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.

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

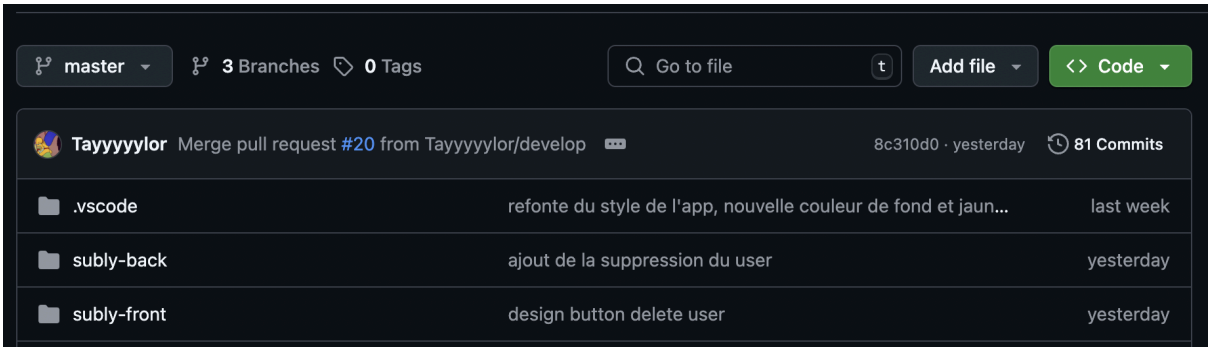
Pour structurer le développement collaboratif et optimiser la gestion du repository, une organisation GitHub a été mise en place. Ce choix permet une centralisation efficace des

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.

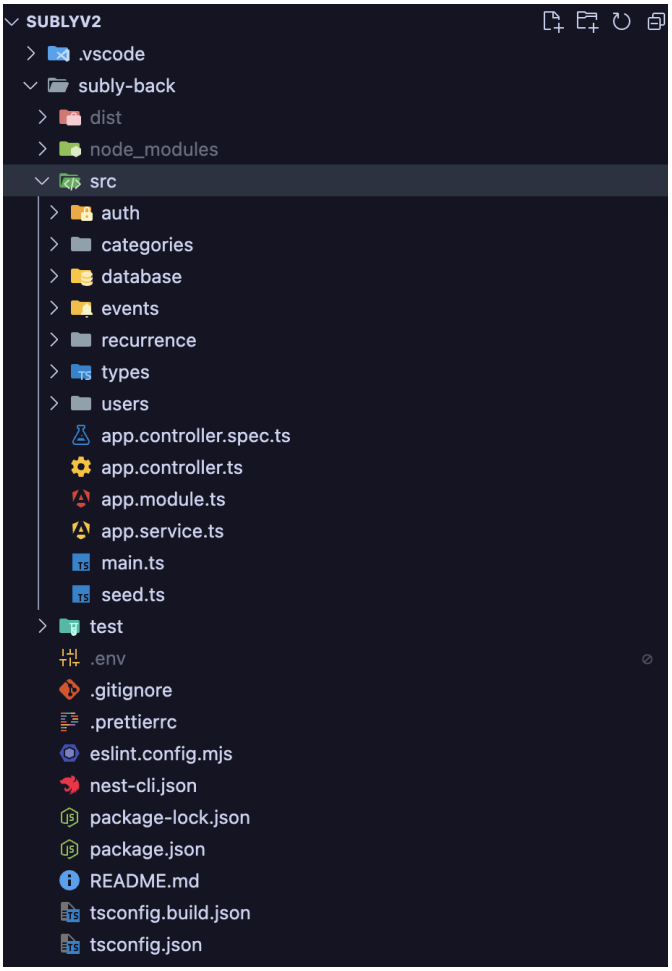
Une fois le projet initialisé, j'ai installé les dépendances nécessaires à la gestion de la navigation avec Expo Router, ainsi que le framework de styles NativeWind (Tailwind CSS).

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 : contient chaque fonctionnalités représenté par un dossier.

/types: pour le typage

/test : contient les tests

Développer des interfaces utilisateur

Front NativewindCSS

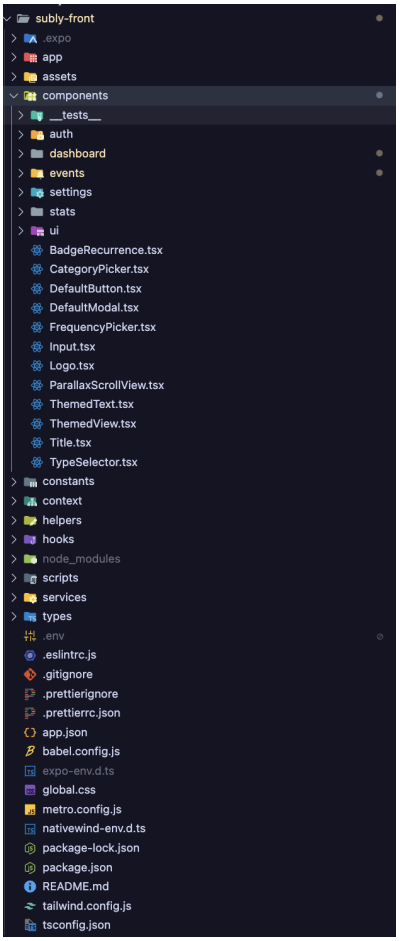
Dans le cadre du développement de cette plateforme, j'ai choisi d'utiliser **React Native** associé à **NativeWind (intégration de Tailwind CSS)** pour la création de l'interface utilisateur.
React Native est une technologie reconnue pour sa capacité à produire des applications mobiles performantes et 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

/constants
/helpers
/services
/types
/scripts



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 noeud. 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 :

```
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.

```
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 :

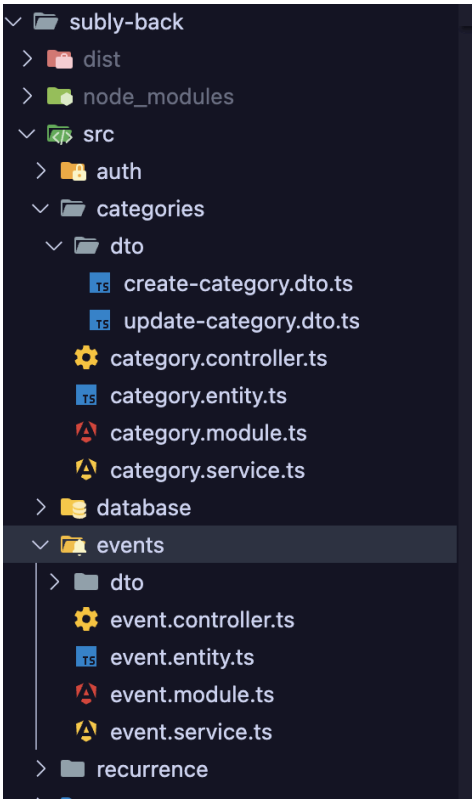
```
  return (
    <SafeAreaView className="flex-1 mt-[100px]">
      <View className="flex-col items-center gap-1 mb-[20px]">
        <Logo />
        <Title label="Connexion" />
      </View>
      <View className="p-10 gap-5 ">
        {inputData.map((input, index) => {
          return (
            <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 :
controller :
entity :
module :
service :



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 coeur 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

Cahier des charges et besoins

Création de wireframes et maquettes

La première étape a consisté en une analyse approfondie des besoins des utilisateurs. Cette analyse a été réalisée à partir d'un **cahier des charges** détaillé, dans lequel j'ai identifié les acteurs principaux, les flux de navigation et les limites du système. L'objectif a été de formaliser les attentes des utilisateurs en les traduisant en fonctionnalités concrètes et en parcours clairs.

Pour traduire cette vision en interfaces tangibles, j'ai conçu des **maquettes interactives** à l'aide d'outils tels que **Figma**. Ces maquettes ont permis de visualiser l'enchaînement des écrans et de valider l'ergonomie générale de l'application. Elles ont servi de support pour recueillir des retours auprès de potentiels utilisateurs et affiner le design des interfaces en fonction de leurs attentes et contraintes spécifiques.

Cette phase de maquettage a été essentielle pour anticiper les points critiques du parcours utilisateur, identifier les éventuels points de friction et garantir une expérience fluide et cohérente. Elle s'est inscrite dans une démarche de **co-conception**, prenant en compte les retours des parties prenantes pour aboutir à un résultat qui répond pleinement aux besoins fonctionnels et ergonomiques du projet.

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 (services de persistance). Chaque couche a été conçue pour garantir la sécurité, la maintenabilité et la performance de l'application.

Pour la mise en œuvre, j'ai choisi le **framework Nest.js**, particulièrement adapté aux architectures modulaires et à 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

Modèle conceptuel de données

Modèle logique de données

Modèle physique de données

Unified Modeling Language

Diagramme de classes

Diagramme de séquences

La conception de la **base de données relationnelle** a été une étape majeure, visant à garantir l'intégrité, la cohérence et la sécurité des informations manipulées par l'application. J'ai opté pour **PostgreSQL**, un système de gestion de base de données robuste, fiable et évolutif, parfaitement adapté aux besoins du projet.

La première étape a consisté à élaborer un **schéma conceptuel des données**, identifiant les principales entités (utilisateur, abonnement, catégorie, prélèvement) ainsi que les relations qui les lient. J'ai veillé à respecter les règles de normalisation, les conventions de nommage de l'entreprise, et à mettre en place les clés primaires et étrangères nécessaires pour garantir l'intégrité référentielle.

Une fois le modèle conçu, j'ai créé la **base de données physique** en utilisant les commandes SQL adaptées, en créant les tables, les index et les contraintes nécessaires. J'ai également défini des **utilisateurs et des rôles** dans PostgreSQL, avec des droits d'accès stricts pour chaque profil, en conformité avec les exigences de sécurité et de confidentialité du projet.

Pour valider la structure de la base et préparer les phases de test, j'ai mis en place un **jeu d'essai complet** dans une base de données de test. Cette base a été sauvegardée afin de permettre une restauration rapide en cas de besoin.

Développer des composants d’accès aux données SQL et NoSQL

Failles de sécurité OWASP

Protection contre les injections SQL

Validation des données

Protection des mots de passe utilisateurs

JSON Web Token

Le développement des composants d’accès aux données a été effectué en utilisant directement les outils et bibliothèques proposés par **Nest.js**, tels que **TypeORM**. Cette couche d’accès aux données est essentielle pour garantir la fiabilité et la sécurité des échanges entre l’application et la base de données.

J’ai conçu des services dédiés pour les opérations courantes : lecture, création, modification et suppression des données. Chaque service a été développé dans un style défensif, avec une attention particulière à la gestion des exceptions et des erreurs. J’ai mis en place des validations strictes des entrées, utilisant les fonctionnalités de **validation des DTOs** proposées par **class-validator**, afin de garantir l’intégrité et la cohérence des données.

La gestion des accès concurrents et des éventuels conflits a été pensée en amont, afin d’éviter toute perte de données ou incohérence. Les cas d’exception ont été identifiés et traités explicitement dans le code, pour assurer la robustesse des traitements.

Bien que l’application ne fasse pas usage de bases NoSQL pour l’instant, l’architecture a été pensée pour rester **ouverte** à l’intégration de telles technologies si nécessaire (par exemple pour des besoins spécifiques en analyse temps réel ou en stockage de données semi-structurées).

L’ensemble de ces développements a été soumis à des **tests unitaires** pour valider leur bon fonctionnement et à des **tests de sécurité** afin de garantir la conformité avec les exigences du projet. Une démarche claire et structurée de résolution de problème a été définie, permettant de corriger efficacement les éventuels dysfonctionnements identifiés lors des tests ou des incidents de production.

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