

Dossier de projet professionnel



Haptics have never been easier

SOMMAIRE

Présentation de l'entreprise.....	4
Haptique - Définition générale.....	5
Fonctionnement technique.....	5
Produits développés par V.RTU.....	6
L'anneau.....	6
Le bracelet.....	8
Haptify.....	9
Projet principal : Haptify.....	9
Objectifs pédagogiques du projet.....	9
Démontrer la capacité à développer une application sécurisée moderne.	9
Concevoir une architecture organisée en couches.....	10
Implémenter l'authentification, les accès restreints et les règles de sécurité.....	10
Mettre en œuvre une base de données cloud avec gestion des droits.	11
Réaliser un déploiement cloud avec intégration continue et hébergement sécurisé.....	11
Description fonctionnelle.....	11
Génération d'effets haptiques à partir d'un prompt.....	12
Personnalisation des effets haptiques.....	14
Comptes utilisateurs avec tokens limités par abonnement.....	15
Système d'abonnement et de paiement récurrent (Stripe).....	16
Interface utilisateur en React / Next.js.....	17
Authentification et gestion de session sécurisée (AWS Cognito).....	18
Appels backend via GraphQL / AppSync.....	18
Stack technique.....	19
Front-end.....	19
Back-end.....	20
Base de données.....	21
Paiement.....	21
CI/CD.....	21
Outils de projet.....	22

Architecture logicielle.....	22
Modélisation des données.....	22
Séparation des couches.....	24
Accès et logique des données.....	24
Sécurité et validation.....	24
Déploiement et qualité.....	25
Intégration continue (CI) et vérification qualité.....	25
Déploiement continu (CD).....	26
Gestion du domaine personnalisé.....	26
Sécurisation des variables d'environnement.....	26
Organisation du projet.....	27
Méthodologie et gestion.....	27
Outils de pilotage.....	28
Composition de l'équipe projet.....	31
Exemples de fonctionnalités clés.....	32
Authentification avec Amplify et withAuthenticator.....	32
Paiement via Stripe + redirections + webhooks sécurisés.....	34
Code - du pricing à l'ajout du plan pour l'utilisateur.....	36
Attribution automatique d'un Plan et génération de tokens.....	39
Conclusion.....	41
Remerciements.....	42

Présentation de l'entreprise



V.RTU est une entreprise de développement de logiciel et de matériel conceptrice d'haptique.

Haptique - Définition générale

L'haptique est un domaine technologique qui concerne le sens du toucher. Elle permet à un utilisateur d'interagir avec un environnement numérique ou physique via des retours tactiles ou kinesthésiques.

Concrètement, l'haptique reproduit des sensations physiques (pression, vibration, texture, force, résistance, etc.) pour simuler le contact ou la manipulation d'objets dans un système virtuel ou réel.

Fonctionnement technique

L'haptique repose sur trois composantes principales :

Les capteurs (input) : ils détectent les actions de l'utilisateur (pression des doigts, position, mouvement...).

Les actionneurs (output) : ils envoient un retour physique à l'utilisateur (vibration, force, friction...).

Le logiciel haptique : il traite les interactions et déclenche les retours selon des règles prédéfinies (moteur physique, API, moteur haptique, etc.).

Produits développés par V.RTU

V.RTU crée du software et du hardware.

Les équipements hardware s'utilisent en conjonction avec des logiciels et des jeux, y compris des serious games et entraînements pour employés :

- en Réalité Virtuelle (Virtual Reality, **VR**)
- en Réalité Augmentée (Augmented Reality, **AR**)
- en Réalité Étendue (eXtended Reality, **XR**)

Ces équipements utilisent principalement la détection des mains pour contrôler le retour haptique aux utilisateurs.

L'anneau



Anneau créant des retour haptiques pour l'utilisateur.

Avantages :

- peu encombrant
- facile à produire
- plastique, moins salissant que le tissu

Le bracelet



Bracelet créant des retour haptiques pour l'utilisateur.

- peu encombrant
- facile à produire
- plastique, moins salissant que le tissu

Haptify

Logiciel permettant aux utilisateurs de créer, générer et personnaliser des effets haptiques.

L'objectif premier d'Haptify est de rendre plus accessible la création d'effets haptiques pour les développeurs et les designers.

V.SION

V.SION est un système de motion capture servant à entraîner des usagers à effectuer des tâches spécifiques.

Il permet d'enregistrer les positions des humains dans l'espace, sans besoin d'utiliser des combinaisons ou des marqueurs.

Ajouter de l'haptique permet des interactions variées avec les utilisateurs, comme par exemple la définition d'un périmètre d'activité.

Projet principal : Haptify

Haptify est une application web conçue pour permettre à ses utilisateurs de générer et personnaliser des effets haptiques à l'aide d'IA génératives.

Elle est destinée aux développeurs et designers souhaitant créer facilement des retours tactiles riches.

Le projet a été réalisé en équipe pluridisciplinaire dans un cadre professionnel, en alternance, au sein du service R&D de l'entreprise V.RTU.

Objectifs pédagogiques du projet

Démontrer la capacité à développer une application sécurisée moderne

À travers Haptify, l'accent a été mis sur la mise en œuvre d'un système complet de gestion d'identité, d'autorisations et de sécurité applicative. L'utilisation de standards reconnus comme AWS Cognito, le respect des bonnes pratiques de validation des entrées, la séparation des rôles et la sécurisation des flux font partie intégrante de la solution.

Concevoir une architecture organisée en couches

Le projet repose sur une architecture logicielle claire : interface utilisateur, logique métier (règles d'abonnement, consommation de tokens), et couche d'accès aux données (API GraphQL/AppSync). Cette séparation permet une évolutivité et une maintenance facilitée.

Implémenter l'authentification, les accès restreints et les règles de sécurité

La protection des routes, le cloisonnement des données utilisateurs, la gestion des jetons JWT ainsi que l'application de règles d'accès par rôles ont été mis en œuvre grâce aux directives `@auth` dans le schéma GraphQL, couplées à Cognito.

Mettre en œuvre une base de données cloud avec gestion des droits

DynamoDB a été utilisé comme système de persistance NoSQL. La modélisation a été faite via Amplify Studio, en intégrant dès la conception les règles d'accès aux entités selon les profils utilisateurs.

Chaque entité dispose de ses propres règles d'autorisation, implémentées à l'aide des directives `@auth` dans le schéma GraphQL.

Réaliser un déploiement cloud avec intégration continue et hébergement sécurisé

Le projet bénéficie d'un pipeline de déploiement automatique via AWS Amplify, intégrant des phases de build, de test, d'analyse de code (ESLint) et de publication.

Le domaine personnalisé est configuré en HTTPS via Route 53 et GoDaddy, garantissant une livraison continue et sécurisée.

Description fonctionnelle

Génération d'effets haptiques à partir d'un prompt

Le cœur fonctionnel de l'application repose sur la génération d'effets haptiques à partir de textes libres rédigés par l'utilisateur. Cette approche permet une grande créativité, tout en offrant un accès intuitif à la création haptique.

Une fois authentifié, l'utilisateur accède à une interface dans laquelle il peut saisir un *prompt* textuel décrivant l'effet souhaité (exemples : « effet de pulsation lente », « retour tactile granuleux »). Ce prompt est envoyé à une

API via une requête GraphQL, laquelle déclenche un processus de génération d'effet haptique.

L'effet est généré par une IA (modèle entraîné en amont), et son résultat est stocké dans une base de données DynamoDB, associé à l'utilisateur, à son plan d'abonnement et au nombre de tokens consommés.

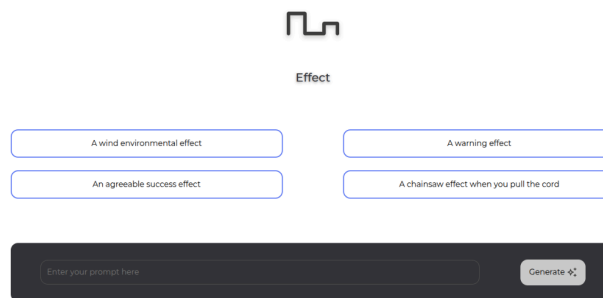
Chaque génération d'effet haptique consomme un certain nombre de tokens.

L'utilisateur peut ensuite visualiser ou télécharger l'effet généré, ou l'intégrer dans un flux de développement d'application externe.

Les tokens disponibles sont affichés en temps réel dans l'interface, et le frontend réagit dynamiquement à chaque mutation via les *subscriptions GraphQL*.

L'accès à cette fonctionnalité est strictement réservé aux utilisateurs authentifiés, conformément aux règles de sécurité mises en place via Amazon Cognito et les directives `@auth` définies dans le schéma GraphQL.

Interface principale de génération d'effets haptiques. L'utilisateur peut soit cliquer sur un exemple prédéfini, soit saisir son propre prompt librement avant de cliquer sur "Generate".



The screenshot shows a web interface for generating haptic effects. At the top center is a logo consisting of a stylized square with a horizontal line extending to the right. Below the logo is the word "Effect". Underneath, there are four rounded rectangular buttons arranged in a 2x2 grid. The top-left button contains the text "A wind environmental effect". The top-right button contains "A warning effect". The bottom-left button contains "An agreeable success effect". The bottom-right button contains "A chainsaw effect when you pull the cord". Below these buttons is a dark gray horizontal bar. Inside this bar, on the left, is a text input field with the placeholder text "Enter your prompt here". On the right side of this bar is a button labeled "Generate" with a small icon of a lightning bolt.

Personnalisation des effets haptiques

Une fois les effets haptiques générés, l'utilisateur a la possibilité de les modifier grâce à une interface intuitive. L'interface de personnalisation permet d'ajuster différents paramètres de l'effet, tels que la durée, l'intensité, la forme d'onde, ou encore la fréquence des impulsions.

L'objectif est de permettre aux utilisateurs de peaufiner leurs créations, que ce soit pour des besoins d'accessibilité, d'ergonomie ou de créativité. Les

modifications peuvent être visuelles (prévisualisation graphique en temps réel), auditives (feedback sonore) ou tactiles (test sur simulateur ou appareil connecté).

Toutes les personnalisations sont stockées et liées au compte utilisateur, avec la possibilité de les rééditer ou les dupliquer. L'interface a été conçue avec une forte attention à l'expérience utilisateur : composants accessibles, navigation fluide, contrôles glissants (sliders), valeurs numériques précises et validations instantanées.

Ces ajustements respectent également les droits d'accès définis via les règles Cognito, garantissant que seul le propriétaire de l'effet peut le modifier ou le supprimer.

Comptes utilisateurs avec tokens limités par abonnement

Chaque utilisateur inscrit dispose d'un nombre de tokens défini par son plan d'abonnement. Ces tokens représentent la capacité de génération d'effets haptiques disponibles.

Le système d'authentification repose sur AWS Cognito, qui permet de sécuriser l'accès aux ressources et d'attribuer à chaque utilisateur un identifiant unique. Lors de l'inscription, un utilisateur se voit automatiquement affecté un plan par défaut (par exemple "Découverte").

Lorsqu'un utilisateur génère un effet, un nombre défini de tokens est automatiquement décrémenté. Ce mécanisme est intégré dans la logique métier de l'application côté backend et est synchronisé avec la base de données DynamoDB. Les tokens restants sont affichés dynamiquement dans l'interface.

Les informations relatives aux abonnements, à la consommation de tokens et aux historiques d'utilisation sont accessibles via des requêtes GraphQL sécurisées, avec filtrage des données selon l'identité de l'utilisateur connecté.

Ce système encourage une utilisation responsable des ressources et favorise l'adhésion à des abonnements plus élevés pour des besoins avancés.

Système d'abonnement et de paiement récurrent (Stripe)

Pour gérer les accès premium et limiter la consommation abusive des ressources, Haptify intègre un système d'abonnement payant basé sur Stripe. Ce système permet aux utilisateurs de souscrire à différents plans offrant chacun un quota de tokens, des fonctionnalités avancées ou une priorité de traitement.

L'utilisateur peut accéder à la page d'abonnement depuis l'interface web sur la route front `/pricing`. Lorsqu'il sélectionne un plan, une requête est envoyée

à une route API `/api/checkout`, laquelle crée dynamiquement une session Stripe Checkout. Stripe prend en charge l'authentification du paiement, les redirections (success / cancel) et la gestion des informations sensibles (PCI DSS).

Une fois le paiement validé, l'utilisateur est redirigé vers une page de confirmation. En arrière-plan, un webhook Stripe appelle `/api/webhook` pour notifier le backend. Le backend vérifie l'authenticité de l'événement (signature secrète) et met à jour le statut de l'abonnement dans la base DynamoDB via une mutation GraphQL. Si l'utilisateur n'existe pas encore en base, il est automatiquement créé.

L'intégration de Stripe permet d'éviter toute gestion directe de données bancaires sensibles, tout en respectant les obligations RGPD. De plus, la synchronisation est conçue pour être robuste : les tokens sont mis à jour automatiquement dès réception du webhook, et l'utilisateur bénéficie immédiatement des droits de son nouveau plan.

Interface utilisateur en React / Next.js

L'interface utilisateur de Haptify a été développée avec React et Next.js, offrant une expérience fluide, rapide et accessible sur tous types de supports. Elle constitue le point d'entrée principal pour toutes les fonctionnalités clés : connexion, génération, personnalisation, suivi des tokens et abonnement.

Le système de routage dynamique de Next.js permet de charger rapidement les pages critiques (comme `/start`, `/pricing`) avec pré-rendu côté serveur ou génération statique selon le contexte.

Enfin, l'intégration du composant `withAuthenticator` d'AWS Amplify assure une protection des pages sensibles, garantissant que seuls les utilisateurs connectés accèdent à leurs effets haptiques ou paramètres d'abonnement. Le frontend consomme toutes les données via Apollo Client, ce qui permet une communication efficace et sécurisée avec AppSync.

Authentification et gestion de session sécurisée (AWS Cognito)

L'authentification constitue une brique fondamentale du projet Haptify. Elle repose sur AWS Cognito, qui gère la création, la connexion, et la sécurisation des comptes utilisateurs. Grâce à l'intégration d'AWS Amplify, l'authentification est totalement intégrée au frontend via le composant `withAuthenticator`, qui encapsule l'ensemble du processus : création de compte, vérification d'adresse email, connexion, gestion de session, récupération de mot de passe, etc.

Les utilisateurs reçoivent un token JWT sécurisé à chaque session, qui est automatiquement utilisé dans les requêtes vers l'API GraphQL. Ce

mécanisme garantit une identification robuste, sans besoin de gérer manuellement les sessions ou les cookies.

L'ensemble des pages sensibles (génération, personnalisation, abonnements) sont protégées par cette authentification, avec redirection automatique en cas d'accès non autorisé.

Appels backend via GraphQL / AppSync

Le projet repose sur un backend basé sur AWS AppSync, qui expose une API GraphQL centralisée. Cette API permet d'effectuer toutes les opérations côté client : requêtes, mutations, et subscriptions temps réel.

Les requêtes sont effectuées via Apollo Client, qui offre un cache local intelligent et une intégration fluide avec React. Les mutations permettent de créer ou mettre à jour des effets, gérer les tokens ou enregistrer les abonnements.

Les subscriptions, quant à elles, permettent d'afficher en temps réel les changements liés à l'utilisateur (par exemple, l'actualisation du quota de tokens après une génération ou un paiement).

Le schéma GraphQL est enrichi avec des directives `@auth` permettant de restreindre les droits d'accès selon l'identité et le rôle de l'utilisateur, en

s'appuyant sur Cognito. L'ensemble garantit une communication sécurisée, fluide et scalable entre le frontend et les données.

Stack technique

Front-end

Le frontend a été conçu avec **Next.js** et **React**, deux technologies modernes permettant de construire des interfaces performantes, réactives et facilement maintenables.

TypeScript est utilisé pour garantir la robustesse du typage statique, limiter les erreurs à la compilation, et améliorer l'auto-complétion durant le développement.

L'interface graphique repose sur les composants **Amplify UI**, qui assurent la cohérence visuelle et l'accessibilité (accessibilité clavier, responsive design, contraste).

Back-end

Le backend repose sur **AWS AppSync**, un service managé exposant une **API GraphQL**. Cette API sert d'intermédiaire entre le frontend et les données, en gérant les requêtes, mutations et subscriptions en temps réel.

Des fonctions **Lambda** sont utilisées pour des traitements métier spécifiques comme la gestion des prompts. L'ensemble est serverless et scalable.

Base de données

La persistance est assurée par **DynamoDB**, base NoSQL hautement disponible, configurée via **Amplify Studio**.

Le schéma est défini en GraphQL, et les directives `@model` et `@auth` permettent de générer automatiquement les tables, les règles d'accès et les endpoints associés.

Paielement

Le paiement est géré via **Stripe**, en mode Checkout hébergé. Cette solution assure la conformité PCI DSS sans manipuler les données bancaires.

Des **webhooks** sécurisés permettent de synchroniser le backend aux événements comme un paiement réussi, en mettant à jour les utilisateurs et leur plan d'abonnement dans la base.

CI/CD

La livraison continue est assurée par **AWS Amplify Hosting**, couplée à **GitHub Actions**. À chaque `push`, une chaîne d'intégration est déclenchée : installation des dépendances, exécution de **ESLint**, build, et déploiement. Le tout est configurable via le fichier `amplify.yml`.

Outils de projet

La gestion de projet repose sur une combinaison d'outils collaboratifs :

- **Figma** pour le design UI/UX,
- **Notion** pour le suivi des tâches (kanban agile),
- **Hoppscotch** pour tester les appels GraphQL,
- **GitHub** pour le versioning et l'intégration continue.

Architecture logicielle

L'architecture de l'application Haptify repose sur une séparation claire des responsabilités, inspirée des principes de conception en couches : chaque couche (présentation, logique métier, accès aux données) est indépendante et testable.

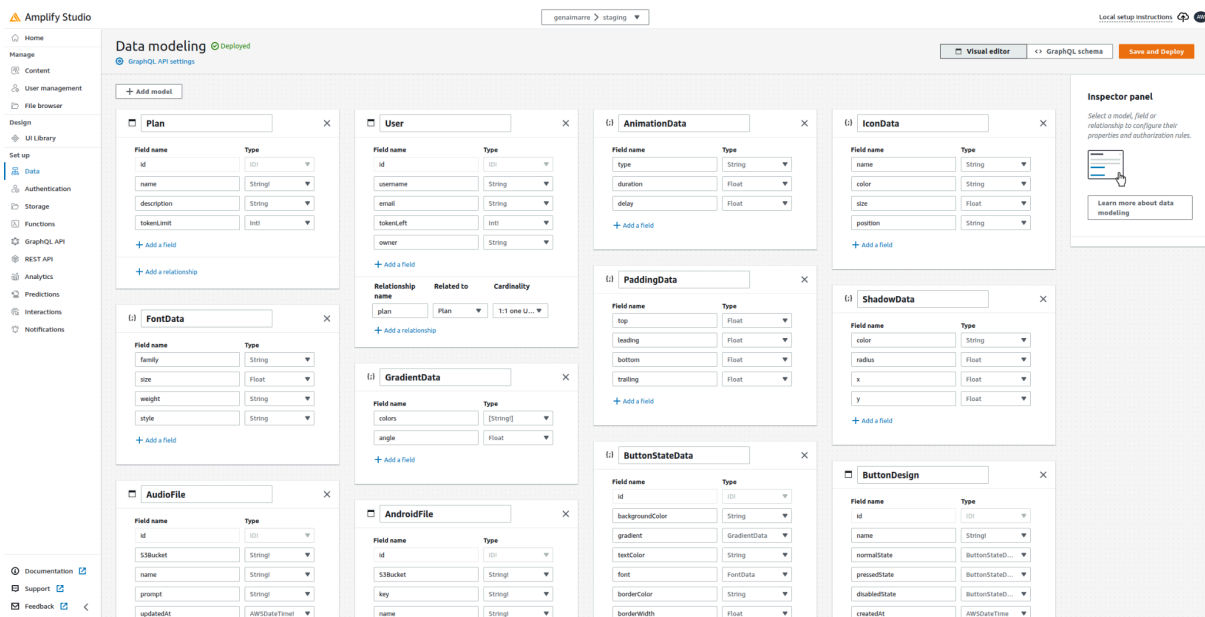
Modélisation des données

Le cœur du modèle repose sur trois entités principales :

- **User** : représente un utilisateur Cognito, identifié par son email, son ID unique et son plan d'abonnement.
- **Plan** : contient les détails du forfait souscrit (nom, quota de tokens, niveau d'accès).
- **HapticComponent** : Gère les éléments haptiques et leurs effets.

Ces entités sont reliées entre elles avec des relations **One-to-One** ou **Owner-based**, modélisées et gérées via **Amplify Studio**, puis générées automatiquement en DynamoDB.

Modèle de données conçu dans Amplify Studio. On y voit les entités principales et leurs relations. Les règles de sécurité (authentication, ownership) sont intégrées directement.



Séparation des couches

- **Couche Présentation** : assurée par Next.js et React. Elle interagit uniquement via Apollo Client avec l'API.
- **Couche Métier** : implémentée via les fonctions Lambda ou les règles définies dans le schéma GraphQL (logique de tokens, vérification d'abonnement).
- **Couche Données** : pilotée par AppSync, DynamoDB, et sécurisée par des directives `@auth` pour l'accès aux ressources.

Accès et logique des données

- Les données sont typées et filtrées dynamiquement côté client, garantissant que chaque utilisateur n'accède qu'à ses propres effets.
- Les subscriptions GraphQL permettent de mettre à jour en temps réel le compteur de tokens ou l'état d'un abonnement.

Sécurité et validation

- Le système Cognito gère l'identification via tokens JWT, transmis automatiquement dans les headers Apollo.
- Les directives `@auth` dans le schéma GraphQL limitent les accès en lecture/écriture par rôle (owner, groups).
- Les entrées utilisateur sont validées à la fois côté frontend (via React + TypeScript) et côté backend (via AppSync ou Lambda).

Déploiement et qualité

Intégration continue (CI) et vérification qualité

Le projet Haptify utilise la configuration `amplify.yml` pour gérer le processus d'intégration et de livraison continue. Ce fichier définit les étapes exécutées

automatiquement lors d'un push sur la branche principale du dépôt GitHub connecté à Amplify.

Les étapes configurées incluent :

- Installation des dépendances avec `npm install`
- Analyse statique du code avec **TypeScript** et **ESLint**, qui vérifient la cohérence du typage et le respect des bonnes pratiques de code
- Compilation de l'application avec `next build`
- Déploiement de la version construite sur l'environnement Amplify

Cette approche permet de valider automatiquement chaque mise à jour avant qu'elle ne soit publiée.

Déploiement continu (CD)

Le déploiement est entièrement automatisé par **AWS Amplify Hosting** dès qu'un commit est poussé sur la branche de production. Amplify :

- reconstruit l'application frontend,
- déploie le backend AppSync (si configuration modifiée),
- publie la nouvelle version sur une URL Amplify dédiée (staging ou production).

Gestion du domaine personnalisé

Le domaine `haptify.io` est relié à l'hébergement via **AWS Route 53**, avec les DNS configurés sur GoDaddy. Un certificat SSL est généré automatiquement pour assurer une navigation sécurisée en **HTTPS**.

Sécurisation des variables d'environnement

Les variables sensibles comme les clés Stripe, secrets webhooks, ou les identifiants Cognito sont gérées dans la console Amplify. Elles ne sont pas versionnées dans Git et ne transitent jamais en clair dans l'application. En développement, elles sont placées dans un fichier `.env` ignoré par Git.

Cette configuration garantit la fiabilité, la sécurité et la fluidité des mises en production pour Haptify.

Organisation du projet

Méthodologie et gestion

Le projet Haptify a été mené selon une approche **agile** allégée. Bien qu'il n'y ait pas eu de sprints formels avec user stories planifiées, l'équipe a tenu des points réguliers (daily meetings informels et réunions hebdomadaires) pour suivre l'avancement et ajuster les priorités. Les livrables ont été définis de

façon incrémentale, avec des jalons intermédiaires (MVP, version bêta, version présentable).

Pour le développement du produit, nous nous sommes organisés de manière agile :

- des réunions courtes quotidiennes le matin pour énumérer nos tâches, nos estimations et les difficultés attendues
- une réunion en fin de semaine pour faire le point sur le travail accompli
- La gestion du projet avec notion, qui nous a servi à organiser nos tâches et suivre nos réunions.

Les axes principaux :

- **Accessible** : le logiciel doit atteindre le public le plus large. Les complexités de l'haptique doivent être abstraites pour pouvoir rendre Haptify accessible au plus grand nombre de développeurs et designers.
- **Générique** : en conséquence, le logiciel n'est pas spécialisé pour concevoir les haptiques les plus précis.

Outils de pilotage

- **Notion** a servi de tableau de bord collaboratif pour centraliser les tâches et les idées et gérer un mini backlog par bloc fonctionnel (génération, paiement, personnalisation, etc.).
- **Figma** a permis de créer les maquettes, de suivre les retours sur l'UI et d'itérer sur l'ergonomie.
- **GitHub** a été utilisé pour le versioning, la collaboration et le déclenchement automatique des déploiements via Amplify.

Gestion de projet avec notion

📁 Achèvement 33,33 %

📅 Dates 31 janvier 2024

✓ 5 autres propriétés

✓ Backend

📁 Demo haptify - mobile test / data collect ● En cours 👤 Lucas Ribard F félix H Heidy Daumas

📁 Generate thumbnails from waveform ● Pas commencé H Heidy Daumas

📁 React App Haptify_2D (sin gen) ● Terminé 👤 Lucas Ribard

📁 React App Build Up ● Terminé 👤 Lucas Ribard

📁 Nouvelle tâche ● Archivé

📁 MUI Register ● Archivé F félix

📁 Database Mongo ● Archivé F félix

📁 React App Base ● Terminé 👤 Lucas Ribard

📁 API Gestio ● En cours F félix

📁 sign in & sign up ● Terminé F félix 👤 Lucas Ribard

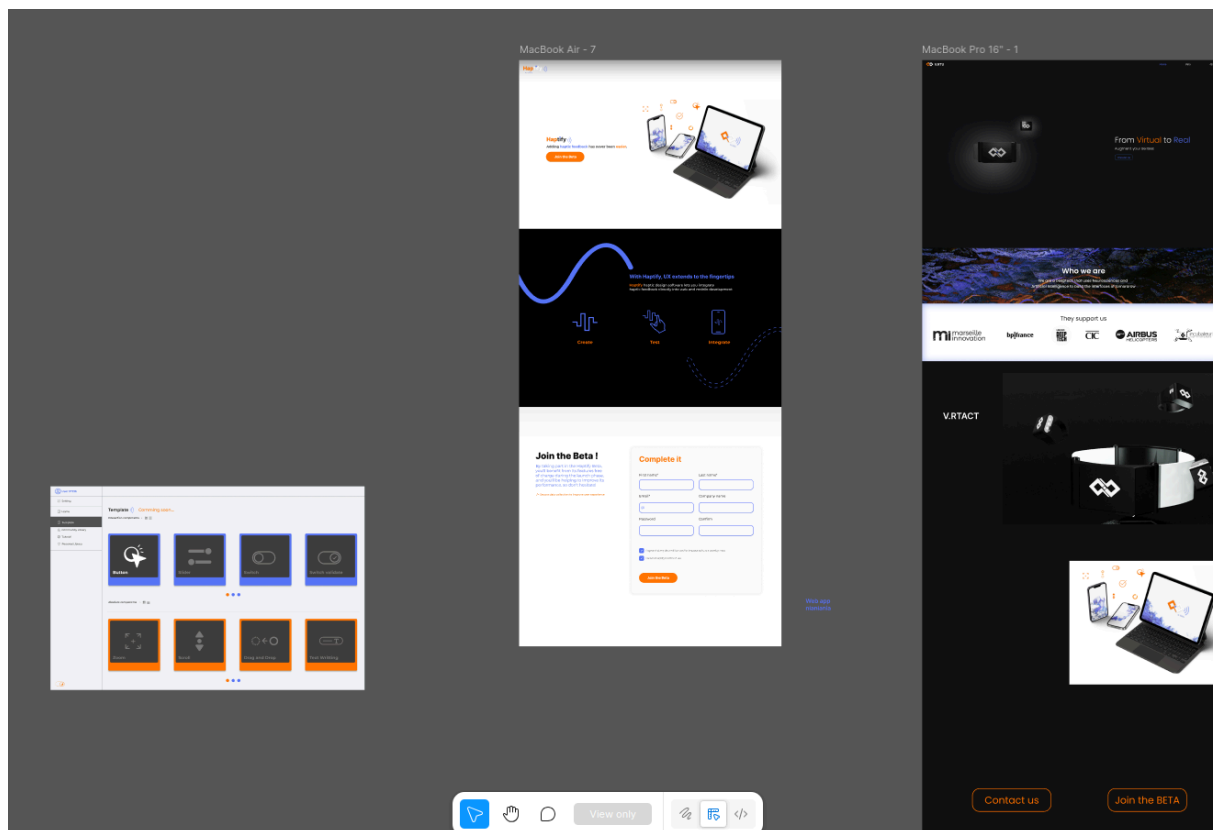
14 de plus...

+ Nouveau

Commentaires

F Ajouter un commentaire...

Espace de travail figma pour les maquettes



Composition de l'équipe projet

Le projet a été réalisé en **équipe pluridisciplinaire** composée de :

- Deux développeurs principaux (alternants CDA, moi inclus)
- Une designer UI/UX
- Une doctorante en haptique
- Une docteure en informatique
- Une scrum master (suivi de la méthodologie)
- Le dirigeant de l'entreprise (vision stratégique et support)

Cette organisation a permis de couvrir à la fois les aspects techniques, créatifs et scientifiques du projet tout en répondant aux contraintes de production.

Exemples de fonctionnalités clés

Authentification avec Amplify et withAuthenticator

L'authentification dans Haptify repose sur le service **Amazon Cognito**, intégré via **AWS Amplify**. Le composant `withAuthenticator`, fourni par la bibliothèque **Amplify UI**, encapsule toute la logique d'authentification côté frontend.

Lorsque l'utilisateur accède à une page protégée, `withAuthenticator` affiche automatiquement :

- l'écran d'inscription (email, mot de passe, confirmation)
- la page de connexion
- la récupération de mot de passe

Une fois connecté, Cognito fournit un **token JWT sécurisé**, automatiquement injecté dans les appels GraphQL effectués avec **AppSync**. Ce token est

vérifié côté backend grâce aux directives `@auth` définies dans le schéma GraphQL (par exemple : `@auth(rules: [{ allow: owner }])`).

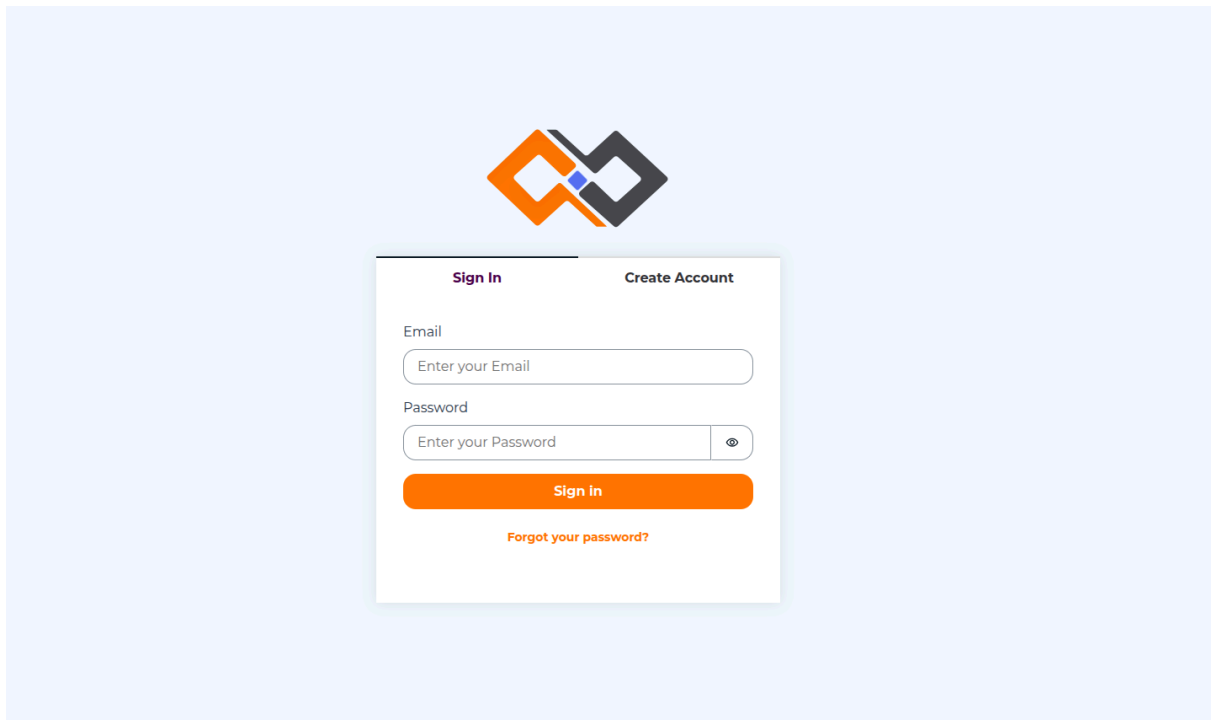
Avantages clés :

- Authentification **sécurisée et standardisée** sans coder manuellement les formulaires.
- Gestion automatique du **stockage des sessions** côté navigateur.
- Possibilité d'adapter les règles d'accès par groupe, rôle ou ownership.
- Simplification du code frontend (l'enveloppe `withAuthenticator` remplace toute logique manuelle).

De plus, les utilisateurs ont accès à leur compte personnel avec affichage conditionnel des interfaces selon leur statut (abonné gratuit ou premium).

Cette gestion dynamique est assurée par la combinaison du token Cognito et des données issues de l'API GraphQL (rôle, plan actif, nombre de tokens restants).

Connexion avec `withAuthenticator` - UI



Paieement via Stripe + redirections + webhooks sécurisés

Le parcours de souscription passe par **Stripe Checkout**. Lorsqu'un utilisateur souhaite accéder à un plan premium, il sélectionne l'offre souhaitée depuis l'application, qui déclenche une requête POST vers une route [/api/checkout](#).

Cette route côté serveur utilise le SDK Stripe pour créer une **session de paiement sécurisée**, avec les paramètres du plan (prix, durée, redirections).

L'utilisateur est redirigé vers l'interface Stripe hébergée, garantissant la **sécurité des données bancaires** (aucune information sensible n'est traitée côté application).

- En cas de succès, Stripe redirige automatiquement l'utilisateur vers <https://haptify.io/checkout/success>
- En cas d'annulation : <https://haptify.io/checkout/cancel>

En parallèle, un **webhook Stripe** configuré sur

<https://haptify.io/api/webhooks> reçoit les événements en temps réel :

- `checkout.session.completed`
- `invoice.paid`
- `customer.subscription.deleted`, etc.

Chaque webhook est signé avec une **clé secrète** (header `Stripe-Signature`)

que l'application vérifie systématiquement. Le payload est parsé uniquement après authentification de la signature, empêchant toute tentative de fraude.

Le backend, à réception d'un paiement validé, effectue une **mutation GraphQL sécurisée** vers AppSync. Celle-ci permet de :

- créer l'utilisateur s'il n'existe pas encore (en fonction de son email)
- associer ou mettre à jour son plan d'abonnement
- initialiser la quantité de tokens associée

Ce mécanisme garantit un parcours fluide, automatisé et conforme au **RGPD**.

Le parcours de souscription passe par Stripe Checkout. L'utilisateur sélectionne un plan, est redirigé vers Stripe, puis revient vers l'app en cas de succès ou d'annulation. Un webhook Stripe permet de notifier le backend AppSync du succès

du paiement. Le backend met alors à jour l'utilisateur (création ou changement de plan) de manière automatique, sécurisée et conforme au RGPD.

Code - du pricing à l'ajout du plan pour l'utilisateur

Bouton de checkout Stripe

```
const CheckoutButton = ({ priceId }) => {
  const router = useRouter();

  const handleClick = async () => {
    try {
      const redirectPath = `/payment/checkout/${priceId}`;
      console.log("redirectPath", redirectPath);
      router.push(redirectPath);
    } catch (err) {
      console.error("Error while redirecting to checkout:", err);
    }
  }
}
```


Pages de paiement avec le routage dynamique de Next.js

```
▼ payment
  ▼ checkout
    ⚙ [...priceId].tsx
  ▼ return
    ⚙ [...priceId].tsx
```

Intégration Stripe dans Next.js

S'abonner à Premium Subscription

29,99 € par mois


Payer avec  link


Ou

E-mail

Moyen de paiement


Informations de la carte





Nom du titulaire de la carte


Pays ou région

France 

☐ **Enregistrer mes informations pour régler plus rapidement**
Réglez vos achats sur ce site plus rapidement ainsi que partout où Link est accepté.

Payer et s'abonner

En confirmant votre abonnement, vous autorisez à débiter votre compte pour les paiements à venir, conformément à ses conditions. Vous pouvez annuler votre abonnement à tout moment.

Propulsé par  | [Conditions d'utilisation](#) | [Confidentialité](#)

Connexion à Stripe via secrets et variables d'environnement

```
const stripe = require('stripe')(process.env.STRIPE_SECRET_KEY);
const webhookSecret = process.env.STRIPE_WEBHOOK_SECRET;
```

Vérification si l'utilisateur existe déjà

```

try {
  switch (eventType) {
    case 'checkout.session.completed': {
      // First payment is successful and a subscription is created (if mode was set to "subscription" in ButtonCheckout)
      // ✅ Grant access to the product
      let user;

      const session = await stripe.checkout.sessions.retrieve(
        data.object.id,
        {
          expand: ['line_items']
        }
      );

      const customerId = session?.customer;
      const customer = await stripe.customers.retrieve(customerId);
      const priceId = session?.line_items?.data[0]?.price.id;

      if (customer.email) {
        // 1. Check if user exists
        const email = customer.email;

        const queryListUserByEmail = gql`
          query ListUsers($filter: ModelUserFilterInput) {
            listUsers(filter: $filter) {
              items {
                id
                email
                createdAt
              }
            }
          }
        `;

        const { data } = await client.query({
          query: queryListUserByEmail,
          variables: { email }
        });

        user = data[0];
      }
    }
  }
}

```

Création d'un utilisateur s'il n'existe pas déjà

```

// 1.1 If no user found, create it
if (!user) {
  console.log("no user found, creating user...");
  // Create user here
  const mutationCreateUser = gql`
    mutation CreateUser($input: CreateUserInput!) {
      createUser(input: $input) {
        id
        email
        customerId
        tokenLeft
      }
    }
  `;

  const { data } = await client.mutate({
    mutation: mutationCreateUser,
    variables: {
      input: {
        email: email,
        customerId: customerId,
        tokenLeft: 0
      }
    }
  });

  res.status(201).json({ success: true, user: data.createUser });
  user = data.createUser;
}

```

Si le plan existe bien en base de données grâce au planId, je donne accès à l'utilisateur aux features associées.

```
// 2. Update user data by giving them access to selected plan/features
(async () => {
  const queryListPlansByPriceId = gql`
    query ListPlans($filter: ModelPlanFilterInput) {
      listPlans(filter: $filter) {
        items {
          id
          stripePriceId
        }
      }
    }
  `;

  const userData = await client.query({
    query: queryListPlansByPriceId,
    variables: { priceId }
  });

  const plan = userData.data.getPlanByPriceId;

  if (!plan) {
    console.error('No matching plan found');
    throw new Error('No matching plan found for the price ID');
  }

  const mutationUpdateUser = gql`
    mutation UpdateUser($input: UpdateUserInput!) {
      updateUser(input: $input) {
        id
        stripePriceId
        planUsersId
        hasAccess
      }
    }
  `;

  const { data } = await client.mutate({
    mutation: mutationUpdateUser,
    variables: {
      input: {
        id: user.id,
        stripePriceId: priceId,
        planUsersId: plan.id,
        hasAccess: true
      }
    }
  });

  res.status(200).json({ success: true, user: data.updateUser });
})();
```

Attribution automatique d'un Plan et génération de tokens

Lorsqu'un nouvel utilisateur s'inscrit (via Cognito), une mutation GraphQL côté AppSync lui attribue automatiquement un **plan gratuit par défaut**. Ce plan lui donne droit à un quota de tokens mensuels pour générer des effets haptiques.

Lorsque l'utilisateur passe à un plan supérieur via Stripe (voir ci-dessus), le **webhook déclenche une mise à jour** de son plan dans la base de données. La mutation associée initialise ou augmente automatiquement le **nombre de tokens disponibles** en fonction du plan choisi.

Cette logique métier est centralisée dans un resolver GraphQL. Elle garantit que :

- les tokens sont générés ou mis à jour sans intervention manuelle
- les changements de plan sont instantanés et cohérents avec Stripe
- la sécurité est assurée par l'authentification des requêtes (JWT Cognito)

Ainsi, tout nouvel utilisateur peut commencer à générer des effets immédiatement après inscription ou paiement validé.

Lors de l'inscription, un plan par défaut (gratuit) est automatiquement attribué à chaque nouvel utilisateur. Suite à un paiement validé, le webhook met à jour le plan et initialise le quota de tokens associé. Ces opérations sont encapsulées dans une mutation GraphQL sécurisée. Cette logique garantit une expérience fluide, sans intervention manuelle.

Conclusion

Ce projet m'a permis de mettre en pratique un grand nombre de compétences techniques, allant du développement d'interfaces sécurisées à la mise en œuvre d'une architecture cloud moderne basée sur les services AWS. J'ai également appris à intégrer des solutions tierces comme Stripe et à automatiser les processus de déploiement dans une logique DevOps. Haptify incarne une application complète, tournée vers l'innovation, la sécurité et l'expérience utilisateur.

Grâce à la collaboration avec une équipe pluridisciplinaire (développeurs, designers, chercheurs), j'ai pu évoluer dans un environnement stimulant et professionnalisant, tout en contribuant à un produit technologique novateur.

Remerciements

Je remercie chaleureusement toute l'équipe de l'entreprise **V.RTU**, et en particulier :

- Ma tutrice en entreprise pour son accompagnement technique
- la designer et les chercheuses en haptique pour leur expertise et leur disponibilité
- la scrum master pour son soutien dans la gestion agile du projet
- Le chef d'entreprise pour avoir investi et cru en moi

Je tiens également à remercier mes proches pour leur soutien constant.