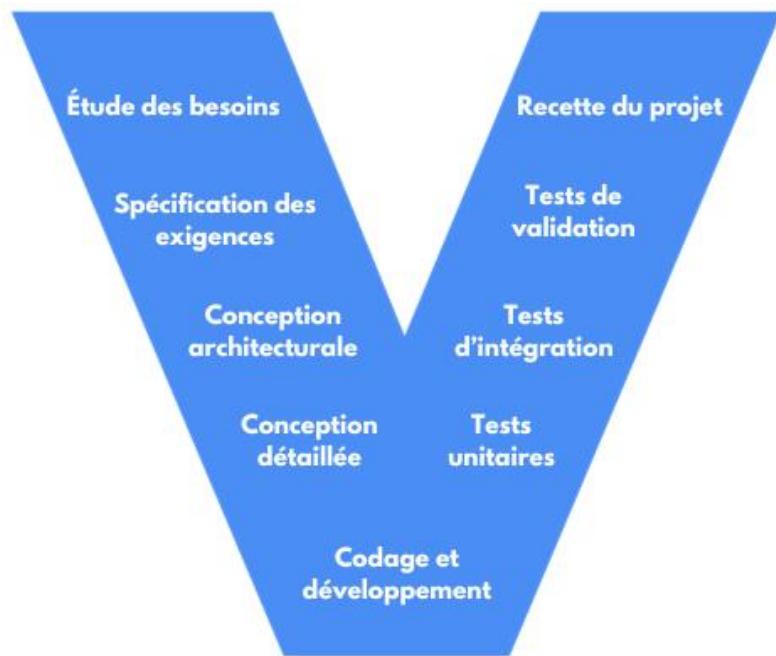


Figure 2 : Schéma représentant les différentes étapes de la méthode en V

La méthode en V permet une meilleure compréhension et définition des besoins du projet dès le début. Elle permet de consacrer du temps à l'analyse approfondie des exigences fonctionnelles et non fonctionnelles, ainsi qu'à la définition claire des fonctionnalités attendues.

Cependant, cette méthode présente des limites : elle est rigide et ne permet pas facilement d'intégrer des changements en cours de projet. Le client ne voit le produit final qu'à la toute fin, ce qui peut générer des incompréhensions.

B – Le choix de la méthode Agile

Mais j'ai choisi une approche plus moderne : **la méthode Agile**.

La méthode Agile repose sur des cycles de développement courts appelés "itérations" ou "sprints". Elle favorise la collaboration, l'adaptation au changement et la livraison régulière de fonctionnalités.

Pourquoi ce choix pour le projet Innov'Events Manager ?

- **Flexibilité** : Possibilité d'ajuster les priorités en fonction des retours
- **Livraisons régulières** : Le client peut voir l'avancement rapidement
- **Gestion du risque** : Les problèmes sont détectés tôt grâce aux itérations courtes
- **Priorisation** : Focus sur les fonctionnalités essentielles (MVP) en premier

Le framework Kanban

Parmi les frameworks Agile existants (Scrum, Kanban, XP...), j'ai opté pour Kanban car il est particulièrement adapté à un projet individuel. Il offre une visualisation claire de l'avancement sans la complexité des cérémonies Scrum (daily meetings, sprint planning, etc.).

Le principe de Kanban repose sur un tableau visuel où les tâches se déplacent de gauche à droite, de "À faire" vers "Terminé". Cela permet d'avoir une vision claire sur les tâches accomplies, celles en cours, et celles restantes.

C – Outil de gestion de projet : Trello

Afin de mettre en place le Kanban et d'organiser mes tâches, j'ai utilisé l'application Trello. Cet outil permet de créer un tableau collaboratif avec des colonnes et des cartes représentant les tâches.

Structure du tableau Kanban

Conformément aux exigences du projet, mon tableau Kanban comprend les colonnes suivantes :

Colonne	Description
Backlog	Toutes les fonctionnalités prévues, ordonnées par priorité
Sprint	Fonctionnalités prévues pour le sprint en cours
In Progress	Fonctionnalités en cours de développement
Done (dev)	Fonctionnalités terminées sur la branche dev
Merged (main)	Fonctionnalités mergées dans la branche principale

Grâce à ce tableau, j'ai pu organiser mes plages de travail en commençant par les tests des tickets en attente de passage à la colonne "Done", pour passer ensuite à la sélection des tickets à traiter dans la journée.

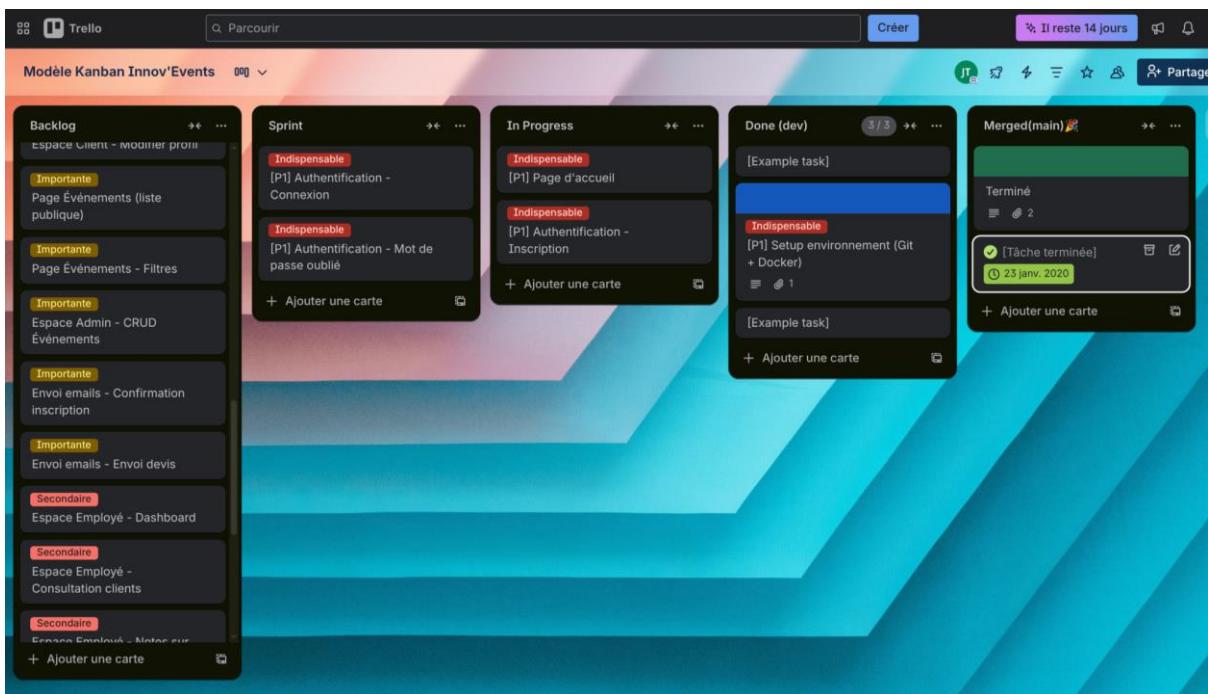


Figure 3 : Capture d'écran de mon tableau Kanban sur Trello

Gestion des priorités

Les cartes du Backlog sont ordonnées par priorité, en cohérence avec la définition du MVP :

- Priorité 1 (P1) : Fonctionnalités indispensables (authentification, demande de devis, admin de base)
- Priorité 2 (P2) : Fonctionnalités importantes (espace client, événements)
- Priorité 3 (P3) : Fonctionnalités secondaires (espace employé, avis, journalisation)

D – Gestion de version : Git et GitHub

Pour le versioning de l'application, j'ai fait le choix de **GitHub** (via Git).

Outre son avantage pour le travail collaboratif, GitHub offre d'une part la disponibilité du projet d'un poste à un autre très aisément grâce au dépôt distant. La commande git clone permet de télécharger l'intégralité du code de l'application.

D'autre part, grâce au système de versionnement, le code déjà validé (commité) est protégé. Si une erreur venait à être publiée, GitHub via Git permet le retour à une version antérieure sur laquelle on est sûr du fonctionnement de l'application.

Stratégie de branches

Le projet utilise deux branches principales :

Branche	Rôle	Règle

Main	Production	Code stable uniquement, merge depuis dev
Dev	Développement	Travail quotidien, tests

Workflow Git

Le workflow suivi est le suivant :

- Développement sur la branche dev
- Commits réguliers avec messages descriptifs
- Push sur le dépôt distant (GitHub)
- Quand une fonctionnalité est stable → merge dans main
- Mise à jour de la colonne "Merged" sur Trello

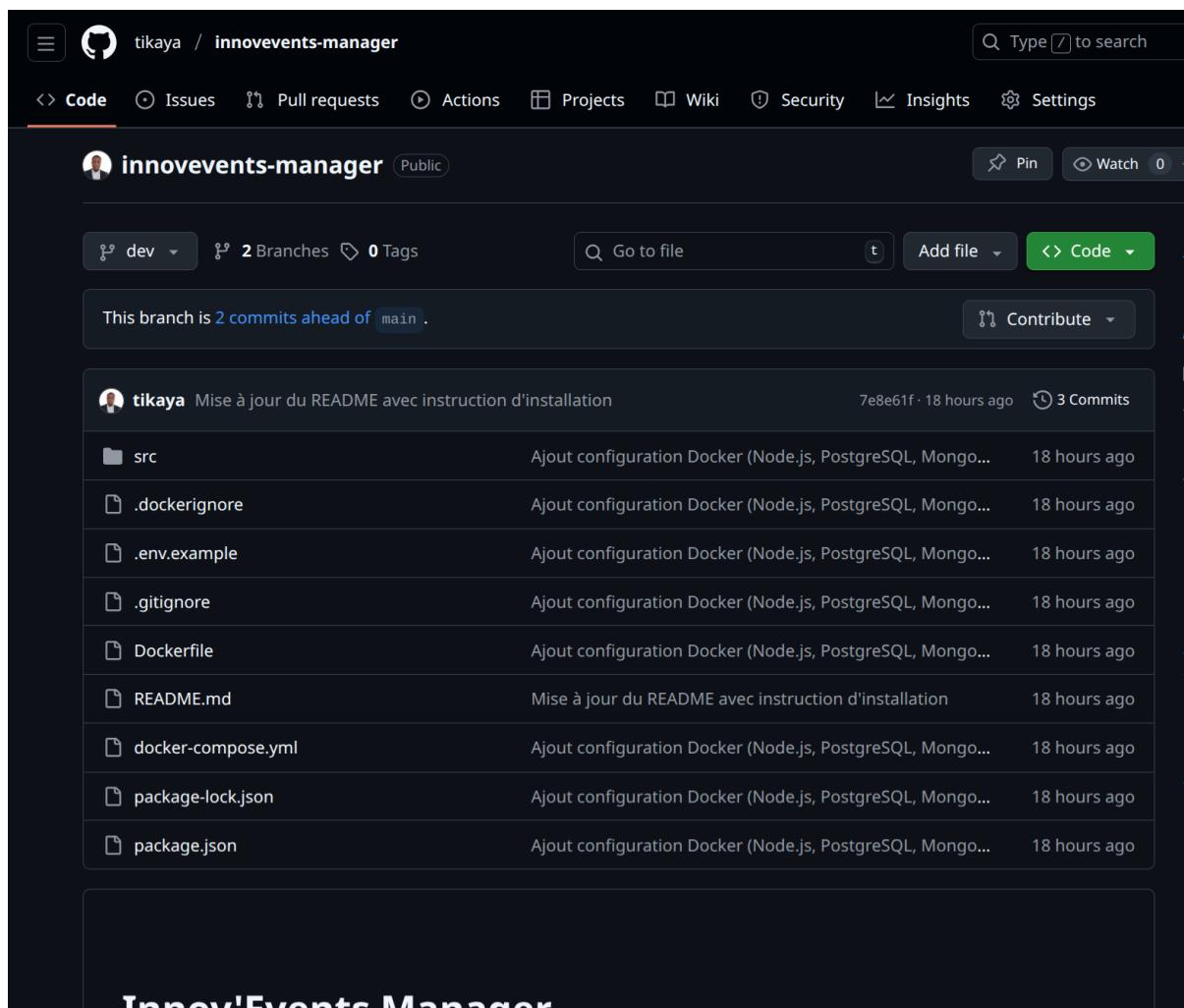


Figure 4 : Capture d'écran du repository GitHub avec les branches

```
tikaya@tikaya-KLVD-WXX9:~/innovevents-manager$ git commit -m "Mise à jour du README avec instruction d'installation"
[dev 3458617] Mise à jour du README avec instruction d'installation
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 e.txt
```

Figure 5 : Exemple de commit sur le projet

E – Environnement de développement

L'environnement de développement a été défini en adéquation avec l'architecture du projet. Il repose sur la conteneurisation avec Docker pour garantir un environnement reproductible.

Composant	Outil	Version
Conteneurisation	Docker Desktop	4.x
Orchestration	Docker Compose	V2
Backend	Node.js	20-alpine
Framework	Express	4.x
Base SQL	PostgreSQL	15-alpine
Base NoSQL	MongoDB	7
Versioning	Git + GitHub	2.x
IDE	Visual Studio Code	Latest

L'utilisation de Docker permet à tout développeur de récupérer le projet et de le lancer avec une simple commande docker compose up, sans avoir à configurer manuellement les dépendances.

F – Récapitulatif des outils collaboratifs

Les outils collaboratifs ont été choisis en fonction de la méthode de développement Agile/Kanban :

Besoin	Outil choisi	Justification

Gestion de projet	Trello	Tableau Kanban visuel et collaboratif
Versioning	Git + GitHub	Historique, branches, collaboration
Conteneurisation	Docker	Environnement reproductible
Communication	Email / Discord	Échanges avec le client si besoin
Documentation	Markdown / PDF	README, documentation technique

Cette organisation permet de planifier les tâches en fonction du délai défini, d'identifier les éventuels retards grâce au tableau Kanban, et de mettre en œuvre les procédures qualité (tests, revue de code, commits structurés).

V – Conception de l'interface graphique

La conception de l'interface utilisateur (UI) et de l'expérience utilisateur (UX) occupe une place primordiale dans le développement d'une application. Ce chapitre se concentre sur cette phase de conception, où nous avons accordé une attention particulière à l'aspect visuel, à l'ergonomie et à l'expérience globale des utilisateurs.

L'objectif principal de la conception UI/UX est de créer une interface intuitive, attrayante et conviviale, qui offre une expérience utilisateur agréable et engageante. Je me suis efforcé de concevoir des interactions fluides et des fonctionnalités intuitives qui répondent aux besoins du public cible, tout en reflétant l'identité de mon projet.

Pour rappel, les étapes de conception graphique sont : Zoning → Wireframe → Maquettes

A – Zoning

Le zoning fait référence à un processus qui consiste à représenter l'interface utilisateur de façon schématisée de manière à faire apparaître les zones qui contiendront chacune un élément de l'interface.

Ce procédé a pour objectif d'organiser de manière claire et structurée les éléments de l'interface utilisateur. En divisant l'écran en zones dédiées, il devient plus facile pour les utilisateurs de comprendre et de naviguer dans l'application.

Le zoning permet aussi de mettre en évidence la hiérarchie de l'information en fonction de son importance.

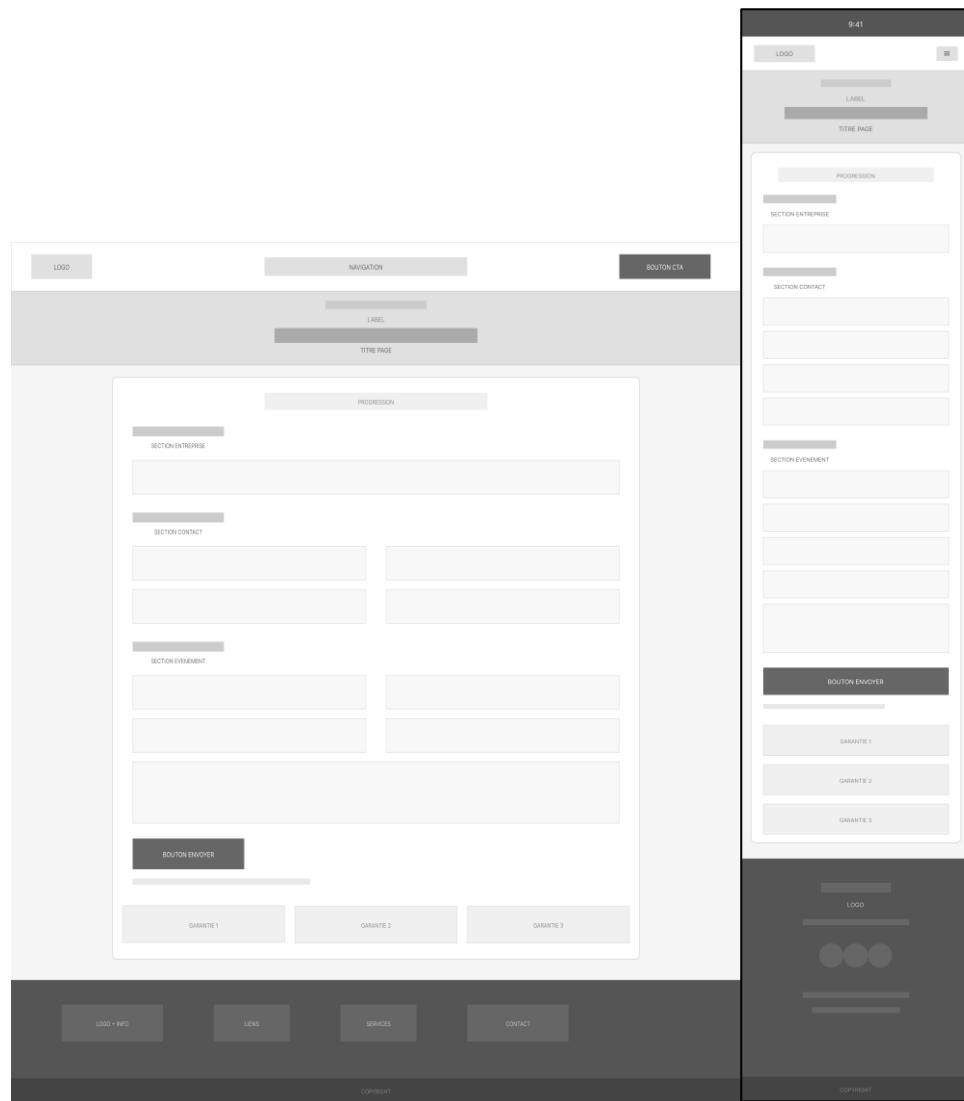


Figure 6 : Zoning de la page de demande devis web et mobile



Figure 7 : Zoning de la page d'accueil web et mobile

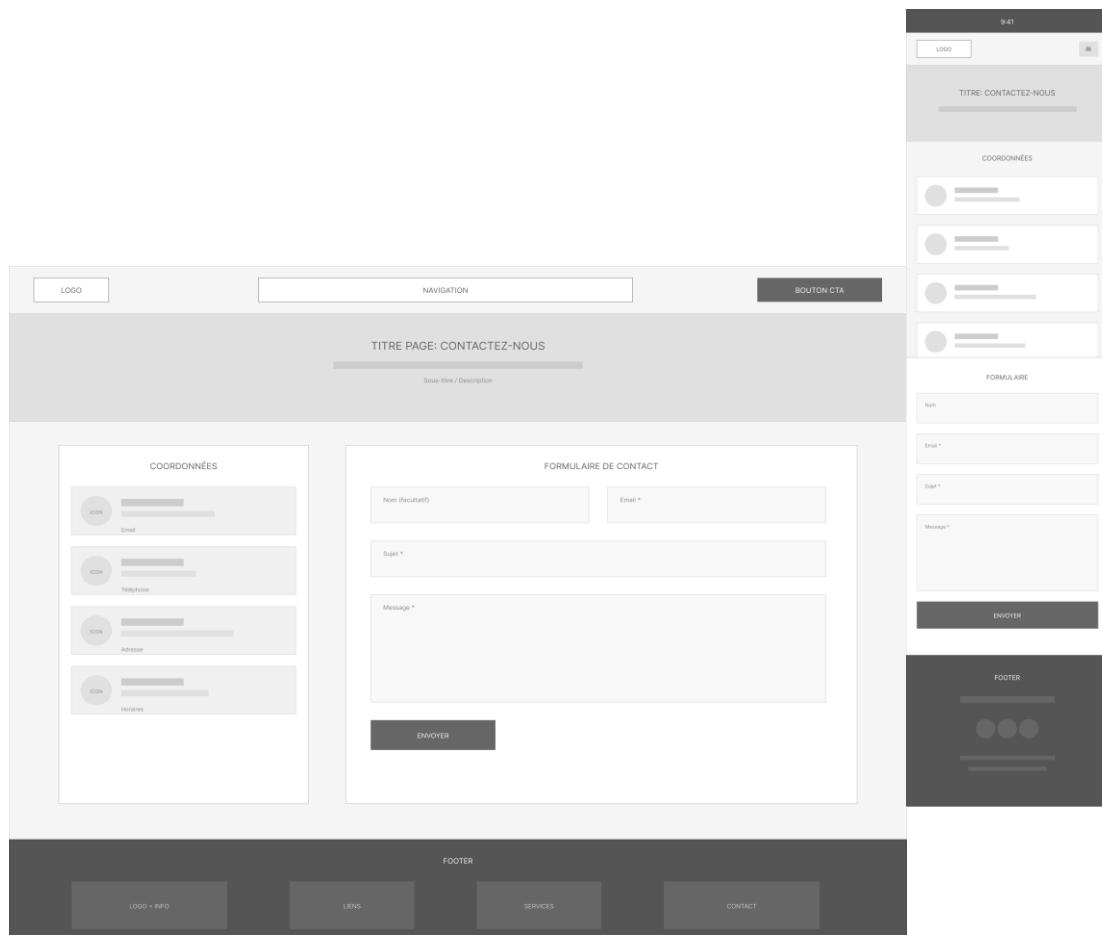


Figure 8 : Zoning de la page de demande de contact web et mobile

B – Wireframe

Le wireframe est une représentation visuelle simplifiée de l'interface utilisateur, toutefois elle rentre plus en détails dans les éléments déterminés lors du zoning. Il s'agit essentiellement d'une esquisse ou d'un schéma qui met en évidence la structure, la disposition et les interactions de base des éléments de l'interface.

Ce schéma permet de définir la structure de l'application en identifiant les différentes sections, les zones d'interaction et les relations entre les éléments. Il met en évidence la disposition générale de l'interface, comme la navigation, les menus, les formulaires, etc.



Figure 9 : Wireframe de la page d'accueil (version web et mobile)

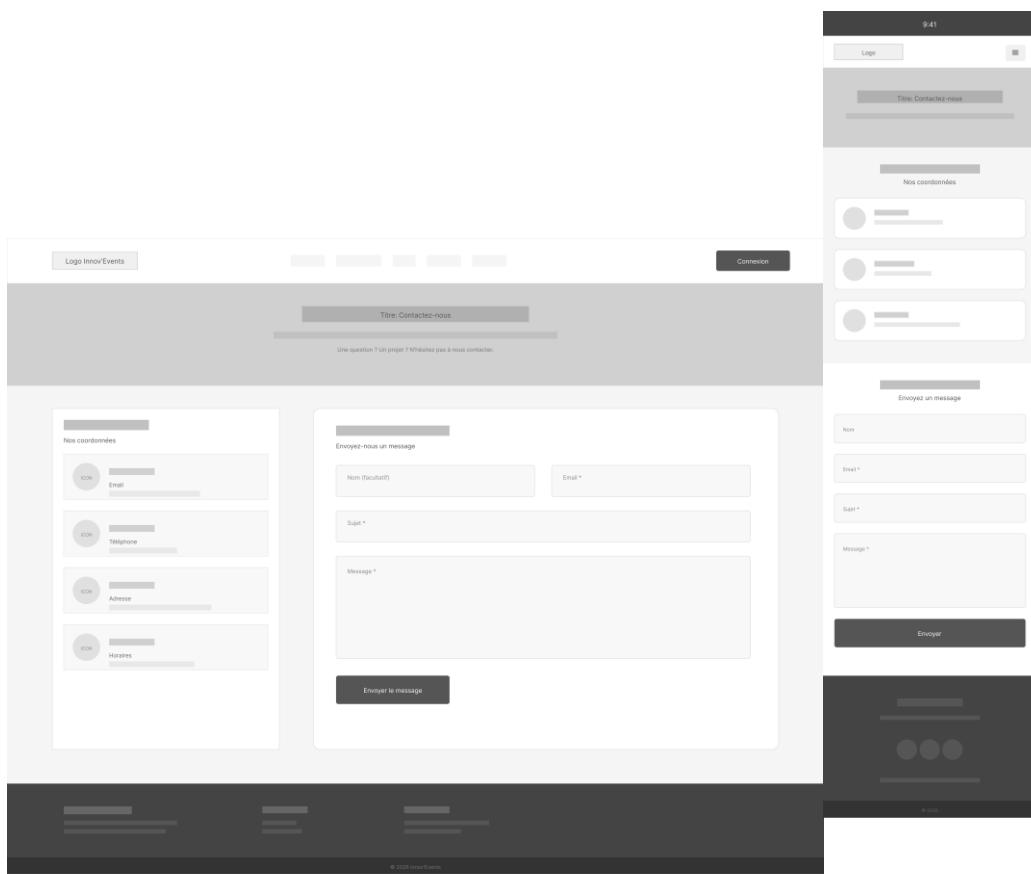


Figure 10 : Wireframe de la page Demande de contact (version web et mobile)

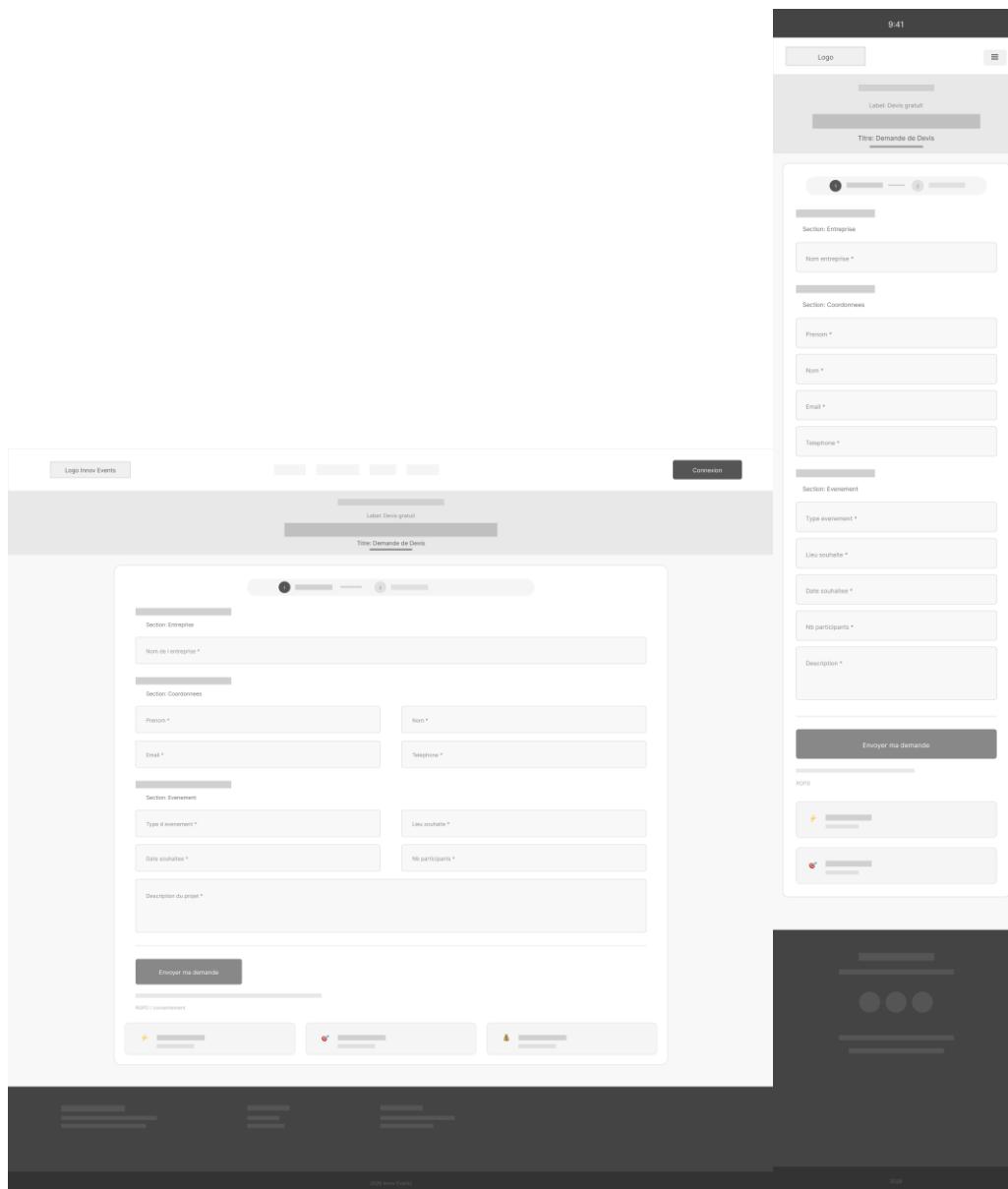


Figure 11 : Wireframe de la page Demande de devis (version web et mobile)

C – Charte graphique

Afin de m'aider à avancer dans le processus de création de mon interface utilisateur, j'ai pris la décision de me lancer dans l'élaboration de ma charte graphique, le but étant de me permettre d'avoir des éléments pour visualiser l'interface que je souhaiterai développer pour mon application.

1 – Les couleurs

L'élément qui selon moi paraissait le plus simple à déterminer a été le choix de la couleur principale. En effet, l'idée de faire une application pour une agence événementielle « haut de

gamme » m'a orienté vers des couleurs qui évoquent le professionnalisme, la confiance et le prestige.

Je me suis inspiré des codes visuels du secteur événementiel de luxe pour définir une palette de couleurs élégante et professionnelle.

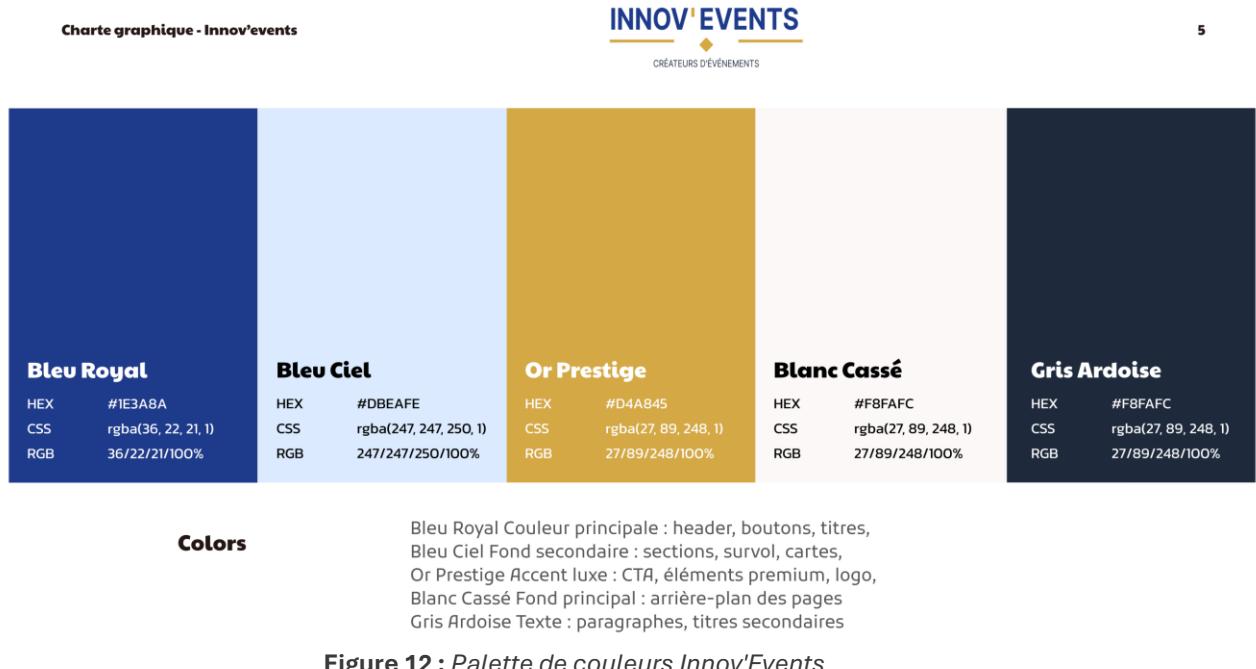


Figure 12 : Palette de couleurs Innov'Events

2 – La typographie

Pour la typographie, j'ai choisi des polices qui reflètent l'image professionnelle et moderne d'Innov'Events tout en garantissant une excellente lisibilité sur tous les supports.

Utilisation	Police	Style	Exemple
Titres principaux	Montserrat	Bold (700)	INNOV'EVENTS
Sous-titres	Montserrat	Semi-Bold (600)	Nos services
Corps de texte	Open Sans	Regular (400)	Lorem ipsum dolor...
Boutons / CTA	Montserrat	Medium (500)	Demander un devis

Ces polices sont disponibles gratuitement sur Google Fonts et sont optimisées pour l'affichage web.

3 – Le logo

Le logo d'Innov'Events a été conçu pour refléter les valeurs de l'entreprise : innovation, élégance et professionnalisme.

Il combine les couleurs principales de la charte graphique (bleu et or) pour créer une identité visuelle forte et reconnaissable. Le logo existe en plusieurs déclinaisons pour s'adapter aux différents supports :



Figure 13 : Logo Innov'Events version bleu et or

D – Maquettage

Le maquettage fait référence à la création de maquettes graphiques détaillées de l'interface utilisateur. Il s'agit d'une représentation visuelle plus élaborée et esthétique de l'application, qui inclut des éléments de design tels que les couleurs, les typographies, les icônes, etc.

Les maquettes permettent une visualisation précise de l'apparence et du ressenti de l'interface utilisateur. Elles donnent une idée réaliste de l'aspect final de l'application, en intégrant les éléments visuels et les détails de conception. Elles servent de référence visuelle pour les développeurs lors de la création de l'interface utilisateur.

Elles définissent les styles, les mises en page et les interactions attendues, ce qui facilite le processus de développement en réduisant les ambiguïtés et les erreurs potentielles.

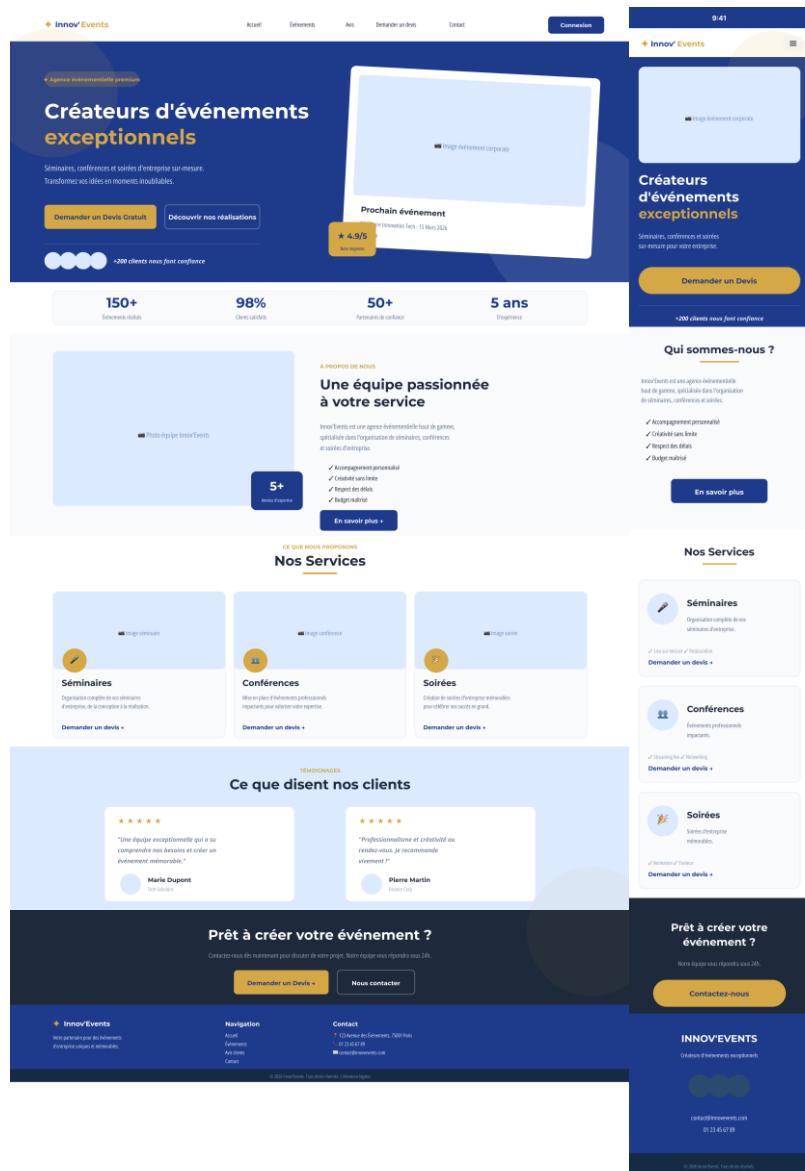


Figure 14 : Maquette de la page d'accueil (version web et mobile)

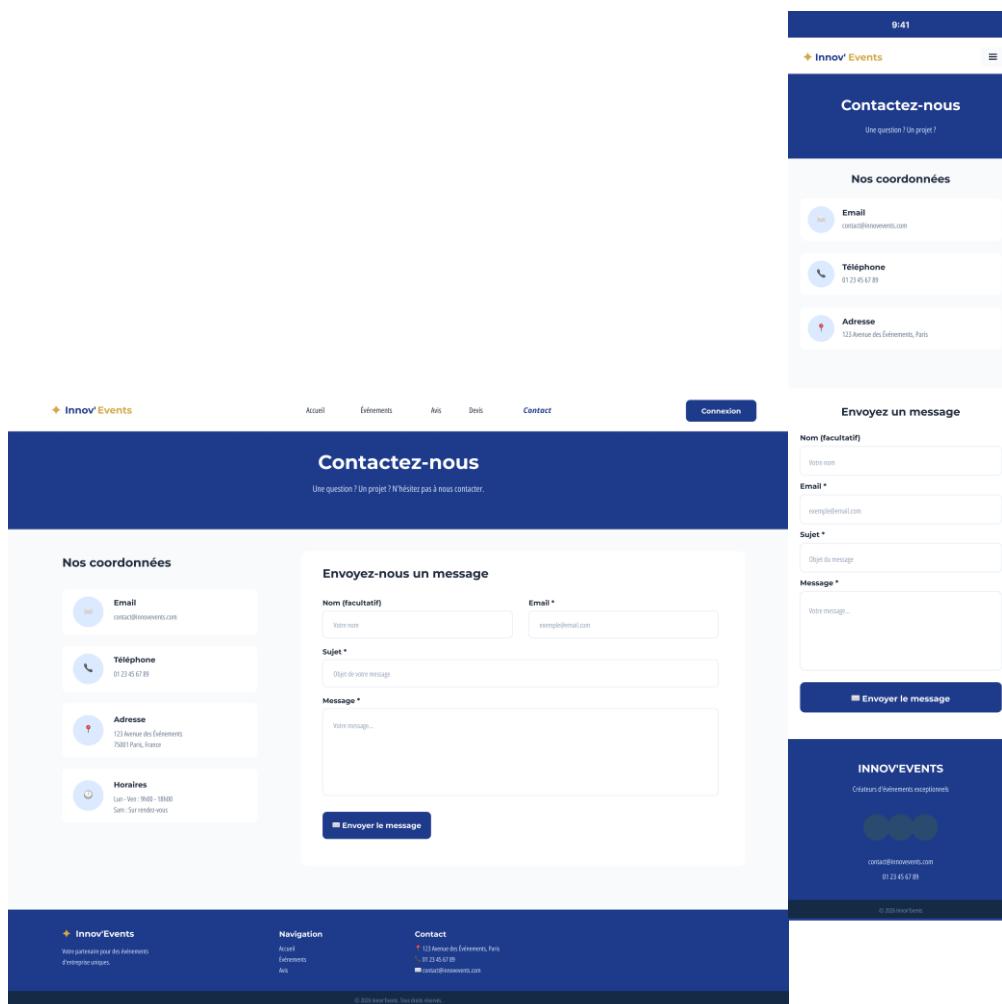


Figure 15 : Maquette de la page de Contact (version web et mobile)

Figure 16 : Maquette de la page Demande de devis (version web et mobile)

VI – Conception de la base de données

Introduction

Une étude visant à structurer la base de données d'une application n'est pas un processus simple mais il est nécessaire si on ne souhaite pas développer à l'aveugle. En effet, réaliser cette étude aide à comprendre les données qui seront utilisées par l'application. Cela permet de modéliser et de représenter les entités, les relations et les attributs clés nécessaires pour le fonctionnement de l'application.

Pour réaliser cette étude, j'ai fait usage de la méthode Merise, il s'agit d'une méthode de conception et de modélisation des systèmes d'information. La méthode Merise se compose de plusieurs étapes clés, notamment l'analyse des besoins, la modélisation conceptuelle des données (MCD), la modélisation logique des données (MLD) et la modélisation physique des données (MPD). La méthode Merise permet une conception rigoureuse des bases de données.

Pour le projet Innov'Events Manager, j'ai identifié 10 entités principales nécessaires au fonctionnement de l'application : Utilisateur, Prospect, Client, Événement, Devis, Prestation, Note, Tâche, Avis et Contact. Une collection MongoDB supplémentaire a été prévue pour la journalisation des actions (Logs).

A – Dictionnaire des données

Le dictionnaire des données est un document essentiel qui recense l'ensemble des données manipulées par le système d'information. Il permet de définir précisément chaque donnée avec son format, sa longueur, son type (élémentaire ou calculé) ainsi que les règles de gestion associées.

Pour le projet Innov'Events Manager, j'ai identifié 109 données réparties dans 11 entités. Parmi celles-ci, 101 sont des données élémentaires (saisies directement) et 8 sont des données calculées (déduites d'autres données).

Les données calculées identifiées sont notamment les montants TTC des devis et des prestations, calculés à partir des montants HT et du taux de TVA.

B - Dépendances fonctionnelles

Les dépendances fonctionnelles permettent d'identifier les liens entre les données. Une dépendance fonctionnelle existe entre deux données A et B lorsque la connaissance de la valeur de A permet de déterminer de façon unique la valeur de B. On note cette relation : $A \rightarrow B$.

Pour le projet Innov'Events Manager, j'ai identifié 11 sources (clés primaires) et 92 dépendances fonctionnelles. Toutes les dépendances fonctionnelles sont élémentaires et directes, ce qui garantit une bonne normalisation de la base de données.

Exemple de dépendance fonctionnelle :

- $\text{id_client} \rightarrow (\text{nom_entreprise_client}, \text{nom_contact}, \text{prenom_contact}, \text{email_client}, \text{telephone_client}, \text{adresse_client}, \text{code_postal_client}, \text{ville_client})$

Cela signifie que si je connais l'identifiant d'un client, je peux déterminer de façon unique toutes les informations qui lui sont associées.

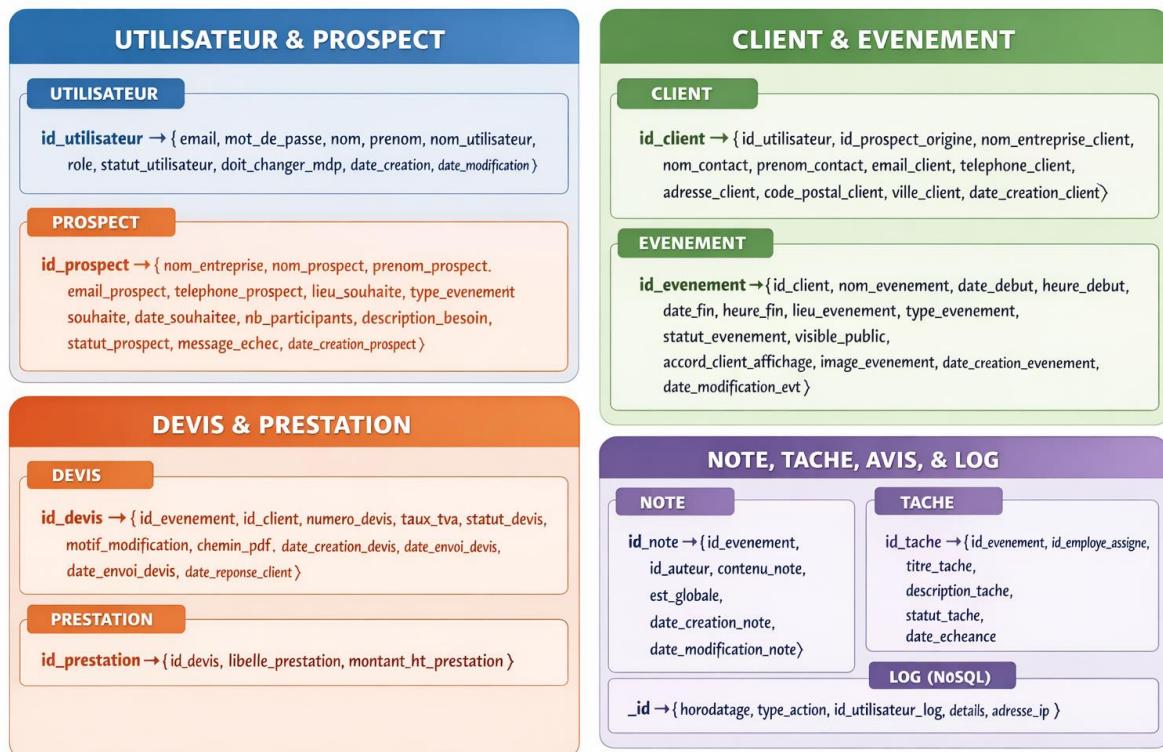


Figure 17 : Liste des Dépendances Fonctionnelles

C – Modèle Conceptuel de Données (MCD)

Le MCD, ou Modèle Conceptuel de Données, est une représentation abstraite des concepts et des relations entre les entités d'un système d'information. Il s'agit d'un schéma conceptuel qui décrit les principales entités, leurs attributs et les associations entre elles. Le MCD est indépendant de tout aspect technique ou de mise en œuvre.

Il utilise des symboles graphiques tels que des rectangles pour représenter les entités et des ovales pour représenter les relations entre ces entités. Les attributs sont spécifiés à l'intérieur des rectangles, et les cardinalités des relations sont indiquées avec des notations appropriées (0,1 – 1,1 – 0,n – 1,n).

Son objectif est de fournir une vue claire et abstraite des données du système, en mettant l'accent sur la structure logique et les relations entre les entités. Il permet de comprendre les besoins métier, d'identifier les entités clés, d'organiser les données de manière cohérente et de préparer la modélisation logique ultérieure.

Pour le projet Innov'Events Manager, le MCD comprend 10 entités et 11 relations :

Entité	Description

UTILISATEUR	Comptes de connexion (admin, employé, client)
PROSPECT	Demandes de devis (clients potentiels)
CLIENT	Clients convertis depuis les prospects
ÉVÉNEMENT	Événements organisés pour les clients
DEVIS	Devis associés aux événements
PRESTATION	Lignes de prestation d'un devis
NOTE	Notes collaboratives sur les événements
TACHE	Tâches assignées aux employés
AVIS	Avis clients sur les événements terminés
CONTACT	Messages du formulaire de contact

Les principales relations identifiées sont :

- Un CLIENT est un UTILISATEUR (1,1 – 0,1)
- Un CLIENT provient d'un PROSPECT (1,1 – 0,1)
- Un CLIENT possède des ÉVÉNEMENTS (1,n – 1,1)
- Un ÉVÉNEMENT génère des DEVIS (0,n – 1,1)
- Un DEVIS contient des PRESTATIONS (1,n – 1,1)

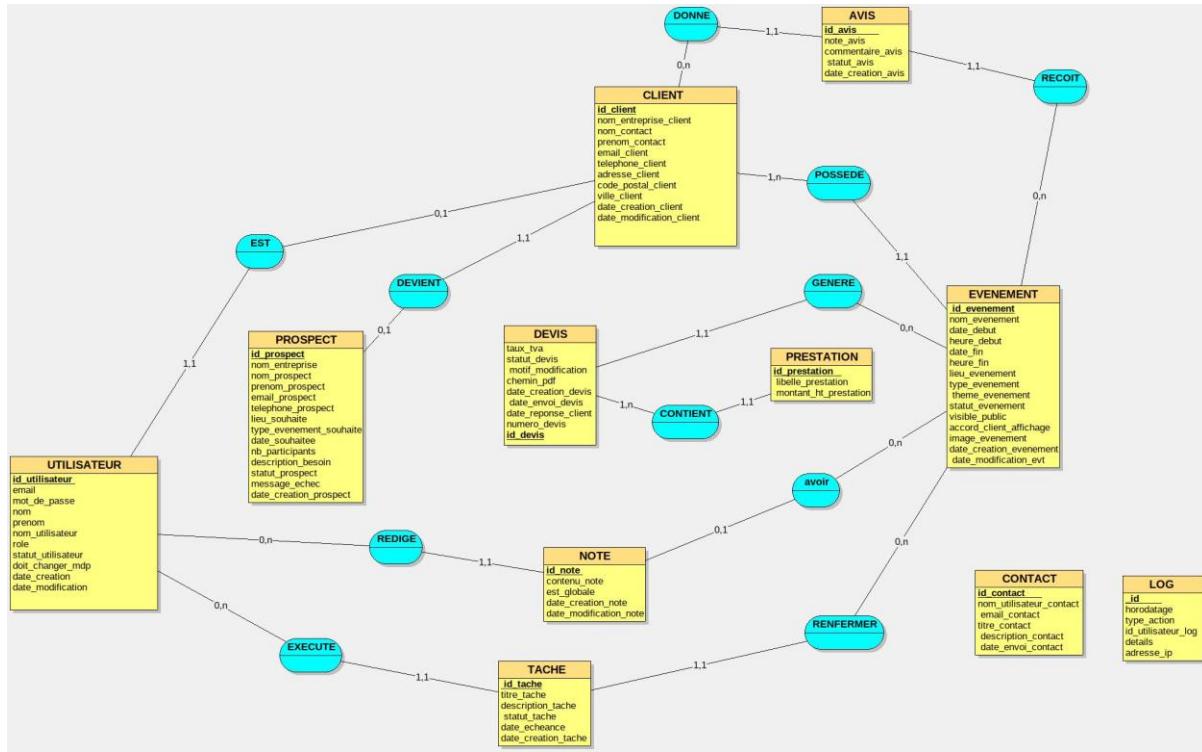


Figure 18 : Capture d'écran du MLD (export Looping)

D – Modèle Logique de Données (MLD)

Le MLD, ou Modèle Logique de Données, est quant à lui, une représentation structurée et formelle des données d'un système d'information. Il s'agit d'une étape intermédiaire entre le MCD (Modèle Conceptuel de Données) et le MPD (Modèle Physique de Données).

Il se concentre sur la traduction du MCD en un modèle plus détaillé et spécifique, en utilisant des concepts et des notations propres à un système de gestion de base de données (SGBD) particulier. Il inclut des éléments tels que les tables, les clés primaires, les clés étrangères, les contraintes d'intégrité et les relations entre les tables.

Ce schéma décrit en effet la structure logique des données.

Les règles de passage du MCD au MLD sont les suivantes :

- L'entité qui possède la cardinalité maximale égale à 1 reçoit l'identifiant de l'entité ayant la cardinalité maximale la plus forte (clé étrangère).
- Les relations ayant toutes leurs entités reliées avec des cardinalités maximales supérieures à 1 se transforment en table en absorbant les identifiants des entités jointes.
- Toute relation porteuse de propriétés se transforme en table.

Pour le projet Innov'Events Manager, le MLD comprend 10 tables avec les clés étrangères suivantes :

Table	Clés étrangères
CLIENT	id_utilisateur, id_prospect
EVENEMENT	id_client
DEVIS	id_evenement
PRESTATION	id_devis
NOTE	id_utilisateur, id_evenement
TACHE	id_utilisateur, id_evenement
AVIS	id_evenement, id_client

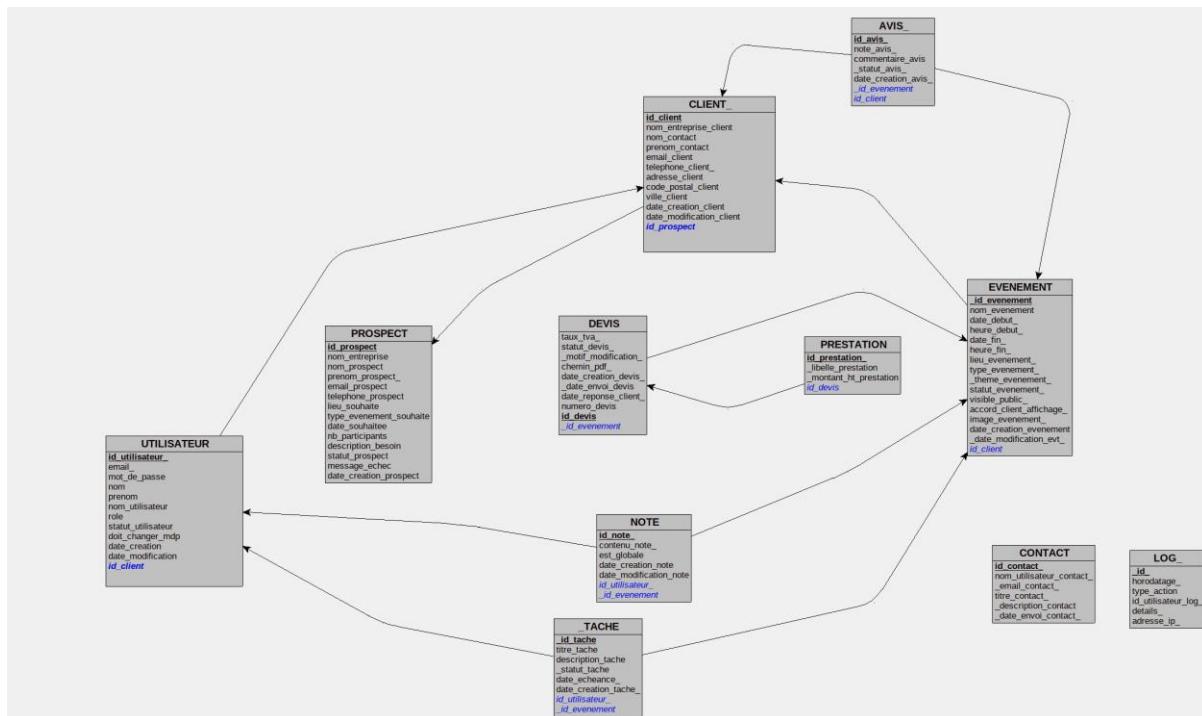


Figure 19 : Capture d'écran du MLD (export Looping)

E – Modèle Physique de Données (MPD)

Le MPD, ou Modèle Physique de Données, est la dernière étape du processus Merise. Il consiste à transformer le Modèle Logique de Données en une suite de requêtes SQL permettant de créer concrètement les tables dans un système de gestion de base de données.

Cette étape finalise le processus de traitement des données. L'implémentation de la base de données peut être réalisée de façon optimale. Le script SQL ainsi généré peut être donné à un développeur pour qu'il puisse créer la base de données correspondante sur un serveur de base de données.

Pour le projet Innov'Events Manager, j'ai choisi d'utiliser PostgreSQL comme système de gestion de base de données relationnelles. Ce choix s'explique par sa robustesse, sa conformité aux standards SQL et sa large adoption dans le monde professionnel.

Le script de création des tables respecte les règles suivantes :

- Utilisation du type SERIAL pour les clés primaires auto-incrémentées
- Définition des contraintes NOT NULL pour les champs obligatoires
- Définition des contraintes UNIQUE pour les champs devant être uniques (email, nom_utilisateur)
- Définition des clés étrangères avec FOREIGN KEY et REFERENCES
- Utilisation de ON DELETE CASCADE pour la suppression en cascade
- Définition de valeurs par défaut avec DEFAULT
- Ajout de contraintes CHECK pour valider les données (ex: note entre 1 et 5)

```
-- =====
-- INNOV'EVENTS - Script SQL PostgreSQL
-- Base de données : innovevents_dev
-- Projet CDA - 2024
-- =====

-- =====
-- SUPPRESSION DES TABLES (ordre inverse des FK)
-- =====

DROP TABLE IF EXISTS avis CASCADE;
DROP TABLE IF EXISTS tache CASCADE;
DROP TABLE IF EXISTS note CASCADE;
DROP TABLE IF EXISTS prestation CASCADE;
DROP TABLE IF EXISTS devis CASCADE;
DROP TABLE IF EXISTS evenement CASCADE;
DROP TABLE IF EXISTS client CASCADE;
DROP TABLE IF EXISTS prospect CASCADE;
DROP TABLE IF EXISTS utilisateur CASCADE;
DROP TABLE IF EXISTS contact CASCADE;

-- =====
-- TABLE UTILISATEUR
-- =====

CREATE TABLE utilisateur (
    id_utilisateur SERIAL PRIMARY KEY,
    email VARCHAR(100) NOT NULL UNIQUE,
    mot_de_passe VARCHAR(255) NOT NULL,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL,
    nom_utilisateur VARCHAR(50) NOT NULL UNIQUE,
    role VARCHAR(20) NOT NULL,
    statut_utilisateur VARCHAR(20) DEFAULT 'actif',
    doit_changer_mdp BOOLEAN DEFAULT FALSE,
    date_creation TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    date_modification TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Figure 20 : Script SQL de création des tables

F – Jeu de données de test

Un jeu de données de test est essentiel pour valider le bon fonctionnement de la base de données et de l'application. Il permet de tester les fonctionnalités, de vérifier les contraintes d'intégrité et de disposer de données réalistes pour les démonstrations.

Pour le projet Innov'Events Manager, j'ai créé un jeu de données complet couvrant tous les cas d'utilisation :

Table	Nombre de lignes	Description
UTILISATEUR	6	1 admin, 2 employés, 3 clients

PROSPECT	5	3 convertis, 1 à contacter, 1 échoué
CLIENT	3	Clients avec comptes utilisateurs
EVENEMENT	4	Différents statuts (brouillon, en cours, terminé)
DEVIS	4	Différents statuts (brouillon, accepté, en étude)
PRESTATION	10	Locations, traiteurs, animations
NOTE	5	Notes sur événements et note globale
TACHE	8	Différents statuts et assignations
AVIS	2	Avis validé et en attente
CONTACT	3	Messages du formulaire

Ce jeu de données respecte l'intégrité référentielle : les clés étrangères font référence à des enregistrements existants, et l'ordre d'insertion respecte les dépendances entre les tables.

La base de données est entièrement restaurable : un seul script SQL permet de supprimer les tables existantes, de les recréer et d'insérer toutes les données de test.

```
-- UTILISATEURS (6 utilisateurs)
-----
INSERT INTO utilisateur (email, mot_de_passe, nom, prenom, nom_utilisateur, role) VALUES
('chloe@innovevents.com', 'motdepasse123', 'Durand', 'Chloé', 'chloe_admin', 'admin'),
('jose@innovevents.com', 'motdepasse123', 'Martin', 'José', 'jose_employe', 'employe'),
('marie@innovevents.com', 'motdepasse123', 'Dupont', 'Marie', 'marie_employe', 'employe'),
('pierre.bernard@entreprise.com', 'motdepasse123', 'Bernard', 'Pierre', 'pierre_client', 'client'),
('sophie.leroy@startup.fr', 'motdepasse123', 'Leroy', 'Sophie', 'sophie_client', 'client'),
('marc.petit@corporate.com', 'motdepasse123', 'Petit', 'Marc', 'marc_client', 'client');

-- PROSPECTS (5 prospects)
-----
INSERT INTO prospect (nom_entreprise, nom_prospect, prenom_prospect, email_prospect, telephone_prospect, lieu_souhaite, type_evenement_souhaite, date_souhaitee, nb_participants, description_besoin, statut_prospect, message_echec) VALUES
('Tech Solutions', 'Moreau', 'Julie', 'julie.moreau@techsolutions.fr', '0612345678', 'Paris', 'Seminaire', '2025-03-15', 50, 'Séminaire annuel de notre équipe commerciale avec team building.', 'converti', NULL),
('Startup Nation', 'Lefebvre', 'Thomas', 'thomas@startuppation.io', '0698765432', 'Lyon', 'Conference', '2025-04-20', 150, 'Conférence sur l''innovation et le digital pour nos partenaires.', 'converti', NULL),
('Corporate Group', 'Dubois', 'Nathalie', 'n.dubois@corporate.com', '0654321987', 'Marseille', 'Soiree', '2025-05-10', 200, 'Soirée de gala pour les 20 ans de l''entreprise.', 'converti', NULL),
('Petit Commerce SARL', 'Girard', 'Francois', 'fgirard@petitcommerce.fr', '0678901234', 'Bordeaux', 'Seminaire', '2025-06-01', 20, 'Petit séminaire pour notre équipe de 20 personnes.', 'a_contacter', NULL),
('Mega Corp', 'Roux', 'Isabelle', 'isabelle.roux@megacorp.com', '0645678901', 'Nice', 'Conference', '2025-02-28', 300, 'Grande conférence internationale.', 'echoue', 'Budget insuffisant pour les prestations demandées.');
```

G – Journalisation (Logs) – MongoDB

Pour la journalisation des actions utilisateurs, j'ai choisi d'utiliser MongoDB, une base de données NoSQL orientée documents. Ce choix se justifie par la nature flexible des logs : chaque type d'action peut nécessiter des informations différentes, et le format JSON de MongoDB permet cette flexibilité.

La collection "logs" stocke les informations suivantes :

- Identifiant unique (_id)
- Horodatage de l'action
- Type d'action (CONNEXION_OK, CREATION_CLIENT, MODIFICATION_DEVIS, etc.)
- Identifiant de l'utilisateur concerné
- Détails supplémentaires (format JSON flexible)
- Adresse IP (conformité RGPD)

Conclusion

La conception de la base de données du projet Innov'Events Manager a été réalisée en suivant rigoureusement la méthode Merise. Cette approche méthodique a permis de :

- Identifier et documenter toutes les données nécessaires au fonctionnement de l'application
- Modéliser les entités et leurs relations de manière cohérente
- Garantir l'intégrité et la sécurité des données grâce aux contraintes appropriées
- Préparer un script SQL opérationnel et un jeu de données de test complet

La base de données ainsi conçue répond aux besoins exprimés dans le cahier des charges et constitue une fondation solide pour le développement de l'application.

VII – Conception de l'application

Pour la conception d'Innov'Events, je me suis appuyé sur le langage UML (Unified Modeling Language) pour représenter le fonctionnement de l'application. En effet, le langage UML est un langage graphique standard utilisé pour modéliser et représenter visuellement les systèmes logiciels. Il permet de capturer les aspects essentiels d'un système, de sa structure à son comportement, en utilisant une notation graphique compréhensible par les développeurs, les concepteurs et les parties prenantes.

Dans le contexte d'Innov'Events, l'utilisation d'UML m'a permis de modéliser les différents flux métier liés à la gestion événementielle : le parcours d'un prospect depuis sa demande de devis jusqu'à sa conversion en client, la création et le suivi des événements, ainsi que la gestion des prestations et des intervenants. Ces diagrammes servent de supports de communication pour analyser, concevoir, documenter et visualiser le système, facilitant ainsi la compréhension et la collaboration entre les membres de l'équipe de développement et les parties prenantes.

Le langage UML est largement utilisé dans l'industrie du développement logiciel pour améliorer la modélisation, la conception et la documentation des systèmes complexes.

A – Diagramme de cas d'utilisation (Use Case)

Le diagramme de cas d'utilisation est une technique de modélisation qui décrit les interactions entre les utilisateurs (acteurs) et le système (l'application web). Il permet de comprendre les fonctionnalités requises et les objectifs de l'application. Les concepteurs peuvent hiérarchiser les fonctionnalités, identifier les besoins des utilisateurs, définir les scénarios d'utilisation et spécifier les exigences du système en utilisant le diagramme de cas d'utilisation. Cela donne une vision claire des fonctionnalités à intégrer et aligne la conception de l'application sur les besoins des utilisateurs.

Le diagramme de cas d'utilisation sert de base pour d'autres activités de conception, telles que la création de maquettes d'interface utilisateur, la spécification des interactions et la définition des exigences fonctionnelles pour le développement de l'application web. En somme, il est un élément clé de la conception d'applications web.

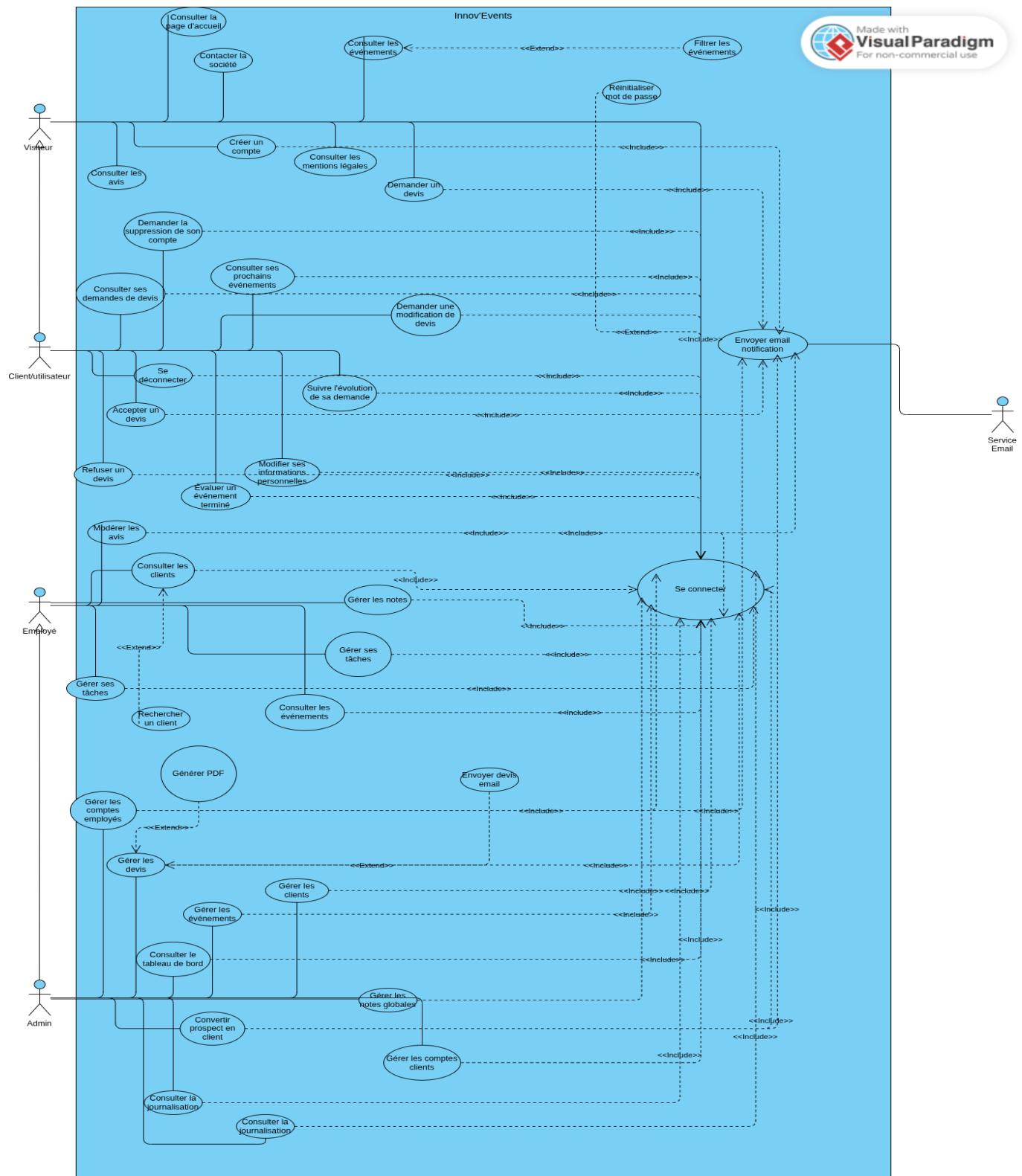


Figure 21 : Schéma du diagramme de cas d'utilisation

B- Diagramme de séquence

Le diagramme de séquence fait également partie de la phase de conception dans la création d'une application web. Il est utilisé pour modéliser les interactions entre les objets ou les composants du système pendant l'exécution d'un scénario spécifique. Dans la phase de conception, une fois que les besoins ont été analysés et que les cas d'utilisation ont été identifiés, le diagramme de séquence peut être créé pour détailler l'ordre chronologique des messages échangés entre les objets ou les composants du système lors de l'exécution d'un cas d'utilisation.

Le diagramme de séquence montre comment les objets interagissent entre eux au fil du temps pour accomplir une fonctionnalité spécifique. Il met en évidence les messages échangés, les appels de méthodes, les retours et les synchronisations entre les différents acteurs et objets.

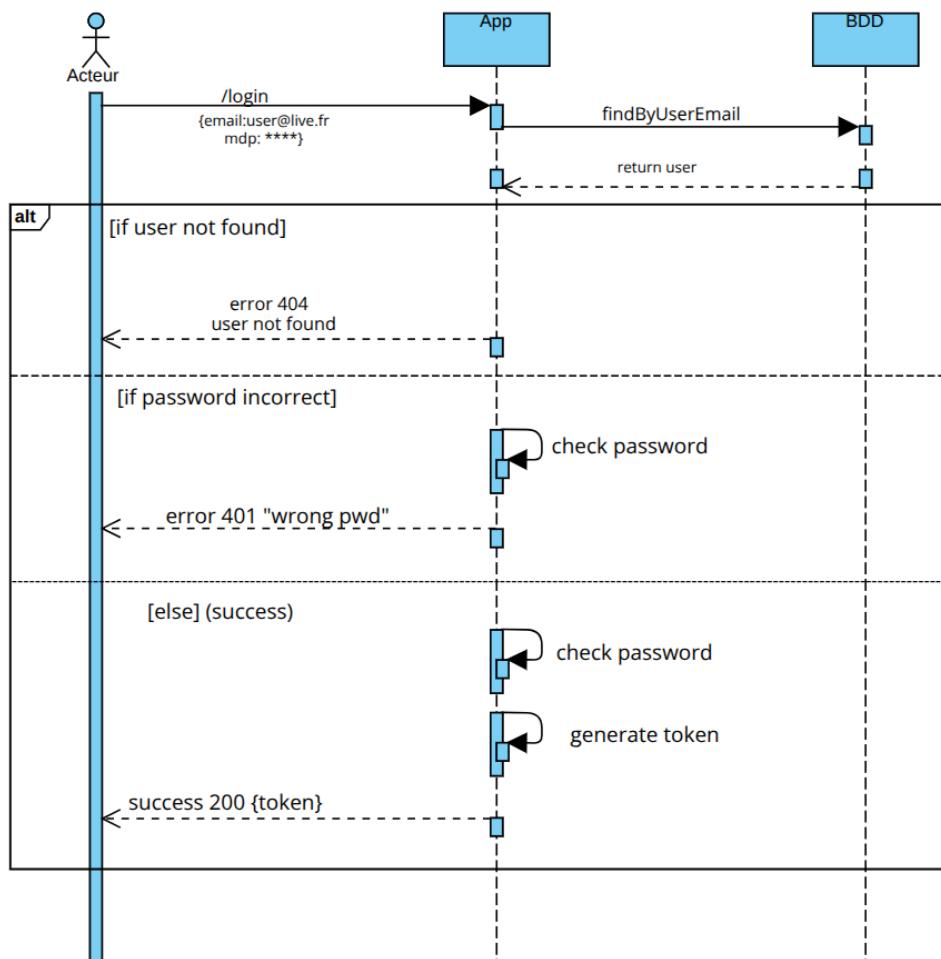


Figure 22 : Schéma du diagramme de séquence de use case : se connecter

VIII- Conception Multicouche : MVC

Le schéma MVC (Modèle-Vue-Contrôleur) est un concept fondamental en développement logiciel, largement utilisé dans la conception d'applications web et d'autres systèmes interactifs. Il offre une structure claire et organisée pour séparer les différentes responsabilités d'une application et favoriser une meilleure maintenabilité, extensibilité et réutilisabilité du code. Je vais ci-dessous vous présenter les composants de ce schéma dans le cadre de l'architecture technique adoptée pour Innov'Events, combinant React pour le frontend et Node.js/Express pour le backend, avec une approche hybride de base de données.

A – Le Modèle

Le Modèle représente la couche responsable de la gestion des données et de la logique métier. Il est chargé d'interagir avec les bases de données, de récupérer et de stocker les informations pertinentes pour l'application. Dans le cadre d'Innov'Events, j'ai adopté une architecture de persistance hybride utilisant deux systèmes de gestion de bases de données complémentaires : PostgreSQL et MongoDB.

PostgreSQL pour les données métier

PostgreSQL est un système de gestion de base de données relationnelle (SGBDR) open source, reconnu pour sa robustesse, sa conformité aux standards SQL et ses fonctionnalités avancées. J'ai choisi PostgreSQL pour stocker les données métier structurées d'Innov'Events : les utilisateurs, les prospects, les clients, les événements, les devis, les prestations et les intervenants.

Pour interagir avec PostgreSQL, j'ai opté pour une approche directe en utilisant le module pg (node-postgres), le driver natif PostgreSQL pour Node.js. Cette approche me permet d'écrire mes requêtes SQL de manière explicite, offrant un contrôle total sur les interactions avec la base de données et une compréhension claire des opérations effectuées.

Les requêtes SQL sont écrites manuellement dans les fichiers du Modèle, permettant d'optimiser finement chaque opération selon les besoins spécifiques. Par exemple, pour récupérer un événement avec les informations de son client associé, j'utilise une jointure SQL explicite (JOIN) plutôt que de déléguer cette logique à une couche d'abstraction. Cette méthode garantit une maîtrise complète des requêtes exécutées et facilite le débogage en cas de problème de performance.

La structure relationnelle de PostgreSQL permet de définir des relations claires entre les entités : un Client possède plusieurs Événements via la clé étrangère client_id, un Événement génère plusieurs Devis, un Devis contient plusieurs lignes de Prestations. Les contraintes d'intégrité référentielle (FOREIGN KEY), les contraintes de validation (CHECK, NOT NULL) et les index sont définis directement dans le schéma SQL, garantissant la cohérence des données au niveau de la base elle-même.

Cette approche sans ORM présente plusieurs avantages : une performance optimale car les requêtes sont exactement celles souhaitées, une transparence totale sur les opérations SQL exécutées, et un apprentissage approfondi du langage SQL qui constitue une compétence fondamentale pour tout développeur.

MongoDB pour les logs de connexion

En complément de PostgreSQL, j'ai intégré MongoDB pour gérer les logs de connexion des utilisateurs. MongoDB est une base de données NoSQL orientée documents, particulièrement adaptée au stockage de données semi-structurées et volumineuses comme les journaux d'activité.

L'utilisation de Mongoose comme ODM (Object Document Mapping) me permet de définir un schéma pour les logs de connexion (ConnectionLog) incluant l'identifiant de l'utilisateur, l'action effectuée (connexion, déconnexion, tentative échouée), l'adresse IP, le user-agent du navigateur et le timestamp. Cette approche offre une grande flexibilité pour enrichir les logs avec des métadonnées supplémentaires sans modifier le schéma de la base relationnelle.

Cette architecture hybride tire parti des forces de chaque système : la rigueur et l'intégrité référentielle de PostgreSQL pour les données métier critiques, et la flexibilité et les performances en écriture de MongoDB pour la traçabilité et l'audit.

```

src > models > JS Avis.js > Avis > findAll
1  /**
2   * Model Avis
3   * @module models/Avis
4   */
5
6 const { query } = require('../config/database');
7
8 class Avis {
9   /**
10    * Trouve tous les avis
11   */
12   static async findAll(filters = {}) {
13     let sql = `
14       SELECT a.*,
15             e.nom_evenement, e.date_debut, e.type_evenement,
16             c.nom_entreprise_client, c.nom_contact, c.prenom_co
17             FROM avis a
18             INNER JOIN evenement e ON a.id_evenement = e.id_evenement
19             INNER JOIN client c ON a.id_client = c.id_client
20             WHERE 1=1
21       `;
22     const params = [];
23     let paramIndex = 1;
24
25     if (filters.statut_avis) {
26       sql += ` AND a.statut_avis = ${paramIndex++}`;
27       params.push(filters.statut_avis);
28     }
}

```

Figure 23 : La couche modèle pattern MVC

B – La Vue

La Vue est responsable de la présentation des informations aux utilisateurs et de l'interaction avec eux. Pour Innov'Events, j'ai opté pour React, une bibliothèque JavaScript moderne permettant de construire des interfaces utilisateur dynamiques et réactives.

Contrairement aux moteurs de templates traditionnels côté serveur, React adopte une approche composant : chaque élément de l'interface est un composant réutilisable possédant son propre état et sa propre logique d'affichage. J'ai ainsi pu concevoir des composants modulaires pour les différentes sections de l'application : le tableau de bord avec ses statistiques, les listes paginées de prospects et clients, les formulaires de création d'événements avec validation en temps réel, et les cartes de visualisation des devis.

L'utilisation de Tailwind CSS en complément de React m'a permis de créer une interface utilisateur esthétique et cohérente, respectant la charte graphique d'Innov'Events définie avec les couleurs bleu royal (#1E3A8A), or (#D4A845), bleu ciel (#DBEAFE) et blanc cassé (#F8FAFC). Les classes utilitaires de Tailwind facilitent la création d'un design responsive s'adaptant aux écrans desktop et mobile.

Les hooks React comme useState et useEffect facilitent la gestion de l'état local des composants et les interactions asynchrones avec l'API backend. Le hook personnalisé useAuth centralise la logique d'authentification et permet de protéger les routes sensibles de l'application.

La Vue dans Innov'Events se structure en deux espaces distincts :

- **La partie publique** : page d'accueil présentant les services, formulaire de demande de devis, page contact et consultation des avis clients. Ces pages sont accessibles à tous les visiteurs sans authentification.
- **La partie privée** : dashboard, gestion des prospects, clients, événements, devis, prestations et intervenants. Cet espace est réservé aux utilisateurs authentifiés (administrateur et commercial).

Cette séparation est gérée par un système de routes protégées vérifiant la présence et la validité d'un token JWT stocké dans le localStorage du navigateur.

```

e.js Dashboard.jsx AdminLayout.jsx AdminSidebar.jsx Home.jsx
frontend > src > pages > public > Home.jsx > Home > services.map() callback
1 import { useState, useEffect } from 'react';
2 import { Link } from 'react-router-dom';
3 import { Mic, Users, PartyPopper, ArrowRight, Star, Calendar, CheckCircle } from 'react-feather';
4 import api from '../../../../../services/api';

5
6 const Home = () => {
7   const [prochainEvenement, setProchainEvenement] = useState(null);
8   const [averageNote, setAverageNote] = useState(null);

9
10  useEffect(() => {
11    fetchProchainEvenement();
12    fetchAverageNote();
13  }, []);

14
15  const fetchProchainEvenement = async () => {
16    try {
17      const response = await api.get('/evenements/public');
18      const evenements = response.data.data || [];
19      const today = new Date();
20      today.setHours(0, 0, 0, 0);
21      const futurs = evenements
22        .filter(e => new Date(e.date_debut) >= today)
23        .sort((a, b) => new Date(a.date_debut) - new Date(b.date_debut));
24      if (futurs.length > 0) {
25        setProchainEvenement(futurs[0]);
26      }
27    } catch (err) {
28      console.error('Erreur événements: ', err);
29    }
30  }
31
32  const fetchAverageNote = async () => {
33    const averageNote = await api.get('/notes/average');
34    setAverageNote(averageNote);
35  }
36
37  return (
38    <div>
39      <AdminLayout>
40        <Home>
41          <div>
42            <h1>Innov'Events</h1>
43            <p>Le meilleur moyen d'organiser vos événements !</p>
44            <div>
45              <Link to="/admin">Administration</Link>
46              <Link to="/auth">Authentification</Link>
47              <Link to="/client">Clients</Link>
48              <Link to="/employe">Employés</Link>
49              <Link to="/public">Public</Link>
50            </div>
51            <div>
52              <Link to="/public/avis">Avis</Link>
53              <Link to="/public/contact">Contact</Link>
54              <Link to="/public/demandeDevis">Demande Devis</Link>
55              <Link to="/public/evenements">Événements</Link>
56              <Link to="/public/home">Accueil</Link>
57              <Link to="/public/mentionsLegales">Mentions Légales</Link>
58            </div>
59          </div>
60        </AdminLayout>
61      </div>
62    </div>
63  );
64}

```

tikaya@tikaya-KLVD-WXX9:~/innovevents-manager\$

Figure 24 : La couche vue du pattern MVC

C – Le Contrôleur

Le Contrôleur joue un rôle crucial dans la gestion des actions de l'utilisateur et l'orchestration des interactions entre le Modèle et la Vue. Dans Innov'Events, cette couche est implémentée avec Express.js, un framework minimaliste et flexible pour Node.js permettant de créer des API RESTful robustes.

Les contrôleurs d'Innov'Events sont organisés par entité métier, suivant le principe de séparation des responsabilités :

- **authController** : gestion de l'authentification (login, register, changement de mot de passe, réinitialisation)
- **prospectController** : CRUD des prospects et gestion des statuts (Nouveau, Contacté, En négociation, Converti, Perdu)
- **clientController** : gestion des clients et de leurs informations
- **evenementController** : création, modification et suivi des événements
- **devisController** : génération et gestion des devis
- **prestationController** : gestion du catalogue de prestations
- **intervenantController** : gestion des prestataires externes
- **contactController** : traitement des messages reçus via le formulaire public

Chaque contrôleur expose des méthodes correspondant aux opérations CRUD (Create, Read, Update, Delete) et aux actions métier spécifiques comme la conversion d'un prospect en client ou le changement de statut d'un événement.

Le contrôleur reçoit les requêtes HTTP entrantes, extrait les données du corps de la requête (req.body) ou des paramètres d'URL (req.params), effectue les validations nécessaires, puis interagit avec le Modèle pour exécuter les requêtes SQL appropriées sur PostgreSQL ou les opérations Mongoose sur MongoDB. Une fois le traitement effectué, il formate la réponse JSON appropriée et la renvoie au client avec le code HTTP correspondant (200, 201, 400, 401, 404, 500).

Express fournit un système de routage déclaratif qui associe les endpoints API aux méthodes des contrôleurs. Par exemple :

- POST /api/prospects → création d'un nouveau prospect
- GET /api/clients/:id → récupération d'un client par son identifiant
- PUT /api/evenements/:id/statut → modification du statut d'un événement
- DELETE /api/prestations/:id → suppression d'une prestation

Les middlewares d'authentification (vérification du token JWT) et d'autorisation (vérification du rôle admin ou commercial) s'intercalent avant l'exécution des contrôleurs pour sécuriser les endpoints sensibles.

```

EXPLORATEUR
...  AdminSidebar.jsx  Home.jsx  JS AuthController.js X  DemandeDevis.jsx  D  □
ÉDITEURS OUVERTS
INNOEVENTS-MANAGER
src > controllers > JS AuthController.js > login > asyncHandler() callback > result
src > controllers > JS AuthController.js
1 /**
2  * Controller Auth
3  * @module controllers/AuthController
4  */
5
6 const AuthService = require('../services/AuthService');
7 const EmailService = require('../services/EmailService');
8 const { LogService, ACTION_TYPES } = require('../services/LogService')
9 const { asyncHandler } = require('../middlewares/errorHandler');
10
11 const login = asyncHandler(async (req, res) => {
12     const { email, mot_de_passe } = req.body;
13     const clientIp = req.clientIp;
14
15     try {
16         const result = await AuthService.login(email, mot_de_passe);
17
18         // Log connexion réussie
19         await LogService.log(
20             ACTION_TYPES.CONNEXION_REUSSIE,
21             result.user.id_utilisateur,
22             {
23                 email: result.user.email,
24                 role: result.user.role
25             },
26             clientIp
27         );
28     }
29 });
30
31 module.exports = {
32     login
33 };
34
35 
```

Détails de l'écran : L'écran montre une interface de développement (IDE ou terminal) avec l'explorateur de fichiers de l'application 'INNOEVENTS-MANAGER'. Le dossier 'controllers' contient plusieurs fichiers de contrôleur (AuthController.js, AvisController.js, ClientController.js, ContactController.js, DashboardController.js, DevisController.js, EmployeController.js, EvenementController.js, index.js, NoteController.js, ProspectController.js, TacheController.js). La fenêtre de code affiche le contenu du fichier 'AuthController.js', spécifiquement la fonction 'login'. La fonction utilise 'asyncHandler' pour gérer les erreurs et 'AuthService' pour effectuer la connexion. Des logs sont générés dans 'LogService' avec les informations de l'utilisateur connecté. En bas de l'écran, une console de terminal montre une session bash sur un serveur nommé 'tikaya@tikaya-KLVD-WXX9'. La ligne de commande indique l'emplacement du répertoire de travail : 'tikaya@tikaya-KLVD-WXX9:~/innoevents-manager\$'.

Figure 25 : La couche contrôleur du pattern MVC

D – Communication entre nos 3 composants

Comme nous venons de le voir, chacun de ces composants a sa fonction propre. Voyons maintenant comment ils opèrent en synergie pour fournir à l'utilisateur une réponse à sa requête dans le contexte d'Innov'Events :

1. **Initiation de la requête** : L'utilisateur interagit avec l'interface React, par exemple en cliquant sur "Créer un événement" dans le dashboard commercial. React déclenche alors une requête HTTP POST vers l'API backend via la bibliothèque Axios, en incluant le token JWT dans l'en-tête Authorization.
2. **Routage et authentification** : Express reçoit la requête sur l'endpoint /api/événements. Le middleware d'authentification extrait et vérifie la validité du token JWT. Si le token est valide et non expiré, la requête est transmise au contrôleur approprié ; sinon, une erreur 401 (Unauthorized) est renvoyée.
3. **Traitements par le Contrôleur** : L'evenementController reçoit la requête et extrait les données de l'événement à créer depuis req.body. Il effectue les validations métier : cohérence des dates (date de fin postérieure à la date de début), existence du client référencé, format des données.
4. **Interaction avec le Modèle** : Le contrôleur exécute une requête SQL INSERT via le module pg pour créer un nouvel enregistrement dans la table "événements" de PostgreSQL. La requête utilise des paramètres préparés (\$1, \$2, \$3...) pour prévenir les injections SQL.
5. **Journalisation** : En parallèle, le système enregistre cette action dans MongoDB via Mongoose pour conserver une trace de l'activité utilisateur à des fins d'audit et de sécurité.
6. **Réponse du Modèle** : PostgreSQL confirme l'insertion et renvoie l'enregistrement créé avec son identifiant unique (id auto-incrémenté) grâce à la clause RETURNING de la requête SQL.
7. **Formatage de la réponse** : Le contrôleur construit une réponse JSON contenant l'événement créé et un message de succès, puis l'envoie au client avec le code HTTP 201 (Created).
8. **Mise à jour de la Vue** : React reçoit la réponse via la promesse Axios, met à jour son état local avec le nouvel événement grâce au hook useState, et déclenche automatiquement un re-rendu du composant pour afficher les données actualisées à l'utilisateur. Une notification de succès peut également être affichée.

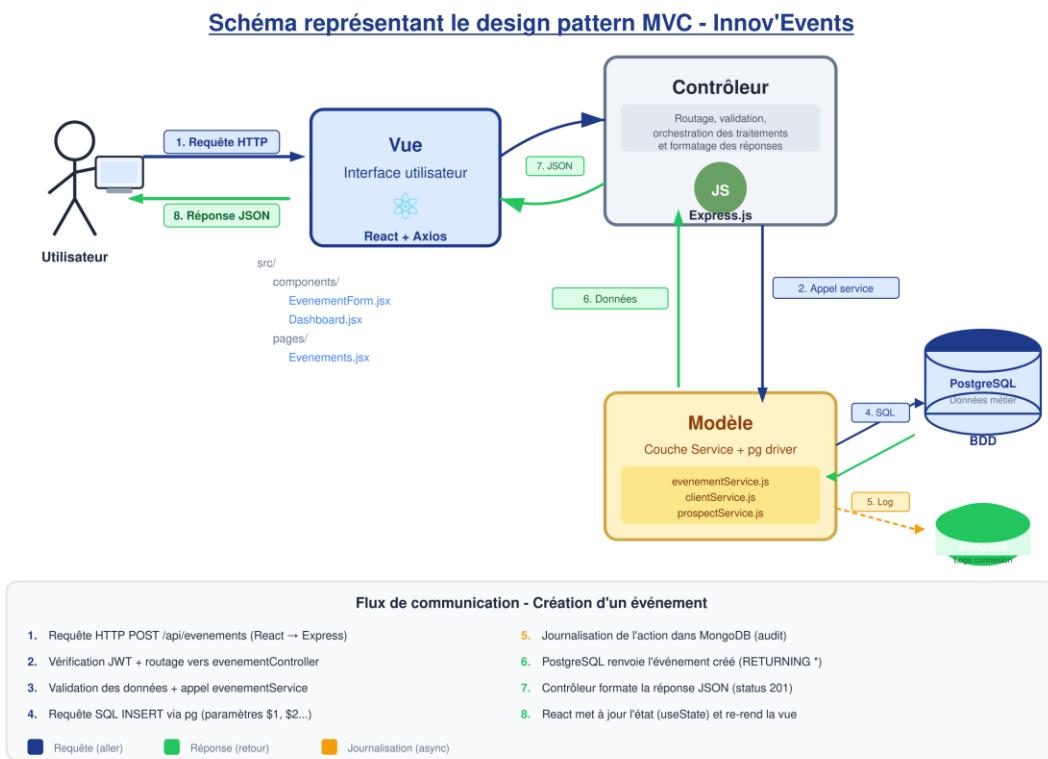


Figure 26 : Design pattern MVC innov'Events

E – L'architecture 3 tiers

Maintenant que nous avons examiné en détail le pattern MVC et compris comment il permet de structurer Innov'Events en séparant les responsabilités de manière efficace, il est temps d'élargir notre vision et d'explorer l'architecture globale du système : l'architecture 3 tiers.

L'architecture 3 tiers est un modèle d'organisation logique qui divise une application en trois couches distinctes : la présentation, la logique métier et les données. Chaque couche est responsable de tâches spécifiques et communique avec les autres selon des protocoles bien définis. Cette séparation permet une meilleure maintenabilité, une évolutivité accrue et une indépendance technologique entre les couches.

Couche Présentation (Frontend)

La première couche, **la présentation** ou couche d'interface utilisateur, est chargée de communiquer avec les utilisateurs finaux. Dans Innov'Events, cette couche est entièrement implémentée en **React** et s'exécute dans le navigateur du client. Elle comprend :

- **Les composants d'interface** : Header avec navigation, Sidebar pour le menu latéral, Cards pour l'affichage des données, Formulaires avec validation, Tableaux paginés et triables
- **La gestion de l'état local** avec les hooks React (useState, useEffect, useContext)
- **Le routage client** avec React Router pour la navigation sans recharge de page

- Les **appels API** via Axios pour communiquer avec le backend
- Le **style visuel** avec Tailwind CSS et la charte graphique Innov'Events

Cette couche ne contient aucune logique métier complexe ni accès direct aux données ; elle se contente d'afficher les informations reçues du backend et de transmettre les actions utilisateur à la couche suivante via des requêtes HTTP.

Couche Logique Métier (Backend)

La deuxième couche, la logique métier ou couche de traitement, constitue le cœur intelligent de l'application. Implémentée avec Node.js et Express, elle héberge :

- Les **contrôleurs** gérant les requêtes API REST et orchestrant les traitements
- Les **règles métier** : calcul automatique des montants TTC à partir du HT et du taux de TVA, gestion des transitions de statuts (un prospect "Perdu" ne peut plus être "Converti"), validation des données métier
- L'**authentification et l'autorisation** : génération et vérification des tokens JWT, hachage des mots de passe avec bcrypt, vérification des rôles (admin vs commercial)
- La **logique de conversion** : transformation d'un prospect en client avec création automatique des entités liées
- Les **middlewares** : validation des entrées, gestion centralisée des erreurs, logging des requêtes

Cette couche agit comme intermédiaire entre l'interface utilisateur et les données persistées, appliquant les règles de gestion propres à Innov'Events avant toute modification en base.

Couche Accès aux Données (Bases de données)

La troisième couche, la **couche d'accès aux données**, est responsable de la persistance et de la gestion des données. Dans Innov'Events, elle repose sur une architecture hybride combinant deux technologies complémentaires :

PostgreSQL pour les données métier structurées :

- Stockage des utilisateurs, prospects, clients, événements, devis, prestations et intervenants
- Requêtes SQL natives via le driver **pg** (node-postgres)
- Intégrité référentielle garantie par les clés étrangères définies dans le schéma
- Transactions ACID pour les opérations critiques
- Index optimisés pour les requêtes fréquentes

MongoDB pour les données de traçabilité :

- Stockage des logs de connexion (ConnectionLog)
- Flexibilité du schéma pour enrichir les métadonnées
- Performances optimales en écriture pour le logging intensif

- Accès via l'ODM **Mongoose**

Avantages pour Innov'Events

En adoptant cette architecture 3 tiers avec une approche hybride de persistance, Innov'Events bénéficie de plusieurs avantages majeurs :

- **Séparation des responsabilités** : Chaque couche a un rôle clairement défini, facilitant la compréhension du code par les développeurs et la maintenance à long terme.
- **Scalabilité** : Le frontend React peut être déployé sur un CDN pour une distribution mondiale, le backend Express peut être répliqué horizontalement derrière un load balancer, PostgreSQL peut être configuré en réplication, et MongoDB supporte nativement le sharding pour les gros volumes de logs.
- **Indépendance technologique** : Chaque couche peut évoluer indépendamment. Il serait possible de remplacer React par Vue.js, Express par Fastify, ou d'ajouter une couche de cache Redis sans impacter fondamentalement les autres couches.
- **Testabilité** : Chaque couche peut être testée isolément grâce à cette séparation : tests unitaires pour les fonctions du Modèle, tests d'intégration pour les contrôleurs Express, tests end-to-end avec Playwright pour l'interface React.
- **Sécurité renforcée** : La séparation en couches permet d'appliquer des mesures de sécurité à chaque niveau : validation côté client ET serveur, authentification JWT avec expiration, requêtes SQL paramétrées pour prévenir les injections, chiffrement des mots de passe avec bcrypt.
- **Maîtrise des requêtes SQL** : L'utilisation directe du driver pg sans ORM permet un contrôle total sur les requêtes exécutées, une optimisation fine des performances, et une compréhension approfondie des interactions avec la base de données.
- **Flexibilité de la persistance** : L'utilisation combinée de PostgreSQL et MongoDB permet de choisir le meilleur outil pour chaque besoin : la rigueur relationnelle pour les données métier critiques, la souplesse documentaire pour l'audit et le logging.

Cette architecture robuste et éprouvée garantit à Innov'Events une base solide pour répondre aux besoins actuels de gestion événementielle de l'entreprise, tout en offrant la flexibilité nécessaire pour intégrer de nouvelles fonctionnalités et s'adapter à l'évolution des besoins métier.

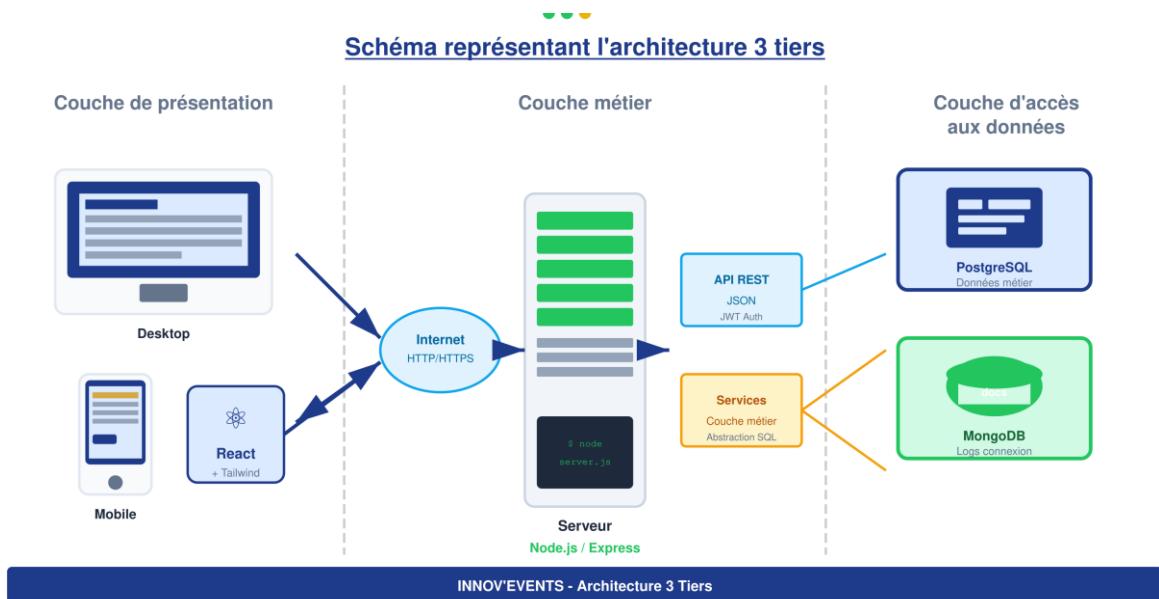


Figure 27 : Représentation de l'architecture 3 tiers

IX – Sécurité de l'application

Dans le paysage numérique actuel, la sécurité des applications web est devenue une préoccupation majeure. Les applications web jouent un rôle essentiel dans la communication, les transactions en ligne et le stockage des données sensibles. Cependant, elles sont également vulnérables aux attaques malveillantes et aux violations de la confidentialité.

Les cybercriminels utilisent des techniques sophistiquées pour exploiter les failles de sécurité des applications web et accéder aux données sensibles des utilisateurs. Les conséquences peuvent être désastreuses pour les entreprises et les utilisateurs, allant de la perte de données à la violation de la vie privée. Dans le contexte d'Innov'Events, qui manipule des informations commerciales sensibles (données clients, devis, informations d'événements), la protection de ces données est primordiale.

La protection des utilisateurs, la préservation de l'intégrité des données et la disponibilité des services sont des aspects essentiels pour assurer la confiance et la fiabilité d'une application web. C'est pourquoi la sécurité est un enjeu crucial que j'ai pris en compte dès les premières phases de développement d'Innov'Events.

A – Faille XSS (Cross-Site Scripting)

Une attaque XSS est une technique courante pour injecter du code malveillant dans des sites web et exploiter les vulnérabilités des navigateurs des utilisateurs. L'attaque XSS peut se produire lorsqu'une application web n'effectue pas une validation ou un échappement approprié des données entrées par les utilisateurs.

Dans le contexte d'une application Node.js/React comme Innov'Events, une attaque XSS pourrait se produire si des données fournies par l'utilisateur (par exemple, le nom d'un événement ou la description d'un prospect) étaient affichées directement dans les pages web sans être correctement filtrées. Un utilisateur malveillant pourrait alors entrer du code JavaScript dans un champ de saisie, et si cette entrée est affichée sans protection, le code serait exécuté par le navigateur, entraînant potentiellement le vol de données sensibles ou la redirection vers des sites malveillants.

Protection mise en place dans Innov'Events :

Pour prévenir les attaques XSS, j'ai mis en place plusieurs mécanismes de protection :

- **Échappement automatique** par React : React, utilisé pour le frontend d'Innov'Events, échappe automatiquement toutes les valeurs insérées dans le JSX avant de les afficher. Ainsi, même si un utilisateur entre du code dans un champ, React convertira automatiquement les caractères spéciaux en entités HTML inoffensives, empêchant l'exécution du script.
- **Validation côté serveur** : Toutes les données reçues par l'API Express sont validées et assainies avant d'être traitées ou stockées en base de données. Les caractères potentiellement dangereux sont échappés ou rejettés.
- **En-têtes de sécurité HTTP** : L'utilisation du middleware Helmet dans Express permet de configurer des en-têtes HTTP sécurisés, notamment Content-Security-Policy qui restreint les sources de scripts autorisées.

B – Injection SQL

L'injection SQL est une technique d'attaque qui consiste à insérer du code SQL malveillant dans les requêtes envoyées à la base de données. Cette faille peut permettre à un attaquant de contourner l'authentification, d'accéder à des données confidentielles, de modifier ou supprimer des enregistrements, voire de prendre le contrôle total de la base de données.

Dans le contexte d'Innov'Events, où PostgreSQL stocke des informations sensibles sur les clients, les devis et les événements, une injection SQL pourrait avoir des conséquences catastrophiques : accès non autorisé aux données commerciales, modification frauduleuse des montants de devis, ou suppression de l'historique des événements.

Protection mise en place dans Innov'Events :

Pour prévenir les injections SQL, j'ai adopté les pratiques suivantes :

- **Requêtes paramétrées** : Toutes les requêtes SQL exécutées via le driver pg (node-postgres) utilisent des paramètres préparés (\$1, \$2, \$3...) plutôt que la concaténation de chaînes. Par exemple

```

javascript

// ✗ Vulnérable à l'injection SQL
const query = `SELECT * FROM clients WHERE id = ${userId}`;

// ✓ Sécurisé avec paramètres préparés
const query = 'SELECT * FROM clients WHERE id = $1';
const result = await pool.query(query, [userId]);

```

- **Validation des entrées** : Les données utilisateur sont validées (type, format, longueur) avant d'être utilisées dans les requêtes, rejetant toute entrée suspecte.
- **Principe du moindre privilège** : Le compte PostgreSQL utilisé par l'application dispose uniquement des droits nécessaires (SELECT, INSERT, UPDATE, DELETE sur les tables spécifiques), sans priviléges d'administration.

C – Authentification et gestion des sessions

L'authentification est un élément critique de la sécurité d'Innov'Events, permettant de s'assurer que seuls les utilisateurs autorisés (administrateur et commercial) peuvent accéder aux fonctionnalités de gestion.

Mécanismes mis en place :

- **Authentification par JWT (JSON Web Token)** : Plutôt que d'utiliser des sessions côté serveur, Innov'Events utilise des tokens JWT pour authentifier les utilisateurs. Lors de la connexion, le serveur génère un token signé contenant l'identifiant et le rôle de l'utilisateur. Ce token est ensuite envoyé dans l'en-tête Authorization de chaque requête pour prouver l'identité de l'utilisateur.
- **Expiration des tokens** : Les tokens JWT ont une durée de validité limitée (par exemple, 24 heures), obligeant l'utilisateur à se reconnecter régulièrement et limitant la fenêtre d'exploitation en cas de vol de token.
- **Hachage des mots de passe avec bcrypt** : Les mots de passe ne sont jamais stockés en clair dans la base de données. L'algorithme bcrypt est utilisé pour générer un hash irréversible avec un salt unique pour chaque utilisateur. Même en cas de compromission de la base de données, les mots de passe originaux ne peuvent pas être récupérés.
- **Changement de mot de passe obligatoire** : Lors de la création d'un compte par l'administrateur, un mot de passe temporaire est généré. L'utilisateur est obligé de le modifier lors de sa première connexion grâce au flag `doit_changer_mdp` stocké en base de données.

D – Gestion des autorisations (RBAC)

Au-delà de l'authentification (vérifier l'identité), Innov'Events implémente un système d'autorisation basé sur les rôles (RBAC - Role-Based Access Control) pour contrôler les actions que chaque utilisateur peut effectuer.

Rôles définis dans Innov'Events :

Rôle	Permissions
Admin	Gestion des utilisateurs, consultation des logs, accès complet au système
Commercial ou employé	Gestion des prospects, clients, événements, devis, prestations, intervenants

Implémentation :

Un middleware d'autorisation vérifie le rôle de l'utilisateur (extrait du token JWT) avant d'autoriser l'accès aux routes protégées. Par exemple, seul un administrateur peut accéder à l'endpoint /api/users pour gérer les comptes utilisateurs.

```
/** 
 * Middleware d'authentification JWT
 * @module middlewares/auth
 */

const jwt = require('jsonwebtoken');
const Utilisateur = require('../models/Utilisateur');

/**
 * Vérifie le token JWT
 */
const authenticate = async (req, res, next) => {
    try {
        // Récupère le token du header
        const authHeader = req.headers.authorization;

        if (!authHeader || !authHeader.startsWith('Bearer ')) {
            return res.status(401).json({
                success: false,
                message: 'Token d\'authentification requis'
            });
        }

        const token = authHeader.split(' ')[1];

        // Vérifie le token
        const decoded = jwt.verify(token, process.env.JWT_SECRET);

        // Vérifie que l'utilisateur existe toujours
        const user = await Utilisateur.findById(decoded.id);

        if (!user) {
            return res.status(401).json({
                success: false,
                message: 'Utilisateur non trouvé'
            });
        }
    } catch (error) {
        return res.status(500).json({
            success: false,
            message: 'Erreur lors de la vérification du token'
        });
    }
}

module.exports = authenticate;
```

Figure 28 : Capture du middleware d'autorisation

E – Protection CORS (Cross-Origin Resource Sharing)

Le CORS est un mécanisme de sécurité qui permet de contrôler quelles origines (domaines) peuvent accéder aux ressources de l'API. Sans configuration CORS appropriée, n'importe quel site web pourrait effectuer des requêtes vers l'API d'Innov'Events.

Configuration mise en place :

Le middleware CORS d'Express est configuré pour n'accepter que les requêtes provenant du domaine frontend autorisé :

```
// =====
// MIDDLEWARES DE SÉCURITÉ
// =====

// Helmet - Sécurité HTTP headers
app.use(helmet());

// CORS
app.use(cors({
  origin: process.env.FRONTEND_URL || 'http://localhost:5173',
  credentials: true
}));

// Rate limiting - Protection brute force
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100, // 100 requêtes par IP
  message: { success: false, message: 'Trop de requêtes, réessayez plus tard' }
});
app.use('/api/auth', limiter);
```

Figure 29 : Capture image implémentation middleware CORS

Cette configuration garantit que seule l'interface React d'Innov'Events peut communiquer avec l'API backend.

F – Journalisation et traçabilité

La traçabilité des actions est essentielle pour détecter les comportements suspects et analyser les incidents de sécurité. Innov'Events intègre un système de journalisation des connexions stocké dans MongoDB.

Chaque tentative de connexion (réussie ou échouée) génère un enregistrement dans la collection `ConnectionLog` contenant :

- L'identifiant de l'utilisateur (si connu)

- L'action effectuée (connexion, déconnexion, échec d'authentification)
- L'adresse IP de l'utilisateur
- Le user-agent du navigateur
- La date et l'heure de l'action

Ces logs permettent à l'administrateur de :

- Déetecter les tentatives de connexion frauduleuses (multiples échecs depuis une même IP)
- Identifier les accès suspects (connexion depuis un pays inhabituel)
- Auditer l'activité des utilisateurs en cas d'incident

G – Protection des données sensibles

Les informations sensibles (identifiants de base de données, clé secrète JWT, URLs de connexion) ne sont jamais codées en dur dans le code source. Elles sont stockées dans des variables d'environnement via un fichier ` `.env` qui n'est pas versionné (ajouté au ` .gitignore`).

H – Validation des données

La validation des données est effectuée à plusieurs niveaux pour garantir l'intégrité et la sécurité :

- **Côté client (React)** : Validation en temps réel des formulaires pour améliorer l'expérience utilisateur et éviter les requêtes inutiles.
- **Côté serveur (Express)** : Validation systématique de toutes les données reçues, indépendamment de la validation client qui peut être contournée. Vérification des types, formats, longueurs et valeurs autorisées.
- **Côté base de données (PostgreSQL)** : Contraintes de schéma (NOT NULL, CHECK, FOREIGN KEY) garantissant l'intégrité des données même en cas de bug applicatif.

Conclusion sur la sécurité

La sécurité d'Innov'Events repose sur une approche de défense en profondeur, combinant plusieurs couches de protection complémentaires. De l'échappement automatique des données par React à la journalisation des connexions dans MongoDB, en passant par l'authentification JWT et les requêtes SQL paramétrées, chaque composant de l'application contribue à créer un environnement sécurisé pour les données commerciales sensibles de l'entreprise.

Cette approche proactive de la sécurité, intégrée dès la conception du projet, permet de réduire significativement les risques d'attaques tout en maintenant une expérience utilisateur fluide et une maintenance simplifiée du code.

X – Politique de test

Le processus de développement d'un logiciel est complexe et comporte de nombreux défis. L'un de ces défis majeurs est de s'assurer que le logiciel développé fonctionne correctement et répond aux exigences spécifiées. Pour garantir la qualité et la fiabilité du logiciel, il est essentiel de mettre en place des tests appropriés.

Dans le cadre du développement d'Innov'Events, j'ai mis en place une stratégie de tests complète couvrant trois niveaux : les tests unitaires, les tests d'intégration (API) et les tests end-to-end (E2E). Les tests unitaires se concentrent sur la vérification du bon fonctionnement de chaque fonction de manière isolée, les tests d'intégration examinent les interactions entre les différents composants (contrôleurs, services, base de données), et les tests E2E simulent le comportement réel d'un utilisateur à travers l'interface graphique.

Résultats globaux des tests :

Type de test	Framework	Nombre de tests	Statut
Tests unitaires	Jest	21	<input checked="" type="checkbox"/> Passés
Tests d'intégration API	Jest + Supertest	16	<input checked="" type="checkbox"/> Passés
Tests E2E	Playwright	10	<input checked="" type="checkbox"/> Passés
Total		47	<input checked="" type="checkbox"/>

A – Les tests unitaires

Les tests unitaires sont une méthode de test qui permet de vérifier le bon fonctionnement d'une partie précise d'un logiciel de manière isolée. Chaque "unité" testée correspond à une fonction spécifique. L'objectif est de s'assurer que chaque brique élémentaire du code fonctionne correctement avant de les assembler.

Pour Innov'Events, j'ai utilisé **Jest**, le framework de test JavaScript le plus populaire, pour implémenter les tests unitaires. Jest offre une syntaxe claire, des assertions puissantes et la possibilité de créer des mocks pour isoler les composants testés.

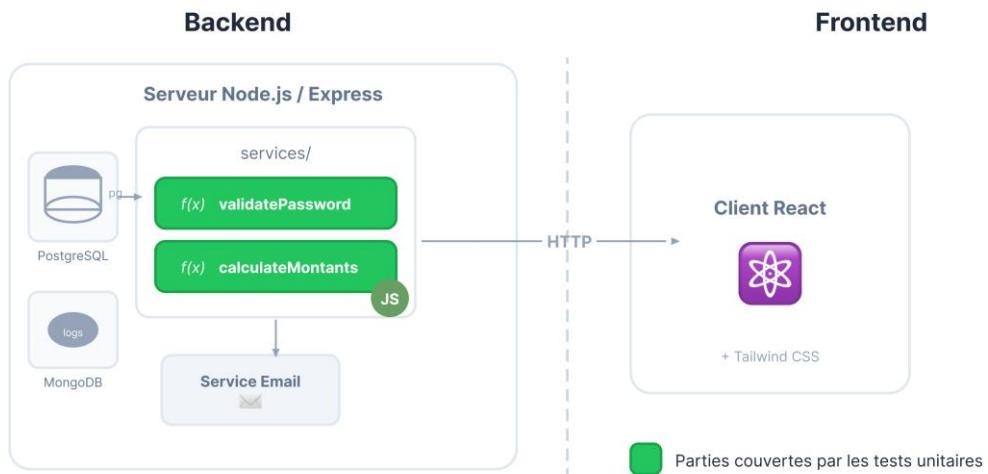


Figure 1
Périmètre des tests unitaires

Figure 30 : Schéma du périmètre des tests unitaires

1 - Tests de validation du mot de passe

La validation du mot de passe est critique pour la sécurité de l'application. J'ai implémenté une série de tests vérifiant les règles de complexité :

```

/* Tests unitaires - Validation
 */

describe('Validation du mot de passe', () => {

  const validatePassword = (password) => {
    const errors = [];

    if (!password || password.length < 8) {
      errors.push('Le mot de passe doit contenir au moins 8 caractères');
    }
    if (!/[A-Z]/.test(password)) {
      errors.push('Le mot de passe doit contenir au moins une majuscule');
    }
    if (!/[a-z]/.test(password)) {
      errors.push('Le mot de passe doit contenir au moins une minuscule');
    }
    if (!/[0-9]/.test(password)) {
      errors.push('Le mot de passe doit contenir au moins un chiffre');
    }
    if (!/[!@#$%^&*(),.?":{}|<>]/.test(password)) {
      errors.push('Le mot de passe doit contenir au moins un caractère spécial');
    }

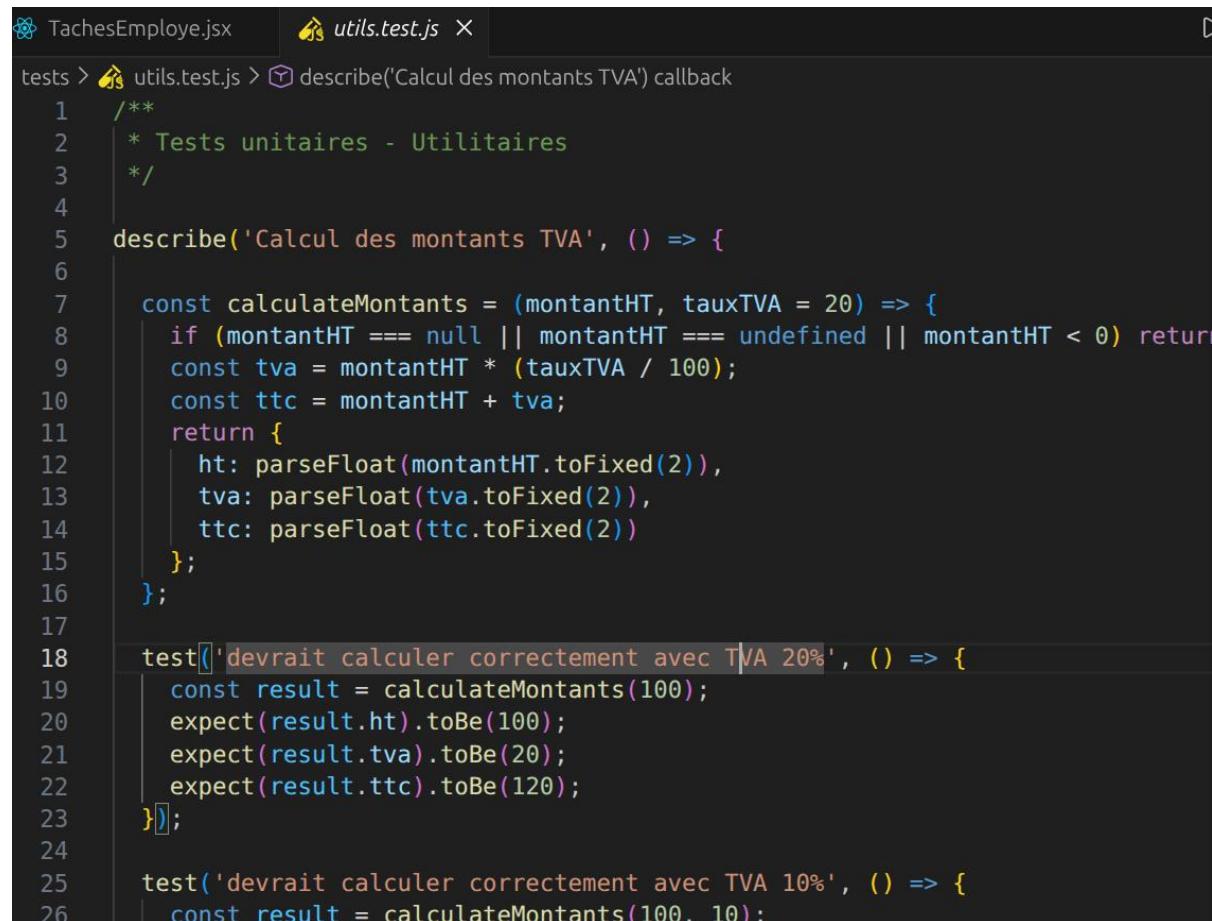
    return { isValid: errors.length === 0, errors };
  }
})

```

Figure 31 : capture du test unitaire de validation des mdp

2 - Tests des calculs de montants TVA

Les calculs financiers étant critiques pour la génération des devis, j'ai implémenté des tests vérifiant l'exactitude des calculs :



The screenshot shows a code editor with two tabs: 'TachesEmploye.jsx' and 'utils.test.js'. The 'utils.test.js' tab is active, displaying a Jest test script. The code defines a function 'calculateMontants' that takes a 'montantHT' and a 'tauxTVA' (defaulted to 20). It checks if 'montantHT' is null or undefined, and if so, returns null. Otherwise, it calculates 'tva' as 'montantHT * (tauxTVA / 100)', then calculates 'ttc' as 'montantHT + tva', and finally returns an object with 'ht', 'tva', and 'ttc' properties, each rounded to two decimal places using 'toFixed(2)'. Two test cases are provided: one for a 20% VAT rate (expecting ht: 100, tva: 20, ttc: 120) and one for a 10% VAT rate (expecting ht: 100, tva: 10, ttc: 110).

```

1  /**
2   * Tests unitaires - Utilitaires
3   */
4
5  describe('Calcul des montants TVA', () => {
6
7    const calculateMontants = (montantHT, tauxTVA = 20) => {
8      if (montantHT === null || montantHT === undefined || montantHT < 0) return null;
9      const tva = montantHT * (tauxTVA / 100);
10     const ttc = montantHT + tva;
11     return {
12       ht: parseFloat(montantHT.toFixed(2)),
13       tva: parseFloat(tva.toFixed(2)),
14       ttc: parseFloat(ttc.toFixed(2))
15     };
16   };
17
18  test(['devrait calculer correctement avec TVA 20%', () => {
19    const result = calculateMontants(100);
20    expect(result.ht).toBe(100);
21    expect(result.tva).toBe(20);
22    expect(result.ttc).toBe(120);
23  }];
24
25  test('devrait calculer correctement avec TVA 10%', () => {
26    const result = calculateMontants(100, 10);
27  });
28});

```

Figure 32 : Capture du test unitaire calculs de montants TVA

Récapitulatif des tests unitaires

Fichier	Module testé	Nombre de tests
validation.test.js	Validation mot de passe	5
validation.test.js	Validation email	3
utils.test.js	Calculs TVA	3
utils.test.js	Génération numéro devis	2
utils.test.js	Vérification rôles	2
statuts.test.js	Statuts événements	4

statuts.test.js	Statuts tâches	2
Total		21 tests

B – Les tests d'intégration (API)

Les tests d'intégration sont une méthode de test qui permet de vérifier que les différentes parties d'un système fonctionnent correctement ensemble. Pour Innov'Events, j'ai utilisé Supertest en combinaison avec Jest pour tester les endpoints de l'API REST.

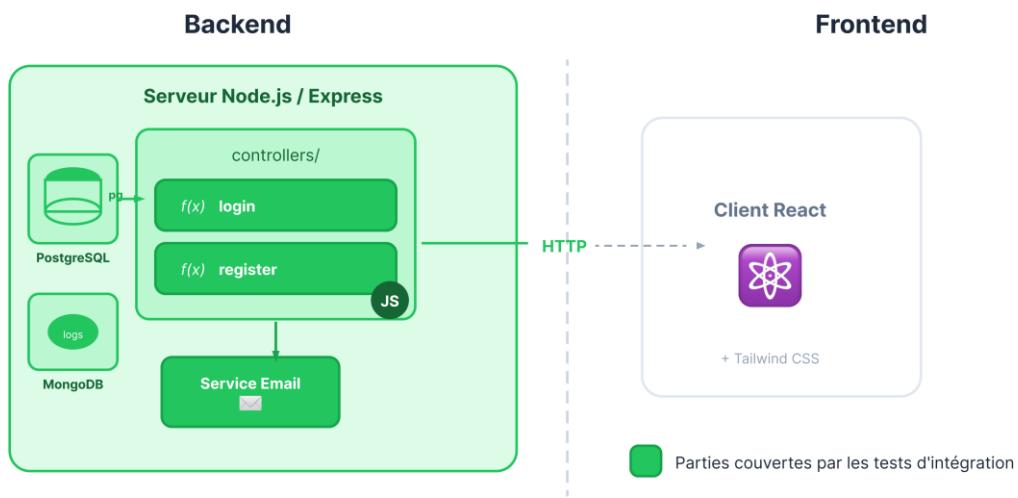


Figure 2
Périmètre des tests d'intégration

Figure 33 : Schéma du périmètre des tests d'intégration

1 - Tests de l'inscription (POST /api/auth/register)

```
javascript

describe('POST /api/auth/register', () => {

    test('devrait créer un nouveau compte client', async () => {
        const response = await request(API_URL)
            .post('/api/auth/register')
            .send(testUser)
            .expect('Content-Type', /json/);

        if (response.status === 201) {
            expect(response.body.success).toBe(true);
            expect(response.body.data).toHaveProperty('token');
            expect(response.body.data).toHaveProperty('user');
            expect(response.body.data.user.role).toBe('client');
        }
    });

    test('devrait refuser un email invalide', async () => {
        const response = await request(API_URL)
            .post('/api/auth/register')
            .send({ ...testUser, email: 'invalid-email' })
            .expect(400);

        expect(response.body.success).toBe(false);
    });
});
```

Figure 34 : capture du test d'intégration de l'inscription (POST /api/auth/register)

2 - Tests de la connexion (POST /api/auth/login)

```
javascript

describe('POST /api/auth/login', () => {

    test('devrait connecter un utilisateur existant (admin)', async () => {
        const response = await request(API_URL)
            .post('/api/auth/login')
            .send({
                email: 'tikaya1999@gmail.com',
                mot_de_passe: 'Admin123!'
            })
            .expect('Content-Type', /json/);

        if (response.status === 200) {
            expect(response.body.success).toBe(true);
            expect(response.body.data).toHaveProperty('token');
            expect(response.body.data).toHaveProperty('user');
            authToken = response.body.data.token;
        }
    });

    test('devrait refuser un mot de passe incorrect', async () => {
        const response = await request(API_URL)
            .post('/api/auth/login')
            .send({
                email: 'tikaya1999@gmail.com',
                mot_de_passe: 'WrongPassword123!'
            });
    });
});
```



Figure 35 : capture du test d'intégration de la connexion (POST /api/auth/login)

3 - Tests du profil utilisateur (GET /api/auth/me)

```
javascript

describe('GET /api/auth/me', () => {

  test('devrait retourner le profil si authentifié', async () => {
    const response = await request(API_URL)
      .get('/api/auth/me')
      .set('Authorization', `Bearer ${authToken}`)
      .expect(200);

    expect(response.body.success).toBe(true);
    expect(response.body.data).toHaveProperty('email');
    expect(response.body.data).toHaveProperty('role');
  });

  test('devrait refuser sans token', async () => {
    const response = await request(API_URL)
      .get('/api/auth/me')
      .expect(401);

    expect(response.body.success).toBe(false);
  });

  test('devrait refuser avec un token invalide', async () => {
    const response = await request(API_URL)
      .get('/api/auth/me')
      .set('Authorization', 'Bearer invalid_token_123')
      .expect(401);
  });
});
```

Figure 36: capture du test d'intégration du profil utilisateur (GET /api/auth/me)

Récapitulatif des tests d'intégration

Endpoint	Méthode	Tests	Scénarios testés
/api/auth/register	POST	4	Création réussie, email invalide, mot de passe faible, champs manquants
/api/auth/login	POST	5	Connexion réussie, email inexistant, mot de passe incorrect, champs manquants

/api/auth/me	GET	3	Profil authentifié, sans token, token invalide
/api/auth/logout	POST	2	Déconnexion réussie, sans authentification
/api/auth/forgot-password	POST	2	Email existant, email inexistant
Total		16 tests	

C – Les tests End-to-End (E2E)

Les tests end-to-end constituent le niveau de test le plus complet, simulant le comportement réel d'un utilisateur interagissant avec l'application via son navigateur. Pour Innov'Events, j'ai utilisé Playwright, un outil moderne de test E2E développé par Microsoft.

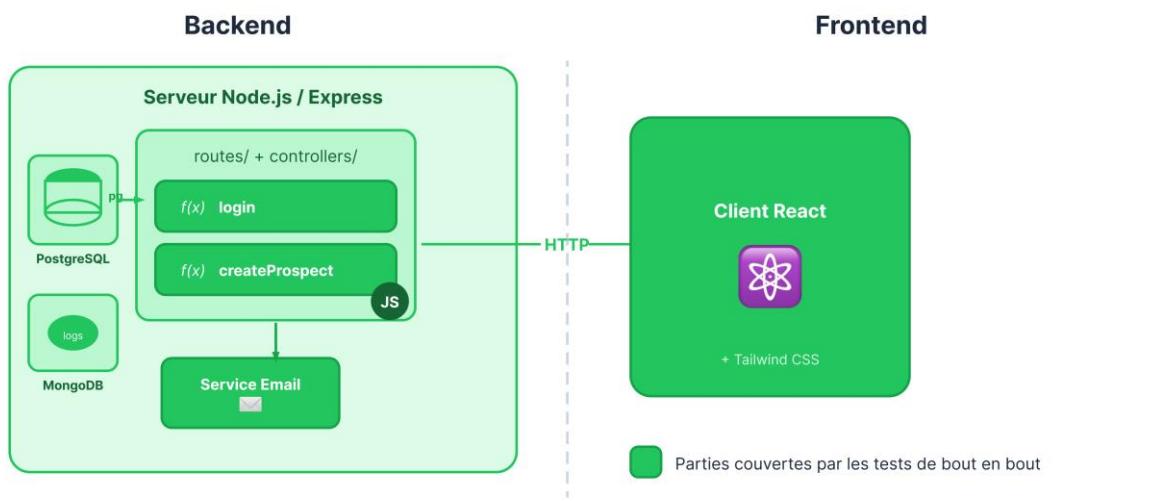


Figure 3
Périmètre des tests de bout à bout

Figure 37 : Schéma du périmètre des tests e2e

Scénarios de tests E2E

```

const { test, expect } = require('@playwright/test');

test.describe('Authentification - Tests E2E', () => {
  test('devrait afficher la page de connexion', async ({ page }) => {
    await page.goto('/');
    await page.click('text=Connexion');

    await expect(page).toHaveURL('/connexion');
    await expect(page.locator('h1')).toContainText('Connexion');
    await expect(page.locator('input#email')).toBeVisible();
    await expect(page.locator('input#mot_de_passe')).toBeVisible();
  });

  test('devrait afficher une erreur avec des identifiants invalides', async ({ page }) => {
    await page.goto('/connexion');

    await page.fill('input#email', 'wrong@example.com');
    await page.fill('input#mot_de_passe', 'WrongPassword123!');
    await page.click('button[type="submit"]');

    await page.waitForTimeout(2000);
    await expect(page).toHaveURL('/connexion');
  });

  test('devrait naviguer vers mot de passe oublié', async ({ page }) => {
    await page.goto('/connexion');

    await page.click('text=Mot de passe oublié');
    await expect(page).toHaveURL('/mot-de-passe-oublie');
  });

  test('devrait afficher/masquer le mot de passe', async ({ page }) => {
    await page.goto('/connexion');

    const passwordInput = page.locator('input#mot_de_passe');
    await expect(passwordInput).toHaveAttribute('type', 'password');

    await page.click('button[aria-label*="mot de passe"]');
    await expect(passwordInput).toHaveAttribute('type', 'text');

    await page.click('button[aria-label*="mot de passe"]');
    await expect(passwordInput).toHaveAttribute('type', 'password');
  });
});

```

Figure 38 : Capture du test de bout en bout

Récapitulatif des tests E2E

Test	Description	Temps
1	Affichage page de connexion	2.0s
2	Erreur identifiants invalides	3.4s
3	Navigation vers inscription	1.9s
4	Navigation mot de passe oublié	1.3s
5	Toggle afficher/masquer mot de passe	785ms
6	Affichage formulaire inscription	999ms

7	Retour à l'accueil	1.2s
8	Affichage page d'accueil	1.1s
9	Navigation au clavier (accessibilité)	669ms
10	Affichage formulaire demande de devis	840ms
Total	10 tests	5.5s

D – Exécution et synthèse

Commandes d'exécution:

```

bash
# Tests unitaires et d'intégration (backend)
cd ~/innovevents-manager && npm test

# Tests E2E (frontend)
cd ~/innovevents-manager/frontend && npm run test:e2e

# Rapport HTML des tests E2E
npm run test:e2e:report
```

Résultats backend (Jest)
```
PASS  tests/auth.functional.test.js
PASS  tests/utils.test.js
PASS  tests/validation.test.js
PASS  tests/statuts.test.js

Test Suites: 4 passed, 4 total
Tests:      37 passed, 37 total
Time:      6.224 s
```

Résultats frontend (Playwright)
```
Running 10 tests using 4 workers
10 passed (5.5s)
```

```

Figure 39 : Capture commande d'exécution

## Synthèse globale

| Type            | Framework        | Fichiers   | Tests    | Temps |
|-----------------|------------------|------------|----------|-------|
| Unitaires       | Jest             | 3          | 21       | ~2s   |
| Intégration API | Jest + Supertest | 1          | 16       | ~4s   |
| E2E             | Playwright       | 1          | 10       | 5.5s  |
| Total           |                  | 5 fichiers | 47 tests | ~12s  |

## Conclusion

Johann KOUAKOU

La mise en place d'une stratégie de tests complète pour Innov'Events, couvrant les trois niveaux (unitaire, intégration, E2E), permet de garantir la qualité et la fiabilité de l'application. Les 47 tests implémentés vérifient les fonctionnalités critiques du système :

- **Tests unitaires** : Validation des données, calculs financiers, gestion des statuts et des rôles
- **Tests d'intégration** : API d'authentification complète (register, login, profil, logout)
- **Tests E2E** : Parcours utilisateur réels incluant la navigation, les formulaires et l'accessibilité

Cette approche de test permet de détecter les régressions dès leur introduction et d'assurer que chaque modification du code ne compromet pas le bon fonctionnement des fonctionnalités existantes. L'exécution rapide des tests (environ 12 secondes au total) facilite leur intégration dans le workflow de développement quotidien.

## XI – Veille Technologique

La veille informatique est une pratique essentielle pour rester informé des dernières évolutions technologiques et des bonnes pratiques de son domaine. Pour faire de la veille informatique efficace, il faut identifier ses objectifs, ses besoins, ses sources et ses outils de veille. On peut utiliser des flux RSS, Twitter/X, Google Alerts, des newsletters, des conférences ou des discussions avec ses pairs.

Pour ma part, passionné par les nouvelles technologies et constamment à l'affût des dernières tendances et innovations dans ce domaine, je suis convaincu que la veille technologique est essentielle pour rester compétitif dans ce secteur en constante évolution. Dans le cadre du développement d'Innov'Events, cette veille m'a permis de faire des choix technologiques pertinents et d'adopter les meilleures pratiques du moment.

### A – Sources de veille en cybersécurité

#### 1 - ANSSI (Agence Nationale de la Sécurité des Systèmes d'Information)

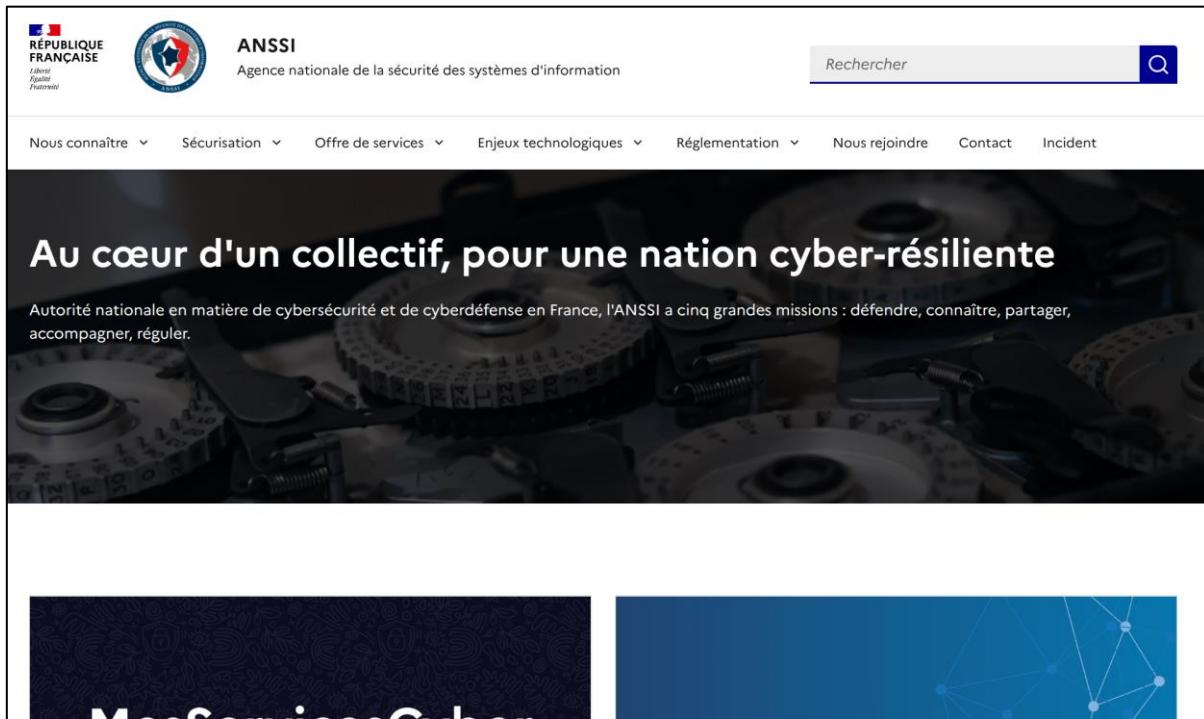
Le site de l'ANSSI est le portail officiel de la cybersécurité en France. Il propose des actualités, des guides, des recommandations, des outils et des services pour aider les acteurs publics et privés à se protéger contre les cyberattaques. Le site de l'ANSSI couvre tous les aspects de la sécurité des systèmes d'information, tels que la gouvernance, la gestion des risques, la défense en profondeur, la sensibilisation et la réaction aux incidents.

Dans le cadre d'Innov'Events, j'ai consulté les recommandations de l'ANSSI concernant:

- Le stockage sécurisé des mots de passe (hachage avec bcrypt)

- La gestion des sessions et des tokens JWT
- Les bonnes pratiques de configuration des serveurs Node.js

Site web : <https://www.ssi.gouv.fr/>



**Figure 40 :** Capture de site de l'Agence Nationale de la Sécurité des Systèmes d'Information

## 2 - OWASP (Open Web Application Security Project)

Le site de l'OWASP est une communauté internationale qui promeut la sécurité des applications web. Il offre des ressources gratuites et ouvertes, telles que des standards, des méthodologies, des projets, des événements et des formations pour aider les développeurs, les testeurs, les auditeurs et les utilisateurs à concevoir, développer, tester et maintenir des applications web sécurisées.

Le site de l'OWASP est notamment connu pour son Top 10 des risques de sécurité applicative, qui est une référence mondiale pour identifier et prévenir les vulnérabilités les plus courantes et les plus critiques. Pour Innov'Events, j'ai particulièrement étudié :

- A01:2025 - Broken Access Control : Mise en place du système RBAC (admin/commercial)
- A02:2025 - Cryptographic Failures : Chiffrement des mots de passe avec bcrypt
- A03:2025 - Injection : Utilisation de requêtes SQL paramétrées avec pg
- A07:2025 - Cross-Site Scripting (XSS) : Échappement automatique avec React

Site web : <https://owasp.org/>

The screenshot shows the OWASP Top 10:2025 website. The header includes a logo, the title "OWASP Top 10:2025", a search bar, and navigation links for "Table of contents", "About This Release", "Main Project Page", "Getting Started", "Navigation", and "Top 10:2025 List". The main content area has a sidebar on the left with links to various sections like Home, Introduction, About OWASP, etc. The main content includes sections for "Welcome to the OWASP Top 10:2025 Release.", "About This Release" (mentioning it's the 2025 version), "Main Project Page" (linking to the main project page), "Getting Started" (linking to the introduction), and "Navigation".

**Figure 41 :** capture d'écran du OWASP Top 10

## B – Documentation officielle des technologies utilisées

### 1 - React.js

La documentation officielle de React (<https://react.dev/>) a été ma source principale pour le développement du frontend d'Innov'Events. J'y ai consulté :

- Les hooks (useState, useEffect, useContext)
- La gestion des formulaires contrôlés
- Le routing avec React Router
- Les bonnes pratiques de structuration des composants

### 2 - Node.js et Express.js

La documentation de Node.js (<https://nodejs.org/docs/>) et Express.js (<https://expressjs.com/>) m'ont permis de :

- Configurer le serveur HTTP

- Mettre en place les middlewares d'authentification
- Gérer les routes API RESTful
- Implémenter la gestion des erreurs

### 3- PostgreSQL

La documentation PostgreSQL (<https://www.postgresql.org/docs/>) m'a guidé pour :

- La création du schéma de base de données
- L'écriture des requêtes SQL optimisées
- La gestion des transactions
- L'utilisation du driver node-postgres (pg)

### 4 - MongoDB et Mongoose

La documentation MongoDB (<https://docs.mongodb.com/>) et Mongoose (<https://mongoosejs.com/docs/>) ont été essentielles pour :

- La configuration de la connexion
- La définition des schémas pour les logs de connexion
- Les opérations CRUD sur les documents

### 5 - Jest et Playwright

Pour la mise en place des tests, j'ai consulté :

- Jest (<https://jestjs.io/docs/getting-started>) : Configuration, assertions, mocks
- Playwright (<https://playwright.dev/docs/intro>) : Tests E2E, sélecteurs, assertions

### 6 - Tailwind CSS

La documentation Tailwind CSS (<https://tailwindcss.com/docs/>) m'a permis de :

- Configurer la charte graphique personnalisée d'Innov'Events
- Utiliser les classes utilitaires pour le responsive design
- Personnaliser les couleurs (bleu royal, or, bleu ciel)

## C – Communautés et forums d'entraide

### 1 - Stack Overflow

Stack Overflow (<https://stackoverflow.com/>) est une ressource incontournable pour tout développeur. Durant le développement d'Innov'Events, j'ai consulté cette plateforme pour résoudre plusieurs problématiques techniques :

- Configuration CORS entre le frontend React et le backend Express

- Gestion des erreurs asynchrones dans Express
- Requêtes SQL complexes avec jointures
- Configuration de Playwright pour les tests E2E

The screenshot shows a Stack Overflow question titled "BCrypt: How to determine whether two hashes refer to the same password". The question was asked 11 years, 2 months ago and modified 11 years, 2 months ago, with 9k views. A sidebar on the left contains navigation links like Home, Questions, AI Assist, Tags, Challenges, Chat, Articles, Users, Jobs, and Companies. A "COLLECTIVES" section lists communities for technologies like Node.js, React, and Angular. A "STACK INTERNAL" section explains the transition from Stack Overflow for Teams to Stack Internal. The main content area includes an advertisement for codecademy and CompTIA PenTest+ certification, and a code snippet showing the result of hashing a password 10 times. A note at the bottom states that the hash is different if stored in a database.

**Figure 41 :** capture d'écran d'une question Stack Overflow consultée

## 2 - GitHub

GitHub (<https://github.com/>) m'a permis de :

- Consulter des projets open source similaires pour m'inspirer
- Lire les issues et discussions sur les repositories des bibliothèques utilisées
- Versionner mon code et collaborer efficacement

## 3 - Dev.to et Medium

Ces plateformes de blogs techniques m'ont fourni des tutoriels et retours d'expérience sur :

- L'architecture d'applications full-stack JavaScript
- Les patterns de conception pour les API REST
- Les stratégies de tests automatisés

## D – Outils de veille utilisés

| Outil         | Usage                                                 | Fréquence    |
|---------------|-------------------------------------------------------|--------------|
| Google Alerts | Alertes sur "React", "Node.js security", "PostgreSQL" | Quotidien    |
| Twitter/X     | Suivi des comptes @reactjs, @nodejs, @oabordeaux      | Quotidien    |
| Newsletter    | JavaScript Weekly, Node Weekly, React Status          | Hebdomadaire |
| YouTube       | Chaînes Fireship, Traversy Media, Web Dev Simplified  | Hebdomadaire |
| Reddit        | r/reactjs, r/node, r/webdev                           | Régulier     |
| Discord       | Serveurs Grafikart, Développeurs Web Français         | Régulier     |

## E – Apports de la veille pour Innov'Events

Cette veille technologique constante m'a permis de :

- Choisir les technologies adaptées** : React pour son écosystème riche, Express pour sa flexibilité, PostgreSQL pour sa robustesse
- Adopter les bonnes pratiques de sécurité** : Grâce à l'ANSSI et l'OWASP, j'ai pu implémenter une authentification sécurisée et prévenir les vulnérabilités courantes
- Résoudre les problèmes efficacement** : Stack Overflow et les documentations officielles m'ont fait gagner un temps précieux
- Implémenter des tests de qualité** : Les ressources sur Jest et Playwright m'ont guidé vers une stratégie de tests complète (47 tests)
- Rester à jour** : Les newsletters et réseaux sociaux m'informent des nouvelles versions et des évolutions des frameworks

## Conclusion

La veille technologique est un pilier fondamental du métier de développeur. Elle permet non seulement de rester compétitif sur le marché de l'emploi, mais aussi de proposer des solutions techniques modernes, sécurisées et performantes. Pour Innov'Events, cette pratique m'a accompagné tout au long du projet, de la phase de conception jusqu'aux tests finaux, garantissant ainsi la qualité et la pérennité de l'application développée.

# XII - Déploiement et Documentation

## A - Architecture de déploiement

L'application Innov'Events Manager utilise une architecture de déploiement moderne et automatisée, combinant conteneurisation Docker, intégration continue (CI/CD) via GitHub Actions, et hébergement cloud sur Render.

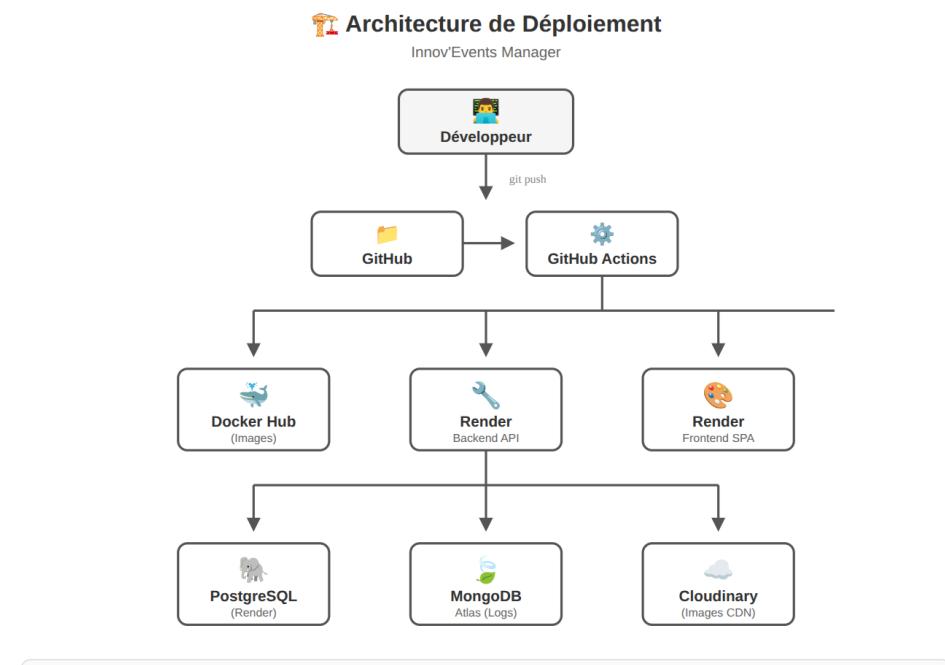


Figure 42 : Vue d'ensemble de l'infrastructure

## 1 - Services déployés

| Service       | Plateforme         | URL / Endpoint                                                                                  |
|---------------|--------------------|-------------------------------------------------------------------------------------------------|
| Backend (API) | Render Web Service | <a href="https://innovevents-manager.onrender.com">https://innovevents-manager.onrender.com</a> |

|                                   |                    |                                                                                                   |
|-----------------------------------|--------------------|---------------------------------------------------------------------------------------------------|
| <b>Frontend (SPA)</b>             | Render Static Site | <a href="https://innovevents-frontend.onrender.com">https://innovevents-frontend.onrender.com</a> |
| <b>Base de données PostgreSQL</b> | Render PostgreSQL  | Connexion interne Render                                                                          |
| <b>Base de données MongoDB</b>    | MongoDB Atlas      | Cluster cloud MongoDB                                                                             |
| <b>Stockage d'images</b>          | Cloudinary         | CDN Cloudinary                                                                                    |
| <b>Registry Docker</b>            | Docker Hub         | tikaya/innovevents-manager                                                                        |

## B - Conteneurisation avec Docker

### 1 - Dockerfile Backend

Le backend Node.js/Express est conteneurisé avec l'image Alpine pour optimiser la taille :

```

Dockerfile > ...
1 | FROM node:20-alpine
2 |
3 | WORKDIR /usr/src/app
4 |
5 | COPY package*.json .
6 |
7 | RUN npm install
8 |
9 | COPY . .
10 |
11 | EXPOSE 3000
12 |
13 | CMD ["npm", "run", "dev"]

```

Figure 43 : Capture du Dockerfile Backend

**Caractéristiques :** - Image de base légère : node:20-alpine (~50 MB vs ~350 MB pour l'image complète) - Installation des dépendances en premier pour optimiser le cache Docker - Port exposé : 3000

### 2 - Dockerfile Frontend

Le frontend React/Vite utilise également une image Alpine :

```

Frontend > Dockerfile > ...
1 FROM node:20-alpine
2
3 WORKDIR /usr/src/app
4
5 COPY package*.json ./
6
7 RUN npm install
8
9 COPY . .
10
11 EXPOSE 5173
12
13 CMD ["npm", "run", "dev", "--", "--host", "0.0.0.0"]
14

```

Figure 44 : Capture du Dockerfile Frontend

**Caractéristiques :**

- Serveur de développement Vite accessible depuis l'extérieur du conteneur
- Port exposé : 5173 - Option --host 0.0.0.0 pour permettre les connexions externes

### 3 - Docker Compose - Environnement complet

Le fichier docker-compose.yml orchestre l'ensemble des services pour le développement local :

```

docker-compose.yml
 ◊ Run All Services
1 services:
2 | # Application Node.js Backend
3 | ◊ Run Service
4 | app:
5 | build:
6 | context: .
7 | dockerfile: Dockerfile
8 | container_name: innovevents-app
9 | restart: unless-stopped
10 | ports:
11 | - "3000:3000"
12 | volumes:
13 | - ./src/app
14 | - ./src/app/node_modules
15 | environment:
16 | - NODE_ENV=development
17 | - POSTGRES_HOST=postgres
18 | - POSTGRES_PORT=5432
19 | - POSTGRES_USER=${POSTGRES_USER}
20 | - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}

```

Figure 45 : Capture du Docker-compose file

### 3.1 Schéma des conteneurs Docker

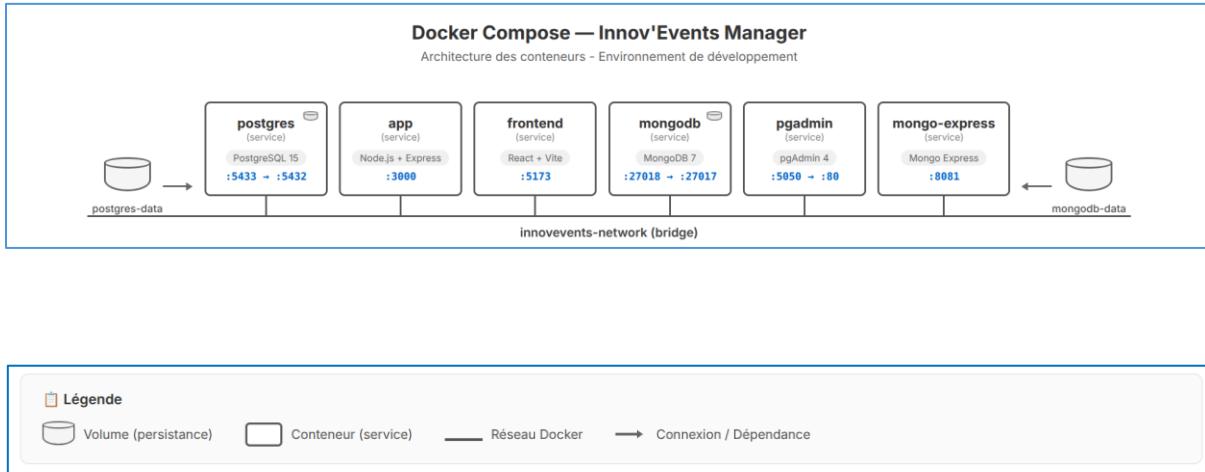


Figure 46 : Schéma Environnement DOCKER COMPOSE NETWORK

## C - Intégration Continue et Déploiement Continu (CI/CD)

### 1 - Pipeline GitHub Actions

Le projet utilise GitHub Actions pour automatiser les tests, le build et le déploiement. Le workflow est défini dans `.github/workflows/ci-cd.yml`.

```
DÉCLENCHEURS
=====
on:
 push:
 branches:
 - main
 - dev

 pull_request:
 branches:
 - main

 workflow_dispatch:

=====
VARIABLES D'ENVIRONNEMENT GLOBALES
=====
env:
 NODE_VERSION: '20'
 DOCKER_IMAGE_NAME: tikaya/innovevents-manager
```

Figure 47 : Capture fichier ci-cd.yml

### 1-1 Étapes du pipeline CI/CD

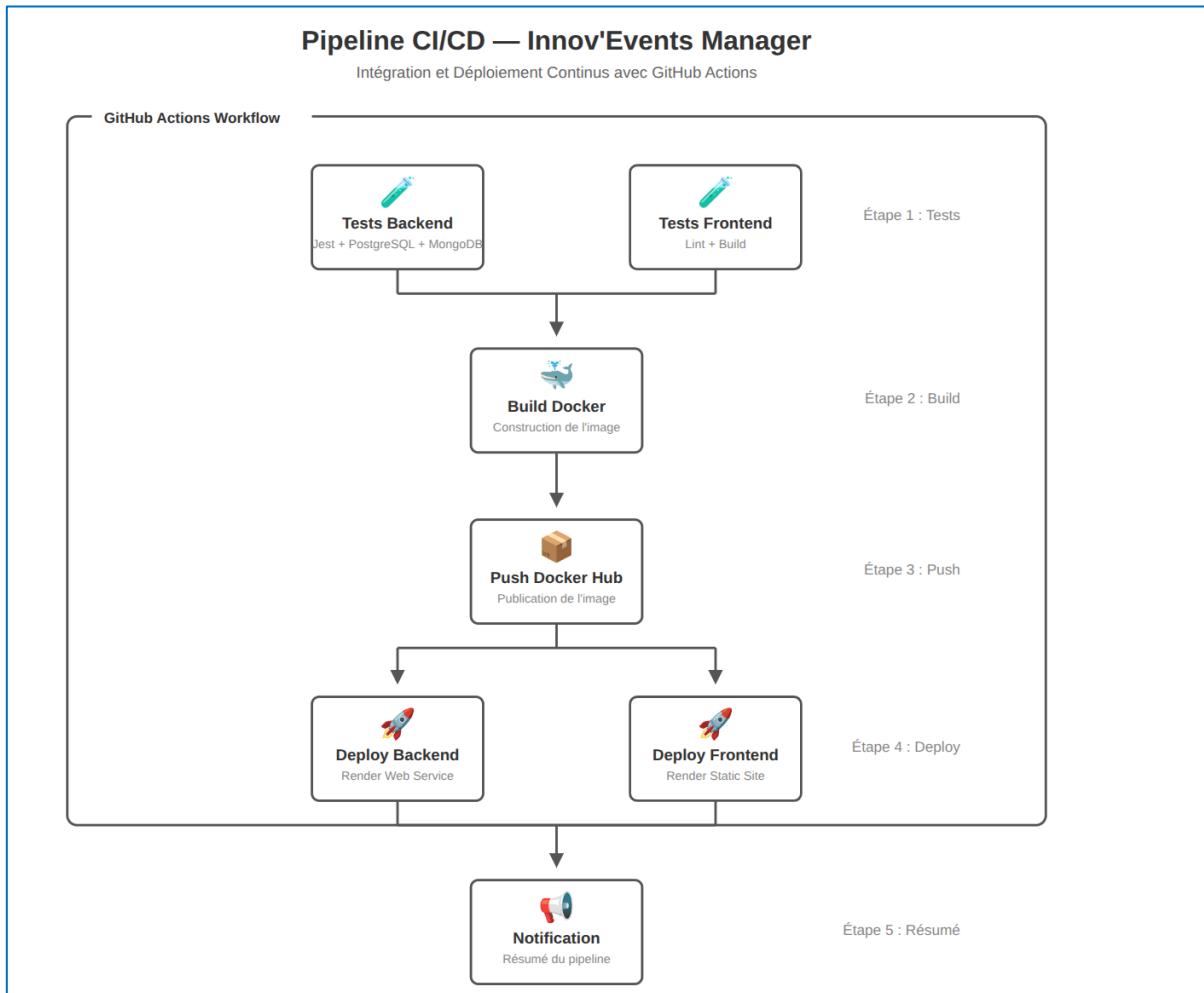


Figure 48 : Architecture du pipeline CI/CD

## D - Hébergement sur Render

### 1- Configuration

Le backend est déployé comme un Web Service sur Render

The screenshot shows the Render interface for the 'innovevents-manager' service. At the top, there are buttons for 'Search', '+ New', 'Upgrade', and help. Below that, the service name 'innovevents-manager' is shown with a 'WEB SERVICE' icon, a 'Docker' tab, and a 'Starter' tab. To the right are 'Connect' and 'Manual Deploy' buttons. The service ID is listed as 'srv-d5n3907gi27c73cimubg'. The repository is 'tikaya / innovevents-manager' with the branch 'main'. The URL is 'https://innovevents-manager.onrender.com'. A 'Filter events' dropdown shows 31 results.

- Deploy live for 8b05ecc: feat: Migration upload images vers Cloudinary - Stockage permanent des images sur Cloudi...** (Success) - January 21, 2026 at 1:08 PM
- Deploy started for 8b05ecc: feat: Migration upload images vers Cloudinary - Stockage permanent des images sur Cloudi...** (Manually triggered by you via Dashboard) - January 21, 2026 at 1:07 PM
- Deploy live for 8b05ecc: feat: Migration upload images vers Cloudinary - Stockage permanent des imag...** (Success) - January 21, 2026 at 12:55 PM
- Deploy started for 8b05ecc: feat: Migration upload images vers Cloudinary - Stockage permanent des images sur Cloudi...** (Manually triggered by you via Dashboard)

**Figure 49 : Capture déploiement backend sur render**

## 2 - Configuration du Frontend (Static Site)

Le frontend est déployé comme un Static Site sur Render :

The screenshot shows the Render interface for the 'innovevents-frontend' service. At the top, there are buttons for 'Search', '+ New', 'Upgrade', and help. Below that, the service name 'innovevents-frontend' is shown with a 'STATIC SITE' icon, a 'Docker' tab, and a 'Starter' tab. To the right are 'Connect' and 'Manual Deploy' buttons. The service ID is listed as 'srv-d5n4guq4d50c73df8ph0'. The repository is 'tikaya / innovevents-manager' with the branch 'main'. The URL is 'https://innovevents-frontend.onrender.com'. A 'Filter events' dropdown shows 28 results.

- Deploy live for 8b05ecc: feat: Migration upload images vers Cloudinary - Stockage permanent des images sur Cloudi...** (Success) - January 21, 2026 at 12:55 PM
- Deploy started for 8b05ecc: feat: Migration upload images vers Cloudinary - Stockage permanent des images sur Cloudi...** (Triggered via Deploy Hook) - January 21, 2026 at 12:54 PM
- Deploy live for dd21888: fix: Autoriser cross-origin pour les images (Helmet)** (Success) - January 21, 2026 at 12:34 PM

**Figure 50 : Capture déploiement du frontend sur render**

### 3 - Configuration PostgreSQL (Render)

The screenshot shows a PostgreSQL service configuration page on Render. At the top, it displays the service name 'innovevents-db' in a purple box, indicating it's free. A link to 'Upgrade your instance' is also present. Below this, the service ID is shown as 'dpg-d5n2v9kmrvns73fgi990-a'. The main section is titled 'Info' and contains a warning message: 'Your database will expire on February 18, 2026. The database will be deleted unless you upgrade'. The 'General' tab is selected, showing the 'Name' field set to 'innovevents-db'. Other details like 'Created' (3 days ago) are also visible.

Figure 51 : Capture implémentation du PostgreSQL sur render

## E - Services externes

### 1 - MongoDB Atlas (Logs)

MongoDB Atlas est utilisé pour stocker les logs d'activité de l'application (connexions, actions RGPD, etc.).

Avantages : - Stockage NoSQL adapté aux logs JSON - Requêtes flexibles pour l'audit - Gratuit jusqu'à 512 MB

The screenshot shows the MongoDB Atlas Data Explorer interface. On the left, there's a sidebar titled "Data Explorer" with sections for "My Queries" and "Data Modeling". Below that is a "CLUSTERS" section with a search bar and a tree view of clusters: "innovevents-cluster" (expanded) containing "admin", "innovevents\_db" (selected), "logs" (highlighted), "local", and "sample\_mflix". The main area shows the "innovevents\_db > logs" collection with 147 documents. A search bar at the top says "Type a query: { field: 'value' } or [Generate query](#)". Below it are buttons for creating, editing, deleting, and comparing documents. A status bar at the bottom left says "System Status: All Good". Three document snippets are displayed:

```

_id: ObjectId('696e72f8d2adc23c95fa0745')
horodatage : 2026-01-19T18:07:52.222+00:00
type_action : "CONNEXION_REUSSIE"
id_utilisateur : 1
details : Object

_id: ObjectId('696e7328d2adc23c95fa0746')
horodatage : 2026-01-19T18:08:40.689+00:00
type_action : "VALIDATION_AVIS"
id_utilisateur : 1
details : Object

_id: ObjectId('696e73d7d2adc23c95fa0747')
horodatage : 2026-01-19T18:11:35.135+00:00
type_action : "MODIFICATION_EMPLOYE"
id_utilisateur : 1
details : Object

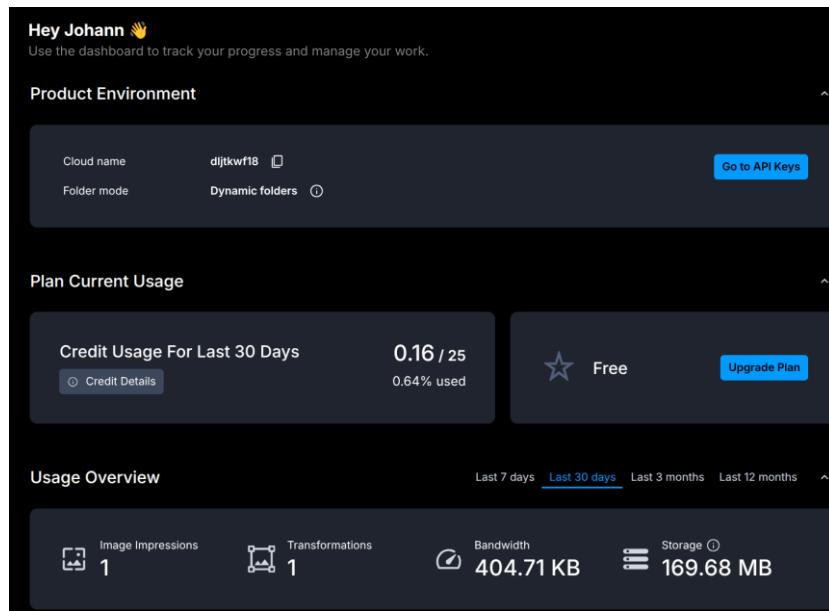
```

**Figure 52 : Capture innovevents\_db sur Mongo Atlas**

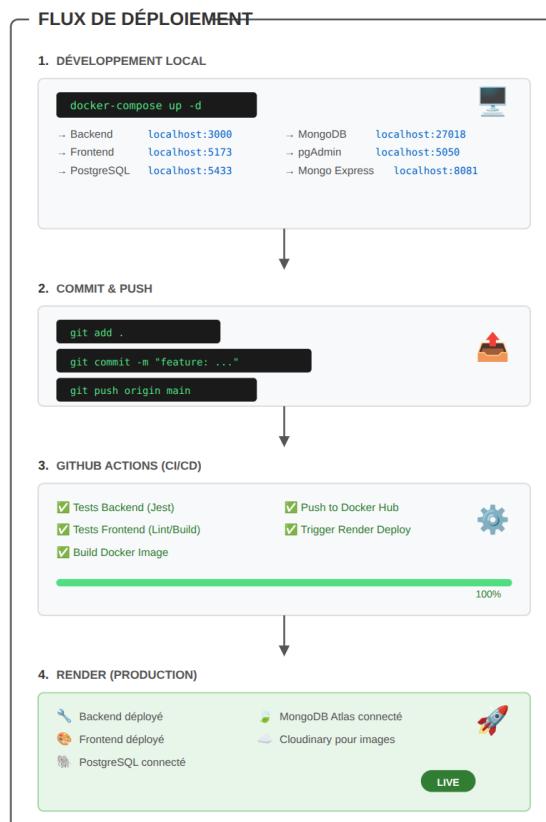
## 2 - Cloudinary (Stockage d'images)

Cloudinary est utilisé pour le stockage permanent des images d'événements.

**Pourquoi Cloudinary ?** Render utilise un système de fichiers éphémère (fichiers perdus à chaque redéploiement) - Cloudinary offre un stockage permanent et un CDN mondial - Pas de problème CORS (URLs HTTPS directes).

**Figure 53 : Capture de cloudfynd**

## F - Flux de déploiement complet

**Figure 55 : Synthèse du flux de déploiement complet**

## XIII - Difficultés Rencontrées

Comme tout projet, celui-ci a eu son lot de galères ! Voici les principales difficultés que j'ai rencontrées et comment je les ai résolues.

### A - Introduction

Tout projet de développement comporte son lot de défis techniques et organisationnels. Cette section présente les principales difficultés rencontrées lors du développement d'Innov'Events Manager, ainsi que les solutions mises en œuvre pour les surmonter.

### B - Les images qui disparaissaient

**Le problème :** Après chaque mise à jour du site, toutes les images uploadées disparaissaient. Frustrant !

**Pourquoi :** Render (mon hébergeur) recrée un nouveau serveur à chaque déploiement. Les fichiers stockés localement sont donc effacés.

**Ma solution :** J'ai migré vers Cloudinary, un service cloud qui stocke les images de façon permanente. Problème réglé !

### C - Les images bloquées par le navigateur

**Le problème :** Même quand les images existaient, le navigateur refusait de les afficher (erreur CORS).

**Pourquoi :** Mon système de sécurité (Helmet) bloquait les ressources venant d'un autre domaine.

**Ma solution :** Avec Cloudinary, les images viennent d'un CDN externe en HTTPS. Plus de blocage !

### D - Gérer deux bases de données

**Le problème :** Mon projet utilise PostgreSQL ET MongoDB. Configurer les deux en local et en production, c'était un peu le bazar.

**Ma solution :**

- En local : Docker Compose lance tout automatiquement
- En production : PostgreSQL sur Render + MongoDB sur Atlas

## E - Le pipeline CI/CD qui plante

**Le problème :** Faire fonctionner les tests automatiques avec les bases de données dans GitHub Actions m'a pris du temps.

**Ma solution :** J'ai configuré des services PostgreSQL et MongoDB directement dans le workflow GitHub Actions. Après plusieurs essais, ça marche !

## F - Le site lent au premier accès

**Le problème :** Avec le plan gratuit de Render, le serveur "s'endort" après 15 min d'inactivité. Le premier visiteur attend parfois 30 secondes.

Ma solution : J'ai accepté cette contrainte pour un projet étudiant. En production réelle, un plan payant réglerait ça.

## Ce que j'ai appris

Ces difficultés m'ont permis de :

- Comprendre les contraintes du cloud (serveurs éphémères)
- Maîtriser Docker et le CI/CD
- Apprendre à chercher et trouver des solutions
- Découvrir des services comme Cloudinary et MongoDB Atlas

**Finalement, chaque problème résolu m'a rendu meilleur développeur !**

# XIV - Conclusion

## 1 - Bilan du projet

Innov'Events Manager est aujourd'hui une application **fonctionnelle et déployée** en production. Elle répond au besoin initial : permettre à une entreprise de gérer ses événements internes et à ses collaborateurs de s'y inscrire facilement.

Le projet m'a permis de toucher à toutes les facettes du développement web moderne :

- **Frontend** avec React et une interface responsive
- **Backend** avec Node.js/Express et une API REST sécurisée
- **Bases de données** avec PostgreSQL et MongoDB
- **DevOps** avec Docker, GitHub Actions et déploiement cloud

Je suis fier d'avoir mené ce projet de A à Z, de la conception à la mise en production.

Johann KOUAKOU

## 2 - Ce qui fonctionne bien

- Authentification sécurisée (JWT + refresh tokens)
- Gestion complète des événements (CRUD)
- Système d'inscription avec places limitées
- Notifications par email
- Interface admin intuitive
- Conformité RGPD (export/suppression des données)
- Pipeline CI/CD automatisé
- Application déployée et accessible en ligne

## 3 - Axes d'amélioration

Avec plus de temps, j'aurais aimé améliorer certains points :

| Axe           | Amélioration possible                                    |
|---------------|----------------------------------------------------------|
| Tests         | Augmenter la couverture de tests (actuellement basique)  |
| Performance   | Optimiser les requêtes et ajouter du cache               |
| Accessibilité | Améliorer l'accessibilité pour les personnes handicapées |
| Mobile        | Créer une vraie application mobile (React Native)        |
| Sécurité      | Ajouter l'authentification à deux facteurs (2FA)         |

## 4 - Évolutions futures envisagées

Si le projet devait continuer, voici les fonctionnalités que j'ajouterais :

### 4.1 Court terme :

- Calendrier interactif pour visualiser les événements
- Système de commentaires sur les événements
- Filtres avancés (par date, lieu, catégorie)

### 4.2 Moyen terme :

- Intégration avec Google Calendar / Outlook
- QR Code pour le check-in le jour de l'événement
- Statistiques avancées pour les administrateurs

### 4.3 Long terme :

- Application mobile native
- Système de recommandation d'événements basé sur les préférences
- Mode multi-entreprise (SaaS)

## 5 - Mot de la fin

Ce projet a été une expérience enrichissante. J'ai appris à résoudre des problèmes concrets, à m'adapter quand ça ne marchait pas, et à livrer une application complète.

Les difficultés rencontrées (images perdues, erreurs CORS, configuration CI/CD...) m'ont poussé à chercher, comprendre et trouver des solutions. C'est ça, le métier de développeur !

**Innov'Events Manager est prêt à accueillir ses premiers utilisateurs.** 