

**CYRIAC DESCUBES DU CHATENET**

21/09/2023



**Travel tailor**

**DOSSIER PROJET**

TITRE RNCP N°31678

# Sommaire

## Introduction

- Parcours 3

## Naissance du projet

- Concept du projet 4
- Schéma de la base de donnée 5

## Ux - Ui

### 1. Ux

- Cible 6
- SEO 7
- User-flows 8
- Wireframes 9

### 2. Ui

- Inspirations 12
- Maquette 14

## Développement

- Workflow 16
- Architecture du projet 18
- Stack Technique
  - NextJS 20
  - Tailwind CSS 21
  - NestJS 22
  - PostgreSQL 23
- Focus Technique
  - Gestion des utilisateurs 24
  - Authentification 28
  - Rôles 31
  - Sécurité 33
  - Documentation 34
  - Création du planning des voyages 36
  - Paiement 37
  - Upload de fichiers 39
  - Mails 41

• Tests	
• Jest	44
<b>Déploiement</b>	
• Frontend	46
• Backend	46
• Base de donnée	47
<b>Evolutions</b>	
• Frontend	48
• Backend	48
• Mobile	48
• Globales	48
<b>Conclusion</b>	
• Ce que ce projet m'a apporté	49

# Parcours

Je m'appelle Cyriac Descubes du Chatenet, j'ai 20 ans, je suis développeur Fullstack React NestJS depuis plus d'un an.

Passionné par le digital depuis tout jeune, j'ai toujours su que je voulais faire carrière dans ce secteur, mais je ne savais pas exactement dans quel domaine.

Issu d'un Baccalauréat STMG spécialité Marketing, j'ai eu la chance de pouvoir créer une mini-entreprise lors de mon année de première.

Le but de cette mini-entreprise était de livrer des boissons pour les soirées grâce à une application nommée "Safe Party 33".

Pendant cette année de première, j'ai eu la chance de faire un stage d'immersion d'une semaine à l'Ecole Supérieure du Digital, où j'ai pu découvrir l'UX-UI mais aussi d'être dans ce domaine. Grâce à l'UX-UI, j'ai pu prototyper une application dans le domaine du sport pour recevoir les scores des matchs.

Avec ces connaissances fraîchement acquises, j'ai pu prototyper l'application mobile de "Safe Party 33".

En mai 2019, nous avons participé au concours régional "Entreprendre pour apprendre", nous avons fini à la deuxième position de tous les projets de la région Nouvelle Aquitaine.

Mon Baccalauréat en poche, je débarque à l'ESD pour effectuer un Bachelor Chef de projet digital. Un Bachelor avec un programme très large contenant de la photographie, vidéo, Ux-Ui, développement web, marketing, philosophie...

C'est au début de ma première année que je me suis découvert une passion pour le développement web. Dans cette matière, nous devions réaliser des intégrations de maquettes en utilisant les langages HTML – CSS et JavaScript.

Dès décembre 2020, j'ai commencé à apprendre le développement web en autodidacte, c'est avec ceci que j'ai construit toute ma stack. C'est lors de cette année que j'ai pu être accompagné d'un étudiant passionné par le développement web qui a pu m'aider à construire ma première roadmap.

Grâce à cette motivation et cette envie, j'ai pu majorer le développement web pendant les deux premières années de mon bachelor.

En septembre 2022, j'arrive en alternance chez Dotmind pour améliorer mes compétences en React et Node.js. À partir d'octobre, je commence mon Bachelor 3 développeur web, où j'ai eu notamment pour mission de coder mon projet fil rouge qui est un gestionnaire de voyage automatisé nommé "Travel Tailor".

# Concept du projet

Travel Tailor est un projet qui a pour but de faciliter la manière dont les gens vont préparer leurs voyages.

À travers une plateforme simple et intuitive, il sera possible de pouvoir concevoir le planning de son voyage en fonction de ses goûts afin d'avoir un voyage personnalisé et une expérience unique.

Sachant que la conception d'un voyage peut être longue et complexe, le but ici sera de passer outre la phase de recherche en fonction de ses goûts, des avis, des prix...

Tous ces éléments seront pris en compte et son algorithme vous permettra de concevoir votre planning personnalisé en un claquement de doigt. L'algorithme ne va générer une proposition de séjour. Contenant différentes activités L'algorithme va pouvoir exclure les activités qui seront fermées pendant le séjour de l'utilisateur.

La plateforme sera divisée en trois parties.

Il y aura une partie voyageur où l'utilisateur pourra créer son compte sur la plateforme et renseigner ses goûts qui seront modifiables par la suite. Il va ensuite créer son voyage grâce à un formulaire dans lequel il devra renseigner sa ville de départ, sa ville d'arrivée, ainsi que sa date de départ et d'arrivée. C'est grâce à toutes ces informations que l'algorithme va générer le planning.

L'utilisateur aura ensuite la possibilité de visualiser le planning ainsi que les activités proposées afin de le valider ou de modifier le planning si besoin.

Il y aura une partie annonceur accessible via un abonnement mensuel, où l'utilisateur pourra créer son compte et renseigner les informations de son agence comme son nom et sa localisation. Après cela, l'annonceur pourra créer, modifier et supprimer ces activités. Chaque activité aura une note ainsi qu'une section commentaire pour que l'utilisateur puisse noter et donner son avis sur l'activité.

C'est grâce à cette partie annonceur et aux activités générées que l'application aura un lac de donnée dans lequel l'algorithme va aller sélectionner les meilleures activités pour le séjour de l'utilisateur.

L'annonceur aura aussi accès à un espace pour consulter les factures liées à son abonnement.

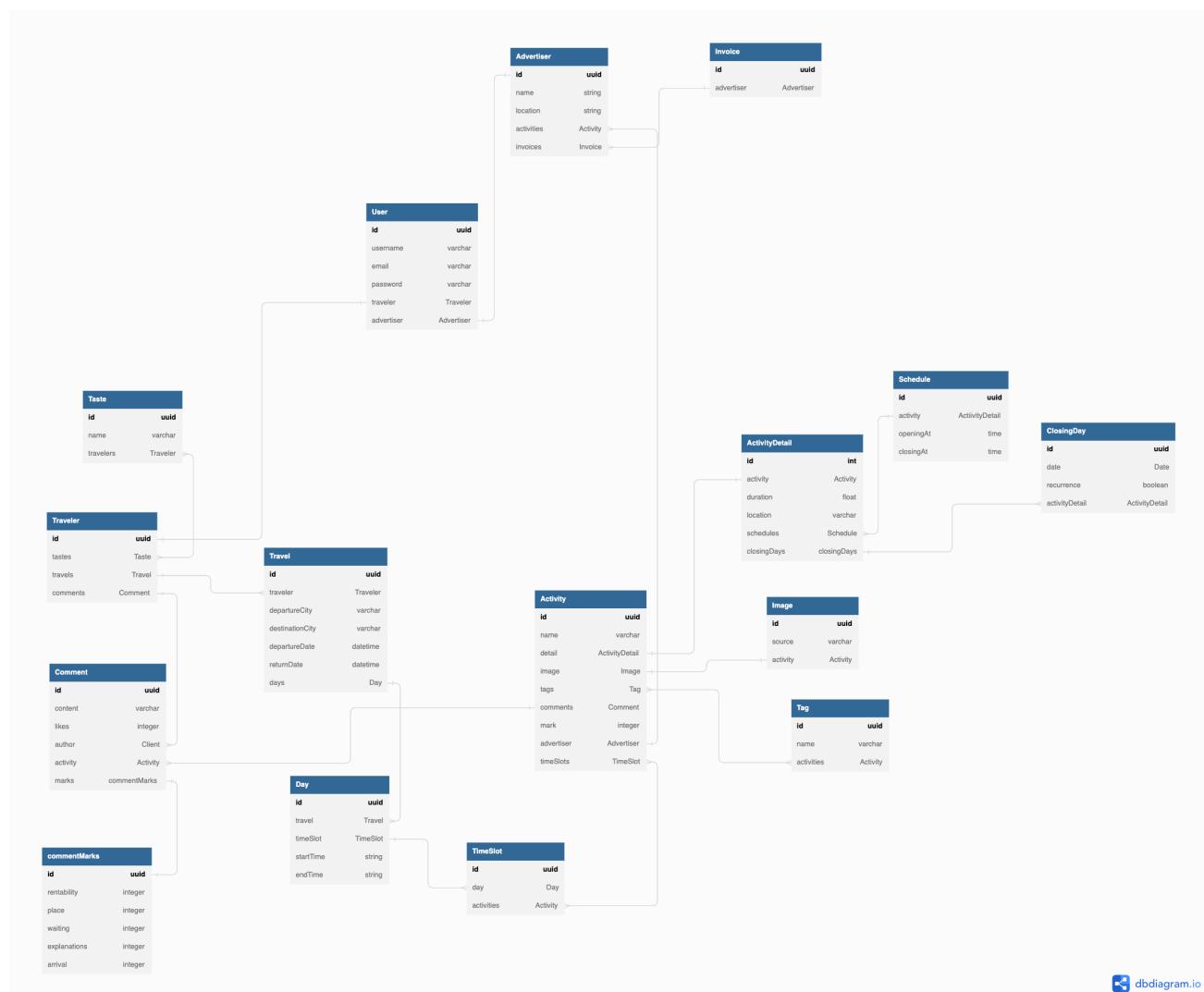
Il y aura une partie administrateur avec un back-office lui permettant de gérer les activités, les utilisateurs, les commentaires, les voyages...

La gestion des utilisateurs ayant le rôle administrateur se fera dans un premier temps depuis base de donnée pour éviter les failles de sécurité côté frontend et côté backend.

# Schéma de la base de donnée

Une fois le cahier des charges rédigé et que tous les gros points de la gestion de projet ont été réalisés, nous allons pouvoir concevoir le schéma de la base de donnée. C'est une étape cruciale pour avoir un aperçu de la structure de l'application.

Pour ce faire, j'ai utilisé l'outil DBdiagram qui permet de faire des schémas de base de donnée très facilement et ainsi d'avoir une vision claire des relations entre les tables. De plus, il est possible d'exporter notre schéma en PostgreSQL pour l'intégrer directement dans notre API.



# UX - Cible

Pour que cette application ait une raison d'être, il faut qu'elle réponde à un besoin précis pour un segment de la population particulier. C'est pour cela qu'il y a une cible, cette dernière va représenter le client type qui va consommer notre solution. Ceci va être étape très importante pour structurer notre application afin qu'elle réponde à un besoin précis.

# Sophie Anderson



**Technologie**  
Smartphone & desktop

**Citations**  
"Je veux que mes voyages soient à la fois bien organisés et remplis d'expériences uniques."  
  
"Le temps est précieux pour moi. Je cherche une solution qui simplifie ma planification de voyage."  
  
"J'apprécie les recommandations de voyage basées sur les avis d'autres voyageurs. Cela me permet de découvrir des endroits inattendus."

Age : 32 ans  
Profession : Consultante en gestion d'entreprise  
Situation familiale : Célibataire

Sophie est une professionnelle ambitieuse et dynamique, toujours à la recherche de nouveaux défis. Elle travaille comme consultante en gestion d'entreprise et voyage fréquemment pour rencontrer des clients et participer à des conférences. Elle est passionnée par les voyages et aime découvrir de nouvelles cultures et expériences.

**Objectif & besoins**

<b>Gain de temps</b> <i>optimiser la planification de ses voyages pour minimiser les perturbations de son emploi du temps chargé. elle a besoin d'une solution rapide prenant en compte ses préférences personnelles</i>	<b>Organisation simplifiée</b> <i>Sophie a besoin d'une application qui centralise toutes les informations pertinentes concernant ses voyages, telles que les détails des vols, les réservations d'hôtel, les itinéraires et les documents de voyage.</i>
<b>Personnalisation</b> <i>Sophie aime personnaliser ses voyages en fonction de ses intérêts et de ses préférences. Elle recherche une application qui lui suggère des activités, des restaurants et des sites touristiques en fonction de ses goûts personnels. Elle apprécierait également des recommandations basées sur les avis d'autres voyageurs.</i>	<b>Assistance en temps réel</b> <i>En raison de ses déplacements fréquents, Sophie a besoin d'une assistance en temps réel pour faire face à d'éventuels imprévus tels que les retards de vol, les annulations ou les changements d'itinéraire. Elle souhaite pouvoir contacter facilement le service client de l'application pour obtenir de l'aide en cas de besoin.</i>

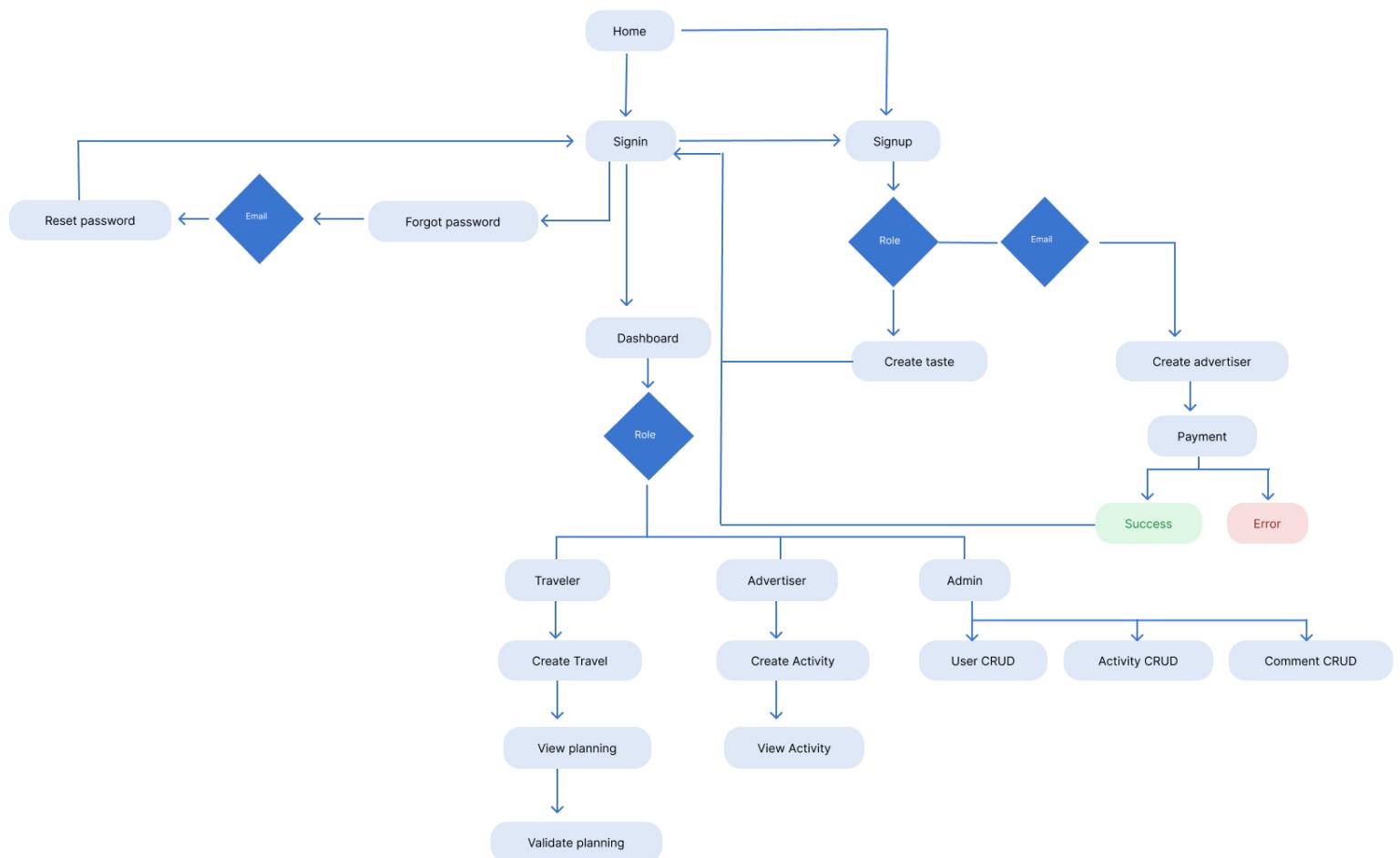
# UX - SEO

Le SEO (search engine optimization) est un ensemble de règles à adopter pour mieux référencer un site web. Il y a notamment l'utilisation des attributs alt pour les images, mais aussi le fait d'avoir qu'une seule balise HTML <h1> par page, ou bien l'utilisation des attributs title et description de la partie <head> des pages HTML.

MOTS-CLÉS	VOLUME	CPC	PD	SD	MISE À JOUR
séjour pas cher	14 800	€0,47	60	53	3 semaines
voyage grece	12 100	€0,74	53	49	3 semaines
voyage voyage	12 100	€0,19	1	48	3 semaines
voyage en italie	12 100	€0,62	31	52	3 semaines
voyage egypte	12 100	€0,43	56	50	3 semaines
voyage japon	12 100	€1,24	56	51	3 semaines
voyage maroc	12 100	€0,56	50	56	La semaine dernière
voyage new york	12 100	€1,13	57	48	3 semaines
voyage thailande	12 100	€1,01	60	47	3 semaines
séjour all inclusive	12 100	€0,77	64	47	3 semaines
voyage de noce	9 900	€0,98	34	45	3 semaines
voyage organisé	9 900	€0,59	51	46	3 semaines

# UX - User-flows

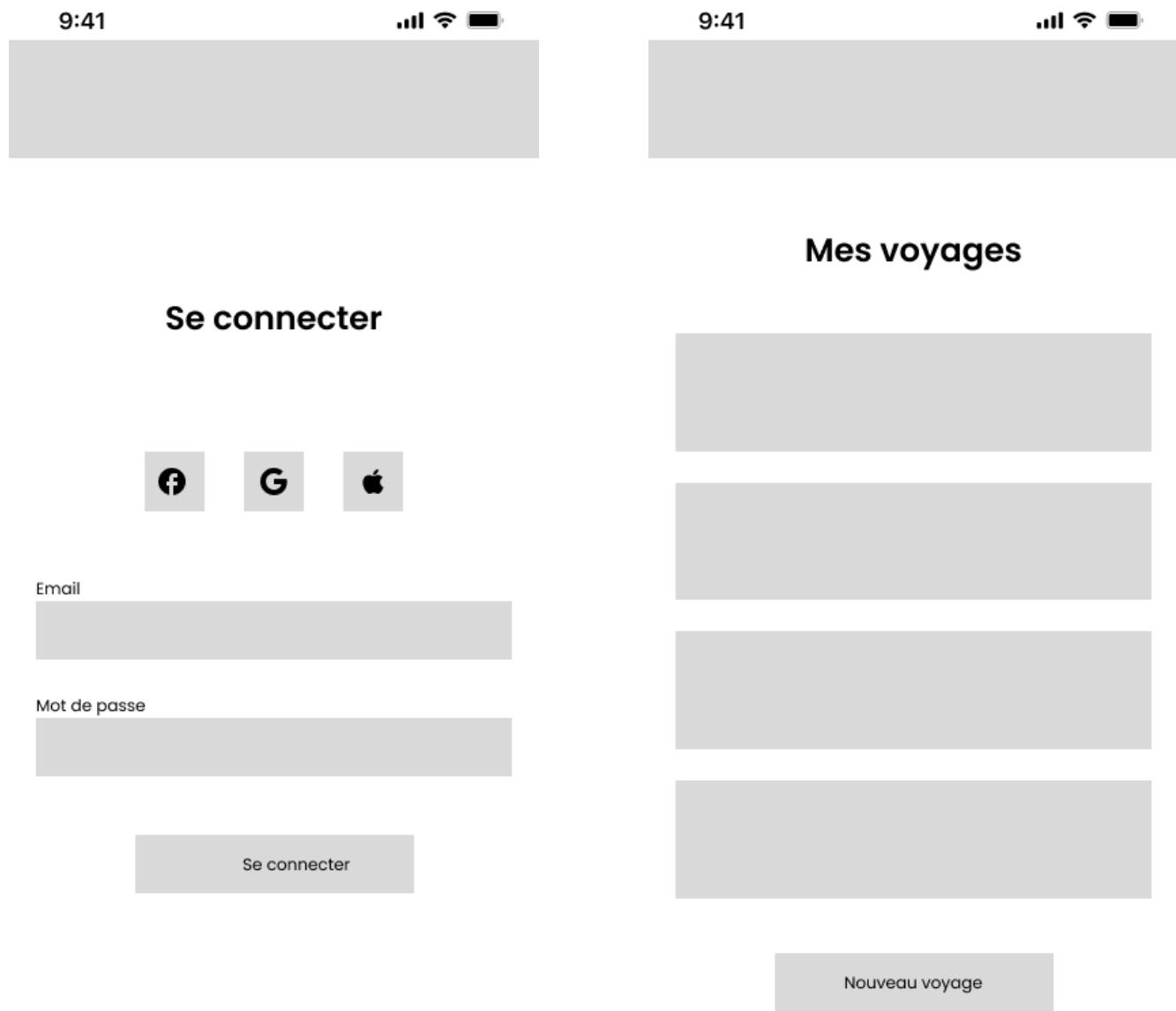
Une fois notre cible définie et ces besoins clairement exprimés, nous allons créer des user-flow qui vont représenter le parcours de l'utilisateur sur la plateforme. Ce user flow représente le parcours de l'utilisateur en fonction des différents rôles.



# UX - Wireframes

Une fois nos user-flow créés, nous allons pouvoir créer des wireframes qui va être la structure visuelle et fonctionnelle de nos pages en fonction des besoins exprimés par la cible et par le cahier des charges.

Ici, il n'y aura pas de couleurs, c'est juste une vision globale de l'application. Les formes grises représentant l'emplacement de certains éléments sur les pages (card, bouton...). Pour ce faire, j'utilise le logiciel Figma. Ce dernier va me permettre de réaliser les maquettes en s'appuyant sur les wireframes.



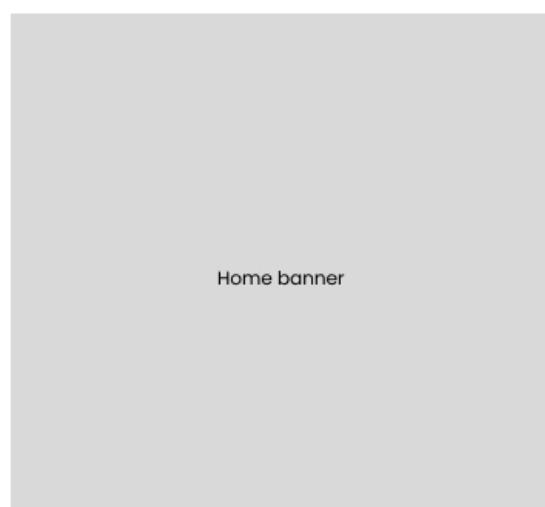


9:41

. . . . .

9:41

. . . . .



Home banner

Input nom de ville



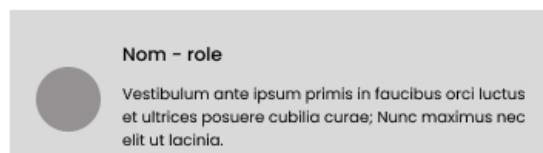
Input date départ



Input date de retour

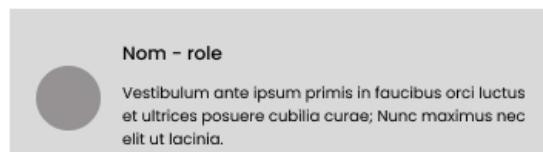


## Commentaires



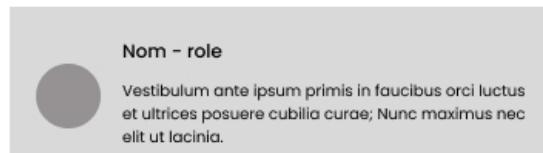
Nom - role

Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Nunc maximus nec elit ut lacinia.



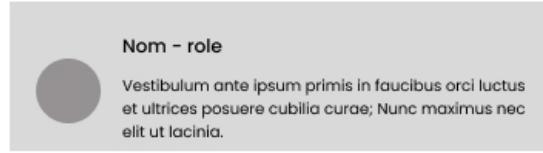
Nom - role

Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Nunc maximus nec elit ut lacinia.



Nom - role

Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Nunc maximus nec elit ut lacinia.



Nom - role

Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Nunc maximus nec elit ut lacinia.

Ajouter un commentaire



## Nouveau voyage

Input nom de ville

Input date départ

Input date de retour

Date picker

Valider



# Ui - Inspirations

Afin d'avoir des visuels homogènes, modernes, mais aussi familiers de l'utilisateur, j'ai décidé de m'inspirer de site de réservations de séjours comme Airbnb, Booking, EasyJet ou encore TripAdvisor.

En m'aidant de ces inspirations et des wireframes, je vais pouvoir concevoir les maquettes de l'application.

The screenshot shows the Airbnb homepage with a search bar at the top. Below it are several filters and categories. The main content area displays a grid of travel listings:

- Drimmin, Royaume-Uni**: Particulier, 16-21 nov. 233 € par nuit. ★ 4,95.
- Ponta Delgada, Portugal**: Professionnel 4-9 déc. 145 € par nuit. ★ 4,95.
- Pelkosenniemi, Finlande**: Professionnel 3-8 janv. 176 € par nuit. ★ 4,81.
- Harlingen, Pays-Bas**: Professionnel 3-8 sept. 382 € par nuit. ★ 4,99.
- Amsterdam, Pays-Bas**: Professionnel 1-6 juil. 1485 € par nuit. ★ 4,92.
- Joncherey, France**: Professionnel 24-29 sept. 258 € par nuit. ★ 5,0.
- Wargnies-le-Petit, France**: Professionnel 3-8 sept. 171 € par nuit. ★ 4,86.
- Sutton, Royaume-Uni**: Particulier 20-26 sept. 281 € par nuit. ★ 4,92. [Afficher la carte](#)
- Caylus, France**: Professionnel 3-9 sept. 136 € par nuit. ★ 4,92.
- Dalerveen, Pays-Bas**: Professionnel 3-8 juil. 91 € par nuit. ★ 4,92.

At the bottom, there's a footer with links to 2023 Airbnb, Inc., Confidentialité, Conditions générales, Plan du site, Fonctionnement du site, Infos sur l'entreprise, Destinations, Français (FR), EUR, Assistance et ressources.

The screenshot shows the Booking.com homepage with a dark blue header featuring the Booking.com logo, currency (EUR), language (Français (FR)), and user options (List your property, Register, Sign in).

The main heading is "Find your next stay" with a sub-instruction "Search deals on hotels, homes, and much more...". Below it are search fields for destination, check-in and check-out dates, and guest information (2 adults, 0 children, 1 room). There's also a checkbox for "I'm traveling for work".

A section titled "Offers" highlights "Promotions, deals, and special offers for you". It includes two cards: "Take your longest vacation yet" (Browse properties offering long-term stays) and "Fly away to your dream vacation" (Get inspired – compare and book flights with flexibility).

Below these are sections for "Explore France" and "Explore Europe". The "Explore France" section shows thumbnails for Paris, Marseille, Nice, Cannes, Lyon, and Montpellier, each with a price of 1 774 €/nuit.

## Explore France

These popular destinations have a lot to offer



Paris  
1 774 €/nuit



Marseille  
1 774 €/nuit



Nice  
2 254 €/nuit



Cannes  
2 774 €/nuit



Lyon  
1 204 €/nuit



Montpellier  
2 274 €/nuit

**Notre guide pour votre deuxième visite à Barcelone**  
Lors de votre premier voyage à Barcelone, vous avez probablement vu bon nombre de sublim...

Lire

**Tripadvisor**

Avis Voyages Alertes Connectez-vous Panier

Hôtels Activités Locations de vacances Restaurants Récits de voyage Plus ...

**easyJet** Infos de vol Hôtels Voitures/Autres services Business Mes réservations S'enregistrer Se connecter Aide fr-FR

Vols Hôtels Voitures

Aller simple

De: Paris Orly (ORY)

À: par ex., Paris

Aller Retour

Adultes (16+): 1

Enfants (2-15): 0

Bébés (< 2): 0

Afficher les vols >

À propos de l'assistance spéciale >

Nous compensons nos émissions carbone >

**VOUS APERCEVEZ L'ÉTÉ À L'HORIZON?**

C'est le moment de réserver pour les grandes vacances !

Réservez maintenant >

Bordeaux à partir de Lille 22,98 € nov. 2023 Réservez >

Toulouse à partir de Lille 22,98 € nov. 2023 Réservez >

Marseille Provence à partir de Bordeaux 24,55 € nov. 2023 Réservez >

Lille à partir de Bordeaux 24,55 € nov. 2023 Réservez >

Le prix indiqué est valable pour un aller simple, taxes incluses, à disponibilité limitée.



# Ui - Maquette

Le but ici est de reprendre les wireframes et d'utiliser les inspirations dont je viens de parler pour créer une identité de marque et des visuels. C'est à partir de là que l'on va avoir des maquettes pour l'application.



## An innovative solution

Our travel management project is a groundbreaking website that revolutionizes the way people plan their trips. By allowing users to input their preferences, the site



### Traveler Dashboard

Departure city	Destination city	Departure date	Return date	<a href="#">Create travel</a>
<input type="text"/>	<input type="text"/>	<input type="text"/> dd/mm/yyyy	<input type="text"/> dd/mm/yyyy	

#### My Travels

Bordeaux, Gironde, France	14/07/2023 - 22/07/2023	Not validated		
Bordeaux, Gironde, France	10/07/2023 - 21/07/2023	Not validated		
Bordeaux, Gironde, France	30/06/2023 - 01/07/2023	Not validated		

1





# Développement - Workflow

Pour ce projet, j'ai choisi d'utiliser Git pour me faciliter la tâche au niveau du versioning du projet, mais aussi de rester organisé.

Dans ce projet, chaque feature à sa propre branche, que ce soit pour de la création de feature, de la résolution de bugs, ou encore du refactoring.

De plus, ce projet étant hébergé sur Github, il y a tout un pipeline pour la CI-CD pour me faciliter la tâche au niveau du déploiement. Tout ceci est rendu possible grâce aux GitHub Actions. Il y a aussi un système de changelog automatisé grâce à Sementic Release qui permet également d'avoir un historique du code par version.

Le fait de travailler avec GitHub apporte un autre avantage que sont les Issues. Je me suis servi de cette fonctionnalité en guise de ticket pour mes différentes features à laquelle je leur assigne un label de priorité afin de toujours rester organisé.

Pour la partie développement, j'utilise Docker dans le but de pouvoir virtualiser la base de donnée ainsi que le serveur permettant d'envoyer les emails aux utilisateurs.

Travaillant sous l'environnement macOS, j'ai décidé d'ajouter un Makefile, pour avoir des commandes raccourcies dans la gestion de Docker.

Pour ce qui est du code en lui-même, j'utilise PNPM qui est un gestionnaire de package utilisant les mêmes commandes que NPM, il possède plusieurs avantages.

Il est plus économique en espace de stockage. Les dépendances seront téléchargées une seule fois sur ma machine avant que PNPM regarde soit le package existe et ainsi va créer une sorte de raccourcis du package dans le dossier node\_modules du projet.

Il est également possible de monter un monorepo avec un système de workspace très facilement, rapidement et simplement.

Voici un exemple du pipeline pour le changelog du projet:

```

● ● ●

name: Release
on:
  push:
    branches:
      - main

jobs:
  install-deps:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Cache node modules
        id: cache
        uses: actions/cache@v2
        with:
          path: node_modules
          key: cache-node-${{ hashFiles('pnpm-lock.yaml') }}
    }{
      - name: Setup Node
        uses: actions/setup-node@v1
        if: steps.cache.outputs.cache-hit != 'true'
        with:
          node-version: 18
      - name: install dependencies
        run: npm i

  semantic-release:
    runs-on: ubuntu-latest
    needs: [install-deps]
    steps:
      - uses: actions/checkout@v3
        name: Checkout main
        with:
          fetch-depth: 0
      - name: Cache node modules
        id: cache
        uses: actions/cache@v2
        with:
          path: node_modules
          key: cache-node-${{ hashFiles('pnpm-lock.yaml') }}
    }{
      - name: Launch Semantic release
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        run: npx semantic-release

```

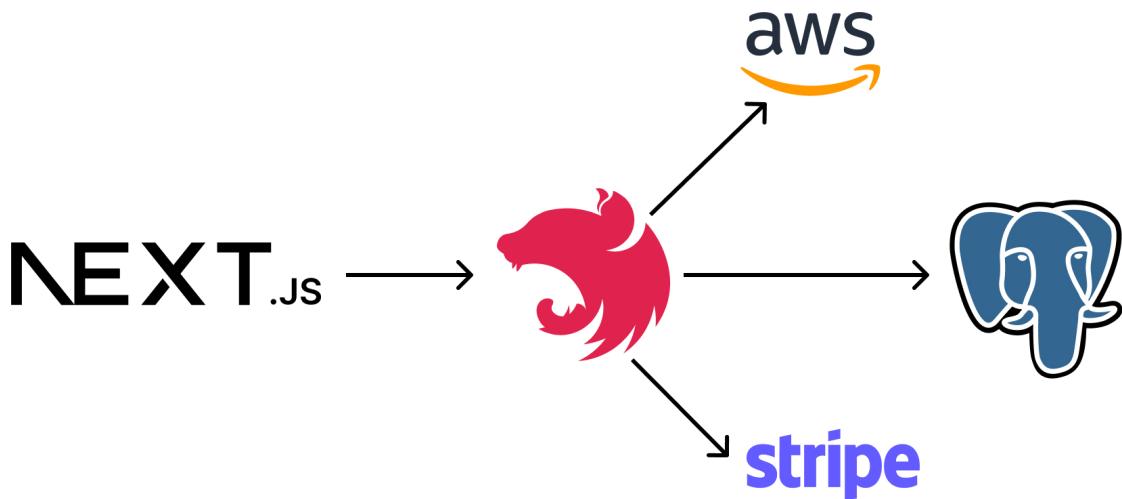
# Développement - Architecture du projet

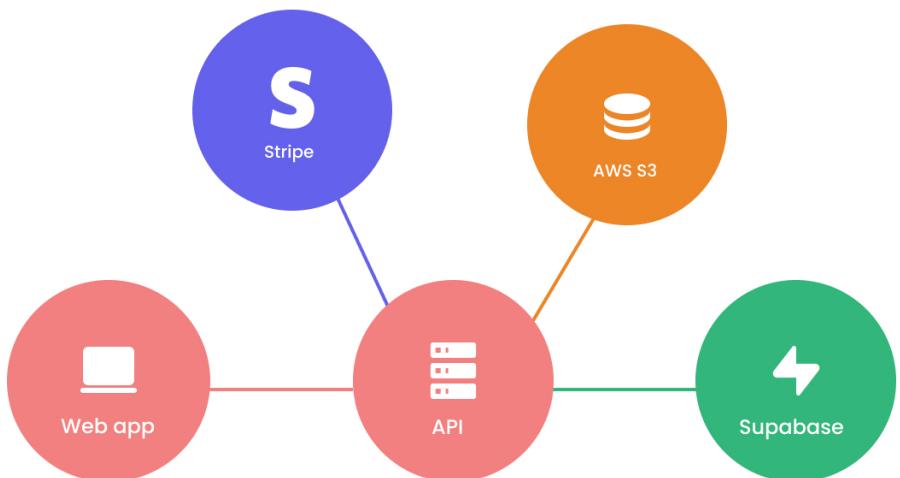
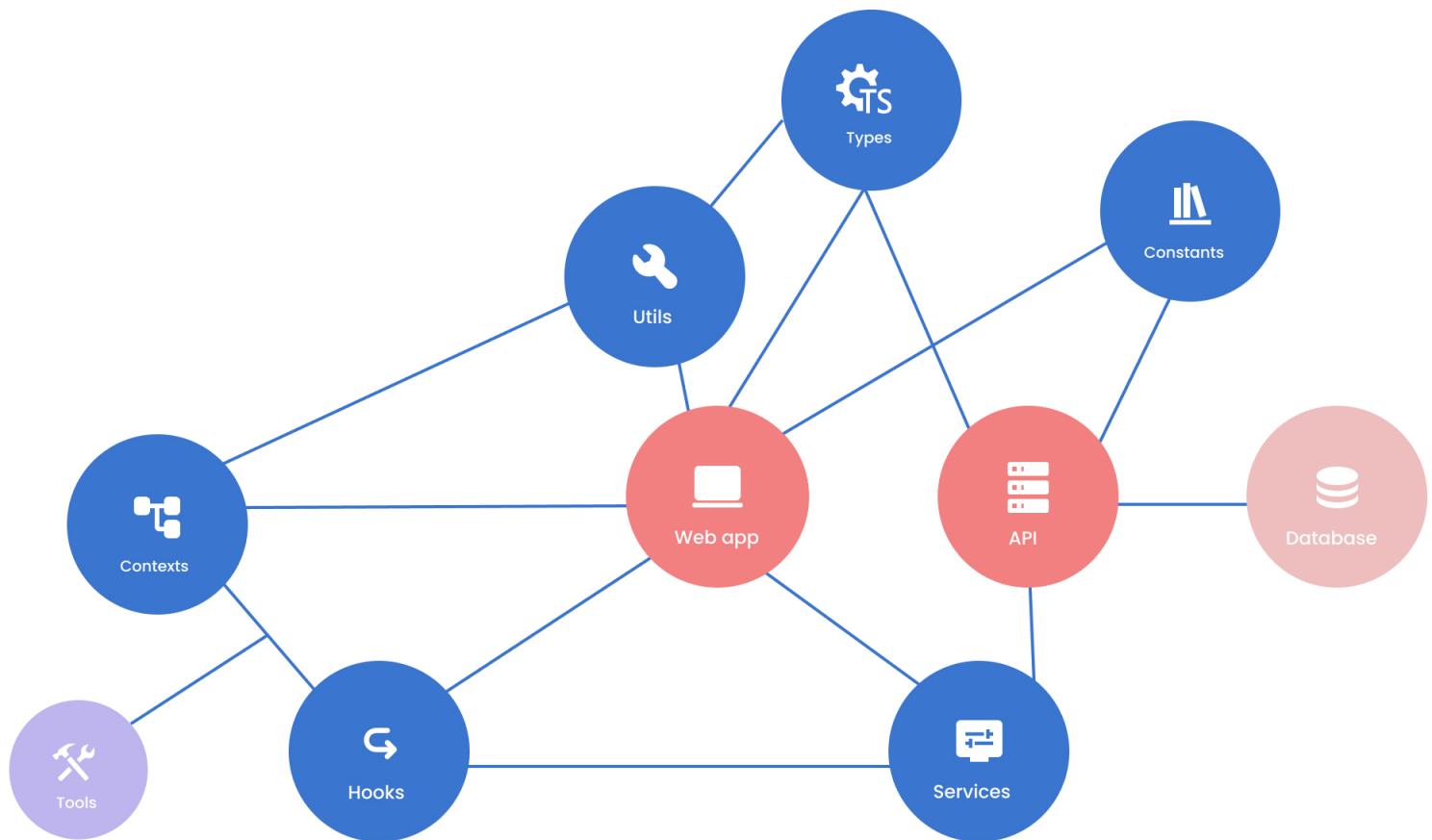
Pour l'architecture de ce projet, j'ai choisi de créer mon application sur un monorepo utilisant les workspace PNPM.

Grâce à ce monorepo, j'utilise TypeScript de bout en bout, sans avoir besoins de compiler le code entre mes différents packages.

L'avantage de cette architecture, c'est qu'elle est modulable, de plus, il est facile de rajouter une application React ou React Native dans cet écosystème, puisque tout est quasiment réutilisable instantanément ou ne nécessite que très peu d'adaptation.

Avant de commencer à coder le projet, il y a donc eu une réflexion pour architecturer l'application pour qu'elle soit le plus simple, modulable et scalable possible.





# Stack - NextJS

Tout d'abord, Next.js est polyvalent avec des fonctionnalités spécifiques. Grâce à sa capacité de rendu côté serveur (SSR) et de rendu côté client (CSR) hybride, j'ai pu générer des pages de manière dynamique et les rendre côté serveur, améliorant ainsi considérablement les performances globales de l'application et offrant une expérience utilisateur fluide. De plus, en utilisant le pré-rendu (pre-rendering), j'ai pu réduire le temps de chargement en générant des pages statiques mises en cache, ce qui a également amélioré l'indexation de mon application par les moteurs de recherche.

Ensuite, j'ai trouvé que Next.js offrait une prise en charge du routage dynamique. Étant donné que mon application nécessite de générer des itinéraires spécifiques en fonction des choix de l'utilisateur, Next.js m'a permis de créer facilement des routes dynamiques en utilisant des paramètres dans les URL, simplifiant ainsi la gestion des différentes pages associées.

Un autre avantage de Next.js par rapport à une application React classique est son amélioration du référencement (SEO). Grâce à son rendu en SSR ou CSR, Next.js génère des pages HTML indépendantes, ce qui améliore la visibilité de mon application dans les moteurs de recherche, contrairement à une SPA (Single Page Application) React qui a un seul fichier généré, ce qui la rend moins performante en termes de SEO.

En outre, la combinaison de Next.js et React était un choix naturel pour développer une application web moderne et réactive. Cela m'a permis d'organiser mon code de manière modulaire, réutilisable et facile à maintenir. Cette structure claire du code facilitera la gestion des différentes fonctionnalités et la collaboration entre les membres de mon équipe de développement.

Enfin, Next.js dispose d'une vaste collection de modules et de bibliothèques prêts à l'emploi, ce qui m'a offert des possibilités d'extension des fonctionnalités de mon application.

En résumé, en utilisant Next.js, j'ai pu profiter de fonctionnalités puissantes telles que le rendu côté serveur, le routage dynamique, l'intégration transparente avec React et la flexibilité pour intégrer d'autres services, ce qui a facilité le développement d'un gestionnaire de voyage automatisé performant, réactif et évolutif.

# Stack - Tailwind CSS

Grâce à l'approche utility-first de Tailwind CSS, j'ai pu créer des styles réutilisables en utilisant des classes directement dans le code JSX. Cela m'a fait gagner énormément de temps dans le développement de l'interface utilisateur, car je n'ai pas eu à écrire des styles CSS personnalisés à partir de zéro (sauf pour les templates d'email). Les classes utilitaires de Tailwind CSS couvrent un large éventail de styles prédéfinis, tels que la mise en page, les couleurs, les polices, les marges, les rembourrages, etc. J'ai ainsi pu personnaliser rapidement l'apparence de mon application en utilisant simplement les classes adaptées à mes besoins.

En outre, l'approche de conception atomique de Tailwind CSS m'a aidé à maintenir une cohérence de design tout au long de mon application. En réutilisant de manière systématique les classes utilitaires, j'ai pu construire une interface utilisateur cohérente et harmonieuse.

Grâce à ce framework, j'ai pu créer facilement des mises en page et des composants qui s'adaptent de manière fluide aux différentes tailles d'écran, offrant ainsi une expérience utilisateur optimale sur les appareils mobiles et les ordinateurs de bureau.

Une autre caractéristique appréciable de Tailwind CSS est sa grande personnalisation et extensibilité. J'ai pu étendre les fonctionnalités de Tailwind CSS en ajoutant mes propres classes utilitaires ou en modifiant les valeurs par défaut du framework pour répondre aux besoins spécifiques de mon application. Cette flexibilité m'a permis d'adapter l'interface utilisateur en fonction de mes préférences et des exigences de conception de mon gestionnaire de voyage automatisé.

En somme, l'utilisation de Tailwind CSS a été un choix judicieux pour mon projet. Grâce à son approche utility-first, sa cohérence de design, sa conception réactive et sa personnalisation avancée, j'ai pu développer une interface utilisateur performante, esthétique et adaptée à mes besoins spécifiques en matière de gestion de voyage automatisé.

# Stack - NestJS

Pour commencer, NestJS est basé sur Node.js et utilise TypeScript comme langage de programmation. Cette combinaison a offert à mon projet une base solide, fiable et performante pour le développement de l'application. Grâce à TypeScript, j'ai pu profiter d'une syntaxe plus robuste, d'une meilleure vérification des types et d'une meilleure lisibilité du code, ce qui a grandement facilité la maintenance et la collaboration au sein de mon équipe de développement.

En outre, l'architecture basée sur les modules proposée par NestJS a été un atout majeur. Elle m'a permis d'organiser clairement et de manière modulaire mon code. Chaque fonctionnalité de mon application a été regroupée dans un module distinct, facilitant ainsi la gestion des dépendances, la réutilisation du code et contribuant à maintenir une base de code propre et bien structurée.

Les fonctionnalités de gestion des requêtes et des réponses basées sur les intercepteurs, les filtres et les middlewares de NestJS ont également été très utiles. J'ai pu gérer efficacement des aspects tels que l'authentification, l'autorisation, la validation des données et la gestion des erreurs. Ces fonctionnalités intégrées ont renforcé la sécurité et la fiabilité de mon application.

Le système de gestion intégré des serveurs HTTP de NestJS a également été un avantage considérable. J'ai pu créer facilement des API RESTful pour communiquer avec d'autres services ou applications externes. Grâce aux décorateurs et aux annotations de NestJS, j'ai pu définir les endpoints, les contrôleurs et les services nécessaires pour interagir avec les différentes fonctionnalités du gestionnaire de voyage automatisé.

Enfin, j'ai apprécié la communauté active et l'écosystème bien développé de NestJS. J'ai pu tirer parti d'un large éventail de modules et d'extensions prêts à l'emploi pour étendre les fonctionnalités de mon application. Cela m'a permis d'intégrer facilement des bases de données, des services d'authentification, des services de messagerie, etc., répondant ainsi aux besoins spécifiques de mon projet de gestionnaire de voyage automatisé. De plus la DX (Developer Experience) de Nest a été un point important pour le choix de cette technologie.

En somme, l'utilisation de NestJS s'est avérée être un choix judicieux pour mon projet. Grâce à sa base solide, son architecture modulaire, ses fonctionnalités de gestion avancées, son support pour les serveurs HTTP et son écosystème actif, j'ai pu développer une application de gestion de voyage automatisé performante, sécurisée et évolutive. Je suis très satisfait des résultats obtenus grâce à ce choix de framework.

# Stack - PostgreSQL

Tout d'abord, la stabilité, la fiabilité et la robustesse de PostgreSQL ont été des atouts majeurs. Je peux désormais compter sur la durabilité des données, ainsi que sur la capacité de PostgreSQL à gérer efficacement de grandes quantités de données. Cette fiabilité est cruciale pour assurer la précision et l'intégrité des données, un aspect essentiel dans un système de gestion de voyage automatisé.

En outre, les fonctionnalités avancées de PostgreSQL ont grandement contribué à améliorer mon projet. La prise en charge des transactions ACID, la gestion des contraintes, les déclencheurs, les procédures stockées et les vues matérialisées m'ont permis de mettre en place des règles métier complexes et de garantir la cohérence et l'intégrité des données. Cette flexibilité est extrêmement précieuse pour les futures évolutions de mon application.

Le langage SQL intégré à PostgreSQL a également été d'une grande aide. Il m'a facilité la manipulation des données et l'extraction d'informations spécifiques. Les requêtes complexes que j'ai pu effectuer ont permis d'obtenir des planning personnalisés avec des relations complexes entre les différentes tables de la base de donnée, ce qui m'a aidé à prendre des décisions éclairées et à optimiser mon système de gestion automatisé.

En outre, les fonctionnalités avancées de recherche et d'indexation offertes par PostgreSQL, notamment la recherche plein texte, ont considérablement amélioré l'expérience utilisateur. Mes utilisateurs peuvent désormais effectuer des recherches rapides et précises sur des destinations, ce qui ajoute une valeur significative à mon application.

Enfin, la large communauté de développeurs et la documentation complète de PostgreSQL ont été d'une grande aide. J'ai pu trouver des ressources, des tutoriels et des forums en ligne pour résoudre d'éventuels problèmes et bénéficier de conseils sur l'optimisation des performances et la conception de la base de données. Cette assistance continue est un avantage précieux pour maintenir mon projet efficace et évolutif.

Dans l'ensemble, l'utilisation de PostgreSQL s'est révélée être une décision très avantageuse pour mon projet, offrant stabilité, fiabilité, fonctionnalités avancées, capacités de recherche performantes et un support continu de la communauté de développeurs.

# Focus technique : gestion des utilisateurs

La gestion des utilisateurs étant un élément principal de l'application, voici un exemple de contrôleur, service et repository, entité et DTO (Data Transfer Object).

- Entité :

```

@Entity()
export class User extends Timestamp {
    @PrimaryGeneratedColumn('uuid')
    id: string;

    @Column({ unique: true, nullable: false })
    email: string;

    @Column({ nullable: false })
    username: string;

    @Column({ nullable: false })
    password: string;

    @Column({
        type: 'enum',
        enum: Role,
        default: Role.Traveler,
        nullable: false,
    })
    roles: Role;

    @OneToOne(() => Advertiser)
    @JoinColumn()
    advertiser: Advertiser;

    @OneToOne(() => Traveler)
    @JoinColumn()
    traveler: Traveler;

    @OneToOne(() => Customer, customer => customer.user, {
        cascade: true
    })
    @JoinColumn()
    customer: Customer;

    @OneToOne(() => ResetPasswordToken)
    @JoinColumn()
    resetPasswordToken: ResetPasswordToken;
}

```

- Contrôleur:

```
@Controller('user')
@UseGuards(ThrottlerGuard)
export class UserController {
    constructor(private readonly userService: UserService) {}

    @Post()
    @Throttle(500, 60)
    async create(@Body() signupUserDto: SignupUserInputDTO):
Promise<User> {
        return await this.userService.create(signupUserDto,
signupUserDto.roles as Role)
    }

    @Get()
    @Throttle(500, 60)
    async findAll(@Query() queries: ApiLimitResourceQuery) {
        return await this.userService.findAll(queries)
    }

    @Get(':email')
    @Throttle(500, 60)
    async findOneByEmail(@Param('email') email: string):
Promise<User> {
        return await this.userService.findOneByEmail(email)
    }
}
```

- DTO (Data Transfer Object):

```
export class LoginUserInputDTO {
    @IsEmail()
    email: string;

    @IsString()
    password: string;
}
```

- Service :

```

● ● ●

@Injectable()
export class UserService {
  constructor(private userRepository: UserRepository, private
travelerService: TravelerService, private customerService:
CustomerService) { }

  async create(signupUserDto: SignupUserInputDTO, roles:
Role): Promise<User> {
    try {
      if (testEmailUtil(signupUserDto.email)) {
        const customer = await
this.customerService.createCustomer({ email:
signupUserDto.email, name: signupUserDto.username })

        const user = new User()
        user.username = signupUserDto.username
        user.email = signupUserDto.email
        user.password = signupUserDto.password
        user.roles = roles
        user.customer = customer

        switch (roles) {
          case Role.Traveler: {
            const traveler = new Traveler()
            user.traveler = traveler
            await this.travelerService.save(traveler)
            return await this.userRepository.save(user)
          }
          break;
        }

        return await this.userRepository.save(user)
      } else {
        throw new BadRequestException('email must contain
***@***.***')
      }
    } catch (error) {
      throw new BadRequestException(error)
    }
  }

  async findAll(queries: ApiLimitResourceQuery) {
    try {
      return await this.userRepository.findAllUser(queries)
    } catch (error) {
      throw new NotFoundException(error)
    }
  }
}

```

- Repository :

```


export class UserRepository extends Repository<User> {
    constructor(@InjectDataSource() datasource: DataSource) {
        super(User, datasource.createEntityManager());
    }

    async createUser(signupUserDto: SignupUserInputDTO): Promise<User> {
        const user = this.create({ ...signupUserDto } as User);
        return await this.save(user);
    }

    async saveUser(user: User) {
        return await this.save(user)
    }

    async findAllUser(queries: ApiLimitResourceQuery) {
        let { page, limit, sortedBy, username, email, roles } = queries;
        page = page ? +page : 1;
        limit = limit ? +limit : 10;

        const query = this.createQueryBuilder('user')
            .leftJoinAndSelect('user.traveler', 'traveler')
            .leftJoinAndSelect('traveler.tastes', 'tastes')
            .leftJoinAndSelect('traveler.travels', 'travels')
            .leftJoinAndSelect('traveler.comments',
            'comments')
            .leftJoinAndSelect('user.customer', 'customer')
            .leftJoinAndSelect('user.advertiser',
            'advertiser')
            .leftJoinAndSelect('user.resetPasswordToken',
            'resetPasswordToken')

        if (sortedBy) {
            query.orderBy('user.createdAt', sortedBy)
        } else {
            query.orderBy('user.createdAt', 'DESC')
        }

        if (username) {
            query.andWhere('user.username LIKE :username', { username })
        }

        if (email) {
            query.andWhere('user.email LIKE :email', { email })
        }

        if (roles) {
            query.andWhere('user.roles = :roles', { roles })
        }

        return {
            page: page,
            limit: limit,
            total: await query.getCount(),
            data: await query.skip((page - 1) *
            limit).take(limit).getMany()
        }
    }
}

```

# Focus technique : Authentification

L'authentification est un point obligatoire dans les applications, ceci est déjà un gros point de sécurité. Ici, j'ai utilisé les JSON Web Tokens pour garantir la sécurité des utilisateurs. Un JWT est une chaîne de caractère encryptée content des informations comme sa date de création et d'expiration, mais aussi d'autres infos sur l'utilisateur, comme son email, son rôle... . Elle contient aussi une signature unique, ce qui fait qu'il est très difficilement violable.

The screenshot shows a JWT token being analyzed on jwt.io. The token is pasted into the 'Encoded' field:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6InRlc3QyQHRLc3QuY29tIiwiaWQi0iJmMDM2MmI0Yi0xY2M4LTQ0ZjQtOTU1My1jODg3OWQ1MmE3NWEiLCJyb2xlcI6InRyYXZlbGVyIiwiaWF0IjoxNjg2MTQzMzUwLCJleHAiOjE2ODc5NTgxNTB9.EALpuq8VhShydZXnPPyjQm9qh1qK0pc7HV8X0y2p0u4|
```

The 'Algorithm' dropdown is set to 'HS256'. The 'Decoded' section shows the following payload:

```
{
  "alg": "HS256",
  "typ": "JWT"
}

{
  "email": "test2@test.com",
  "id": "f0362b4b-1cc8-44f4-9553-c8879d52a75a",
  "roles": "traveler",
  "iat": 1686143758,
  "exp": 1687958150
}
```

The 'VERIFY SIGNATURE' section contains the HMACSHA256 verification code:

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) □ secret base64 encoded
```

Voici un exemple de module, service et strategy pour l'authentification:

- **Strategy:**

```
● ● ●  
@Injectable()  
export class JwtStrategy extends PassportStrategy(Strategy) {  
  constructor() {  
    super({  
      jwtFromRequest:  
        ExtractJwt.fromAuthHeaderAsBearerToken(),  
        ignoreExpiration: false,  
        secretOrKey: process.env.JWT_SECRET,  
    });  
  }  
  
  async validate(payload: ValidateUserDto) {  
    return { userId: payload.sub, username: payload.username  
  };  
  }  
}@Injectable()  
export class JwtStrategy extends PassportStrategy(Strategy) {  
  constructor() {  
    super({  
      jwtFromRequest:  
        ExtractJwt.fromAuthHeaderAsBearerToken(),  
        ignoreExpiration: false,  
        secretOrKey: process.env.JWT_SECRET,  
    });  
  }  
  
  async validate(payload: ValidateUserDto) {  
    return { userId: payload.sub, username: payload.username  
  };  
  }  
}
```

- **Service :**

```


    @Injectable()
    export class AuthService {
        constructor(
            private userService: UserService,
            private jwtService: JwtService,
            private resetPasswordTokenService:
ResetPasswordTokenService,
            private mailService: MailService
        ) {}

        public async validateUser(email: string, password: string)
{
    const user = await this.userService.findOneByEmail(email)

    if (!user) {
        throw new HttpException('User is not found',
HttpStatus.NOT_FOUND)
    }

    const isMatch = await bcrypt.compare(password,
user.password)
    if (user && isMatch) {
        const { password, ...result } = user
        return result
    }
}

public async signin(user: SigninDTO) {
    try {
        const findUser = await
this.userService.findOneByEmail(user.email)

        if (!findUser) {
            throw new HttpException(`User isn't exist`,
HttpStatus.NOT_ACCEPTABLE)
        }

        const payload = {
            email: findUser.email,
            id: findUser.id,
            roles: findUser.roles,
        }
        return {
            accessToken: this.jwtService.sign(payload),
        }
    } catch (err) {
        throw new UnauthorizedException(err)
    }
}
}

```

- **Module:**



```
@Module({
  imports: [
    ConfigModule.forRoot(),
    PassportModule,
    JwtModule.register({
      secret: process.env.JWT_SECRET,
      signOptions: { expiresIn: '21d' },
    }),
    UserModule,
    MailModule,
    ResetPasswordTokenModule
  ],
  controllers: [AuthController],
  providers: [AuthService, JwtStrategy],
  exports: [AuthService],
})
export class AuthModule {}
```

## Focus technique : Rôles

Le système de rôle est très important dans une application cela permet de gérer les permissions des utilisateurs. C'est notamment avec ça que l'on peut protéger des ressources de notre API, cela apporte une sécurité supplémentaire sur la partie backend de mon application.

Voici un exemple de décorateur, guard, enum et contrôleur:

Décorateur:



```
export const Roles = (...roles: Role[]) =>
  SetMetadata('roles', roles);
```

- Guard :

```

@Injectable()
export class RolesGuard implements CanActivate {
  constructor(private reflector: Reflector) {}

  canActivate(context: ExecutionContext): boolean {
    const requireRoles =
      this.reflector.getAllAndOverride<Role[]>('roles', [
        context.getHandler(),
        context.getClass(),
      ]);

    if (!requireRoles) {
      return true;
    }

    const request = context.switchToHttp().getRequest();
    const token: string =
      request.headers['authorization'].split(' ')[1];
    const decodedToken: AccessToken = jwt_decode(token);

    const { user } = context.switchToHttp().getRequest();
    return requireRoles.some((role) =>
      decodedToken.roles.includes(role));
  }
}

```

- Enum:

```

export enum Role {
  Traveler = 'traveler',
  Advertiser = 'advertiser',
  Admin = 'admin',
}

```

- Contrôleur:

```

@Controller('comment')
@UseGuards(ThrottlerGuard)
export class CommentController {
    constructor(private readonly commentService:
CommentService) {}

@Post()
@Throttle(500, 60)
@UseGuards(JwtAuthGuard)
@Roles(Role.Traveler, Role.Advertiser, Role.Admin)
    create(@Body() createCommentDto: CreateCommentDto,
@Body('activity') activity: string) {
        return this.commentService.create(createCommentDto,
activity)
    }
}

```

## Focus technique : Sécurité

Pour garantir un maximum de sécurité et limiter le nombre de failles, plusieurs sécurités, on était ajoutés.

- CORS permettant d'autoriser la réception de requêtes d'un ou plusieurs noms de domaines précis
- Helmet permettant de protéger les headers HTTP contre les vulnérabilités
- Rate limit permettant de limiter les attaques par brute force ou DDOS (Distributed Denial of Service), on définit un certain nombre de requêtes autorisé dans un temps donné, si la limite est dépassée, le serveur renvoie une erreur 429.

# Focus technique : Documentation

Avec Swagger, j'ai pu générer automatiquement une documentation interactive pour mon API NestJS. Cette fonctionnalité s'est avérée très précieuse, car elle permet aux développeurs (dont moi si je retourne sur le projet longtemps après les derniers ajouts) de comprendre rapidement les différentes routes, les paramètres acceptés et les réponses renvoyées par l'API. L'avantage majeur est qu'il évite le travail fastidieux de création et de maintenance d'une documentation séparée, ce qui a grandement simplifié le processus de développement.

En utilisant Swagger, les équipes de développement et de conception ont pu bénéficier d'une compréhension claire et précise des fonctionnalités de l'API. Grâce à la documentation générée par Swagger, moi même et les futurs développeurs pourront facilement apprendre à interagir avec l'API et utiliser ses fonctionnalités correctement.

Une autre caractéristique intéressante de Swagger est son interface utilisateur interactive (UI), qui permet aux développeurs de tester aisément les différentes routes de l'API sans avoir besoin d'un client HTTP externe. La possibilité d'envoyer des requêtes directement depuis l'interface Swagger a considérablement simplifié les tests et a accéléré le processus de développement.

Une fonctionnalité particulièrement pratique est la capacité de Swagger à générer automatiquement des clients HTTP dans différents langages de programmation à partir de la spécification OpenAPI. Cela a grandement facilité l'intégration de mon API NestJS avec d'autres services ou applications, car les clients générés suivent les conventions définies par Swagger et sont prêts à être utilisés.

Enfin, j'ai apprécié le fait que Swagger suive la spécification OpenAPI, un standard largement accepté pour la description des API. En utilisant Swagger, mon API NestJS a donc été rendue conforme aux normes de l'industrie, ce qui la rend simplement compréhensible par d'autres développeurs utilisant des outils compatibles OpenAPI.

En résumé, l'utilisation de Swagger pour générer la documentation de mon API NestJS s'est avérée être une décision judicieuse. Grâce à cette documentation interactive, claire et conforme aux normes de l'industrie, mon projet a été rendu plus facilement accessible et compréhensible pour l'équipe de développement, tout en améliorant sa scalabilité et son intégration avec d'autres services.

**Exemple de documentation Swagger:**

The screenshot shows the Swagger UI interface for the "Travel tailor API". At the top, there's a dark header bar with the "Swagger" logo and "Supported by SMARTBEAR". Below it, the title "Travel tailor API" is displayed with a "1.2 OAS3" badge. A sub-header "The travel tailor API" follows. The main content area is organized into sections: "travel tailor" and "default". Under "travel tailor", there are several API endpoints listed with their methods and URLs. Some endpoints are collapsed (indicated by a downward arrow icon) and some are expanded (indicated by an upward arrow icon). The expanded sections show detailed descriptions or examples. One endpoint, "DELETE /user/{id}", is highlighted with a red background.

Method	Path
GET	/
POST	/auth/signin
POST	/auth/signup
POST	/auth/forgot-password
POST	/auth/reset-password/{token}
POST	/user
GET	/user
GET	/user/{email}
PATCH	/user/{id}
DELETE	/user/{id}
POST	/traveler
GET	/traveler
GET	/traveler/{id}

# Focus technique : Création d'un planning

La création d'un planning étant la fonctionnalité principale de l'application, il y a tout un algorithme qui va permettre de générer un planning en fonction de plusieurs critères dont la ville d'arrivée, la date de départ et d'arrivée, mais aussi les goûts de l'utilisateur. L'algorithme va donc filtrer les activités en fonction de toutes ces informations. Les dates vont permettre d'exclure toutes les activités étant fermés à ces dates.

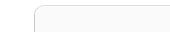
Voici un exemple de code dans le projet permettant de générer un planning de voyage.

- Contrôleur:



```
@Throttle(500, 60)
@UseGuards(JwtAuthGuard)
@Roles(Role.Traveler, Role.Admin)
@Patch('/activity/:travel_id')
async updatePlanningActivity(@Param('travel_id') travel_id: string, @Body()
updatePlanningActivityDto: UpdatePlanningActivityDto) {
    return await this.planningService.updatePlanningActivity(travel_id,
updatePlanningActivityDto);
}
```

- Service:



```
async create(userConnected: User, travel) {
    const user = await
this.userService.setTravelerIfNotSet(travel)
    const tastes = user.traveler.tastes

    const availableActivities: Activity[] = []
    const activities = await this.filterActivities(
        travelInDB,
        tastes,
        availableActivities
    )
    await this.createPlanning(travelInDB, activities)
}
```

# Focus technique : Paiement

Ce système de paiement permet de facturer les annonceurs afin que leurs activités soient référencées sur la plateforme.

Lorsqu'un annonceur crée son compte, il doit s'abonner à un plan de 100€ par mois pour accéder pleinement à la plateforme. Cet abonnement bénéficie également d'une période d'essai de 30 jours à partir de la date de début de l'abonnement.

Pour ce système de paiement, j'ai opté pour l'utilisation de Stripe. L'intégration de Stripe dans mon application NestJS présente de nombreux avantages que voici :

Stripe offre une API puissante et conviviale, simplifiant ainsi la gestion des transactions financières. Grâce à cette intégration, j'ai pu créer facilement des paiements, effectuer des remboursements, gérer les abonnements et suivre les transactions en temps réel. Stripe prend également en charge la sécurité liée aux paiements, allégeant ainsi l'application de cette responsabilité.

Stripe propose une interface de paiement optimisée et conviviale pour les utilisateurs finaux. Les clients peuvent effectuer des paiements de manière transparente en utilisant diverses méthodes de paiement prises en charge par Stripe, comme les cartes de crédit/débit et les portefeuilles numériques. Cette intégration garantit une expérience utilisateur fluide et transparente lors du processus de paiement.

Vu que mon application comprend des fonctionnalités d'abonnement, l'intégration de Stripe facilite grandement leur gestion pour les utilisateurs. Stripe propose des fonctionnalités avancées de gestion des cycles de facturation, des mises à jour des informations de paiement et de gestion des essais gratuits. Ainsi, les abonnements sont gérés efficacement, offrant une expérience utilisateur flexible et personnalisée.

Stripe dispose d'intégrations prêtées à l'emploi avec de nombreux outils populaires tels que Salesforce, QuickBooks et Mailchimp. Cette intégration permet de synchroniser automatiquement les données de paiement avec d'autres systèmes utilisés dans mon application NestJS. Par exemple, les données de paiement peuvent être intégrées au système de gestion des clients ou des e-mails de confirmation peuvent être envoyés après chaque paiement.

Enfin, Stripe prend en charge une large gamme de modèles commerciaux, des ventes ponctuelles aux abonnements en passant par les paiements à la consommation et en plusieurs étapes. Cette flexibilité m'a permis de créer des fonctionnalités de paiement

personnalisées adaptées aux besoins spécifiques de mon application NestJS. Je peux définir des règles de tarification flexibles et gérer différentes options de paiement selon les cas d'utilisation spécifiques. J'ai pu donc délégué la partie paiement à stripe ce qui m'a fait gagner du temps, tout en limitant les failles de sécurité supplémentaires.

En somme, grâce à l'intégration de Stripe, mon projet dispose d'un système de paiement performant, convivial et adaptable, soutenant ainsi un modèle économique viable pour mon application de gestion de voyage automatisé.

Voici un exemple de code :

- Service:



```

@ Injectable()
export class PaymentService {
  constructor(
    @InjectStripe() private readonly stripeClient: Stripe,
    private configService: ConfigService,
  ) { }

  async createCheckoutSession(createCheckoutDto: CreateCheckoutDto): Promise<string>
  {   try {
      const session = await this.stripeClient.checkout.sessions.create({
        payment_method_types: ['card'],
        customer: createCheckoutDto.customer,
        line_items: [
          {
            price: 'price_1Mw40MFXjkZ2xKS6EgnyEo60',
            quantity: 1,
          },
        ],
        mode: 'subscription',
        subscription_data: {
          trial_period_days: 30,
        },
        success_url: `${this.configService.get('CLIENT_APP_URL')}/payment/success`,
        cancel_url: `${this.configService.get('CLIENT_APP_URL')}/payment/cancel`,
      });
      return session.id;
    } catch (err) {
      throw new HttpException(err.message, 402);
    }
  }
}

```

# Focus technique : Upload de fichier

j'ai mis en place un système d'upload de fichier pour héberger les différentes images du contenu de l'application. Cela nous permet d'avoir des images uniques pour chaque activité. Pour cette application, j'ai choisi d'utiliser AWS S3 comme solution de stockage pour mes fichiers.

AWS S3 (Simple Storage Service) offre un stockage évolutif et durable pour mes fichiers. Il est spécialement conçu pour gérer de grands volumes de données tout en garantissant la disponibilité et la durabilité des fichiers stockés. Cette solution me permet de facilement augmenter la capacité de stockage selon mes besoins sans se soucier de la gestion de l'infrastructure sous-jacente.

Grâce à AWS S3, je bénéficie d'une haute disponibilité pour nos fichiers. Ces derniers sont automatiquement répliqués dans différentes régions, assurant ainsi leur accessibilité même en cas de défaillance d'un centre de données. Ainsi, mon application NestJS peut offrir une expérience utilisateur ininterrompue, même en cas de problèmes au niveau de l'infrastructure.

La sécurité de mes fichiers est également bien prise en charge par AWS S3. Il propose des fonctionnalités de chiffrement des données en transit et au repos, assurant ainsi la confidentialité des fichiers stockés.

Une grande force d'AWS S3 réside dans son intégration étroite avec d'autres services AWS. Cela nous permet d'étendre les fonctionnalités de notre application NestJS en utilisant ces services complémentaires.

Grâce à l'API simple et complète fournie par AWS S3, je peux facilement gérer mes fichiers. Je peux télécharger, récupérer, supprimer et mettre à jour des fichiers en utilisant des opérations basées sur l'API. De plus, S3 prend en charge la gestion des métadonnées des fichiers, ce qui me permet d'ajouter des informations supplémentaires aux fichiers pour une organisation et une recherche plus efficaces.

En conclusion, l'utilisation d'AWS S3 comme système de stockage pour les fichiers de mon application NestJS s'est avérée être une décision judicieuse. Cette solution offre une grande évolutivité, une haute disponibilité, des fonctionnalités de sécurité avancées, une intégration avec d'autres services AWS et une API facile à utiliser, ce qui a grandement contribué au succès et à la robustesse de notre projet.

Voici un exemple de service d'upload de fichier dans l'API du projet :



```
@Injectable()
export class UploadFileService {
    private s3
    private bucketName

    constructor( private readonly uploadFileRepository: UploadFileRepository) {
        this.s3 = new S3({
            region: process.env.API_BUCKET_REGION,
            accessKeyId: process.env.API_ACCESS_KEY,
            secretAccessKey: process.env.API_SECRET_KEY,
        })
        this.bucketName = process.env.API_BUCKET_NAME
    }

    async create(filesData, user, activityImage) {
        try {
            const file = await this.uploadFileAws(user, filesData)
            return await this.uploadFileRepository.createUploadFile(file,
activityImage)
        } catch(error) {
            throw new UnauthorizedException(error)
        }
    }
}
```

# Focus technique : Mails

Dans mon projet récemment achevé, la partie d'envoi d'e-mails revêt une importance capitale, car elle permet d'informer les utilisateurs et de leur fournir une trace de certaines actions qu'ils ont effectuées sur l'application, comme un changement de mot de passe ou un achat.

Pour cette fonctionnalité, j'ai choisi d'utiliser Nodemailer, une bibliothèque JavaScript spécialisée dans l'envoi d'e-mails. Les templates des e-mails sont rédigés dans des fichiers Handlebars, ce qui offre une syntaxe similaire au HTML, facilitant ainsi la maintenance des templates.

Nodemailer s'intègre aisément à mon application NestJS, fournissant une interface simple pour l'envoi d'e-mails, qu'ils soient transactionnels, de notification ou de marketing. La flexibilité est au rendez-vous, car Nodemailer prend en charge différents protocoles d'envoi, tels que SMTP, Sendmail et Direct, offrant ainsi le choix du fournisseur de messagerie adapté à mes besoins.

Une des forces de Nodemailer réside dans la personnalisation des e-mails selon les besoins. Je peux facilement personnaliser le contenu, la mise en forme, les pièces jointes et les en-têtes des e-mails. De plus, l'utilisation de modèles d'e-mails avec Handlebars me permet de générer dynamiquement le contenu des e-mails en fonction des données spécifiques à chaque utilisateur.

L'ajout de pièces jointes et d'images dans les e-mails est également possible avec Nodemailer. Cette fonctionnalité me permet d'inclure des fichiers, tels que des factures en format PDF ou des images, dans les e-mails sortants, facilitant ainsi l'envoi de contenus importants et multimédias.

Nodemailer offre une gestion avancée des erreurs liées à l'envoi d'e-mails, ce qui est essentiel pour suivre et gérer efficacement les problèmes éventuels, tels que les échecs d'envoi, les erreurs de connexion au serveur de messagerie ou les problèmes de délivrabilité.

L'intégration de Nodemailer avec des services tiers tels que Gmail, Outlook ou AWS SES (Simple Email Service) est aisée. J'ai configuré Nodemailer pour utiliser les identifiants de mon compte de messagerie existant, ou utiliser des services de messagerie gérés par des fournisseurs cloud, facilitant ainsi l'envoi d'e-mails à grande échelle (déployé en utilisant le SMTP de Gmail).

Enfin, Nodemailer facilite également les tests unitaires de la logique d'envoi d'e-mails grâce à ses fonctionnalités de mock. Cela me permet de simuler l'envoi d'e-mails

pendant les tests, sans réellement les envoyer, rendant ainsi les tests plus rapides, isolés et fiables, sans dépendre d'un serveur de messagerie externe.

En résumé, l'utilisation de Nodemailer dans mon application NestJS pour la gestion de l'envoi d'e-mails a été une décision judicieuse, car elle offre une solution flexible, complète et simple à utiliser, répondant parfaitement aux besoins de mon projet.

Voici des exemple de code utilisant Nodemailer :

- Module

```



```

@Module({
  imports: [
    ConfigModule.forRoot(),
    MailerModule.forRoot({
      transport: process.env.NODE_ENV === 'production' ?
      {
        service: process.env.MAILER_SERVICE,
        host: process.env.MAILER_HOST,
        port: Number(process.env.MAILER_PORT),
        secure: true,
        auth: {
          user: process.env.MAILER_EMAIL,
          pass: process.env.MAILER_PASSWORD,
        },
      } :
      {
        host: process.env.MAILER_HOST,
        port: Number(process.env.MAILER_PORT),
        secure: false,
        auth: {
          user: process.env.MAILER_EMAIL,
          pass: process.env.MAILER_PASSWORD,
        },
      },
      template: {
        dir: __dirname + '/templates',
        adapter: new HandlebarsAdapter(),
        options: {
          strict: true,
        },
      },
    }),
  ],
  providers: [MailService],
  exports: [MailService],
})
export class MailModule {}

```


```

- Service

```
● ● ●
@ Injectable()
export class MailService {
  constructor(private mailerService: MailerService, private configService: ConfigService)
{}
  public async sendSignupMail(reciever: string) {
    await this.mailerService.sendMail({
      to: reciever,
      from: this.configService.get('MAILER_EMAIL'),
      subject: 'Welcome to Travel Tailor',
      template: 'signup',
      context: {
        reciever,
        url: this.configService.get('CLIENT_APP_URL')
      }
    })
  }
}
```

- Template

```
● ● ●
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Welcome to Travel Tailor!</title>
</head>
<body>
  <div class="container">
    <div class="header">
      
      <h1>Welcome to Travel Tailor!</h1>
    </div>
    <div class="content">
      <p>Dear {{reciever}},</p>
      <p>Thank you for joining Travel Tailor, the ultimate travel management platform. We're excited to have you on board!</p>
      <p>With Travel Tailor, you can easily plan your trips, track your itineraries, and stay organized throughout your journey.</p>
      <p>Get started now and explore the world with Travel Tailor!</p>
      <p>
        <a href="{{url}}/signin" class="cta-button">Sign In</a>
      </p>
      <p>If you have any questions or need assistance, feel free to contact our support team at support@travel-tailor.com.</p>
      <p>Happy travels!</p>
      <p>The Travel Tailor Team</p>
    </div>
  </div>
</body>
</html>
```

# Test: Jest

Tout d'abord, Jest offre une syntaxe simple et expressive pour écrire des tests, rendant son utilisation aisée même pour les développeurs moins familiers avec les tests unitaires ou d'intégration. Son approche basée sur des assertions permet de décrire clairement les comportements attendus des différentes parties du code.

En intégrant Jest de manière transparente dans NestJS, il est fourni avec une configuration par défaut adaptée à la plupart des projets. Cela facilite sa mise en place et réduit la nécessité de configurations complexes, permettant ainsi aux développeurs de se concentrer sur l'écriture des tests sans être encombrés par des tâches de configuration fastidieuses.

La capacité de Jest à exécuter les tests de manière parallèle est un avantage considérable dans un projet de gestionnaire de voyage automatisé, où les tests peuvent être nombreux. Cela accélère l'exécution globale des tests et garantit la réactivité du système, contribuant ainsi à une meilleure expérience utilisateur.

Jest propose également des fonctionnalités avancées telles que les mocks, les spies et les snapshots. Ces outils facilitent la création de tests isolés, la simulation de comportements spécifiques et la vérification des résultats attendus. Ils permettent également de tester des composants individuels indépendamment, favorisant ainsi la modularité et une maintenance simplifiée du code.

En tant qu'outil compatible avec les outils d'intégration continue couramment utilisés tels que les GitHub Actions, Jest offre des rapports de test détaillés, une intégration avec des outils de couverture de code et prend en charge les tests en continu. Cela simplifie l'intégration des tests Jest dans un pipeline d'intégration continue, garantissant ainsi une validation continue de la qualité du code tout au long du développement.

- Exemple de test avec Jest:

● ● ●

```
describe('findOne', () => {
  it('should return the taste with the given id', async () => {
    const id = 'example-id';
    const expectedResult: Taste = {
      id: '',
      name: '',
      traveler: new Traveler,
      createdAt: new Date(),
      updatedAt: new Date(),
      deletedAt: null
    }; // Mocked expectedResult

    jest.spyOn(tasteRepository, 'findOneTaste').mockResolvedValue(expectedResult);

    const result = await service.findOne(id);

    expect(result).toBe(expectedResult);
    expect(tasteRepository.findOneTaste).toHaveBeenCalledWith(id);
  });

  it('should throw NotFoundException if an error occurs', async () => {
    const id = 'example-id';

    const errorMessage = 'Error finding taste';
    jest.spyOn(tasteRepository, 'findOneTaste').mockRejectedValue(new Error(errorMessage));
    await expect(service.findOne(id)).rejects.toThrowError(NotFoundException);
    expect(tasteRepository.findOneTaste).toHaveBeenCalledWith(id);
  });
});
```

# Déploiement: Frontend & Backend

J'ai utilisé Vercel pour déployer mon application de gestion de voyage automatisé, qui combine les frameworks NextJS et NestJS. Voici comment Vercel a été un atout majeur pour ce projet :

Tout d'abord, Vercel simplifie grandement le déploiement de l'application. Sa plateforme intuitive prend en charge nativement NextJS et NestJS, ce qui signifie qu'elle comprend les configurations et les dépendances spécifiques à ces frameworks. En connectant simplement mon référentiel Git à Vercel, j'ai pu déployer automatiquement mon application en quelques clics.

Un avantage clé de Vercel est son intégration étroite avec NextJS, étant le créateur de ce framework. Cette intégration optimisée garantit des performances élevées, notamment grâce à la mise en cache automatique des pages et à l'optimisation du rendu côté serveur. Cela se traduit par des temps de chargement rapides et une expérience utilisateur fluide pour mon gestionnaire de voyage automatisé.

La prise en charge des fonctions serverless par Vercel a été particulièrement utile pour certaines fonctionnalités spécifiques de mon application, telles que l'envoi de notifications ou l'intégration avec des services tiers. Ces fonctions serverless peuvent être déployées facilement et mises à l'échelle de manière transparente en fonction des besoins de mon application.

Avec son réseau de distribution de contenu (CDN) mondial, Vercel assure un déploiement global de mon application. Cela permet de garantir des temps de réponse rapides et une disponibilité élevée, essentiels pour un gestionnaire de voyage automatisé, dont les utilisateurs sont répartis à travers le globe.

Les fonctionnalités d'aperçu et de collaboration de Vercel ont grandement facilité le processus de révision et de validation des modifications apportées à mon application. Les liens temporaires d'aperçu permettent de prévisualiser les changements avant leur déploiement, tandis que la collaboration permet à plusieurs développeurs de travailler simultanément sur des branches distinctes et de les prévisualiser avant la fusion.

Quant à la partie envoi d'e-mails, bien que Vercel n'offre pas de partie SMTP, j'ai opté pour l'envoi des e-mails via le SMTP de Gmail pour sa simplicité, tout en gardant à l'esprit que d'autres solutions sont également envisageables.

# Déploiement: Base de donnée

Supabase offre une interface conviviale qui facilite grandement la gestion de la base de données PostgreSQL. Les outils visuels qu'il propose permettent de créer des tables, de définir des relations, de gérer les autorisations et d'effectuer des requêtes SQL de manière intuitive. Grâce à cela, le processus de développement et de maintenance de la base de données a été simplifié.

Un autre avantage clé de Supabase est sa scalabilité élastique. Il a été conçu pour être hautement évolutif, ce qui signifie que j'ai pu faire évoluer ma base de données en fonction des besoins de mon gestionnaire de voyage automatisé. La possibilité d'ajuster les ressources de la base de données en fonction de la charge de travail, sans compromettre les performances ou la disponibilité, s'est avérée très utile.

Supabase génère automatiquement une API RESTful pour la base de données PostgreSQL, ce qui m'a permis d'accéder facilement aux données depuis mon application.

La sécurité des données est également une priorité avec Supabase. J'ai pu mettre en place des mécanismes avancés d'authentification et d'autorisation pour protéger mes données. Supabase prend en charge la définition de rôles et de permissions granulaires pour accéder aux données, et il chiffre les données en transit et au repos, ce qui garantit la confidentialité et l'intégrité des informations stockées.

## Evolution: Frontend

Pour toute cette partie, il y aura certainement de nouveaux packages dans le workspaces notamment pour les composants. Si le projet devait être repris par une équipe de plusieurs personnes dont des Ux-Ui mais aussi des développeurs, j'aurais ajouté une partie designOps avec des design tokens afin de monter une design API.

Il y aura aussi une refonte de la partie frontend sera à prévoir.

À noter qu'il y aura aussi une réécriture partielle ou totale de cette partie du projet.

Enfin, il est prévu que je rajoute Redux sur le projet lors de la réécriture.

## Evolution: Backend

Pour ce qui est de la partie backend, j'ai prévu de décomposer l'API en micro-services afin d'améliorer la scalabilité de l'API.

L'algorithme de création de planning serait à revoir. Je pourrais essayer d'implémenter une IA déjà existante, ce qui m'allégera le code.

Ou bien, je pourrais essayer de créer ma propre IA. Là, je n'ai pas encore les compétences pour le faire, mais cela pourrait être un beau challenge.

## Evolution: Mobile

Pour ce qui est de la partie mobile, elle sera nécessaire en fonction des résultats qui vont être apportés par le frontend grâce à Google Analytics. Si les KPI qui ressortent Google Analytics m'indique que le site est plus utilisé sur mobile, il y aura certainement une application mobile à créer. Elle pourrait être développée soit en React Native pour compléter le workspace actuel et ne pas avoir à tous réécrire tout le code. Soit en Flutter pour avoir une application multiplateforme scalable avec une seule codebase à maintenir.

## Evolution Globale

Pour ce qui est des évolutions globales, il y aura d'abord une couverture de tests plus importante.

Il y aura aussi différents abonnements avec des prix différents pour que chaque annonceurs est une offre adapté à ces besoins et à son budget.

Ayant déjà mis en place à un système de note sur les activités, je pourrais les réutilisés pour créer un système d'activités recommander pour que l'utilisateur est des activités plus affines en fonction de ces goûts.

En bref, il y a encore plein d'idées dans les cartons pour continuer à faire grossir le projet, mais avant ça il y aura une réécriture avec une grosse refactorisation obligatoire pour continuer à faire grandir le projet sur des bases saines.

## Conclusion

Pour conclure, ce projet m'a apporté beaucoup de connaissances, mais aussi une autre manière de travailler et de percevoir la conception de gros projet. Avec le recul, je me dis que certaines choses auraient pu être faites différemment.

Même si dans l'ensemble le maître mot pour réussir ce projet était de rester organisé.

Pour sortir cette première version, j'ai dû faire des choix, afin d'avoir un MVP (Minimum Viable Product) convaincant et correspondant à mes attentes, il y a des processus qui on était mis en pause et d'autres qui on était repensés.

Ce projet était aussi l'occasion d'avoir un challenge personnel, pour sortir un projet avec autant de fonctionnalités et de processus. S'il y a quelques mois, quelqu'un me montrait le résultat, je pense que j'aurais été un peu époustouflé de tout ce qui a pu être mis en place comme le système d'envoi d'email, le système de paiement ou encore le système d'upload de fichiers. Ces trois fonctionnalités, d'apparence très compliquées, ont été moins compliqués à mettre en place que prévu.

Même si je ne le cache pas, certaines fonctionnalités ont été compliquées à mettre en place comme l'algorithme de création de planning.

Mais, une fois le problème décomposés en petites tâches, le cheminement paraît plus accessible.

Pour finir, ce projet m'a fait grandir au niveau de mes compétences, de mon workflow mais aussi dans ma manière d'aborder les nouvelles problématiques. Ce qui va me permettre de pouvoir aborder de nouveaux gros projets plus sereinement.