Dossier projet

Bachelor Concepteur - Développeur d'applications

Bryan **HOUBLON**
Étudiant à l'ETNA
Développeur front-end à Razorfish

Vue synoptique de l'emploi-type

N° Fiche AT	Activités types	N° Fiche CP	Compétences professionnelles
1	Développer une application sécurisée	1	Installer et configurer son environnement de travail en fonction du projet
		2	Développer des interfaces utilisateur
		3	Développer des composants métier
		4	Contribuer à la gestion d'un projet informatique
2	Concevoir et développer une application sécurisée organisée en couches	5	Analyser les besoins et maquetter une application
		6	Définir l'architecture logicielle d'une application
		7	Concevoir et mettre en place une base de données relationnelle
		8	Développer des composants d'accès aux données SQL et NoSQL
3	Préparer le déploiement d'une application sécurisée	9	Préparer et exécuter les plans de tests d'une application
		10	Préparer et documenter le déploiement d'une application
		11	Contribuer à la mise en production dans une démarche DevOps

Sommaire

L’entreprise

J’ai effectué mon alternance au sein de l’agence digitale nommée Razorfish (groupe Publicis). L’agence a plusieurs clients et accompagne le client dans sa transformation digitale axée vers une expérience utilisateur accrue et idéale. Ajouter choses

Introduction

Dans un contexte où la transformation numérique bouleverse les usages quotidiens, la conception d’applications mobiles représente un levier stratégique pour répondre aux besoins utilisateurs. Les entreprises comme les particuliers attendent des solutions fiables, intuitives, sécurisées, capables de simplifier et optimiser la gestion de leurs activités.

Cette dynamique s’inscrit au cœur de mon parcours de formation en Conception et Développement d’Applications à l’ETNA, une école qui valorise la pratique, l’autonomie et l’innovation au service de la performance numérique.

Porté par ces dynamiques, le projet présenté ici représente l’aboutissement d’une réflexion globale, comment allier la rigueur technique, la sécurité et une bonne expérience utilisateur tout en répondant à un besoin réel ? Au travers de ce dossier, je détaillerai les étapes de conception, les choix techniques et les pratiques mises en œuvre pour construire une solution respectueuse des standards de qualité, de sécurité et de performance.

Présentation du projet

Problématiques et naissance du projet

Today, the majority of the services we use are based on a subscription model, insurance, mobile or internet plans, streaming platforms, health insurance and many others. These expenses, often monthly, quarterly or annually, accumulate over time and become increasingly difficult to track. For many users, it’s challenging to have a comprehensive and accurate view of all the deductions. This complicated management leads to a lack of visibility over the monthly budget, potential financial disorganization and sometimes even forgotten payments.

Solution

In response to this challenge, I wanted to design Subly, an intuitive and secure mobile application that provides a clear and modern solution to this growing need. Subtly offers users a comprehensive and practical tool to centralize the management of their subscriptions and recurring payments.

The application enables each user to view all their recurring expenses in the form of an interactive calendar with the possibility to organize them by category (such as phone, health insurance, streaming, etc) for better clarity. Thanks to this clean and fluid interface, users gain an overview of their monthly and yearly budgets allowing them to anticipate and better control their finances.

Subtly goes beyond simply listing deductions. It also offers analysis and statistical tools that help users identify their main areas of spending, detect unnecessary or costly subscriptions and make informed decisions. The application incorporates smart notifications to remind users of upcoming due dates and prevent any missed payments.

Above all, Subly is designed for daily use and tailored to the needs of young adults and digital-native users. It aims to be simple to use while integrating essential security standards (secure authentication, data encryption, etc).

Finally, in a more advanced version, Subtly aims to go even further by offering bank synchronization, automatically retrieving payment information and providing an even more seamless and automated experience.

Présentation technique

Pour répondre aux besoins de performances, de fiabilité et de sécurité, l'architecture de l'application repose sur un modèle en microservices. Cette approche permet de découper l'application en plusieurs services autonomes, chacun dédié à un domaine fonctionnel spécifique. Elle présente de nombreux avantages, une meilleure évolutivité, une maintenance facilitée et une capacité à isoler les composants afin de réduire les risques liés aux pannes. Cette flexibilité technique est essentielle pour garantir la robustesse et l'évolutivité de la solution sur le long terme.

Pour la partie Front-end, le développement mobile a été réalisé à l'aide de React Native, un framework permettant de concevoir des interfaces cross-platform pour Android et iOS en utilisant une base de code unique. L'interface utilisateur est pensée pour être simple, ergonomique et accessible, avec l'intégration de NativeWind CSS pour un design cohérent et moderne. La navigation est assurée par Expo Router facilitant ainsi la structuration des écrans et l'expérience utilisateur.

L'application offre un calendrier interactif permettant de visualiser l'ensemble des prélèvements à venir et de consulter la liste des abonnements actifs. Ces fonctionnalités sont complétées par la possibilité de filtrer et de trier les dépenses par catégorie ainsi que par l'affichage de statistiques et de graphiques permettant une meilleure compréhension de la répartition et de l'évolution des dépenses. Enfin, un système de notifications a été intégré pour rappeler les échéances à venir et éviter les oublis.

La partie Back-end de l'application est développée en Nest.js , un framework Node.js moderne et modulaire particulièrement adapté à la création d'API REST robustes et performantes. Le choix de Nest.js s'appuie sur ses nombreux atouts, une architecture claire, une grande modularité et compatibilité native avec les standards de sécurité web. Le backend gère l'authentification des utilisateurs grâce au mécanisme des JSON Web Tokens (JWT), garantissant ainsi la confidentialité et l'intégrité des échanges entre l'application mobile et le serveur.

L'architecture microservices repose sur une séparation des responsabilités, chaque service est conçu pour répondre à un besoin fonctionnel précis, tel que la gestion des utilisateurs, la gestion des abonnements ou encore la gestion des notifications. Cette découpe permet d'optimiser les performances, de réduire les interdépendances et de faciliter les mises à jour ou les évolutions futures. Les services communiquent entre eux via des interfaces API REST, ce qui assure un découplage fort et une évolutivité accrue de l'ensemble du système.

Pour le stockage des données, le projet s'appuie sur PostgreSQL, un système de gestion de base de données relationnelle reconnu pour sa fiabilité et sa robustesse. L'hébergement est assuré par Supabase, une solution managée qui garantit la sécurité, la disponibilité et la pensée pour assurer l'intégrité des données, avec la mise en place des relations appropriées, de clés étrangères et d'index optimisant les performances.

La sécurité des données est une priorité, des bonnes pratiques sont mises en place pour sécuriser les accès à la base, protéger les informations sensibles et prévenir les vulnérabilités telles que les injections SQL. L'ensemble de ces choix techniques contribue à la conformité de l'application avec les exigences de sécurité et de confidentialité des données des utilisateurs.

Le projet adopte une démarche DevOps avec la mise en place d'outils et de processus visant à garantir la qualité et la continuité des livraisons. Docker est utilisé pour la conteneurisation des différents microservices, permettant ainsi de déployer l'application dans un environnement homogène et maîtrisé, tout en facilitant les mises à jour et la maintenance.

L'automatisation des tests et des déploiements est assurée par des pipelines CI/CD mis en place par GitHub Actions. Cette approche garantit un développement plus rapide, plus fiable et une meilleure traçabilité des évolutions apportées au projet. Le backend est hébergé sur Heroku, une plateforme qui offre à la fois flexibilité et performance, tandis que la base de données est prise en charge par Supabase, assurant une haute disponibilité et une gestion optimisée des accès.

Développer une application sécurisée

Le développement d'une application sécurisée a nécessité une approche rigoureuse et structurée en conformité avec les exigences de sécurité, de qualité et de performance définies par l'entreprise et la réglementation en vigueur. Cette démarche s'inscrit dans les recommandations de l'ANSSI pour la sécurité informatique et respecte les principes du RGPD en matière de protection des données personnelles.

Installer et configurer son environnement de travail en fonction du projet

La mise en place de l'environnement de travail a débuté par la définition et l'installation des outils nécessaires pour créer un cadre de développement stable, cohérent et sécurisé. Cela a permis de garantir l'efficacité et la fiabilité des développements tout en respectant les bonnes pratiques professionnelles.

Environnement Git

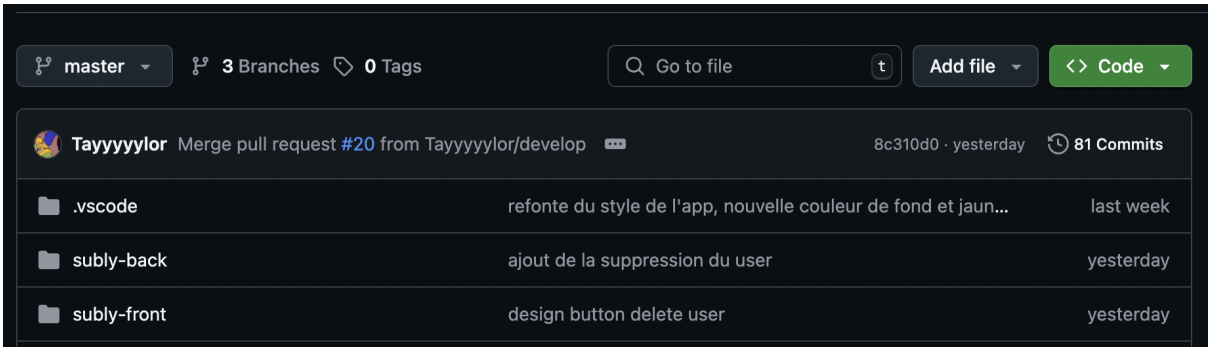
Pour structurer le développement collaboratif et optimiser la gestion du repository, une organisation GitHub a été mise en place. Ce choix permet une centralisation efficace des

projets, une gestion des membres simplifiée et une meilleure visibilité des contributions, tout en facilitant l'intégration de workflows automatisés et la sécurité des dépôts.

Pour ce projet, un monorepo a été privilégié. Cette approche consiste à regrouper l'ensemble des composants de l'application (frontend, backend, scripts d'infrastructure, etc) au sein d'un seul dépôt Git. Le choix du monorepo présente plusieurs avantages significatifs, il centralise l'ensemble du code et des dépendances dans un seul espace de travail, ce qui facilite la cohérence globale du projet et réduit les risques de divergence entre les différents modules.

Le monorepo offre également une meilleure gestion des versions et des évolutions, les mises à jour ou les correctifs peuvent être appliqués de manière simultanée à tous les composants, garantissant une intégration fluide et rapide. Cette architecture simplifie par ailleurs l'automatisation des tests et des déploiements (via des workflows CI/CD), permettant d'assurer une qualité continue et une plus grande sécurité.

Enfin, le monorepo facilite la collaboration entre les développeurs en leur offrant une vue d'ensemble unique et en simplifiant la gestion des branches et des workflows Git, contribuant ainsi à une meilleure productivité et à un suivi des contributions plus clair.



Une mise en place d'environnement a été fait, la branche master (branche de production) et la branche develop (branche de développement et de test).

Initialisation Back et Front

Pour le développement de l'application mobile, j'ai choisi de m'appuyer sur React Native et l'outil Expo afin de simplifier les premières phases de prototypage et de déploiement sur les terminaux Android et iOS.

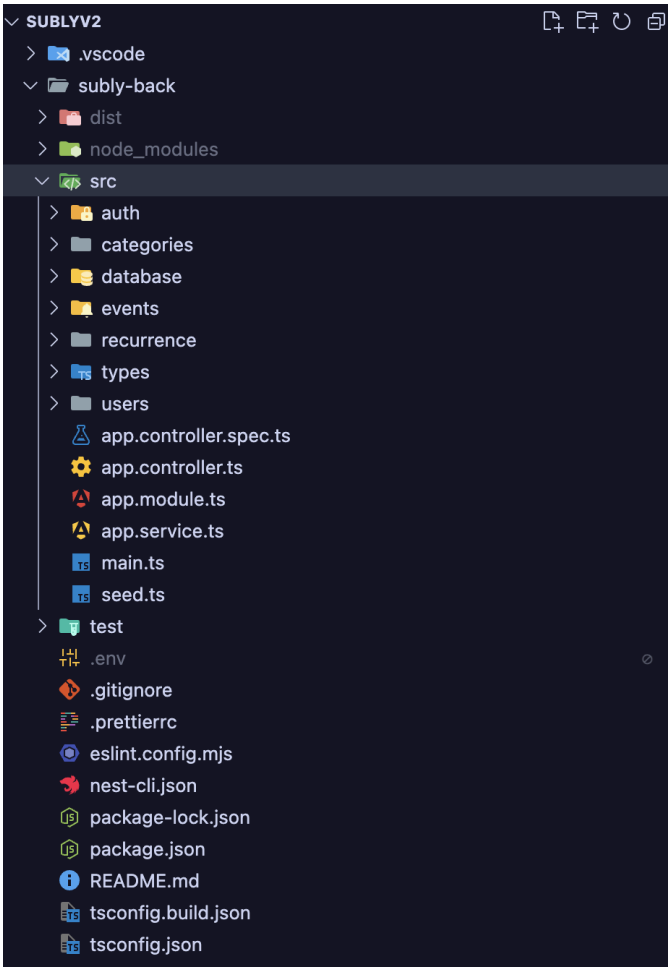
Une fois le projet initialisé, j'ai installé les dépendances nécessaires à la gestion de la navigation avec Expo Router, ainsi que le framework de styles NativeWind (Tailwind CSS).

Configuration du fichier .gitignore, tailwind.config.js.

Pour garantir la qualité et l'uniformité du code, j'ai mis en place les outils ESLint et Prettier dès le début du projet. Ces outils permettent de détecter les erreurs de syntaxe et d'assurer le respect d'un style de code homogène au sein du projet.

Le backend a été développé avec Node.js et le framework Nest.js, un choix particulièrement adapté aux projets structurés et évolutifs. Nest.js offre une architecture modulaire et en couches qui favorise la lisibilité du code, la réutilisation des composants et la mise en place de bonnes pratiques de développement notamment en matière de sécurité et de performances. Il est également conçu pour gérer efficacement les projets de type API RESTful en intégrant nativement des fonctionnalités de validation, de gestion des requêtes HTTP et de sécurisation des échanges.

Pour organiser le code source, la structure suivante a été adoptée :



/src : Ce dossier constitue le coeur de l'application. Il est organisé en sous-dossiers correspondant aux fonctionnalités ou modules métier (gestion des utilisateurs, abonnements, ets). Chaque dossier regroupe les contrôleurs, services et entités associés, conformément à l'architecture en couches de Nest.js. Cette organisation favorise l'isolation des responsabilités et facilite l'évolution du projet.

/types : Ce répertoire regroupe les définitions de types utilisées dans le projet. L'adoption de Typescript pour le typage fort permet de renforcer la sécurité du code et de limiter les erreurs potentielles lors de la manipulation des données.

/test : Ce dossier contient les tests unitaires et les test d'intégration, assurant la fiabilité et la robustesse des fonctionnalités développées. Les tests sont essentiels pour garantir la qualité de l'application et pour faciliter la détection et la correction des éventuels dysfonctionnements.

Développer des interfaces utilisateur

Front NativewindCSS

Dans le cadre du développement de cette plateforme, j'ai choisi d'utiliser React Native associé à NativeWind (intégration de Tailwind CSS) pour la création de l'interface utilisateur. React Native est une technologie reconnue pour sa capacité à produire des applications mobiles performantes et multiplateformes (Android et iOS) à partir d'un seul code source en Javascript. Cette approche permet de réduire le temps de développement et de simplifier la maintenance en partageant la logique métier et les composants graphiques sur l'ensemble des plateformes.

NativeWind vient compléter React Native en apportant les avantages de Tailwind CSS dans l'univers mobile, une approche utilitaire qui permet de créer des interfaces modernes, épurées et cohérentes, tout en accélérant significativement le développement grâce à la simplicité et la clarté des classes Tailwind.

Cette combinaison offre ainsi un équilibre parfait entre rapidité de développement, cohérence graphique et performances optimisées, tout en garantissant une expérience utilisateur de qualité.

Structure du projet :

/components : Ce dossier regroupe l'ensemble des composants visuels réutilisables. Chaque fonctionnalité de l'application est organisée dans un sous-dossier dédié, facilitant ainsi la gestion et l'évolution des différentes parties de l'interface utilisateur.

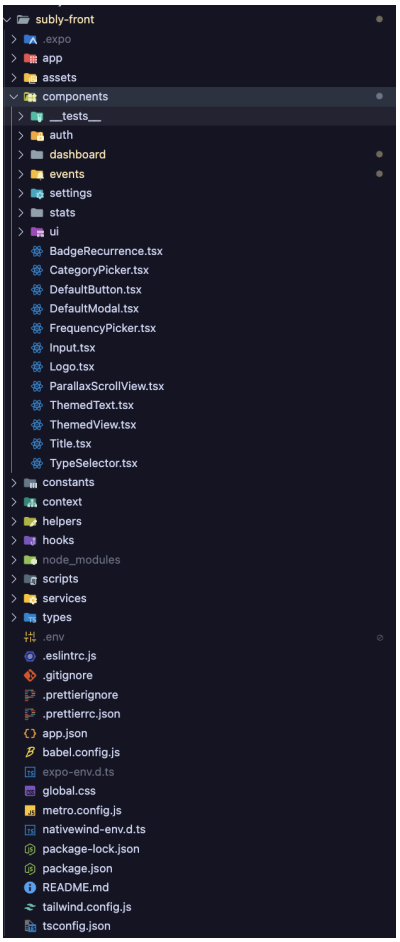
/constants : Ce répertoire centralise les constantes de l'application (valeurs fixes, couleurs, polices, marges, etc). Cette organisation permet de maintenir une cohérence visuelle et fonctionnelle sur l'ensemble des écrans et d'adapter facilement les styles en cas d'évolution des besoins.

/helpers : Les fonctions utilitaires partagées par plusieurs composants sont placées dans ce dossier. Elles facilitent la factorisation du code et réduisent les duplications, contribuant ainsi à la clarté et à l'efficacité du projet.

/services : Ce dossier regroupe les fonctions qui assurent la communication avec le backend (appels API, gestion des données, etc). Cette séparation des responsabilités permet de maintenir une architecture claire et facilite l'implementation des évolutions futures.

/types : Ce répertoire contient les définitions de types TypeScript utilisés pour structurer les données manipulées dans l'application. Cette pratique renforce la sécurité du code et limite les erreurs potentielles.

/script : Les scripts spécifiques à l'automatisation des tâches (par exemple les scripts de build, de génération de fichiers ou d'analyse de performance) sont regroupés dans ce dossier. Cette organisation contribue à la maintenabilité et à l'efficacité du projet.



Manipulation du DOM

Le DOM (Document Object Model) est une interface de programmation pour les documents HTML, XML et SVG. Il représente la structure d'un document sous forme d'un arbre hiérarchique où chaque élément (comme une balise HTML) est un nœud. Cela permet aux développeurs d'accéder, de manipuler et de modifier dynamiquement le contenu, la structure et le style d'une page.

Une étape clé du parcours utilisateur est l'affichage des informations d'une transaction, comme le montre l'exemple ci-dessous, où est présenté une transaction :

Dans le cadre du développement avec Typescript et React Native, j'ai utilisé la force de React pour gérer l'affichage des données et interagir avec le DOM. Grâce à Typescript, nous avons pu bénéficier du typage statique, ce qui a facilité la détection d'éventuelles erreurs et amélioré la robustesse du code. Grâce à Typescript, nous profitons également d'une meilleure intégration avec les outils de développement modernes. Vous trouverez ci-dessous le fichier EventDetails.tsx, dans lequel ces concepts sont appliqués pour gérer l'affichage dynamique et l'interaction avec les données de la transaction.

```tsx
const EventDetails = () => {
  const { id } = useLocalSearchParams();      Tayyyyylor, le mois dernier
  const router = useRouter();
  const [event, setEvent] = useState<EventType | null>(null);
  const isExpense = event?.type === 'EXPENSE';

  const formatDate = (date: Date | undefined) => {
    if (!date) return '';
    return format(new Date(date), 'dd MMMM yyyy', { locale: fr });
  };

  const displayFrequency = event?.recurrence?.frequency
    ? translateFrequency(event?.recurrence?.frequency).toLowerCase()
    : undefined;

  const infos = [
    {
      label: 'Fréquence',
      value: displayFrequency || '',
      hasBorder: true,
    },
    {
      label: 'Catégories',
      value: event?.category?.icon + ' ' + event?.category?.name || '',
      hasBorder: true,
    },
    {
      label: 'Date de début',
      value: formatDate(event?.startDate),
      hasBorder: true,
    },
    {
      label: 'Date de fin',
      value: formatDate(event?.endDate),
      hasBorder: false,
    },
  ];
```

```tsx
return (
  <SafeAreaView className="relative flex-1 items-center p-4">
    <View className="flex-row justify-between items-center w-full p-4 mt-5">
      <Pressable onPress={handleClickBack}>
        <Text className="text-white text-[18px] font-bold">Retour</Text>
      </Pressable>
      <Pressable
        className="bg-[#FBBF24] p-3 rounded-[5px]"
        onPress={handleClickEdit}
      >
        <Text className=" text-[18px] font-bold">Modifier</Text>
      </Pressable>
    </View>
    <Text className="text-white text-[24px] font-bold mb-5">
      {event?.name}
    </Text>
    <View
      className={` ${isExpense ? 'bg-red-300' : 'bg-green-300'} p-2 rounded-[8px] mb-3`}
    >
      <Text
        className={`${isExpense ? 'text-red-800' : 'text-green-800'} text-[18px] font-bold`}
      >
        {isExpense ? `-${event?.amount}€` : `+${event?.amount}€`}
      </Text>
    </View>
    <View className="flex-column w-[95%] rounded-[5px] bg-gray-800 mt-[30px]">
      {infos.map((info, index) => (
        <View
          key={index}
          className={`flex-row justify-between w-full ${info.hasBorder ? 'border-gray-600 border-b' : ''} p-4`}
        >
          <Text className="text-[#FBBF24] text-[18px] font-bold">
            {info.label}
          </Text>
          <Text className="text-white text-[18px] font-bold">
            {info.value === '' || info.value === undefined ? '-' : info.value}
          </Text>
        </View>
      ))}
    </View>
```

Développer des composants métier

La création des composants métier a constitué l'étape centrale du développement, en apportant une réponse directe aux besoins fonctionnels des utilisateurs. Chaque composant a été conçu dans le respect des principes de séparation des responsabilités et de sécurité des données.

Création de formulaire côté front-end

J'ai décidé de prendre en exemple le composant Signin.tsx.
La syntaxe TSX permet d'intégrer du TypeScript à ce qui est rendu par le React DOM, ici nous avons toute la logique du composant qui se découpe en plusieurs parties :

```tsx
const Signin = () => {
  const router = useRouter();
  const { signIn } = useAuth();

  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [showPassword, setShowPassword] = useState(false);

  const inputData = [
    { placeholder: 'email', value: email, onChangeText: setEmail },
    {
      placeholder: 'Mot de passe',
      value: password,
      onChangeText: setPassword,
      id: 'password',
      secureTextEntry: !showPassword,
      showPassword,
      togglePassword: () => setShowPassword(!showPassword),
    },
  ];

  const handleLogin = async () => {
    if (!email || !password) {
      Alert.alert('Erreur', 'Tous les champs sont obligatoires.');
      return;
    }
    ⌘L to chat, ⌘K to generate
    try {
      const userData = {
        email,
        password,
      };
      await loginUser(userData, signIn);
      router.replace('/(tabs)');
    } catch (error) {
      Alert.alert('Erreur', 'Impossible de se connecter.');
    }
  };

  const handleRedirectSignUp = () => {
    router.replace('/signup');
  };
```

Création d’un formulaire

Pour avoir un contrôle complet des données, le formulaire est fait de façon personnalisée, l’utilisation de composants Input ou autres qui permet aux utilisateurs de sélectionner leurs choix.

Gestion de la soumission du formulaire

Les informations du formulaires sont stockées dans des états React, qui seront ensuite envoyé au backend via une fonction asynchrone qui appelle le service crée dédié aux users On vérifie la validité des données envoyés au back et on redirige si la connexion est success.

```tsx
export const loginUser = async (
  userData: User,
  signIn: (token: string) => void,
) => {
  try {
    const response = await axiosInstance.post('/auth/login', userData);
    const token = response.data.access_token;
    if (!token) throw new Error('Token manquant dans la réponse API');
    signIn(token);
    router.replace('/(tabs)');
  } catch (error: any) {
    console.error(
      'Erreur de connexion :',
      error.response?.data || error.message,
    );      Tayyyyylor, il y a 3 mois • add connexion front/back and spread c
    throw error;
  }
};
```

Maintenant que nous avons détaillé la partie “logique” du composant, il faut également parler de la manière dont est rendu la vue :

```
return (
  <SafeAreaView className="flex-1 mt-[100px]">
    <View className="flex-col items-center gap-1 mb-[20px]">
      <Logo />
      <Title label="Connexion" />
    </View>
    <View className="p-10 gap-5 ">
      {inputData.map((input, index) => {
        return (
          <Input
            secureTextEntry={input?.id === 'password' ? true : false}
            placeholder={input.placeholder}
            key={index}
            onChangeText={input.onChangeText}
            value={input.value}
            showPassword={input.showPassword}
            togglePassword={input.togglePassword}
          />
        );
      })}
    </View>
    <View className="flex-col items-center gap-3 p-10">
      <ButtonAuth
        label="Se connecter"
        onPress={handleLogin}
        text="Pas encore inscrit ?"
        labelSecondButton="Créer un compte"
        onPressSecondButton={handleRedirectSignUp}
      />
    </View>
  </SafeAreaView>
);
};
```

inputData : on map un tableau avec les labels des champs pour éviter de se répeter, pour chaque élément du tableau, on retourne un composant personnalisé Input.
onChangeText : permet de détecter quand le texte de l'input change.
showPassword : état booléen pour permettre de voir le mot de passe en clair.

Création de composants back-end

Dans notre projet, le backend joue un rôle central en prenant en charge des fonctionnalités essentielles au bon fonctionnement de l'application telles que la gestion de l'authentification et les opérations CRUD (Create, Read, Update, Delete) pour nos différents modèles de données.
Développé avec Nest.js, ce backend repose sur une architecture modulaire qui facilite la maintenabilité, la collaboration entre développeurs et garantit un haut niveau de sécurité et de performance.
Notre organisation du code est pensée pour refléter une structure claire et logique.
Chaque fonctionnalité a son propre dossier avec tout les fichiers nécessaires à son bon fonctionnement :

Dto : contient les Data Transfer Objects qui définissent la structure des données échangées entre le client et le serveur. Ces fichiers assurent la validation des entrées et la cohérence des données manipulées contribuant ainsi à la sécurité globale de l'application.

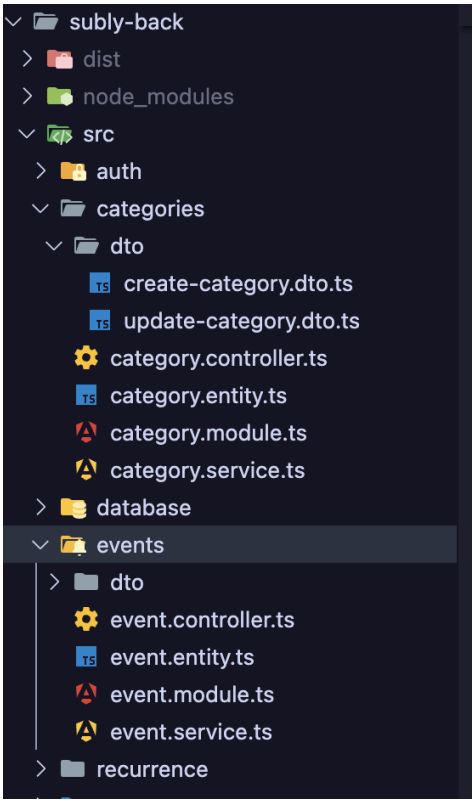
Controller : regroupe les contrôleurs responsables de la gestion des requêtes HTTP entrantes. Ils assurent le lien entre les utilisateurs finaux et la logique métier en appelant les services adéquats pour exécuter les opérations demandées.

Entity: contient les entités qui représentent les tables de la base de données. Ces entités définissent les attributs et les relations entre les différents modèles, facilitant ainsi les échanges avec la couche de persistence des données.

Module : Chaque module regroupe l'ensemble des composants liés à une fonctionnalité spécifique. Il déclare les dépendances, les services et les contrôleurs permettant ainsi une organisation claire et une isolation fonctionnelle conforme aux bonnes pratiques de développement.

Service : Regroupe les services qui contiennent la logique métier. Ces fichiers centralisent les traitements (calculs, accès aux données, appels externes, etc) et assurent l'implémentation des fonctionnalités attendues en veillant à respecter les contraintes de sécurité et de performance.

Cette architecture modulaire et en couches permet une séparation nette des responsabilités renforçant la lisibilité du code et la facilité de maintenance. Elle s'inscrit également dans une démarche de qualité et de sécurité conformément aux recommandations de l'ANSSI et aux exigences du projet.



Contribuer à la gestion d’un projet informatique

Choix d’une méthode de gestion de projet

Dans le cadre du développement de Subly, j’ai opté pour une approche dîtes agile, basée sur la méthodologie Scrum. Cette méthodologie offre une structure flexible et itérative, permettant de gérer efficacement les projets complexes en les divisant en plusieurs itérations appelés “sprints”. Chaque sprint dure en général deux à quatre semaines. Pour ce projet, chaque sprint dure exactement deux semaines et se concentre sur la réalisation des objectifs fixés à l’avance. La méthodologie agile suit le processus suivant :

- Planification des objectifs : Au début, ou les jours précédant un sprint, une réunion de planification est organisée pour définir les objectifs du sprint suivant et sélectionner les tâches à réaliser.
- Réunions hebdomadaires : Des réunions hebdomadaires sont tenues pour discuter de l’avancement du projet et pallier aux éventuels obstacles rencontrés.
- Revue de fin de sprint : La revue de fin de sprint est organisée pour examiner les fonctionnalités développées et obtenir des retours d’utilisation. Elle permet également de définir si l’état d’avancement global du projet prend du retard ou non.

Outil de suivi et de gestion

La réalisation de ce projet a été menée dans un cadre structuré en respectant les principes fondamentaux de la gestion de projet et les contraintes liées aux délais, à la quantité et aux coûts.

roadmap et diagramme de gantt

Le suivi du projet a été organisé grâce à des outils de gestion de projet tels que Trello permettant de créer et d'identifier les tâches, d’en suivre l’avancement et de prioriser les développements.
Notion, pour la documentation et une centralisation des informations nécessaire au bon fonctionnement de l’application.
Cette organisation a facilité la traçabilité des décisions prises et permis d’adapter la planification en fonction des éventuels aléas techniques.

La gestion des versions et des évolutions a été assurée par l'utilisation de Git, qui a permis de structurer le dépôt en branches spécifiques pour chaque fonctionnalité majeure. Cette méthode a facilité l'intégration des nouvelles fonctionnalités et assuré un historique clair des modifications.

Enfin, la démarche qualité a été au cœur du projet, des tests automatisés ont été mis en place pour valider les fonctionnalités critiques et une pipeline CI/CD a été configuré à l'aide de github Actions pour automatiser les phases de build, de tests et de déploiement. Cette approche a permis de garantir la fiabilité et la stabilité de l'application à chaque étape de son développement.

Concevoir et développer une application sécurisée organisée en couches

La conception et le développement d'une application sécurisée organisée en couches constituent un volet fondamental de mon projet alliant à la fois rigueur technique et écoute attentive des besoins des utilisateurs. Ce travail s'inscrit pleinement dans les recommandations de l'ANSSI pour la sécurité informatique ainsi que dans le respect du RGPD et des principes d'éco-conception et d'accessibilité (RGAA).

Analyser les besoins et maquetter une application

Dans un environnement où les services numériques sont omniprésents, la gestion des abonnements et des prélèvements récurrents s'impose comme un enjeu majeur pour les utilisateurs. De la téléphonie mobile aux plateformes de streaming en passant par les assurances et les mutuelles. Les dépenses récurrentes sont devenues incontournables et se multiplient de manière exponentielle. Cette réalité engendre une complexité croissante, il devient difficile de suivre l'ensemble des abonnements, de prévoir les échéances à venir et d'avoir une vision claire de l'impact financier global.

Pour beaucoup d'utilisateurs, cette gestion éparse et dispersée se traduit par une perte de visibilité sur le budget mensuel, une désorganisation potentielle et parfois même des oublis de paiements qui peuvent entraîner des frais supplémentaires ou des interruptions de service.

C'est pour répondre à cette problématique concrète que j'ai conçu Subly, une application mobile intuitive, moderne et sécurisée, dédiée à la centralisation et à l'optimisation de la gestion des abonnements. Subly ambitionne de simplifier la vie quotidienne des utilisateurs en leur offrant une solution complète et fiable pour visualiser, catégoriser et anticiper l'ensemble de leurs prélèvements récurrents.

Cahier des charges et besoins

L’objectif principal de Subly est de permettre à ses utilisateurs de reprendre le contrôle sur leurs abonnements et leurs dépenses récurrentes. Pour ce faire, l’application s’articule autour de plusieurs axes essentiels.

- Offrir une visibilité globale sur les prélèvements récurrents en les centralisant dans un calendrier interactif et dynamique.
- Faciliter l’organisation et la gestion grâce à des outils de catégorisation et de filtrage adaptés.
- Aider à la prise de décision avec des analyses visuelles (statistiques, graphiques) pour identifier les postes de dépenses prioritaires.
- Garantir la sécurité et la confidentialité des données des utilisateurs conformément aux réglementations en vigueur (RGPD, recommandations de l'ANSSI).
- Proposer une expérience utilisateur intuitive, accessible et fluide, adaptée aux attentes des jeunes adultes et des utilisateurs connectés.
- Anticiper les besoins futurs avec l'intégration d’outils de synchronisation bancaire pour automatiser la récupération des prélèvements.

L’application Subly est conçue en priorité pour les jeunes adultes et les utilisateurs technophiles qui multiplient les abonnements et utilisent leur smartphone comme outil principal de gestion quotidienne. Elle vise un public soucieux de maîtriser ses finances personnelles, de gagner en visibilité sur ses charges récurrentes et de simplifier la gestion de ses abonnements.

Besoins fonctionnels

Pour répondre aux attentes identifiées, Subly doit intégrer plusieurs fonctionnalités.

La gestion des utilisateurs :

- Création d’un compte utilisateur
- Authentification sécurisée
- Gestin du profil et des paramètres utilisateur

La gestion des transactions :

- CRUD d’une transaction
- Attribution d’une catégorie à chaque abonnement
- Visualisation des abonnements dans un calendrier interactif pour anticiper les échéances.
- Filtrages des transactions pour une navigation rapide et intuitive.

Suivi et statistiques :

- Affichage d'un tableau de bord récapitulatif des dépenses mensuelles, trimestrielles et annuelles.
- Génération de graphiques et d'analyses pour identifier les principaux postes de dépenses et repérer les abonnements inutiles ou coûteux.

Notifications :

- Envoi de notifications intelligentes pour prévenir les utilisateurs des échéances à venir.
- Paramétrage des rappels pour éviter tout oubli de paiement

Sécurité

- Authentification sécurisée via JWT
- Chiffrement des données sensibles
- Gestion des accès et des autorisations selon les profils utilisateurs
- Respect strict des recommandations ANSSI et des obligations du RGPD

Accessibilité et eco-conception

- Interfaces et parcours utilisateur conformes au Référentiel Général d'Amélioration de l'Accessibilité (RGAA)
- Optimisation des performances et des ressources dans une démarche d'éco-conception des services numériques

Perspectives d'évolutions

- Intégration d'une synchronisation bancaire pour récupérer automatiquement les données des prélèvements et offrir une gestion encore plus fluide

Besoins non fonctionnels

- L'application doit être compatible avec Android et iOS
- Les performances doivent garantir une navigation fluide et une réactivité optimale
- L'architecture technique doit assurer la sécurité, la scabilité et la fiabilité du système
- Le projet doit prévoir des mises à jour régulières pour intégrer les évolutions fonctionnelles et corriger les éventuels dysfonctionnements
- La traçabilité des actions et la clarté du code source doivent permettre une évolution future de l'application, facilitée par une architecture modulaire et claire

Contraintes techniques

- Frontend mobile : React Native avec NativeWind (Tailwind CSS)
- Backend : Nest.js avec une base de données relationnelle PostgreSQL
- Conteneurisation : Docker pour l'isolation des environnements
- Automatisation : CI/CD via GitHub Actions pour sécuriser les livraisons
- Notifications : Firebase Cloud Messaging (FCM)
- Gestion de projet : Trello et Notion pour le suivi et la coordination des tâches
- Versioning : Git, structuré en monorepo pour centraliser et coordonner le développement

RGAA et RGPD

Le développement de l'application Subly s'inscrit dans une démarche résolument respectueuse des réglementations et recommandations en vigueur notamment en matière d'accessibilité et de protection des données personnelles. Ces deux volets sont devenus incontournables pour garantir une expérience utilisateur à la fois inclusive, éthique et sécurisée.

RGAA

L'accessibilité numérique constitue aujourd'hui un enjeu majeur de la conception d'applications mobiles, il ne s'agit pas seulement de répondre à des obligations légales mais aussi de permettre a tous les utilisateurs, quelles que soit leurs capacités, de profiter pleinement des fonctionnalités offertes. Le Référentiel Général d'Amélioration de l'Accessibilité (RGAA) fournit un cadre de référence précieux pour atteindre cet objectif.

Dans le cadre de Subly, j'ai porté une attention particulière à la conception des écrans et des parcours utilisateurs pour m'assurer que l'application soit intuitive et simple à utiliser par tous. Cela s'est traduit par un travail sur les contrastes et les couleurs en veillant à garantir une lisibilité optimale des textes et des éléments interactifs quel que soit l'éclairage ambiant ou la capacité visuelle de l'utilisateur. De même, les boutons et les zones de clic ont été pensés pour offrir un confort d'utilisation en évitant toute ambiguïté dans la navigation et en réduisant les risques d'erreurs lors des interactions.

La compatibilité avec les outils d'assistance tels que les lecteurs d'écran a également été intégrée dès les premières phases de conception. Les composants utilisés comme les titres et les zones interactives ont été structurés pour faciliter la lecture par les technologies d'assistance. Chaque action importante est accompagnée d'un retour clair et compréhensible, qu'il agisse d'un message de confirmation ou d'une alerte en cas d'erreur.

Cette démarche de conception accessible sera illustrée dans le dossier par des captures d'écran démontrant la cohérence visuelle et la facilité d'utilisation de l'application pour tous

les profils d'utilisateurs. Ces visuels mettront en lumière les choix ergonomiques réalisés pour assurer une expérience inclusive et respectueuse des recommandations du RGAA.

screen d’une page

RGPD

Au-delà de l’accessibilité, la question de la protection des données personnelles a été placée au cœur du projet. Le Règlement Général sur la Protection des Données (RGPD) impose en effet des obligations strictes pour garantir la confidentialité, la sécurité et la transparence des données collectées.

Dès la conception de Subly, j’ai pris soin de limiter la collecte de données au strict nécessaire en veillant à ne demander que les informations indispensables au fonctionnement de l’application. Cette approche de minimisation des données vise à limiter les risques et à rassurer les utilisateurs quant à l’utilisation qui est faite de leurs informations.

La sécurité des échanges a été assurée par la mise en place de mécanismes robustes, l’authentification sécurisée par JWT garantit l’intégrité et la confidentialité des sessions utilisateurs tandis que les communications entre le client et le serveur sont protégées par un chiffrement approprié. Les rôles et les droits d'accès ont été soigneusement définis pour éviter toute fuite ou utilisation abusive des données conformément aux exigences du RGPD.

La transparence est un autre pilier fondamental de cette démarche. Une page dédiée aux Politique de confidentialité a été intégrée à l’application exposant de manière claire et accessible les finalités de la collecte des données, les droits dont disposent les utilisateurs (accès, rectification, suppression) ainsi que les moyens mis à leur disposition pour les exercer.

Ainsi en intégrant les exigences RGAA et du RGPD dès la phase de conception, Subly s’inscrit dans une démarche de qualité globale qui ne se limite pas aux aspects fonctionnels mais englobe aussi les dimensions éthiques et sociétales du numérique.L’application entend ainsi offrir à ses utilisateurs une expérience complète, respectueuse de leurs droits et de leurs attentes dans un cadre sécurisé et inclusif.

screen des politiques de confidentialité

Création de wireframes et maquettes

Pour accompagner ces aspects fonctionnels, j’ai également travaillé sur l’ergonomie et la structure de l’interface dès les premières étapes du projet. A l’aide de wireframes, j’ai pu concevoir rapidement les bases de l’architecture de l’application, tester les parcours utilisateurs et ajuster l’organisation des éléments en fonction des retours. Ces prototypes simples ont permis de valider la disposition et la fluidité de la navigation avant de passer à l'étape de design détaillé. Ensuite, avec l’outil de conception figma, j’ai pu affiner chaque fonctionnalité, optimiser l'expérience utilisateur et garantir une cohérence visuelle sur l’ensemble de l’application grâce à la charte graphique définie.

Wireframes
Charte graphique
Maquettes.

Définir l’architecture logicielle d’une application

Choix de type d’architecture

Après la phase d’analyse et de maquettage, j’ai défini une architecture logicielle multicouche répartie. Cette architecture repose sur une séparation claire des responsabilités, la couche de présentation(frontend), la couche métier (services applicatifs) et la couche d'accès aux données (service de persistance). Chaque couche a été conçue pour garantir la sécurité, la maintenabilité et la performance de l'application.

Pour la mise en œuvre, jai choisi le framework Nest.js, particulièrement adapté aux architectures modulaires et a la gestion sécurisée des flux de données. Cette structure multicouche permet de limiter les dépendances directes entre les modules et de renforcer la résilience du système face aux incidents ou aux évolutions.

Lors de la définition de l’architecture, j’ai veillé à intégrer les principes d’éco-conception visant à optimiser l’utilisation des ressources et à réduire l’empreinte environnementale de l’application. Chaque couche a ainsi été pensée pour consommer le minimum de ressources tout en assurant la meilleure expérience utilisateur possible.

Enfin, la sécurité a été intégrée dès la conception, j’ai défini les rôles et responsabilités de chaque couche en tenant compte des enjeux de confidentialité, d’authentification et de gestion des droits d'accès en conformité avec les directives de l’ANSSI.

Concevoir et mettre en place une base de données relationnelle

Merise est une méthode d’analyse et de conception des systèmes d’information, largement utilisée dans les projets informatiques pour structurer et organiser le développement des applications. Créée dans les années 1970 en France, cette méthode repose sur une démarche systématique et rigoureuse visant à modéliser les données et les traitements de manière cohérente et exhaustive.

Merise se distingue par sa capacité à représenter les différents aspects d'un projet grâce à une séparation claire entre trois niveaux d'abstraction, le conceptuel, le logique et le physique. Cette approche multi-niveaux permet de passer progressivement d'une compréhension générale des besoins métier à une implémentation technique détaillée tout en garantissant la traçabilité des choix effectués tout au long du projet.

La méthode s'appuie sur plusieurs outils et modèles comme le Modèle Conceptuel de Données (MCD) pour décrire les entités et leurs relations ou encore le Modèle Logique de Données (MLD) et le Modèle Physique de Données (MPD) pour transformer ces concepts en structures adaptées à une base de données.

La méthode s'appuie sur plusieurs outils et modèles, comme le Modèle Conceptuel de Données (MCD) pour décrire les entités et leurs relations, ou encore le Modèle Logique des Données (MLD) et le Modèle Physique des Données (MPD) pour transformer ces concepts en structures adaptées à une base de données.
Utilisée dans des projets allant des applications simples aux systèmes d'information complexes, Merise se révèle particulièrement utile pour structurer le travail des équipes, clarifier les besoins des utilisateurs et assurer une bonne communication entre les différents acteurs d'un projet informatique.

Modèle conceptuel de données

Le MCD constitue la première étape de la modélisation. Il permet de représenter de manière abstraite les principales entités métier, leurs attributs et les relations qui unissent sans se préoccuper des aspects techniques de la base de données.

Pour notre projet, les principales entités identifiées sont :

- Utilisateur : représente les personnes utilisant l'application. Cette entité regroupe des informations essentielles comme l'identifiant, le nom, l'email et les paramètres de profil.
- Transaction : regroupe les informations relatives aux transactions que l'utilisateur souhaite suivre (nom, montant, périodicité, date de début etc).
- Catégorie : permet de classifier les abonnements pour offrir une vision plus claire et organiser des dépenses.
- Récurrence : correspond à chaque occurrence de paiement lié à un abonnement permettant de suivre précisément les flux financiers dans le temps.

Les relations entre ces entités sont clairement identifiées :

- Un utilisateur peut avoir plusieurs transactions
- Chaque transaction appartient à un seul utilisateur et une catégorie
- Une transaction est lié à une fréquence unique

MCD

Modèle logique de données

Le MLD traduit le modèle conceptuel en un schéma plus proche de la logique relationnelle. Il précise les clés primaires, les clés étrangères et les types de données, tout en conservant la structure définie au niveau conceptuel.

Dans notre MLD, chaque entité identifiée dans le MCD devient une table principale :

- La table users comporte les attributs d'identification et de contact. La clé primaire est id_user
- La table transactions contient les informations sur les transactions avec comme clé primaire id_transac et des clés étrangères pointant vers id_users et id_cat
- La table category référence les différentes catégories possibles avec une clé primaire id_category
- La table frequency définit les fréquences disponibles (journalier, mensuel, annuel, etc.), avec une clé primaire id_frequency. Elle est reliée à la table events pour indiquer la périodicité de chaque événement.

Chaque relation est traduite par des contraintes d'intégrité référentielle (clés étrangères) afin de garantir la cohérence des données et d'éviter toute ambiguïté ou perte d'information.

MLD

Modèle physique de données

Le Modèle Physique de Données (MPD) constitue l'étape finale de la conception de la base de données en traduisant le MLD dans un langage concret et en tenant compte des contraintes techniques spécifiques au SGBD choisi, en l'occurrence PostgreSQL.

Le MPD précise les types exacts des données utilisées pour chaque colonne, par exemple, des types VARCHAR pour les chaînes de caractères (comme les noms), DATE pour les dates liées aux transactions récurrentes.

Il définit également la création des index sur les colonnes les plus sollicitées (comme la recherche d'événements par utilisateur ou par catégorie) afin d'optimiser les performances lors des requêtes.

La gestion des droits et des rôles est spécifié dans le MPD pour garantir la sécurité et la confidentialité des données stockées. Cette configuration inclut la définition des permissions pour les utilisateurs ou services accédant à la base de données.

Enfin, le MPD formalise les contraintes d'unicité (par exemple pour garantir l'unicité des emails dans la table users) et les regles de cascading (mise à jour ou suppression en

cascade) qui assurent l'intégrité référentielle lors des modifications ou suppressions de données liées.

MPD

Unified Modeling Language

Dans le cadre de la conception de l'application Subly, j'ai eu recours à la méthode UML (Unified Modeling Language), un langage de modelisation standardisé qui permet de representer graphiquement les differents aspects d'un systeme. UML

Dans le cadre de la conception de l'application Subly, j'ai eu recours à la méthode **UML (Unified Modeling Language)**, un langage de modélisation standardisé qui permet de représenter graphiquement les différents aspects d'un système. UML offre une **approche universelle et structurée**, facilitant la communication entre les membres d'une équipe de développement et offrant une vision claire et cohérente du fonctionnement global de l'application.

UML se distingue par la richesse et la précision de ses diagrammes, qui permettent de modéliser à la fois les **structures statiques** du système (comme les classes et leurs relations) et les **comportements dynamiques** (comme les interactions entre les objets au fil du temps). Cette capacité à représenter à la fois les données et les processus est un atout majeur pour garantir la **cohérence** et la **fiabilité** du projet.

Ce diagramme UML de cas d'utilisation fournit une vue d'ensemble des fonctionnalités clés de Learn-E, organisées en fonction des rôles des utilisateurs et des actions qu'ils peuvent effectuer sur la plateforme. Il sert de base pour structurer les exigences fonctionnelles et s'assurer que le système répond aux besoins des différents types d'utilisateurs.

Diagramme de classes

Le **diagramme de classes** est un outil fondamental d'UML, qui permet de décrire les structures statiques du système. Il représente les différentes **classes** qui composent l'application, ainsi que leurs **attributs**, leurs **méthodes** et les **relations** qui les lient entre elles (comme les associations, les héritages ou les dépendances).

Dans le cas de Subly, le diagramme de classes a permis de structurer les principales entités telles que l'**Utilisateur**, l'**Abonnement**, la **Catégorie** et le **Prélèvement**, en précisant leurs caractéristiques et leurs interactions. Cette représentation graphique facilite la compréhension des données manipulées et permet de vérifier la cohérence de la conception avant même la mise en place de la base de données ou des composants métier.

Le diagramme de classes offre ainsi une vision claire et synthétique de l'**architecture logique** de l'application, servant de référence commune tout au long du développement.

diagramme

Diagramme de séquences

Le **diagramme de séquences**, quant à lui, s'attache à représenter les **interactions dynamiques** entre les différents objets ou composants du système au cours de l'exécution d'un scénario précis. Il décrit la **chronologie** des échanges, en illustrant les messages échangés, l'ordre d'exécution et les interactions nécessaires pour atteindre un objectif donné.

Pour l'application Subly, le diagramme de séquences a été particulièrement utile pour modéliser des cas d'usage tels que la **création d'un nouvel abonnement** ou l'**envoi d'une notification de rappel**. Il met en lumière les échanges entre le frontend (l'utilisateur), les contrôleurs du backend, les services métier et la base de données, offrant ainsi une vision complète de la dynamique de l'application.

Ce diagramme permet de clarifier les responsabilités de chaque composant dans le déroulement d'un processus, de détecter les éventuelles incohérences ou doublons et de s'assurer de la fluidité et de la cohérence de l'**expérience utilisateur**.

diagramme

Développer des composants d'accès aux données SQL et NoSQL

La mise en place des composants d'accès aux données a représenté une étape clé dans le développement de l'application Subly. L'objectif était de garantir à la fois la **fiabilité des échanges avec la base de données** et la **sécurité des informations manipulées**. J'ai adopté une démarche rigoureuse, en veillant à ce que chaque requête soit optimisée, chaque accès sécurisé et chaque exception soigneusement gérée.

Côté SQL, l'accès aux données a été pensé pour s'appuyer sur des requêtes structurées et une logique métier claire. Les services ont été conçus pour centraliser les traitements liés à la persistance, garantissant ainsi la séparation des responsabilités et la cohérence de l'architecture logicielle. Les entités relationnelles (telles que les utilisateurs, les abonnements, les catégories et les prélèvements) ont été liées entre elles par des clés primaires et étrangères, permettant de préserver l'**intégrité référentielle** et d'assurer la cohérence des données à chaque étape du cycle de vie de l'application.

Bien que l'application n'intègre pas directement de bases **NoSQL** dans sa première version, j'ai veillé à ce que l'**architecture logicielle** reste ouverte et évolutive. Cette flexibilité permettra, si nécessaire, d'intégrer ultérieurement des bases NoSQL pour des besoins spécifiques tels que la gestion des notifications en temps réel ou l'archivage de données

semi-structurées. Cette approche assure à Subly une capacité à évoluer et à s'adapter aux futurs besoins, tout en restant stable et performant.

Création entités

La création des entités a été un moment déterminant, car elle constitue le socle de l'application et de sa base de données relationnelle. Chaque entité a été soigneusement définie pour refléter fidèlement la structure des données métier et permettre leur exploitation de manière efficace et sécurisée.

L'entité **Utilisateur** a été pensée pour centraliser les informations essentielles au profil des utilisateurs, comme l'identifiant unique, l'email, le mot de passe (de manière sécurisée) et les préférences éventuelles.
L'entité **Abonnement** regroupe quant à elle les données liées aux services souscrits par l'utilisateur : le nom du service, le montant, la périodicité des paiements, la date de début et le lien avec la catégorie correspondante.
L'entité **Catégorie** permet de regrouper les abonnements par thématique, afin d'offrir à l'utilisateur une vision organisée et claire de ses dépenses.
Enfin, l'entité **Prélèvement** trace chaque opération financière réelle liée à un abonnement, précisant la date et le montant de chaque débit.

Ces entités ont été structurées dans une logique relationnelle, en s'appuyant sur des clés primaires et étrangères, garantissant ainsi la robustesse des relations et la fiabilité des données. Des captures d'écran viendront illustrer ces entités, permettant de visualiser leur conception et leur intégration dans l'architecture globale de l'application.

screens des entités

Création controllers

Les **contrôleurs** constituent l'interface principale entre les requêtes des utilisateurs et la logique métier de l'application. Leur création a été guidée par les principes fondamentaux de l'**architecture REST**, en veillant à la clarté, à la cohérence et à la sécurité des échanges.

Chaque contrôleur a été pensé pour gérer les requêtes liées à un domaine fonctionnel précis. Le contrôleur des utilisateurs, par exemple, prend en charge les opérations liées à la création de compte, à l'authentification et à la gestion du profil. Le contrôleur des

abonnements, quant à lui, assure la création, la mise à jour, la consultation et la suppression des abonnements, en intégrant la logique de filtrage et de catégorisation.

Les routes ont été définies avec précision pour garantir la **lisibilité** et la **maintenabilité** du code. À chaque requête correspond un traitement clair, réalisé dans le respect des bonnes pratiques de programmation défensive. Des **screens** des contrôleurs seront intégrés au dossier pour illustrer la façon dont ces composants ont été conçus et organisés dans le projet.

screens des controllers

Validation des données

La validation des données a été intégrée dès les premières phases de développement afin de garantir la qualité et la fiabilité des informations traitées. En utilisant les outils et décorateurs de validation de **Nest.js**, j'ai mis en place un système qui vérifie systématiquement les entrées avant tout traitement ou enregistrement en base.

Cela a permis de s'assurer que chaque donnée saisie par l'utilisateur respecte les contraintes attendues (format, longueur, type de donnée, etc.), réduisant ainsi les risques d'erreurs et de vulnérabilités. Cette démarche de validation contribue non seulement à la robustesse de l'application, mais aussi à l'amélioration de l'expérience utilisateur, en offrant un retour clair et immédiat en cas d'erreur de saisie.

screen verif data

Protection des mots de passe utilisateurs

La sécurité des données personnelles, et notamment des **mots de passe**, a constitué une priorité absolue. Aucun mot de passe n'est stocké en clair : ils sont systématiquement **hachés** à l'aide d'algorithmes éprouvés tels que **bcrypt**.
Cette méthode de hachage garantit que même en cas de compromission de la base de données, les mots de passe des utilisateurs ne pourront pas être exploités directement.
Cette protection s'inscrit dans les recommandations de l'ANSSI et les obligations fixées par le RGPD en matière de protection des données sensibles.

screen bcrypt

JSON Web Token

Pour assurer l'**authentification** et la **gestion sécurisée des sessions**, j'ai intégré le système de **JSON Web Token (JWT)**. Ce mécanisme permet de délivrer un **jeton sécurisé**

lors de la connexion d’un utilisateur, qui sera ensuite utilisé pour authentifier ses requêtes lors des interactions avec l’API.

L’utilisation de JWT offre plusieurs avantages : elle renforce la sécurité des échanges en limitant les accès aux seuls utilisateurs authentifiés, et elle simplifie la gestion des sessions en supprimant la nécessité de maintenir un état côté serveur. Chaque jeton contient les informations strictement nécessaires à l’identification de l’utilisateur, dans un format compact et sécurisé.

screen jwt

Préparer le déploiement d’une application sécurisée

Préparer et exécuter les plans de tests d’une application

Ajout de tests unitaires et fonctionnels

Ajout d’un script d'exécution de tests dans la CI

Préparer et documenter le déploiement d’une application

Rédaction d’un processus de déploiement

Contribuer à la mise en production dans une démarche DevOps

Création de scripts d’automatisation pour le déploiement

Création d’un Docker Compose