

ECOLE NATIONALE DE LA STATISTIQUE ET DE L'ANALYSE DE L'INFORMATION



RAPPORT FINAL

Application du Shotgun



Elèves :

Yann AHUIE

Mohamed DOUZI

Jade JAUVERT

Khalid JERRARI

Tutrice :

Raya BEROVA

Enseignant :

Ludovic DENEUVILLE

Table des matières

1 Contexte de l'application	3
1.1 Présentation générale	3
1.2 Objectifs initiaux et aboutissements	3
1.3 Utilisateurs de l'application	5
1.4 Données manipulées par l'application	6
1.4.1 Données internes	6
1.4.2 Données externes (API Brevo)	6
2 Architecture générale	8
2.1 Type d'application	8
2.2 Technologies utilisées	9
2.3 Organisation en couches	10
2.3.1 Couches de présentation : les Vues	11
2.3.2 Couche métier : les Services	13
2.3.3 Couche de persistance : les DAO	14
2.4 Les objets métiers et leur cycle de vie	15
2.4.1 La distinction modèle entrant (IN) / sortant (OUT)	15
2.4.2 Cycle de vie d'un objet métier dans l'application	15
3 Fonctionnement de l'application	17
3.1 Processus central : description détaillée de l'inscription d'un participant à un événement	17
3.1.1 Étape 1 : authentification du participant	18
3.1.2 Étape 2 : choix d'un événement	19
3.1.3 Étape 3 : sélection des créneaux de bus	19
3.1.4 Étape 4 : options supplémentaires	19
3.1.5 Étape 5 : création de la réservation	19
3.1.6 Étape 6 : envoi du mail de confirmation (Brevo)	20
3.2 Description sommaire des autres processus de l'application	20
3.2.1 Création d'un événement (admin)	20
3.2.2 Modification d'un événement	22
3.2.3 Annulation ou suppression d'un événement (admin)	22
3.2.4 Consultation des réservations personnelles	23
3.2.5 Modification d'une réservation	23
3.2.6 Suppression d'une réservation	23
3.2.7 Visualisation statistique	24
4 Système de stockage	25
4.1 Rôle du système de stockage	25
4.2 Modèle relationnel et justification	25
4.3 Logique d'accès aux données (DAO)	26
4.4 Transactions et cohérence	27

5	Outils et démarches de développement	29
5.1	Organisation du groupe	29
5.2	Outils utilisés	30
5.3	Démarche qualité	31
5.3.1	Cas particuliers testés	32
5.3.2	Phase de recette	32
6	Performance et temps de traitement	34
6.1	Mesure de la performance	34
6.2	Robustesse face aux changements externes	35
7	Conclusion	36
8	Notes Individuelles	37
8.1	Yann Ahuie	37
8.2	Mohamed Douzi	38
8.3	Jade Jauvert	38
8.4	Khalid Jerrari	39
9	Annexes	41

1 Contexte de l'application

1.1 Présentation générale

Le Bureau des Étudiants (BDE) de l'ENSAI organise tout au long de l'année de nombreux événements destinés à animer la vie étudiante, renforcer la cohésion entre les promotions et favoriser l'intégration des nouveaux arrivants. Qu'il s'agisse d'activités festives, de tournois sportifs, de sorties culturelles ou d'actions menées en partenariat avec d'autres associations, ces événements rencontrent chaque année un fort engouement de la part des étudiants.

Cependant, la gestion logistique de ces activités repose encore largement sur un ensemble d'outils disparates : formulaires Google créés au cas par cas, fichiers Excel partagés entre les membres du BDE, inscriptions collectées manuellement, ou encore informations transmises via différents canaux de communication (messageries instantanées, réseaux sociaux, e-mails). Cette hétérogénéité complique la coordination et oblige les organisateurs à effectuer de nombreuses tâches répétitives et sujettes à l'erreur.

Ce mode de fonctionnement présente plusieurs limites :

- absence de suivi automatisé du nombre de places restantes, ce qui peut conduire à des surréservations ou à des listes d'attente gérées manuellement ;
- risque important d'erreurs, notamment des doublons, des omissions ou une mauvaise saisie, en raison de la multiplicité des supports et de l'absence de centralisation ;
- difficulté à diffuser des rappels ou mises à jour aux participants, les organisateurs devant souvent passer par des envois individuels ou des annonces sur plusieurs plateformes ;
- impossibilité de générer des confirmations personnalisées, ce qui limite le professionnalisme perçu et complique la vérification des inscriptions lors de l'événement.

Afin de faciliter la gestion des inscriptions pour le BDE tout en offrant aux participants un système de réservation plus simple et plus intuitif, nous proposons de développer une application spécialement conçue pour cet usage.

1.2 Objectifs initiaux et aboutissements

L'objectif principal du projet Shotgun est de concevoir et développer une application permettant au BDE de l'ENSAI de centraliser et de simplifier la gestion des inscriptions aux événements qu'il organise (Chartres, WEI, Gala, etc.). Cet objectif de base consiste à offrir un outil fiable permettant aux administrateurs de créer un événement, de définir ses capacités globales ainsi que les différents créneaux de bus aller et retour, chacun doté d'un nombre maximal de places.

L'application doit également améliorer l'expérience des participants en leur permettant de s'inscrire facilement, de sélectionner leurs créneaux de bus et d'indiquer certaines préférences (comme leur consommation ou non d'alcool). À la fin de l'inscription, un e-mail de confirmation personnalisé doit être envoyé automatiquement, récapitulant les choix effectués,

le montant à payer, les modalités de paiement et un code de réservation unique.

Pour atteindre ces objectifs, le projet repose sur :

- un développement en Python utilisant la programmation orientée objet,
- une base de données PostgreSQL pour stocker l'ensemble des informations (événements, créneaux, utilisateurs, réservations),
- une couche DAO pour une interaction structurée et sécurisée avec la base de données,
- l'envoi automatique d'e-mails via l'API Brevo.

Les fonctionnalités essentielles, qui constituent l'objectif de base, incluent :

1. la création d'un compte administrateur,
2. la création et la gestion d'un événement,
3. l'inscription des participants,
4. la génération d'un code de réservation unique et l'envoi d'un mail de confirmation,
5. l'accès en temps réel à la liste des inscrits,
6. la gestion automatique des capacités des créneaux et des événements,
7. la suppression d'une réservation via le code associé.

En complément, des objectifs optionnels peuvent être intégrés afin d'enrichir l'application et de proposer une solution encore plus complète. Parmi ces options figurent notamment :

1. la création facultative d'un compte client pour faciliter les inscriptions ultérieures,
2. la possibilité de modifier une réservation à l'aide du code,
3. la mise en place d'un frontend simple pour les administrateurs et les participants,
4. l'affichage de statistiques sur le nombre d'inscrits,
5. l'envoi de rappels automatiques pour les paiements,
6. le déploiement de l'application,
7. ou encore l'envoi d'un mail d'alerte aux utilisateurs lorsqu'un nouvel événement est créé.

Ainsi, le projet combine un socle fonctionnel indispensable destiné à répondre aux besoins immédiats du BDE, et une série d'objectifs optionnels visant à améliorer l'ergonomie, l'automatisation et la portée de l'application selon le temps disponible et les priorités du développement.

Non seulement nous avons développé l'ensemble des fonctionnalités essentielles et optionnelles prévues dans le cahier des charges, mais nous avons également intégré plusieurs fonctionnalités avancées conçues pour améliorer l'expérience utilisateur et renforcer la gestion globale de la plateforme.

Tout d'abord, nous avons mis en place une gestion complète du compte participant. Celui-ci peut désormais créer un compte, modifier ses informations personnelles à tout moment, ou encore supprimer définitivement son profil. Cette approche offre une autonomie accrue aux utilisateurs et facilite leur interaction avec l'application.

Nous avons également mis en place une gestion entièrement autonome et centralisée des réservations. Depuis son espace personnel, chaque participant peut créer une réservation, la modifier en cas de changement, la supprimer si nécessaire, et consulter l'intégralité de son historique. Cette fonctionnalité assure ainsi un suivi clair, intuitif et efficace des participations passées comme futures.

En ce qui concerne la sécurisation du code de réservation, celle-ci n'est plus nécessaire dans notre architecture. En effet, toute action sensible — qu'il s'agisse de créer, modifier ou supprimer une réservation — exige obligatoirement l'authentification préalable de l'utilisateur. La sécurité repose donc avant tout sur l'identité vérifiée du participant, et non sur la robustesse cryptographique du code de réservation. Une sécurisation renforcée n'aurait été pertinente que si certaines opérations avaient pu être réalisées sans authentification, ce qui n'est pas le cas dans notre système.

De plus, les utilisateurs ont désormais la possibilité de laisser un avis sur un événement auquel ils ont participé, ce qui enrichit le retour d'expérience et peut aider les organisateurs à adapter leurs futures activités.

Nous avons également intégré une recherche avancée des événements, permettant à l'utilisateur de filtrer les résultats par catégorie, par ville ou encore par date. Cette fonctionnalité optimise considérablement la navigation et la pertinence des résultats proposés.

Par ailleurs, un système complet de notifications par e-mail a été développé. Des courriels sont envoyés automatiquement lors de la création, de la modification ou de la suppression d'un compte client, ainsi que lors de la création, de la modification ou de la suppression d'une réservation. Ce mécanisme garantit un suivi clair et informatif pour l'utilisateur.

Enfin, il est à noter que la seule fonctionnalité optionnelle que nous n'avons pas implémentée concerne l'envoi automatique de rappels par courriel liés aux paiements. Cette fonctionnalité, bien qu'envisagée, n'a pas été retenue car elle ne constituait pas une priorité dans le périmètre fonctionnel initial et n'avait pas d'impact significatif sur l'expérience principale des utilisateurs.

1.3 Utilisateurs de l'application

L'application Shotgun s'adresse à la fois aux administrateurs et aux participants. Du côté des administrateurs, elle met à leur disposition les fonctionnalités nécessaires pour créer, modifier ou supprimer des événements, en définissant l'ensemble de leurs paramètres : créneaux de bus aller et retour, capacité propre à chaque bus et capacité globale de l'événement. Un administrateur peut également participer aux événements ; à ce titre, il possède aussi l'intégralité

des fonctionnalités offertes aux participants.

Pour les participants, l'application permet de consulter les différents événements disponibles et de réserver leur place en ligne. Lors de l'inscription, ils doivent sélectionner leurs créneaux de bus, indiquer s'ils consomment de l'alcool et préciser leur statut : adhérent BDE, non-adhérent ou SAM. Ce dernier correspond à un rôle spécifique, indépendant de l'adhésion au BDE : il s'agit du conducteur désigné, sobre, chargé de raccompagner certains participants à la fin de l'événement.

L'application est principalement destinée aux étudiants de l'ENSAI, qu'ils soient nouveaux arrivants ou membres des promotions précédentes.

1.4 Données manipulées par l'application

1.4.1 Données internes

Les données internes manipulées par l'application sont :

- Les informations sur les utilisateurs : nom, prénom, e-mail, mot de passe ;
- Les informations sur les événements : titre, lieu, date, capacité globale ;
- Les données de transport : nombre de places pour chaque bus aller/retour ;
- Les réservations : date d'inscription, options choisies (sam, boisson, bus), code unique ;
- Le suivi statistique : nombre d'inscrits, taux de remplissage par bus.

Toutes ces données sont stockées dans une base PostgreSQL. Un diagramme physique des données est disponible dans l'annexe.

1.4.2 Données externes (API Brevo)

L'application intègre un service externe : l'API d'envoi d'e-mails Brevo. Cette API permet d'ajouter dans des applications les services de communication proposés par Brevo. Dans le cadre de notre projet, l'API permet l'envoi automatique d'e-mails de confirmation, l'envoi d'e-mails d'alerte lors de la création d'un nouvel événement, ou encore l'envoi potentiel de relances de paiement (cette dernière étant une fonctionnalité optionnelle).

Les données envoyées à Brevo suivent un schéma JSON défini : destinataire, objet, contenu textuel, contenu HTML, nom de l'expéditeur, etc. Aucune donnée sensible n'est transmise (les mots de passe restent internes).

La première étape est de générer la clé API dans un compte Brevo.

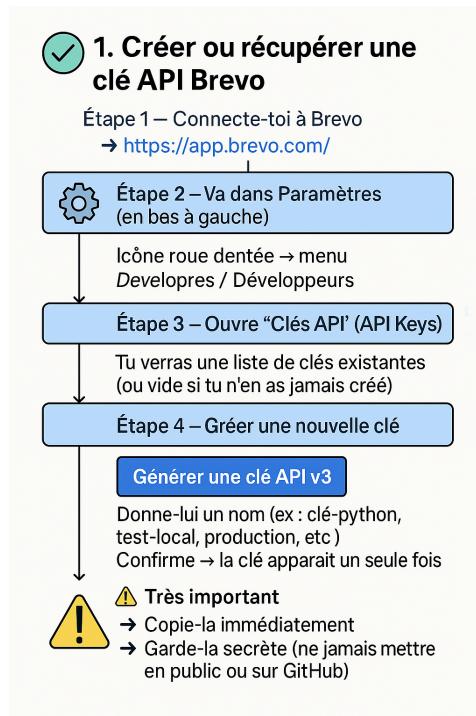


FIGURE 1 – Procédure pour générer une clé API Brevo

Pour l'envoi de courriels depuis un script Python, il suffit d'utiliser la méthode suivante :

```

def send_email_brevo(to_email, subject, message_text):
    url = "https://api.brevo.com/v3/smtp/email"
    headers = {
        "accept": "application/json",
        "api-key": os.environ["TOKEN_BREVO"],
        "content-type": "application/json"
    }
    data = {
        "sender": {"name": "no_reply BDE Ensai", "email": os.environ["EMAIL_BREVO"]},
        "to": [{"email": to_email, "name": "Destinataire"}],
        "subject": subject,
        "textContent": message_text
    }

    response = requests.post(url, headers=headers, json=data)
    return response.status_code, response.text

```

FIGURE 2 – Méthode pour envoyer un courriel via Brevo

2 Architecture générale

2.1 Type d'application

L'application est un programme Python fonctionnant en mode console et structuré selon une architecture multi-couche (Vue → Service → Data Access Object [DAO] → Base de données), tout en intégrant une API externe pour l'envoi d'e-mails (Brevo).

Elle a été conçue en suivant une architecture modulaire et robuste, inspirée des pratiques largement utilisées dans l'industrie logicielle afin de garantir une organisation claire du code et une meilleure gestion de la complexité.

Le système s'appuie sur une séparation nette des responsabilités entre plusieurs couches complémentaires :

- Les vues : elles constituent l'interface console et sont responsables des interactions avec l'utilisateur. Elles affichent les informations essentielles, collectent les entrées et transmettent les actions demandées aux services métier.
- Les services métier : cette couche regroupe la logique applicative. Elle vérifie la cohérence des actions (capacités restantes, droits d'accès, règles métier), orchestre le dialogue entre la vue et les DAO et assure le bon déroulement des opérations comme la création d'un événement, l'inscription d'un participant ou la génération d'un e-mail de confirmation.
- Les DAO (Data Access Objects) : les services s'appuient sur des DAO pour réaliser les opérations de lecture et d'écriture dans la base PostgreSQL. Les DAO assurent l'accès aux données, isolant complètement la logique applicative des détails techniques liés à la base de données. Chaque DAO est responsable d'un ensemble d'opérations CRUD (Create, Read, Update, Delete) sur une table ou un ensemble de tables spécifiques.
- La base de données PostgreSQL : elle stocke l'ensemble des informations persistantes (utilisateurs, événements, créneaux, réservations, rôles) et garantit l'intégrité des données. Les schémas et contraintes ont été conçus pour refléter fidèlement les règles métier tout en évitant les incohérences.
- Les composants externes : en particulier l'API Brevo, utilisée pour l'envoi d'e-mails automatisés (confirmations, notifications). Cette intégration permet de se rapprocher d'un fonctionnement réel et d'anticiper l'évolution vers une application déployée à plus grande échelle.

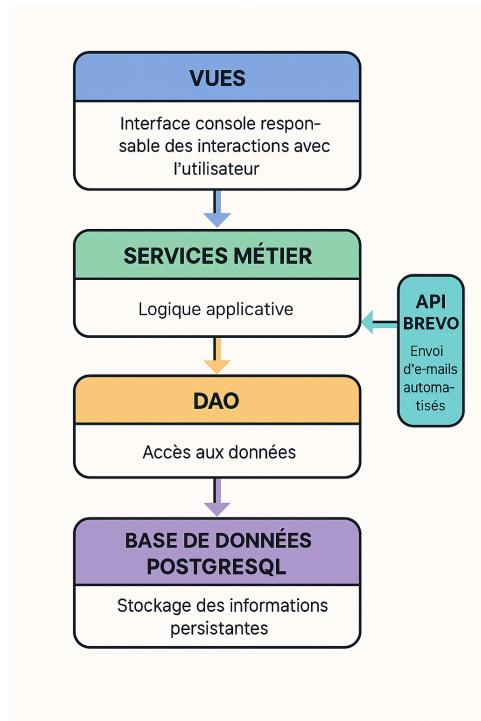


FIGURE 3 – Différentes couches

Cette structure a été choisie pour rendre l’application évolutive, testable et maintenable, tout en permettant d’ajouter facilement de nouvelles fonctionnalités (statistiques, rappels e-mail, front-end dans le futur, etc.).

2.2 Technologies utilisées

Comme indiqué précédemment, l’application Shotgun est développée en Python, en utilisant la version 3.13.8. L’ensemble des dépendances nécessaires à son exécution est listé dans le fichier requirements.txt, qui doit être installé avant le lancement du programme.

Parmi ces bibliothèques, plusieurs jouent un rôle technique central dans l’architecture de l’application. InquirerPy est utilisée pour construire des interfaces utilisateurs en ligne de commande (CLI) interactives, facilitant la navigation et la saisie des données dans un environnement textuel.

Pour assurer la communication avec le système de gestion de base de données PostgreSQL, l’application s’appuie sur psycopg2, une bibliothèque Python permettant d’établir des connexions, d’exécuter des requêtes SQL et de gérer les transactions de manière fiable.

L’envoi des e-mails repose sur l’utilisation de requests, bibliothèque permettant d’effectuer des requêtes HTTP ; elle est notamment utilisée pour appeler l’API Brevo et transmettre les informations nécessaires à la génération et à l’envoi des messages.

La gestion des paramètres de configuration sensibles (comme les clés API ou les identifiants de connexion), stockés dans un fichier .env, est assurée par la bibliothèque python-

dotenv, qui permet de charger ces variables d'environnement sans les intégrer directement dans le code source..

L'application s'appuie sur PostgreSQL comme système de stockage principal. Cette base de données garantit la persistance, la cohérence et l'intégrité des informations relatives aux événements, aux réservations, aux participants et aux transports. Les interactions avec PostgreSQL sont entièrement encapsulées dans la couche DAO, ce qui permet aux services métier de manipuler les données via des méthodes dédiées, sans avoir à gérer directement les requêtes SQL.

Enfin, l'application intègre l'utilisation d'une API externe. Une API (Interface de Programmation d'Application) désigne un ensemble de règles permettant à différents systèmes logiciels d'échanger des informations. Dans notre cas, nous utilisons l'API Brevo, qui prend en charge l'envoi des e-mails de confirmation et de notification mentionné précédemment.

2.3 Organisation en couches

Les fichiers des couches sont organisés dans des dossiers structurés. L'arborescence de ces fichiers et de leurs dossiers associés est présentée ci-dessous :



FIGURE 4 – Architecture multi-couches du projet

2.3.1 Couches de présentation : les Vues

Les vues constituent l'ensemble des interfaces console de l'application et sont regroupées dans le répertoire `view/`. Leur rôle est strictement limité à l'interaction avec l'utilisateur : elles affichent les informations pertinentes, recueillent les saisies via `InquirerPy`, transmettent les actions demandées aux services correspondants et redirigent vers les vues appropriées après traitement.

Elles ne contiennent aucune logique métier ; elles se concentrent exclusivement sur la gestion de l'interface homme-machine. Cette séparation nette des responsabilités garantit leur simplicité, facilite leur testabilité et contribue à la cohérence de l'architecture multi-couche.

La couche `view` est structurée en huit sous-dossiers — accueil, administrateur, auth, client, commentaires, consulter, evenement et reservations — chacun regroupant les vues associées aux différentes fonctionnalités proposées par l'application en console.

Le sous-dossier `auth` contient l'ensemble des vues liées aux mécanismes d'authentification et de gestion de compte. On y retrouve notamment :

- `connexion_vue.py` : permet à tout utilisateur, qu'il soit administrateur ou participant, de s'authentifier en saisissant son identifiant et son mot de passe ;
- `creation_compte_vue.py` : gère la création de compte pour les participants (nom, prénom, courriel, mot de passe). Les comptes administrateurs, quant à eux, sont définis directement dans le système ;
- `modification_compte_vue.py` : offre aux participants la possibilité de modifier leurs informations de connexion ;
- `suppression_compte_vue.py` : permet la suppression d'un compte utilisateur, fonctionnalité réservée aux participants.

Le sous-dossier `evenement` regroupe les vues dédiées à la gestion des événements. On y trouve :

- `creer_evenement_vue.py` : permet uniquement à un administrateur de créer un nouvel événement ;
- `modifier_evenement_vue.py` : permet uniquement à un administrateur de modifier les informations d'un événement existant ;
- `supprimer_evenement_vue.py` : autorise uniquement l'administrateur à supprimer un événement ;

La création d'un événement est gérée par la classe `CreerEvenementVue`, située dans le fichier `src/view/evenement/creer_evenement_vue.py`. Cette vue, réservée aux administrateurs, permet non seulement de créer un événement mais également de configurer les transports associés.

Dès son initialisation, la vue instancie deux services :

- `EvenementService`, chargé de la création de l'événement dans la base de données,
- `BusService`, utilisé pour ajouter les créneaux de bus (aller et retour) après la création de l'événement.

Lors de l'appel à la méthode choisir_menu(), la vue commence par vérifier les droits d'accès via la session utilisateur : seule une personne connectée en tant qu'administrateur peut accéder à cette fonctionnalité. En cas d'accès refusé, l'utilisateur est redirigé vers la page d'accueil.

Lorsque l'administrateur est authentifié, la vue procède à la collecte interactive des données de l'événement à l'aide de la bibliothèque InquirerPy. Les champs suivants sont alors saisis :

- titre (obligatoire),
- adresse et ville (optionnels),
- date de l'événement, validée au format YYYY-MM-DD,
- description (optionnelle),
- capacité totale (entier strictement positif),
- catégorie (optionnelle),
- capacité totale (entier strictement positif),
- statut (parmi une liste définie : « pas encore finalisé », « disponible en ligne », « déjà réalisé », « annulé »).

Une fois les informations saisies, la vue construit un objet EvenementModelIn (modèle Pydantic). Toutes les validations sont prises en charge par Pydantic : en cas d'erreur de validation, un message explicite est affiché à l'utilisateur et la création est annulée.

Lorsque l'objet EvenementModelIn est correctement construit, l'appel au service EvenementService.create_event() assure la création de l'événement dans la base PostgreSQL via la couche DAO. En cas de succès, l'identifiant de l'événement fraîchement créé est affiché, et une notification automatique est envoyée à l'ensemble des participants (via l'API Brevo, déclenchée dans les services).

La vue passe ensuite à la configuration des bus. L'administrateur peut définir :

- le nombre de places pour le bus aller,
- une description (arrêt, horaire, consignes),
- le nombre de places pour le bus retour,
- et une description associée.

Les capacités renseignées doivent être des entiers numériques ; si l'administrateur saisit "0", cela signifie qu'aucun bus n'est prévu pour ce trajet. Une fois les informations collectées, la vue appelle la méthode ajouter_bus_evenement() du BusService, qui crée les entrées correspondantes en base de données.

En cas d'erreur lors de la configuration des bus, l'événement reste créé, mais sans transport associé. Finalement, la vue renvoie l'utilisateur vers le menu d'accueil administrateur avec un message de confirmation.

2.3.2 Couche métier : les Services

La couche service encapsule toute la logique métier. Son rôle est essentiel car elle applique les règles internes (capacités des bus, unicité des réservations, formats des entrées, cohérence), délègue l'accès aux données au DAO, communique avec les API externes, et transforme les données pour les envoyer ou les afficher.

Chaque domaine possède son service :

- **EvenementService** : création, mise à jour, listage, suppression.
- **ReservationService** : gestion des inscriptions, code unique, envoi du mail de confirmation.
- **ParticipantService** : comptes utilisateurs, authentification.
- **BusService** : configuration et gestion des transports liés à un événement.

La logique métier reste ainsi parfaitement isolée et réutilisable.

Description de la couche Service : AdministrateurService

La classe AdministrateurService constitue la couche métier responsable de la gestion des administrateurs au sein de l'application Shotgun. Comme pour l'ensemble des services de l'architecture, son rôle est d'assurer l'interface entre les vues — qui collectent les interactions utilisateur — et la couche DAO, qui gère l'accès aux données et la communication avec la base PostgreSQL. Elle englobe ainsi la logique métier, les règles de validation et les mécanismes de contrôle nécessaires pour garantir la cohérence et la sécurité des opérations effectuées sur les comptes administrateurs.

Dès son initialisation, le service instancie un objet AdministrateurDao, qui lui permet d'accéder aux méthodes de persistance. La responsabilité de cette couche n'est pas d'interagir directement avec la base de données, mais de vérifier que chaque opération demandée respecte bien les contraintes métier avant d'être déléguée au DAO. Les méthodes de lecture (get_all_admins, get_admin_by_id, get_admin_by_email) illustrent ce principe : elles appellent le DAO pour récupérer les informations, mais effectuent systématiquement des vérifications supplémentaires. Par exemple, lorsqu'un administrateur est recherché à partir d'un identifiant ou d'un courriel, le service vérifie qu'un résultat existe réellement ; à défaut, il lève une exception explicite afin de signaler une anomalie fonctionnelle.

Le service assure également la création de nouveaux administrateurs via la méthode create_admin. Avant de déléguer l'ajout au DAO, il applique une règle métier fondamentale : il vérifie qu'aucun autre administrateur n'utilise déjà la même adresse courriel. Cette vérification préventive permet d'éviter les doublons et de garantir l'unicité des comptes. Le même principe est appliqué lors de la mise à jour d'un administrateur : la méthode update_admin s'assure que l'administrateur existe avant de lancer la modification. De même, une suppression (delete_admin) n'est possible que si le compte ciblé est retrouvé dans la base, ce qui évite les opérations incohérentes sur des identifiants inexistantes.

La couche service gère également les aspects liés à l'authentification. La méthode authenticate_admin délègue au DAO la vérification du couple email/mot de passe, mais c'est bien le service qui contrôle la cohérence du résultat et lève une exception en cas d'identifiants

invalides. Cela permet à la vue d'afficher un message d'erreur clair sans exposer la logique interne. De même, le changement de mot de passe (`change_admin_password`) repose sur une étape préalable de validation : l'existence de l'administrateur doit être confirmée avant de procéder à la mise à jour.

Ainsi, la classe `AdministrateurService` incarne parfaitement le rôle attendu de la couche métier dans une architecture multi-couche : elle centralise les règles fonctionnelles, sécurise les opérations sensibles, garantit l'intégrité des données manipulées et maintient une séparation stricte entre les aspects techniques et les interactions utilisateur. En isolant les vues des détails de persistance et en structurant les échanges avec le DAO, cette couche renforce la maintenabilité, la testabilité et la robustesse globale de l'application.

2.3.3 Couche de persistance : les DAO

La couche DAO (Data Access Object) est responsable de l'interaction directe avec la base de données PostgreSQL. Chaque DAO contient les opérations CRUD (Create, Read, Update, Delete) pour un domaine spécifique, offrant une interface claire et cohérente pour la couche service. Cette séparation permet aux services de rester indépendants de la technologie de persistance sous-jacente et de se concentrer uniquement sur la logique métier.

Les DAO de l'application Shotgun sont organisés par domaine :

- **AdministrateurDao** : fournit les méthodes pour créer, lire, mettre à jour et supprimer des administrateurs dans la base. Elle inclut également des fonctions utilitaires comme la recherche par email ou par identifiant.
- **EvenementDao** : gère la persistance des événements, y compris la création d'événements, la modification des informations, la suppression et le listage selon différents critères (date, catégorie, statut).
- **ReservationDao** : responsable des réservations des utilisateurs aux événements. Elle permet de créer une réservation, vérifier l'unicité d'un code, lister les réservations par événement ou par participant, et supprimer une réservation.
- **ParticipantDao** : gère les comptes participants, incluant la création, la modification, la suppression et la recherche de participants. Elle sert également à vérifier l'existence d'un compte pour l'authentification.
- **BusDao** : enregistre les données des transports liés à un événement, notamment les créneaux, la capacité et les descriptions.

Chaque DAO est responsable de la traduction des objets métiers (modèles Pydantic) en opérations SQL, ainsi que de la récupération et de la transformation des résultats de la base en objets manipulables par les services. Les DAO effectuent des opérations techniques et garantissent la cohérence des transactions avec la base de données.

Cette organisation multi-couche permet de séparer les responsabilités, chaque couche à son rôle bien défini. Les services peuvent être réutilisés par différentes interfaces (console, API, interface web) sans modification. Cette disposition des couches assurent également la testabilité, c'est-à-dire que les vues peuvent être testées indépendamment des services, et les services peuvent être testés indépendamment des DAO grâce à l'injection de dépendances ou à des mocks. Et enfin, la couche service valide toutes les opérations avant leur persistance,

limitant ainsi le risque d'incohérences dans la base de données et garantissant ainsi la sécurité et la robustesse.

Ainsi, la couche DAO constitue la base technique fiable de l'application, assurant la persistance des données tout en restant totalement découpée de la logique métier et des interactions utilisateur.

2.4 Les objets métiers et leur cycle de vie

Pour structurer les données qui circulent dans l'application, nous avons refusé d'utiliser de simples dictionnaires Python (trop fragiles). Nous avons opté pour des Objets Métiers typés en utilisant la librairie Pydantic. Ces objets agissent comme des contrats d'interface entre les différentes couches (Vue <-> Service <-> DAO).

2.4.1 La distinction modèle entrant (IN) / sortant (OUT)

Nous avons appliqué une séparation stricte pour chaque entité (Utilisateur, Événement, Réervation, Bus) :

Modèle in (ex : UtilisateurModelIn) :

Il représente les données brutes saisies par l'utilisateur ou préparées par le service. Sa spécificité : il ne contient pas d'ID (car la donnée n'est pas encore en base) et peut contenir des données sensibles en clair (ex : mot de passe) avant traitement. Son rôle est de valider automatiquement les types de données à l'entrée

Modèle out (ex : UtilisateurModelOut) :

Il représente une donnée extraite de la base de données. Contrairement au Modèle In, le Modèle Out possède un ID unique mais aussi des dates de création, et les données sensibles sont nettoyées ou hachées. Il garantit que la Vue reçoit des données formatées et sécurisées pour l'affichage.

2.4.2 Cycle de vie d'un objet métier dans l'application

Prenons l'exemple de la création d'un événement pour illustrer comment ces objets sont utilisés :

1. Vue (CreerEvenementVue) : L'administrateur saisit les informations. La vue instancie un objet EvenementModelIn. Si les données sont invalides (ex : capacité négative), Pydantic lève une erreur immédiate et bloque le processus.
2. Service (EvenementService) : Il reçoit cet objet ModelIn. Il applique des règles métier supplémentaires (ex : vérifier que la date n'est pas dans le passé).
3. DAO (EvenementDao) : Il reçoit l'objet ModelIn, le "déplie" en paramètres SQL et l'insère en base.
4. Retour (BDD -> DAO) : La base de données renvoie la ligne créée (avec le nouvel ID généré et la date de création). Le DAO transforme immédiatement cette ligne brute (tuple/dict) en un objet EvenementModelOut.
5. Retour à la vue : L'objet ModelOut remonte jusqu'à la vue, qui peut alors afficher : "Événement n°42 créé avec succès". Ce découplage fort permet de modifier la structure de la base de

données sans casser toute l’application : il suffit souvent de mettre à jour le modèle Pydantic et le DAO, sans toucher au reste du code.

3 Fonctionnement de l'application

Cette partie présente le fonctionnement interne de l'application, en détaillant le cheminement d'un utilisateur lors des principales opérations. L'objectif est de montrer comment l'ensemble des composants décrits dans la partie précédente interagissent pour satisfaire les besoins exprimés.

Dans un premier temps, nous nous concentrerons sur l'analyse approfondie d'un processus clé : l'inscription d'un participant à un événement. Nous présentons ensuite les autres fonctionnalités principales afin d'en esquisser la logique globale, mais de façon succincte.

3.1 Processus central : description détaillée de l'inscription d'un participant à un événement

La classe `ReservationVue` gère l'ensemble du processus d'inscription d'un utilisateur à un événement et constitue l'interface console dédiée à la réservation. Lorsqu'un utilisateur accède à cette vue, l'application commence par vérifier qu'il est bien authentifié grâce à la session active ; en l'absence de connexion, il est immédiatement redirigé vers la vue de consultation en lui indiquant qu'une authentification est nécessaire. Si aucun événement n'a été préalablement passé à la vue, l'utilisateur se voit proposer la liste des événements auxquels il n'est pas encore inscrit, afin d'éviter toute duplication de réservation. Une fois l'événement choisi, la vue affiche ses informations essentielles — titre, date et éventuellement capacité — puis procède automatiquement à une vérification de la disponibilité des places.

L'application interroge le `ReservationService` pour connaître le nombre d'inscrits et déterminer si l'événement est complet ; si c'est le cas, l'utilisateur est informé et renvoyé vers la vue précédente. Si des places restent disponibles, le processus se poursuit par la gestion des transports. La vue interroge alors le `BusService` afin de déterminer la capacité des bus aller et retour, ainsi que le nombre de places déjà prises ; en fonction de ces données, elle indique si un bus est disponible, complet ou non prévu, et propose à l'utilisateur de réserver ou non une place à bord.

L'utilisateur renseigne ensuite plusieurs options liées à son statut ou à ses préférences : indication de son adhésion au BDE, déclaration d'être SAM (conducteur désigné) ou ajout d'une boisson. Une fois toutes ces informations collectées, la vue construit un objet métier `ReservationModelIn`, encapsulant l'ensemble des paramètres nécessaires à la création d'une réservation. Cet objet est transmis au `ReservationService`, qui se charge de valider la demande, de vérifier les contraintes (notamment l'unicité de la réservation via la combinaison utilisateur/événement) et finalement d'enregistrer la réservation en base de données.

En cas d'erreur — comme un conflit, une réservation existante ou un problème de capacité — la vue affiche un message explicite, tandis qu'en cas de succès, elle confirme la réservation et déclenche l'envoi d'un e-mail de confirmation si l'API Brevo est disponible. Cet e-mail reprend toutes les options choisies et confirme officiellement l'inscription de l'utilisateur à l'événement. Enfin, une fois la réservation finalisée, l'utilisateur est redirigé vers son menu personnel, administrateur ou client selon son profil, dans une continuité fluide de navigation.

L'ensemble de ce processus illustre la séparation nette entre la couche vue, chargée des interactions, et les services responsables de la logique métier, assurant ainsi une structure claire, robuste et conforme aux principes de l'architecture multi-couche.

L'exemple suivant présente, sous forme de diagramme, le processus d'inscription d'un participant à un événement :

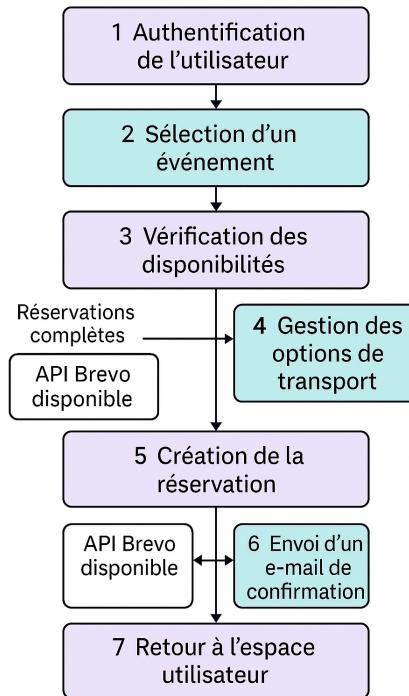


FIGURE 5 – Diagramme réservation d'un événement

L'inscription à un événement constitue l'élément central de l'application Shotgun. Ce processus implique :

- l'utilisateur (participant ou administrateur),
- le système d'authentification,
- la vue d'inscription,
- les services EvenementService, BusService et ReservationService,
- la base PostgreSQL,
- l'API Brevo pour l'envoi d'un e-mail de confirmation.

3.1.1 Étape 1 : authentification du participant

Dans le cadre de la présentation de ce processus, l'utilisateur sera ici un participant. À son arrivée dans l'application, un participant doit s'identifier via la vue ConnexionClientVue. Cette vue utilise le ParticipantService, lequel s'appuie sur le ParticipantDao pour vérifier que :

1. l'adresse e-mail existe ;

2. le mot de passe saisi correspond au hash stocké (bcrypt) ;
3. l'utilisateur n'est pas administrateur (séparation stricte des rôles).

En cas de succès, un objet Session est créé pour stocker le participant connecté. Nous rappelons qu'un utilisateur ne peut pas s'inscrire à un événement sans se connecter. Donc cette étape est obligatoire si une personne souhaite faire une réservation.

3.1.2 Étape 2 : choix d'un événement

Après s'être authentifier, le participant peut désormais choisir un événement parmis tous ceux qui ont été mis en ligne. La vue ReservationVue appelle EvenementService.list_events_with_places() qui retourne la liste des événements ouverts aux inscriptions ainsi que leurs places restantes. Ce calcul repose sur une requête SQL optimisée utilisant un LEFT JOIN sur les réservations existantes.

Le participant sélectionne un événement. S'il n'y a plus de places, l'application refuse immédiatement l'inscription.

3.1.3 Étape 3 : sélection des créneaux de bus

Une fois que l'événement a été choisi, l'application affiche la liste des bus aller associés à l'événement et la liste des bus retour.

Chaque créneau possède :

- une description (horaire, lieu de départ),
- une capacité maximale,
- le nombre de places restantes,
- un identifiant unique.

Le participant choisit indépendamment un bus aller et/ou un bus retour ou aucun.

3.1.4 Étape 4 : options supplémentaires

Pour finaliser sa réservation, l'application invite le participant à renseigner plusieurs informations complémentaires, telles que son adhésion au BDE (oui ou non), son statut éventuel de SAM pour l'événement, ainsi que son intention de consommer ou non des boissons alcoolisées. Il va de soi que, dans le cas où le participant se déclare SAM, la réponse affirmative concernant la consommation d'alcool n'est pas attendue. Ces données additionnelles permettent d'ajuster automatiquement le tarif final et fournissent au BDE des éléments pertinents pour optimiser l'organisation de l'événement.

3.1.5 Étape 5 : création de la réservation

La vue appelle ensuite : ReservationService.create_reservation(...).

Le service effectue les contrôles suivants :

1. L'utilisateur n'a pas déjà une réservation pour cet événement.
2. Les bus choisis ne sont pas complets.
3. Les données sont cohérentes (types, contraintes de capacité).

Une fois la validation effectuée, le service fait appel à la DAO pour enregistrer la réservation dans PostgreSQL.

Nous rappelons qu'il était initialement prévu de générer un code de réservation sécurisé. Toutefois, dans le cadre de notre projet, nous avons fait le choix d'imposer l'authentification systématique de l'utilisateur pour procéder à une réservation. Dans cette perspective, la mise en place d'un code de réservation complexe n'est plus nécessaire, puisque toutes les opérations sensibles — création, modification ou suppression d'une réservation — sont effectuées par un utilisateur dûment identifié. Ce parti pris ne compromet en rien la sécurité des actions réalisées, l'authentification constituant à elle seule une garantie suffisante.

L'objet `ReservationModelOut` est ensuite renvoyé à la couche vue, qui peut l'afficher ou l'exploiter pour mettre à jour l'interface utilisateur.

3.1.6 Étape 6 : envoi du mail de confirmation (Brevo)

Une fois la réservation enregistrée dans la base PostgreSQL, le service déclenche automatiquement l'envoi d'un e-mail de confirmation au participant via l'API Brevo (`api_brevo.py`). Un email est envoyé dans les cas suivants :

- confirmation d'inscription avec un récapitulatif des choix (bus aller, bus retour, adhérent, SAM, boisson)
- lorsqu'un nouvel événement est mis en ligne
- confirmation de modification de réservation ou de compte
- confirmation de suppression de réservation ou de compte
- confirmation d'annulation de réservation

L'envoi de courriels constituait un besoin fortement exprimé par les utilisateurs — en l'occurrence les étudiants de l'Ensai — ce qui en faisait une fonctionnalité incontournable à intégrer au sein de notre application.

3.2 Description sommaire des autres processus de l'application

L'inscription à un événement fait partie des nombreuses fonctionnalités que gère l'application. Nous allons donc en présenter quelques une.

3.2.1 Création d'un événement (admin)

La création d'un événement est réservée aux administrateurs et suit une logique structurée en plusieurs étapes. Après vérification des droits d'accès via la Session, l'administrateur saisit les informations essentielles (titre, date, capacité, statut) via la vue `CreerEvenementVue`. Les données sont validées par le modèle Pydantic `EvenementModelIn` avant insertion. Le `EvenementService` coordonne ensuite deux opérations complémentaires :

- la création de l'événement en base via `EvenementDao`,
- la configuration des transports (bus aller/retour) via `BusService`, si l'administrateur spécifie une capacité supérieure à zéro.

Une fois l'événement créé, le système déclenche automatiquement l'envoi d'un e-mail de notification à tous les participants inscrits (fonctionnalité F08). Le ParticipantService récupère la liste des e-mails, et l'API Brevo effectue l'envoi en boucle. Cette notification automatique garantit que tous les utilisateurs sont informés des nouveaux événements disponibles.

La classe CreerEvenementVue constitue l'interface console permettant à un administrateur de créer un nouvel événement. Elle s'inscrit dans la couche View de l'architecture et se contente de gérer l'interaction avec l'utilisateur, tandis que la logique métier est entièrement déléguée aux services EvenementService et BusService. Lorsqu'elle est appelée, la vue commence par vérifier que l'utilisateur connecté est bien un administrateur; à défaut, l'accès est refusé et l'application renvoie vers l'accueil. En cas d'autorisation, la création de l'événement se déroule sous forme de formulaire interactif géré par InquirerPy, qui permet de saisir les différentes informations nécessaires : titre, adresse, ville, date, description, capacité globale, catégorie et statut de publication. À chaque étape, des règles de validation sont appliquées afin d'éviter des entrées incorrectes, comme un champ vide ou un format de date invalide. Les données collectées sont ensuite encapsulées dans un objet EvenementModelIn, servant de modèle d'entrée pour la couche métier.

Une fois le modèle construit, la vue appelle EvenementService pour tenter la création en base. Ce service se charge de vérifier les contraintes, telles que la validité des données ou la cohérence métier, et enregistre l'événement dans PostgreSQL. En cas d'erreur, un message explicite est affiché à l'utilisateur, et la vue annule le processus pour retourner au menu principal. Si l'événement est correctement créé, la vue affiche un message de confirmation avec l'identifiant de l'événement. À ce stade, une seconde phase s'engage : la configuration des bus associés. L'administrateur peut choisir d'ajouter un bus aller, un bus retour, ou les deux, en indiquant pour chacun la capacité maximale et une description détaillant le lieu et l'horaire de départ. La vue interroge alors le BusService, responsable de créer ces entrées dans la base et de les rattacher à l'événement tout juste créé.

Tout au long du processus, les éventuelles erreurs — telles qu'une capacité invalide ou une mauvaise manipulation — sont capturées par des blocs de gestion d'exceptions qui permettent de préserver la stabilité de l'application et de fournir des retours pertinents à l'administrateur. Si la configuration des bus échoue, la création de l'événement n'est pas annulée, mais un message indique que les transports n'ont pas pu être enregistrés. Une fois toutes les étapes terminées, la vue redirige l'utilisateur vers le menu administrateur accompagné d'un message de succès. Ce fonctionnement illustre la séparation stricte des responsabilités entre les vues, limitées à l'interaction avec l'utilisateur, et les services, responsables de la logique métier, garantissant ainsi une structure claire, modulaire et facilement maintenable.

L'exemple ci-après montre le déroulement du processus de création d'un événement, représenté sous forme de diagramme :

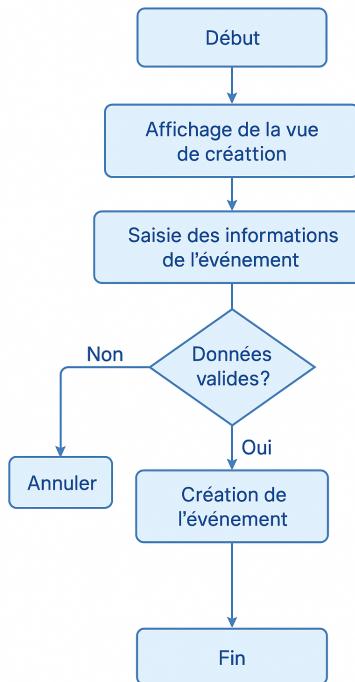


FIGURE 6 – Diagramme du processus de création d'un événement

3.2.2 Modification d'un événement

L'administrateur peut modifier la date, la capacité, ou le statut d'un événement. La modification suit une logique similaire :

1. chargement des valeurs existantes ;
2. saisie et validation ;
3. mise à jour via EvenementDao.update() ;
4. affichage de la nouvelle configuration.

3.2.3 Annulation ou suppression d'un événement (admin)

L'administrateur peut également décider de supprimer un événement. Mais la suppression est contrôlée : l'administrateur ne peut pas supprimer un événement si des réservations sont encore actives. Si aucune réservation n'est active, l'événement en question peut être supprimé via une requête SQL appropriée.

Ci-après, un aperçu du menu de la console en mode terminal accessible à l'administrateur :

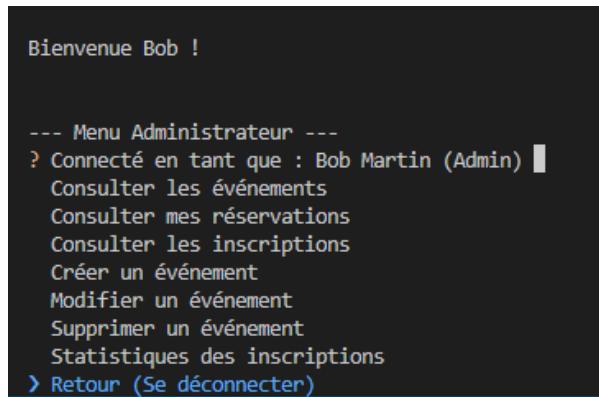


FIGURE 7 – Imprime écran du compte administrateur

3.2.4 Consultation des réservations personnelles

L'utilisateur connecté peut consulter l'ensemble de ses réservations grâce à MesReservationsVue. Cette vue interroge ReservationService qui récupère les réservations avec ReservationDao.find_by_user(). Pour chaque réservation, le système affiche le titre et la date de l'événement (récupérés grâce à EvenementService), les options sélectionnées (bus aller/retour, adhérent, SAM, boisson), ainsi que la date de réservation. L'utilisateur peut également consulter ou modifier les avis qu'il a laissés sur ses événements passés en passant par CommentaireService.

3.2.5 Modification d'une réservation

La modification d'une réservation permet à un utilisateur connecté d'adapter ses choix après son inscription. La vue ModificationReservationVue affiche la liste des réservations de l'utilisateur via ReservationService.get_reservations_by_user(), le titre de l'événement et les options actuelles (bus, adhésion, SAM, boisson). Une fois la réservation sélectionnée, l'utilisateur peut modifier ces options via un formulaire pré-rempli. Le ReservationService met à jour les données en appelant ReservationDao.update_flags(), qui effectue une requête SQL ciblée. Un e-mail de confirmation récapitulant les nouvelles options est ensuite envoyé via l'API Brevo, garantissant ainsi la traçabilité des modifications.

3.2.6 Suppression d'une réservation

La suppression d'une réservation met en œuvre des mécanismes de sécurité pour éviter les annulations accidentelles. Le participant connecté sélectionne une réservation parmi celles listées par ReservationService.get_reservations_by_user(). L'application exige une double confirmation :

1. une validation par bouton (Oui/Non),
2. la saisie du mot "SUPPRIMER" en lettres capitales.

Une fois confirmée, le ReservationDao supprime la réservation en base. Cette opération libère automatiquement les places de bus et déclenche la suppression en cascade des commentaires associés. Enfin, un e-mail de confirmation est envoyé au participant via l'API Brevo, incluant les détails de l'événement. L'envoi est effectué en mode "best-effort" : en cas d'échec, la suppression reste effective.

3.2.7 Visualisation statistique

L’application Shotgun offre aux administrateurs une vue d’ensemble des inscriptions via la StatistiquesInscriptionsVue. Cette fonctionnalité s’appuie sur des requêtes SQL agrégées exécutées par le ConsultationEvenementDao, qui calcule pour chaque événement : le nombre total d’inscrits, le taux d’occupation (pourcentage capacité/inscrits), le nombre de participants SAM et d’adhérents, ainsi que la note moyenne des avis. Les résultats sont affichés sous forme de tableau synthétique incluant l’identifiant, la date, le titre et les statistiques clés de chaque événement. Cette vue permet aux administrateurs de suivre en temps réel la popularité des événements et d’ajuster leur offre en conséquence, s’il le souhaite.

4 Système de stockage

4.1 Rôle du système de stockage

Le système de stockage joue un rôle central dans l'application, car il garantit la cohérence des inscriptions, le respect des capacités des événements et la persistance des données. Ce chapitre présente l'architecture de la base de données PostgreSQL, les choix de modélisation effectués, ainsi que les interactions entre les DAO et la base de données.

L'objectif du stockage est double : assurer la durabilité et la cohérence des données malgré des opérations concurrentes ; et structurer les données de manière à éviter toute redondance ou anomalie.

Pour ce projet, nous nous servons de PostgreSQL car il offre une gestion robuste des contraintes d'intégrité, la prise en charge intégrée des transactions fiables (ACID), des performances élevées pour les requêtes complexes et une bonne compatibilité avec Python via psycopg2. L'ensemble du schéma repose sur une modélisation relationnelle stricte, assurant solidité et extensibilité de l'application.

4.2 Modèle relationnel et justification

Le système repose sur un schéma relationnel structuré autour de quatre tables principales :

- utilisateur : stockage des participants et administrateurs ;
- evenement : description complète d'un événement ;
- bus : créneaux aller/retour associés à un événement ;
- reservation : inscription d'un utilisateur à un événement.

Les relations sont illustrées sur la Figure ??.

Les choix retenus visent à éviter les duplications d'information, limiter les anomalies de mise à jour et assurer la cohérence des données.

Chaque table répond à un rôle précis :

utilisateur : centralisation des données d'identité et des rôles, simplifiant l'authentification.

evenement : gestion complète des informations publiques ainsi que des règles essentielles (capacité positive, statut contrôlé, cohérence lors de la suppression du créateur).

bus : représentation flexible des créneaux de transport, sans duplication des informations liées à l'événement.

reservation : cœur de l'application, garantissant notamment l'unicité d'une réservation par utilisateur et par événement.

Pour garantir cette cohérence, le schéma s'appuie sur plusieurs contraintes d'intégrité critiques. Par exemple, la table reservation implémente une contrainte "UNIQUE(fk_utilisateur, fk_evenement)" qui empêche techniquement un utilisateur de créer deux réservations pour le même événement. Couplée à la gestion transactionnelle, cette contrainte évite les doubles inscriptions même en cas d'accès concurrent. De même, la contrainte "CHECK (capacite > 0)" sur la table evenement refuse toute capacité invalide, tandis que les clauses "ON DELETE

"CASCADE" et "ON DELETE SET NULL" assurent la cohérence référentielle lors de suppressions.

4.3 Logique d'accès aux données (DAO)

L'accès au stockage se fait exclusivement par les DAO (Data Access Objects), afin que les services ne manipulent pas directement SQL. Cette architecture en couches suit un processus standardisé en quatre étapes :

1. Ouverture d'une connexion via un gestionnaire centralisé (DBConnection);
2. Exécution de requêtes paramétrées (protection contre les injections SQL);
3. Gestion automatique des transactions (commit en cas de succès, rollback en cas d'erreur);
4. Conversion des résultats en objets métier (modèles Pydantic).

Cette séparation garantit la sécurité des opérations, la testabilité et la possibilité de changer ultérieurement de type de stockage sans réécrire la logique métier.

Le code suivant illustre la création d'une réservation dans reservation_dao.py :

Listing 1 – Extrait de la méthode `create()` du DAO Reservation

```
def create(self, reservation_in: ReservationModelIn):
    query = """
        INSERT INTO reservation (
            fk_utilisateur, fk_evenement, bus_aller,
            bus_retour, adherent, sam, boisson
        )
        VALUES (
            %(fk_utilisateur)s, %(fk_evenement)s,
            %(bus_aller)s, %(bus_retour)s,
            %(adherent)s, %(sam)s, %(boisson)s
        )
        RETURNING id_reservation, date_reservation
    """
    with DBConnection().getConnexion() as con:
        with con.cursor() as curs:
            try:
                curs.execute(query,
                            reservation_in.model_dump())
                row = curs.fetchone()
                con.commit()
                return ReservationModelOut(
                    **row,
                    **reservation_in.model_dump()
                )
            except Exception as e:
                con.rollback()
                logger.error(f"Erreur: {e}")
                return None
```

On observe ici trois mécanismes clés : l'utilisation de paramètres nommés (% (nom) s) qui empêche les injections SQL, la conversion automatique des objets Pydantic en dictionnaires via `model_dump()`, et la gestion transactionnelle explicite avec `commit()` ou `rollback()`.

selon le résultat de l'opération. Le gestionnaire de contexte (`with`) garantit par ailleurs la fermeture propre de la connexion.

4.4 Transactions et cohérence

Les opérations sensibles utilisent des transactions ACID pour garantir l'intégrité des données. PostgreSQL assure nativement ces propriétés, supervisées par les DAO via `psycopg2` :

- **Atomicité** : une opération est entièrement validée ou annulée (pas d'état intermédiaire) ;
- **Cohérence** : impossibilité de dépasser les capacités ou de laisser des données orphelines ;
- **Isolation** : pas d'interférence entre réservations simultanées ;
- **Durabilité** : la base reste cohérente même en cas d'incident système.

Trois cas d'usage illustrent concrètement ces garanties :

Cas 1 : Création d'événement avec transport

Lors de la création d'un événement (`creer_evenement_vue.py`), la transaction englobe plusieurs opérations :

1. Insertion de l'événement dans la table `evenement`
2. Si `places_aller > 0` : insertion du bus aller dans la table `bus`
3. Si `places_retour > 0` : insertion du bus retour

Si l'une de ces étapes échoue (contrainte violée, erreur de connexion), PostgreSQL effectue un rollback automatique : aucun événement orphelin n'est créé, garantissant l'atomicité de l'opération complète.

Cas 2 : Réservation simultanée de la dernière place

Deux utilisateurs A et B tentent de réserver simultanément la dernière place d'un événement :

1. A démarre sa transaction → PostgreSQL verrouille la ligne `evenement`
2. B tente d'accéder à la même ligne → mise en attente automatique
3. A valide sa réservation → COMMIT et libération du verrou
4. B vérifie la capacité → détection du dépassement → ROLLBACK

La combinaison de l'isolation transactionnelle et de la contrainte "UNIQUE(`fk_utilisateur, fk_evenement`)" empêche toute double inscription, même en cas d'accès concurrent massif.

Cas 3 : Suppression en cascade

Si un événement est supprimé (`supprimer_evenement_vue.py`), les clauses ON DELETE définies dans le schéma assurent la cohérence :

- ON DELETE CASCADE : toutes les réservations associées sont automatiquement supprimées
- ON DELETE SET NULL : les bus voient leur `fk_evenement` passer à NULL

Cette gestion automatique évite les données orphelines et garantit que la base reste dans un état cohérent après chaque suppression.

Notre application repose sur une persistance des données robuste, assurée par un Système de Gestion de Base de Données Relationnelle (SGBDR) PostgreSQL. Le lien entre notre code Python et la base de données est assuré par le driver psycopg2. Cependant, pour ne pas polluer le code métier avec du SQL, nous avons encapsulé toute la logique de persistance dans la couche DAO (Data Access Object). Voici les piliers de notre système de persistance :

1. La Gestion des transactions (Commit / Rollback)

Un point clé de notre apprentissage a été la gestion des transactions. Chaque action de modification (INSERT, UPDATE, DELETE) est atomique.

- Nous ouvrons une transaction via le DAO.
- Si toutes les étapes se passent bien, nous exécutons un `connection.commit()` pour valider les changements de manière permanente.
- Si une erreur survient (ex : contrainte UNIQUE violée, bug SQL), le bloc “try … except” capture l’erreur et exécute automatiquement un `connection.rollback()`. Cela garantit que la base de données ne reste jamais dans un état incohérent ou partiel.

2. Le Pattern singleton pour la connexion

Pour éviter d’ouvrir et fermer des connexions coûteuses à chaque milliseconde, nous avons utilisé une classe DBConnection implémentant le pattern Singleton. Cela nous permet d’avoir un point d’accès unique à la base de données, configuré via des variables d’environnement sécurisées (fichier .env), protégeant ainsi les mots de passe en dehors du code source.

3. Automatisation et tests (ResetDatabase)

La persistance implique aussi la gestion du cycle de vie des données. Nous avons développé un script utilitaire `reset_database.py` capable de détruire et reconstruire le schéma complet (tables, contraintes) et d’injecter un jeu de données initial (`pop_db.sql`). Ce système nous a permis de travailler sur une base "propre" avant chaque session de test unitaire, garantissant la reproductibilité de nos résultats.

5 Outils et démarches de développement

5.1 Organisation du groupe

Afin d'assurer une organisation claire et efficace, nous avons réparti les responsabilités du projet de la manière suivante : Yann en tant que chef de projet métier, Mohamed en tant que chef de projet technique, une équipe de développeurs (toute l'équipe projet), Khalid en tant que responsable recette et Jade en tant que responsable rédaction.

Chef de projet métier (MOA) :

Le chef de projet métier, également appelé MOA (Maîtrise d’Ouvrage), occupe une place centrale dans la conduite du projet. Il est le garant de la vision fonctionnelle et veille à ce que la solution développée réponde aux besoins exprimés par les utilisateurs finaux.

À ce titre, il assume plusieurs responsabilités principales :

- Définition des besoins métier : il recueille, formalise et valide les exigences auprès des parties prenantes.
- Priorisation des fonctionnalités : il arbitre les choix en fonction des objectifs stratégiques et des contraintes du projet.
- Interface entre métiers et technique : il assure la communication et la compréhension mutuelle entre les utilisateurs et l'équipe de développement, facilitant ainsi la traduction des besoins en solutions techniques.

Le chef de projet métier garantit que le produit final sera aligné sur les attentes fonctionnelles et apportera une valeur ajoutée réelle aux utilisateurs.

Chef de projet technique (MOE) :

Le chef de projet technique, ou MOE (Maîtrise d’Œuvre), est responsable de la dimension technique du projet. Il conçoit et supervise la mise en œuvre de la solution, tout en assurant la qualité et la robustesse de l'architecture logicielle. Ses principales missions sont :

- Conception technique : il définit l'architecture et les choix technologiques.
- Encadrement de l'équipe technique : il coordonne le travail des développeurs, attribue les tâches et suit leur avancement.
- Garantie de la qualité : il veille au respect des bonnes pratiques de développement, des standards de sécurité et des délais de livraison.

Ainsi, le chef de projet technique joue un rôle essentiel pour transformer les besoins fonctionnels en solutions informatiques fiables et performantes.

Responsable recette :

Le responsable recette a pour mission d'assurer que la solution livrée est conforme aux attentes et de garantir sa qualité avant sa mise en production. Ses principales tâches sont :

- Élaboration des plans de test : définir les scénarios et jeux de données nécessaires pour vérifier les fonctionnalités.

- Exécution et suivi : réaliser les tests fonctionnels, détecter les anomalies et assurer leur suivi jusqu'à résolution.
- Validation finale : donner son accord pour la mise en production une fois que le produit répond aux exigences métier.

Ce rôle est essentiel pour sécuriser la phase de livraison et renforcer la confiance des utilisateurs dans le produit.

Développeurs :

Les développeurs constituent le cœur opérationnel de l'équipe projet. Leur rôle est de traduire les spécifications fonctionnelles et techniques en un logiciel concret et utilisable.

Leurs principales responsabilités sont :

- Implémentation : coder les fonctionnalités conformément aux spécifications définies.
- Tests unitaires : vérifier la robustesse et la cohérence de leur code avant l'intégration.
- Maintenance et évolution : corriger les anomalies et proposer des améliorations continues du produit.

Ils travaillent en étroite collaboration avec le chef de projet technique et participent activement à la réussite du projet.

Responsable rédaction :

Le responsable rédaction est chargé de la production et de la mise à jour de la documentation du projet.

Son travail contribue à assurer la compréhension et la pérennité de la solution. Ses missions sont :

- Rédaction de la documentation technique : décrire l'architecture, les composants et les procédures pour l'équipe informatique.
- Rédaction des guides utilisateurs : produire des supports clairs et accessibles pour faciliter la prise en main du logiciel.
- Mise à jour continue : maintenir une documentation cohérente et actualisée tout au long du cycle de vie du projet.

Il garantit ainsi la transmission des connaissances et la bonne appropriation de la solution par ses différents utilisateurs.

5.2 Outils utilisés

Dans un premier temps, nous avons établi notre planning de travail ainsi que les différentes échéances du projet. Pour cela, nous avons utilisé le logiciel Libre Project afin de concevoir notre diagramme de Gantt, qui nous a permis de visualiser clairement les tâches à réaliser ainsi

que les délais associés.

Nous avons ensuite réalisé l'ensemble des diagrammes d'analyse et de conception, notamment les diagrammes de classes, de bases de données et de cas d'utilisation. Leur création a été effectuée avec l'outil en ligne draw.io (également connu sous le nom de diagrams.net), particulièrement adapté pour la modélisation visuelle.

Dans le cadre du développement et de la gestion des bases de données, nous avons travaillé sur la plateforme Onyxia, qui met à disposition plusieurs outils de programmation. Nous avons ainsi utilisé :

- Visual Studio Code pour le développement en Python,
- PostgreSQL pour le stockage et la gestion des données,
- CloudBeaver pour une visualisation et une interaction simplifiées avec les bases de données.

Pour la gestion de versions, nous avons utilisé la plateforme GitHub, qui a facilité la collaboration au sein du groupe. Elle nous a permis de versionner efficacement le code, de fusionner nos contributions respectives et de conserver un historique détaillé des modifications tout au long du projet.

Enfin, afin d'assurer une communication continue et efficace, nous avons créé un groupe WhatsApp dès la première séance en septembre. Ce groupe nous a servi à partager nos avancées, signaler nos difficultés, coordonner le travail à venir et même organiser des réunions lorsque cela était nécessaire.

5.3 Démarche qualité

Dans un projet informatique de cette nature, il est essentiel de garantir en permanence le bon fonctionnement de l'ensemble des composants. L'étape de test constitue donc une phase incontournable du développement. Tout au long du projet, nous avons mené une série de tests manuels et scénarisés, comprenant des tests unitaires sur les principales méthodes DAO, des tests d'intégration simulant le comportement d'un utilisateur réel dans la console, des tests de charge simples visant à contrôler la gestion des capacités, ainsi que des tests d'exceptions permettant de vérifier la robustesse du système face aux erreurs.

Nous avons également réalisé des tests d'envoi d'e-mails à l'aide d'un compte Brevo dédié. Bien que ces tests n'aient pas été automatisés via Pytest, ils ont été conçus de manière rigoureuse et structurée, afin de cibler les zones critiques de l'application, telles que la création et la gestion des événements, la vérification des contraintes SQL ou encore la suppression des réservations. Leur réalisation progressive au fil du développement a permis d'assurer la stabilité et la fiabilité du système dans son ensemble.

5.3.1 Cas particuliers testés

Les tests se sont concentrés sur les zones à risque :

- double réservation d'un même utilisateur ;
- dépassement de capacité d'un bus ;
- tentative de suppression d'un événement avec réservations ;
- gestion d'un email déjà utilisé ;
- envoi d'emails avec connexion API incorrecte ;
- déconnexions accidentielles pendant l'inscription.

L'application a été conçue afin de garantir une réaction correcte, maîtrisée et adaptée à chaque scénario potentiel :

- message d'erreur explicite ;
- rollback automatique lors d'un problème SQL ;
- aucune donnée incohérente en base.

5.3.2 Phase de recette

La phase de recette représente un moment clé du projet Shotgun ENSAI. Elle a pour objectif de vérifier que l'application développée répond correctement aux besoins exprimés par le BDE, qu'elle couvre l'ensemble du périmètre fonctionnel défini dans le cahier des charges et qu'elle garantit un fonctionnement fiable, stable et cohérent pour les utilisateurs finaux. Cette étape s'appuie directement sur les documents préparés au préalable par l'équipe projet, notamment le plan de recette, le plan de tests et le suivi des anomalies.

La recette consiste en premier lieu à vérifier la conformité fonctionnelle de l'application. Le Plan de recette précise le périmètre à valider, en distinguant les fonctionnalités de base (F1 à F8), qui incluent par exemple la création d'un événement, la gestion des réservations ou encore l'envoi automatique de mails, et les fonctionnalités optionnelles (FO1 à FO8), comme l'affichage de statistiques ou l'envoi de rappels. L'objectif est de garantir que toutes les exigences obligatoires sont correctement prises en charge et que les fonctionnalités optionnelles produisent le comportement attendu lorsqu'elles ont été implémentées.

Les différentes phases de la recette sont présentées ci-après :

Chronologie de la phase de recette

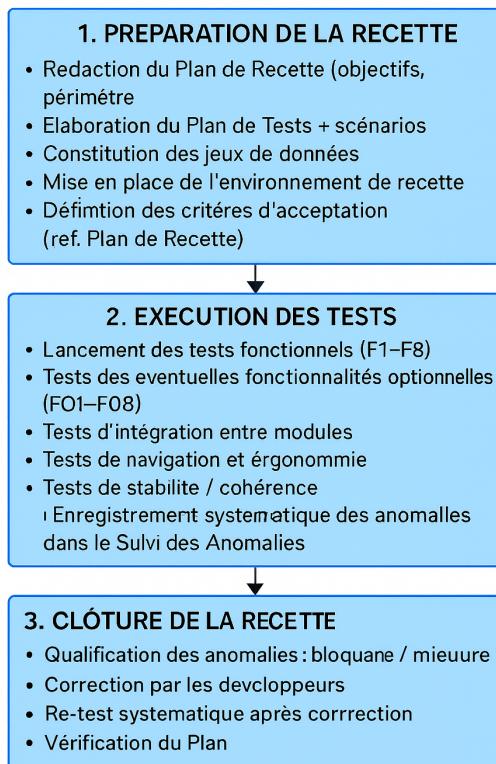


FIGURE 8 – Diagramme de la phase de recette

L'exécution des tests constitue le cœur de la phase de recette. Les cas de tests définis dans le plan de tests sont exécutés de manière systématique afin de contrôler le bon fonctionnement de chaque module du système. Ces tests couvrent les aspects fonctionnels, l'intégration entre les différentes couches applicatives, la navigation au sein de l'interface console ainsi que la stabilité générale du système. Durant cette étape, chaque test est associé à des jeux de données permettant de reproduire des scénarios réalistes. Toute anomalie détectée est immédiatement consignée dans le fichier de suivi, où elle est catégorisée selon sa criticité (bloquante, majeure ou mineure), puis transmise à l'équipe de développement pour correction.

Le suivi des anomalies joue un rôle essentiel dans la validation de la recette. Les anomalies sont analysées, corrigées et vérifiées à nouveau jusqu'à ce que le système atteigne l'état attendu. Selon le plan de recette, la recette n'est déclarée valide que lorsque toutes les anomalies bloquantes ont été corrigées et que le taux d'échec des tests non critiques reste inférieur à un seuil défini (moins de 5% d'échecs permis). Ce cycle itératif test → anomalie → correction → re-test permet d'assurer la qualité et la robustesse du produit final.

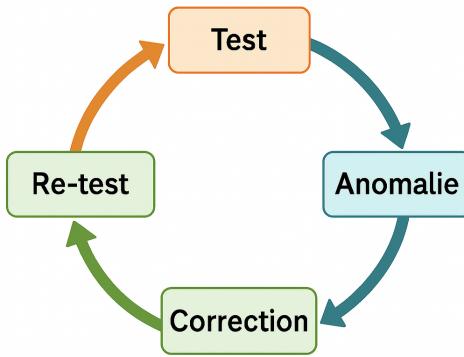


FIGURE 9 – Cycle de tests

Une fois l'ensemble des tests validés et les anomalies critiques corrigées, une revue de recette est organisée en présence du responsable de recette qui assure le rôle de validatrice. Cette étape permet de vérifier que les critères d'acceptation sont satisfaits et que l'application respecte bien les attentes exprimées initialement. À l'issue de cette revue, le Procès-Verbal de recette est signé, marquant la validation officielle de la phase de recette et autorisant la mise en production de l'application.

Enfin, la phase de recette se clôture par l'archivage des documents associés, la mise à jour de la documentation technique ainsi que la réalisation d'un retour d'expérience. Cette dernière étape permet de capitaliser sur les bonnes pratiques, d'identifier les points d'amélioration et de préparer sereinement la livraison finale de l'application.

6 Performance et temps de traitement

6.1 Mesure de la performance

Les performances ont été mesurées empiriquement au cours du développement :

- la création d'un événement : ~ 30 ms ;
- l'inscription d'un utilisateur : ~ 40 ms ;
- l'affichage de la liste des événements : $\sim 10\text{--}15$ ms ;
- l'envoi d'un email Brevo : $\sim 250\text{--}600$ ms (limité par le réseau et l'API externe).

Les opérations les plus coûteuses, comme l'envoi de courriels, sont entièrement externalisées. L'ensemble des autres traitements est exécuté localement et s'effectue de manière quasi instantanée.

L'utilisation d'index sur les clés étrangères (`fk_utilisateur`, `fk_evenement`) a fortement contribué à maintenir d'excellentes performances.

6.2 Robustesse face aux changements externes

De plus, plusieurs évolutions futures ont été rendues possibles grâce à la structure modulaire de l'application. Il serait ainsi très simple de remplacer l'API Brevo par n'importe quel autre service d'envoi d'e-mails, sans modifier la logique métier. L'architecture permet également une migration aisée vers une base de données SQL distante, hébergée en ligne. Par ailleurs, l'ajout d'une couche front-end web pourrait être réalisé sans la moindre modification du backend existant. Enfin, de nouvelles fonctionnalités telles que l'intégration de statistiques avancées ou la génération de QR codes pourraient être ajoutées ultérieurement, sans nécessiter de modifications du schéma principal.

Les données sont centralisées au sein d'une base PostgreSQL entièrement normalisée, ce qui facilite les évolutions futures tout en garantissant une cohérence durable de l'ensemble du système.

Cette section met en évidence que le projet ne se limite pas à une simple réalisation fonctionnelle, mais qu'il s'inscrit dans une démarche de conception rigoureuse conduisant à une application structurée, robuste, éprouvée et conçue pour évoluer. L'ensemble des outils de développement mobilisés, l'organisation méthodique du travail ainsi que la démarche d'assurance qualité mise en œuvre ont permis de garantir un système fiable et cohérent, répondant de manière satisfaisante aux exigences formulées par le BDE.

7 Conclusion

Le projet Shotgun a abouti à une application fonctionnelle répondant aux besoins exprimés, intégrant les fonctionnalités optionnelles ainsi que d'autres fonctionnalités issues de notre propre initiative. L'application met à disposition l'ensemble des fonctionnalités essentielles attendues : création et gestion d'événements par les administrateurs, système d'inscription avec gestion automatique des capacités, envoi d'e-mails de confirmation via l'API Brevo, et suivi en temps réel des inscrits. Ces fonctionnalités de base ont été complétées avec l'intégration de certaines fonctionnalités facultatives comme la modification des réservations, l'affichage de statistiques , l'envoi d'un mail d'alerte à tous les clients lors de la mise en ligne d'un nouvel événement, ainsi qu'une interface console intuitive développée avec InquirerPy.

Le système de gestion des comptes, développé de notre propre initiative, permet à chaque utilisateur de modifier ses informations personnelles ou de supprimer son profil en toute autonomie et en toute sécurité. La gestion centralisée des réservations et de la publication des événements facilitent les modifications pour les participants comme pour les administrateurs. Le système d'avis et de commentaires complète l'expérience utilisateur en permettant un retour constructif sur les événements passés. Enfin, la fonction de recherche avancée avec des filtres (choix par ville, catégorie ou date) améliore la navigation et l'accessibilité aux événements souhaités.

L'application repose sur une architecture multi-couche rigoureuse qui garantit une séparation claire des responsabilités et facilite grandement la maintenance et l'évolution du code. L'utilisation de modèles Pydantic pour la validation des données assure la cohérence et la robustesse du système. Grâce à PostgreSQL, les transactions sont correctement prises en charge et les données demeurent cohérentes, même en cas d'accès simultanés.

La sécurité étant un point clé du projet, tous les mots de passe sont protégés par un hashage systématique avec bcrypt, l'authentification est obligatoire pour toutes les opérations sensibles, et les requêtes SQL paramétrées offrent une protection efficace contre les injections. Le système de notifications automatiques par e-mail, basé sur l'API Brevo, couvre l'ensemble des opérations essentielles afin d'informer le participant de chaque action effectuée sur son compte ou sur ses réservations.

En somme, l'application Shotgun apporte une réponse complète à la problématique initiale : centraliser et gérer efficacement les événements ainsi que les réservations associées. Elle offre une interface intuitive facilitant la prise en main par les utilisateurs, tout en garantissant un haut niveau de fiabilité et de sécurité pour l'ensemble des données et des opérations. Grâce à son architecture structurée et à ses fonctionnalités avancées, elle constitue un outil performant et adapté aux besoins du projet.

8 Notes Individuelles

8.1 Yann Ahuie

En tant que chef de projet, je tiens tout d'abord à exprimer ma profonde gratitude envers les membres de l'équipe avec lesquels j'ai eu l'honneur de mener ce projet à bien : Jade, Khalid et Mohamed. Leur engagement, leur sens du professionnalisme et leur rigueur ont été déterminants. Entre phases d'apprentissage, explications mutuelles et autonomie opérationnelle, chacun a su contribuer de manière exemplaire à l'accomplissement de notre objectif commun. Le respect des échéances, la qualité du travail fourni et l'esprit de collaboration dont ils ont fait preuve m'ont particulièrement marqué. Ce projet a posé une base fondamentale dans mon apprentissage continu, au contact de mes collaborateurs. Partant sans aucune notion préalable en Git, en DAO, ou encore dans certaines technologies spécifiques, j'ai pu consolider et mettre en pratique les connaissances théoriques acquises en cours. Pour les futurs projets, un temps plus long consacré à la prise en main des outils front-end serait une amélioration bénéfique afin d'optimiser et embellir encore davantage notre application.

Mon rôle dans le projet

Ma mission principale consistait à concevoir et implémenter la base de données, élément crucial pour la structure et la cohérence de l'application. Après avoir élaboré collectivement le modèle conceptuel de données, j'ai assuré la réalisation du modèle logique, puis son implantation technique.

Par ailleurs, j'ai utilisé Git afin de documenter et partager mes modifications, notamment dans la rédaction du README et la gestion d'anomalies. Ma participation aux tests utilisateurs a également été essentielle pour permettre à l'équipe de développer d'ajuster certaines fonctionnalités et corriger des erreurs d'usage identifiées. **Leadership et coordination**

Dans mon rôle de chef de projet, j'ai assuré une coordination fluide entre les membres de l'équipe, en communiquant via WhatsApp sur les échéances, les objectifs, les tâches à réaliser et les difficultés rencontrées. Cet échange constant nous a permis, collectivement, de trouver rapidement des solutions aux problèmes rencontrés. Cette démarche collaborative a largement contribué à l'atteinte de nos objectifs.

Difficultés rencontrées

La principale difficulté est survenue en cours de projet, lorsque nous avons constaté que notre modèle initial de données pour la gestion des transports n'était plus adapté. Cela faisait suite à des ajustements structurels de ma part lors de l'implémentation, notamment au niveau de la table reservation et des cardinalités, afin de respecter au mieux les formes normales. Ces modifications ont nécessité une refonte partielle de la base. Grâce au calme et au soutien de mes collaborateurs — et particulièrement l'aide de Khalid, chargé de la DAO. Nous avons pu repenser ensemble la structure et procéder à une révision efficace du modèle. Cette séance de travail a été extrêmement enrichissante, tant sur le plan technique que méthodologique.

Conclusion

En somme, ce projet m'a permis de comprendre concrètement les étapes d'un projet informatique, de la conception jusqu'à la phase de recette. J'ai pu manipuler des outils essentiels tels que Git, PostgreSQL, ainsi que renforcer ma compréhension des scripts Python.

Mon leadership s'en est trouvé renforcé, grâce à la collaboration avec une équipe formidable et très professionnelle. Ce projet restera pour moi une expérience formatrice, structurante et particulièrement motivante pour la suite de mon parcours.

8.2 Mohamed Douzi

Je tiens tout d'abord à remercier Jade, Khalid et Yann pour ce projet très enrichissant que nous avons réalisé ensemble. J'ai beaucoup apprécié notre collaboration, où chacun a su apporter sa contribution essentielle. Yann, notre chef de projet, a parfaitement tenu son rôle en veillant au respect des échéances. L'expérience et le calme de Khalid m'ont particulièrement marqué, tout comme la rigueur de Jade, avec qui j'ai collaboré étroitement lors de l'analyse fonctionnelle.

Cette phase a posé les fondations de l'application : j'ai enfin compris l'importance cruciale de l'analyse fonctionnelle, qui permet de modéliser nos données, de fixer des objectifs clairs et de répartir les rôles.

Mon rôle s'est principalement axé sur le développement pur ("code"). Je devais à la fois identifier et recenser les anomalies avec l'aide de Yann, puis les corriger avec Khalid. Le projet a fini par me passionner, ce qui m'a poussé à aller plus loin en ajoutant de nombreuses fonctionnalités optionnelles. Travailler à quatre sur les mêmes fichiers a parfois engendré des conflits de versions. J'ai ainsi appris qu'une bonne communication avant de modifier un fichier critique est tout aussi importante que le code lui-même.

Mon apport principal a été ma capacité à naviguer à travers toutes les couches de l'application. J'ai acquis une compréhension claire du cheminement de l'information :

- Comment la donnée est structurée dans PostgreSQL (le stockage).
- Comment le DAO l'extrait et la transforme en objets Python.
- Comment le Service applique les règles de gestion (vérifications, calculs).
- Et enfin, comment la Vue présente le résultat final à l'utilisateur.

Le moment le plus marquant de mon apprentissage a été lorsque nous avons réalisé, en cours de projet, que notre modèle de données initial concernant les transports n'était pas adapté. Nous avons dû modifier la structure même de la base de données. Ce projet "Shotgun ENSAI" m'a permis de dépasser le stade de la simple écriture de scripts Python pour entrer dans une véritable logique de développement logiciel. Je ne vois plus l'application comme une somme de fichiers isolés, mais comme un système interconnecté où chaque décision prise au niveau de la base de données a des répercussions directes sur l'expérience utilisateur finale. C'est cette compréhension globale du cycle de vie de la donnée qui restera mon acquis le plus précieux.

8.3 Jade Jauvert

Le projet a été enrichissant mais ce n'était pas évident. J'ai beaucoup appris en travaillant en groupe, plus que si j'avais été seule. Et les personnes avec lesquelles j'ai travaillé se sont révélées particulièrement compétentes et agréables. Par exemple, Khalid est le plus expérimenté de nous quatre, lorsque je ne comprenais pas ce que l'on faisait, je me dirigeais souvent vers lui pour poser mes questions. Et de manière générale, on s'est tous entraînés lorsqu'on avait

un souci. Je suis contente d'avoir pu participer à un projet comme celui-ci avec Khlaid, Yann et Mohamed, d'autant plus que le sujet a été très bien choisi !

Durant ce projet, j'ai pu faire un peu de tout : du code avec les tests et quelques fonctions dans certaines classes business, de la documentation et de la relecture du code mais aussi de la création de diagrammes de classes et de bases de données. La principale tâche que j'avais été portée sur la rédaction, avec le dossier d'analyse, les suivis, la documentation du code et le rapport final.

En ce qui concerne mes difficultés : j'ai été limité par mes compétences informatiques, ce qui fait que j'avais l'impression de prendre beaucoup de temps pour faire une seule petite tâche. L'informatique et notamment l'utilisation de python ne sont pas mes points forts, bien que je fasse de mon mieux. Les premières semaines, la communication avec tous n'a pas été simple mais ce problème s'est très vite réglé au cours du projet. Et enfin, prévoir un planning sur 3 mois pour un projet d'application et s'y tenir n'étaient pas facile.

Si un tel projet était à refaire :

- Il faudrait améliorer la répartition des tâches en termes de quantité horaire et que chacun puisse faire un peu de tout si possible.
- Prévoir les réunions à l'avance et pas deux jours avant.

8.4 Khalid Jerrari

Au cours de ce projet, j'ai participé activement à la conception, au développement et à la coordination des tâches au sein de notre équipe. J'ai travaillé sur plusieurs couches de l'application : la couche métier, la couche DAO, la couche service ainsi que la couche présentation. J'ai également mis en place et animé la phase de recette, cruciale pour valider la conformité fonctionnelle et technique du produit.

Mon rôle et ma contribution

Mon rôle principal a été celui de support transversal, un véritable « électron libre » capable d'épauler chaque membre de l'équipe en fonction des difficultés rencontrées. J'ai ainsi participé à la conception, au développement, aux tests, et parfois joué un rôle de chef de projet AMOA ou technique lorsque la situation l'exigeait.

J'ai régulièrement été sollicité pour trouver des solutions aux problèmes techniques, logiques ou méthodologiques rencontrés par le groupe. J'ai accompagné mes camarades sur la conception de la base de données, sur la résolution d'erreurs applicatives et sur les choix d'architecture.

Mon objectif allait au-delà du simple développement : j'ai souhaité transmettre l'expérience acquise dans ma carrière professionnelle, qu'elle soit liée au développement Java, au pilotage de projets ou à la gestion d'entreprise.

Vie d'équipe et organisation

Le travail d'équipe s'est très bien déroulé. Nous avons fonctionné avec une organisation inspirée de Scrum, en réalisant des réunions courtes et régulières pour suivre l'avancement. Notre communication via WhatsApp a été un véritable atout.

J'ai pris beaucoup de plaisir à travailler avec Jade, Mohammed et Yann, qui se sont énormément investis et ont fourni un travail considérable pour atteindre nos objectifs.

Une difficulté importante est apparue lorsque nous avons découvert vers la fin du projet une erreur d'implémentation dans la structure initiale de la base de données. Cette tâche ayant été confiée à une seule personne sans contrôle croisé, l'erreur s'est ensuite répercutée sur les classes métiers, les DAO et les services. Cela nous a fait perdre du temps et aurait pu être évité grâce à une validation collective plus rigoureuse.

Enseignements et retours d'expérience

Ce projet m'a permis de renforcer ma capacité à travailler sur un code collaboratif, à formaliser mes choix techniques et à anticiper la cohérence entre les différentes couches d'une application.

Si je devais recommencer, je :

- contrôlerais davantage la qualité du développement au fil de l'avancement ;
- testerais la base de données et les requêtes avant d'implémenter la logique applicative ;
- mettrais en place des jeux de tests SQL dès le début du projet.

En résumé

Ce projet a été une expérience à la fois stimulante, enrichissante et formatrice. Il m'a permis de consolider mes compétences techniques et organisationnelles, mais aussi d'expérimenter pleinement la dynamique d'un travail d'équipe sur un projet applicatif complet. J'en retiens que la réussite d'un projet repose autant sur la qualité du code que sur la qualité de la communication et de la collaboration au sein du groupe.

9 Annexes

Fonctionnalités de l'application

Fonctionnalités de base

- F1 : Création d'un compte admin
- F2 : Création d'un événement par un admin (titre, créneaux de bus aller/retour avec capacité)
- F3 : Inscription d'un participant à un événement
- F4 : Lister tous les événements auxquels le client peut s'inscrire
- F5 : Génération d'un code de réservation unique et envoi d'un mail de confirmation
- F6 : Pour les admin, avoir la liste des inscrits en temps réel à l'évènement
- F7 : Suppression d'une réservation à partir du code de réservation
- F8 : Gestion des capacités maximales des créneaux de bus et des évènements

Fonctionnalités optionnelles

- FO1 : Création facultative de compte client pour pré-remplissage à la prochaine inscription
- FO2 : Modification d'une réservation à partir du code de réservation
- FO3 : Frontend simple pour les admins/participants
- FO5 : Afficher des statistiques sur le nombre d'inscrits
- FO6 : Envoyer des rappels par mail pour les paiements
- FO7 : Déployer l'application
- FO8 : Envoyer un mail d'alerte à tous les clients lorsqu'un nouvel événement est créé

Diagramme physique des données

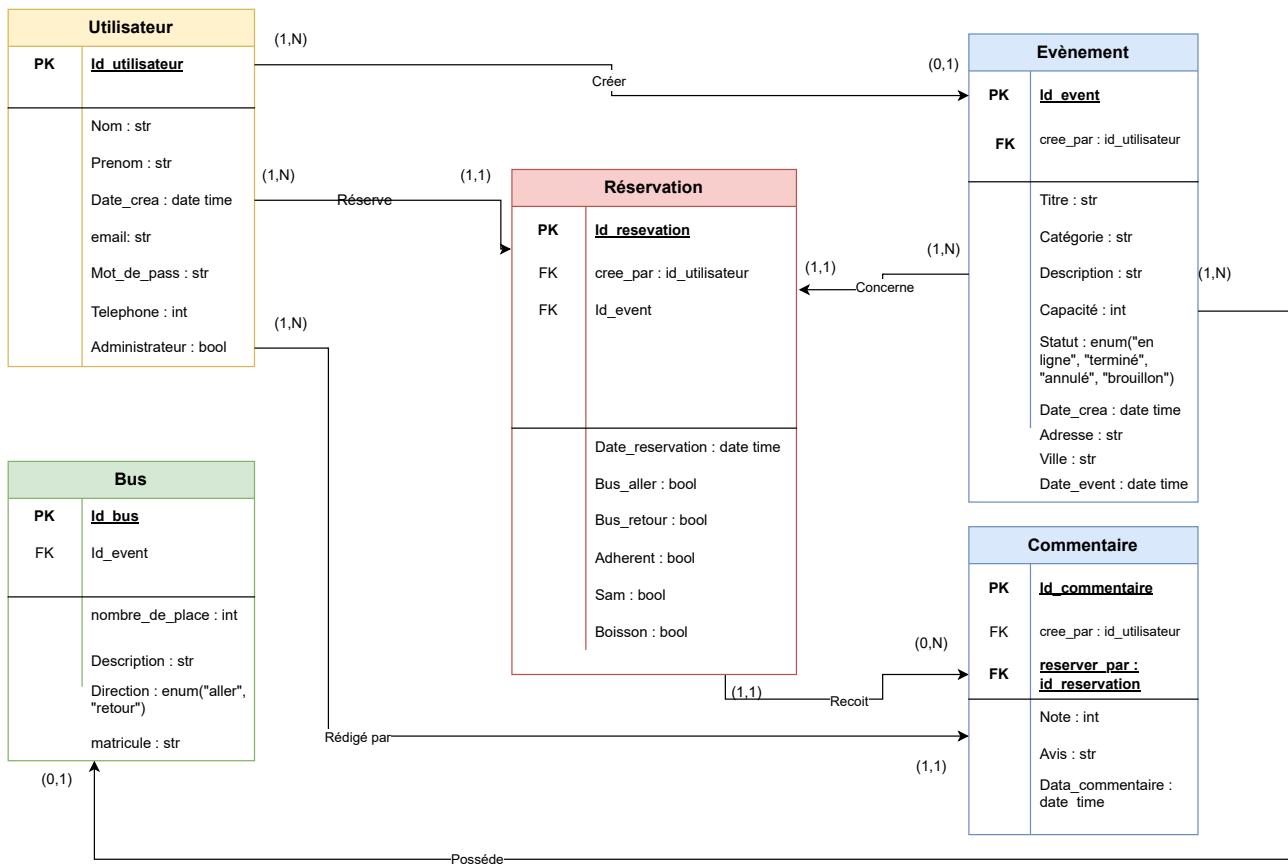


FIGURE 10 – Diagramme physique des données

Au cours du développement, nous avons ajouté une table Commentaire, qui permet aux utilisateurs de laisser un avis sur un événement auquel ils se sont inscrits.