Julien Bédu

Dossier projet

SKATERQUEST

Titre RNCP 6 · Avril 2025

Développeurs :

Julien Bédu · Thomas Poillion · Baptiste Zuber

La Capsule

SOMMAIRE

PARTIE I · CONCEPTION ET PROTOTYPAGE

I.1 Le projet SkaterQuest

Les skateurs sont amateurs de défis et aiment se surpasser. Et s'il existait enfin une application mobile permettant à un skateur de suivre sa progression, de défier ses pairs et de partager facilement ses prouesses ? C'est l'ambition de SkaterQuest, une application mobile réalisée pour les skateurs en quête de motivation, de progrès et de compétition !

Nous partons en effet du constat qu'actuellement, les quelques applications aspirant à accompagner les skateurs dans leur pratique ne sont pas assez complètes et ergonomiques pour totalement satisfaire leur cible. Par exemple, l'application *Skate Tricks* (éditée par DMFV) propose des outils d'apprentissage de figures de skate mais aucune fonctionnalité sociale, et l'application *Smap* (éditée par Artium) permet uniquement de localiser et enregistrer des spots.

De plus, le skate en tant que discipline sportive connaît un regain de popularité depuis 2020, grâce à l'apparition des premières épreuves de skate aux jeux olympiques de Tokyo.

L'objectif de SkaterQuest est de proposer un outil accessible et ludique pour encourager un skateur à améliorer son niveau, en l'aidant dans son apprentissage et en lui permettant de partager ses progrès et d'estimer ceux des autres.

Notre application se veut à destination de tous les skateurs. Elle s'adresse autant au débutant qui cherche à apprendre les bases qu'au skateur chevronné en quête de défi et de compétition. Pour un ciblage plus précis, l'utilisateur typique de SkaterQuest est un skateur ayant entre 13 et 40 ans et vivant en milieu urbain.

SkaterQuest offre notamment les possibilités suivantes :

- Un suivi d'apprentissage des figures de skate, sous la forme d'un "livre de tricks" (le terme anglais "tricks", plus courant dans le milieu des skateurs, sera employé dans ce dossier comme dans l'application)
- Un outil pour faciliter et améliorer l'expérience du *game of skate*, un jeu populaire chez les skateurs, consistant à enchaîner des tricks jusqu'à ce que l'accumulation de 5 erreurs (représentées par les lettres du mot "SKATE") désigne un perdant
- Une carte interactive permettant de repérer et enregistrer de nombreux spots (sites favorables à la pratique du skate)
- Un système de publication de vidéos des performances des utilisateurs, avec une fonction de vote et de classement pour encourager la compétition et la validation des pairs
- La possibilité de créer une équipe (nous utiliserons le terme "crew") pour réunir ses amis skateurs sous une bannière commune

La réalisation de ce projet a été soumise aux contraintes suivantes :

- Temps limité :
  - 3 jours de pré-production (storyboards, modélisation de la base de données, etc.)
  - 9 jours de développement
  - 1 jour de présentation vidéo
- Équipe limitée : 3 développeurs en formation
- Technologies imposées :
  - Node.js pour le back-end (langage : JavaScript)
  - Expo pour le front-end (langage : React Native)
- Panel d'utilisateurs à sélectionner uniquement parmi nos camarades de formation

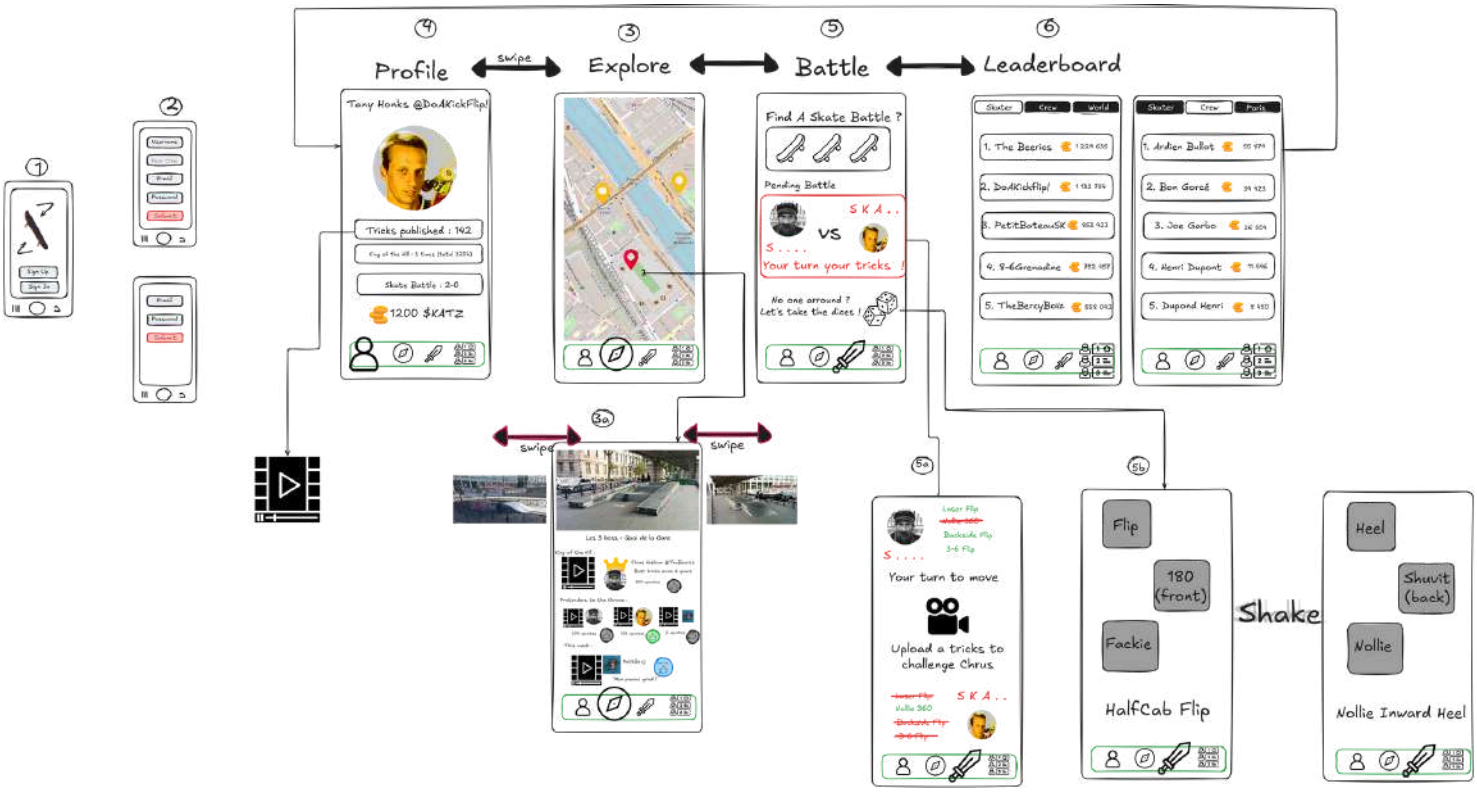
Ces contraintes, notamment la courte durée de développement, nous ont encouragés à travailler de manière particulièrement méthodique. Il était surtout important de bien hiérarchiser l'importance de chaque fonctionnalité de l'application, afin d'y consacrer un temps adéquat.

I.2 Storyboards

Afin de clarifier notre vision de SkaterQuest, nous avons commencé par étudier différentes propositions d'interface en établissant des storyboards. À ce stade, nous avions en tête les fonctionnalités principales que nous souhaitions développer dans l'application, mais nous n'étions pas encore fixés sur leur importance respective.

Nous avons alors réparti notre trio en deux groupes de une et deux personnes, chaque groupe proposant son propre storyboard sans consulter l'autre. Ce travail préparatoire nous a ainsi permis, en plus de préciser l'architecture de l'application, de comparer deux versions de SkaterQuest. Ce fut intéressant car chaque storyboard mettait en avant une fonctionnalité différente, nous avions donc deux propositions d'interface bien distinctes.

La version ci-dessous propose un interface centré autour de la compétition entre les utilisateurs, qui utiliseraient SkaterQuest avant tout pour filmer leurs prouesses sur certains spots et consulter leur classement à l'aide d'un système de votes :

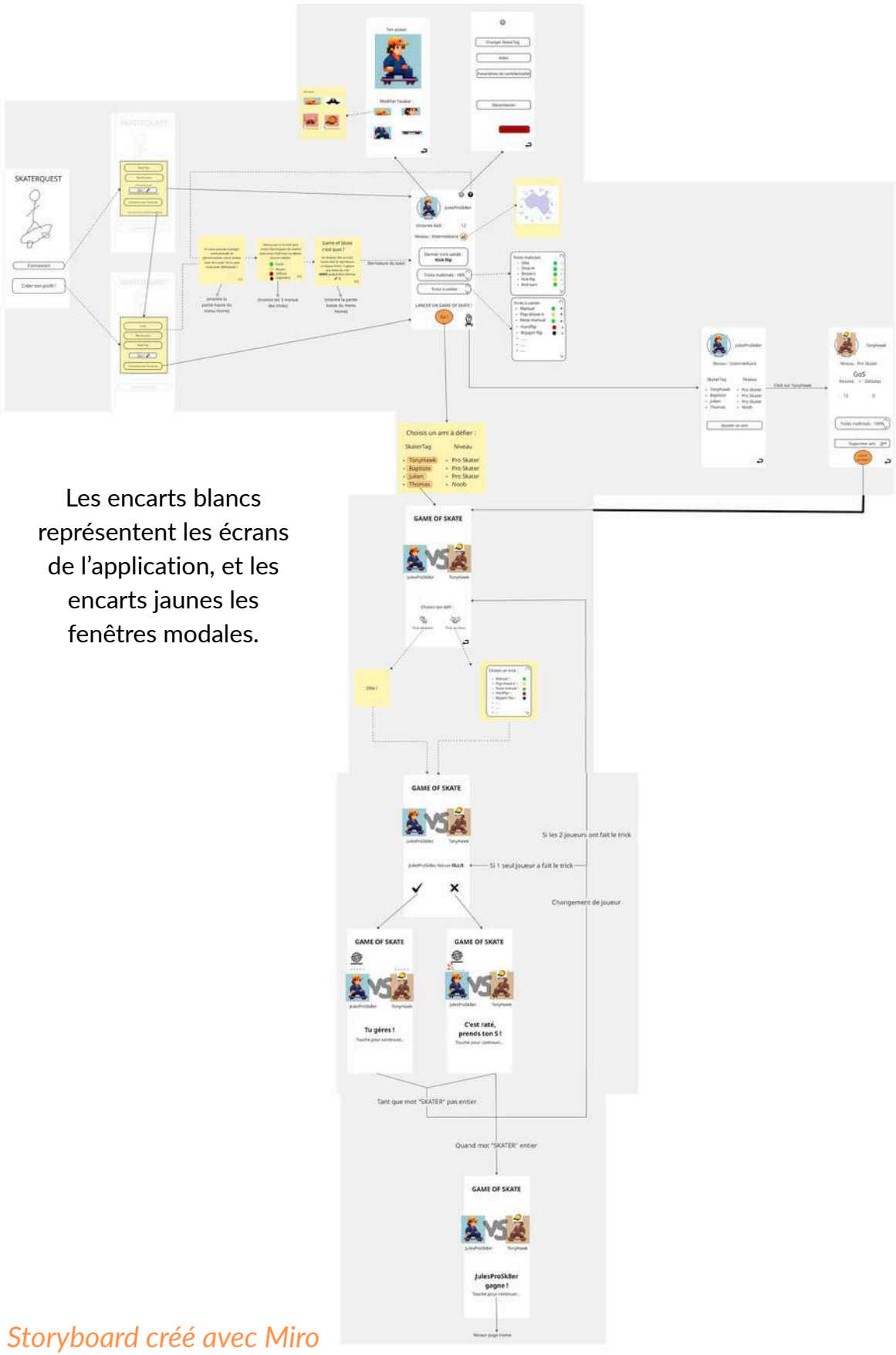


Storyboard créé avec Excalidraw

SKATERQUEST

Notre autre storyboard met davantage en valeur la fonctionnalité du *game of skate* et de la progression personnelle dans l'apprentissage des tricks :

Aperçu global :

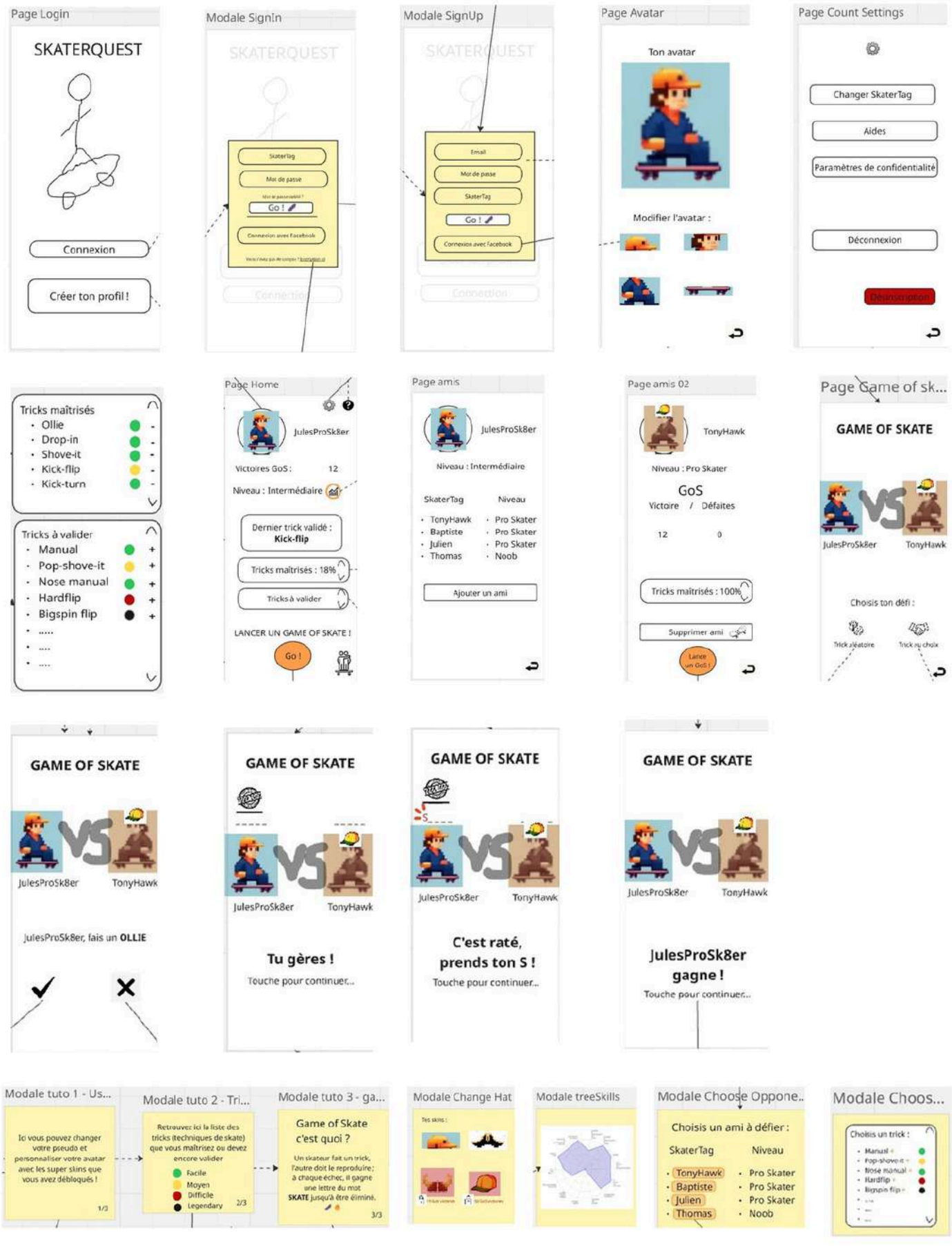


Les encarts blancs représentent les écrans de l'application, et les encarts jaunes les fenêtres modales.

Storyboard créé avec Miro

SKATERQUEST

Aperçu rapproché :



I.3 Panel d'utilisateurs

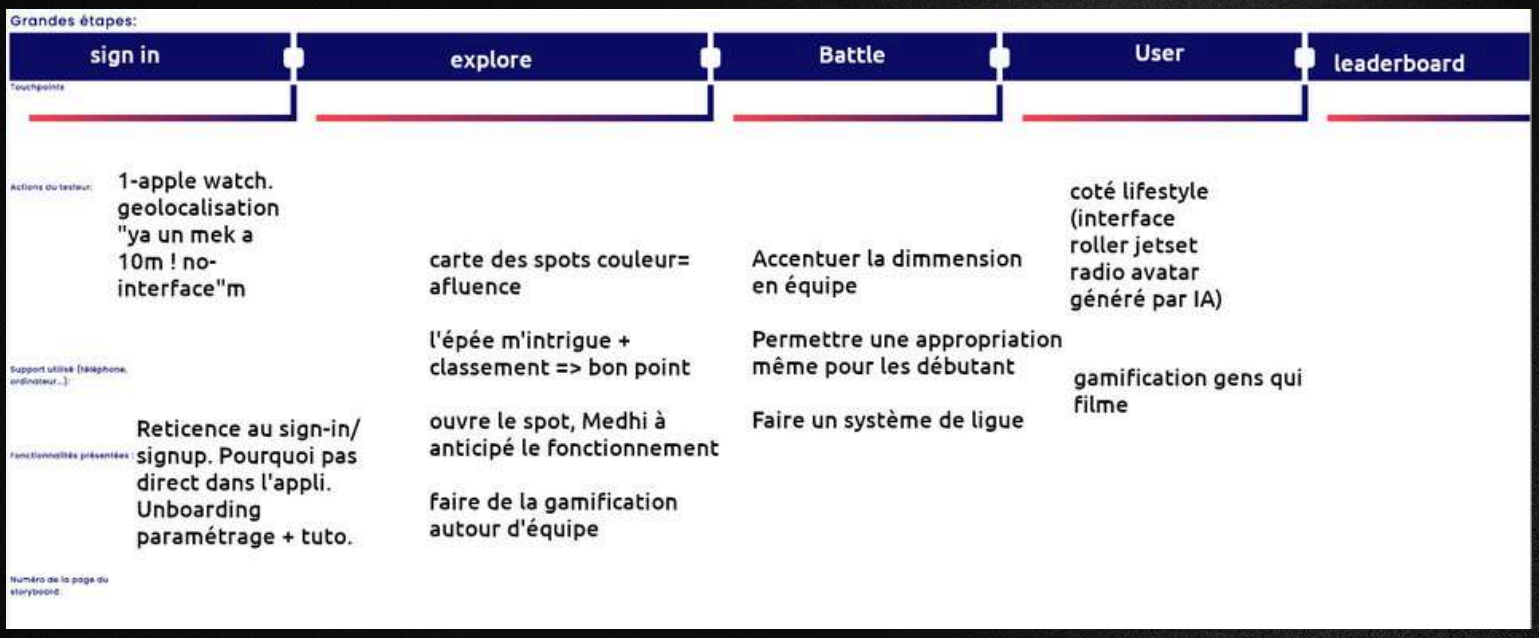
Nous avons ensuite entrepris d'interroger des utilisateurs potentiels afin de tester les storyboards précédents. Notre sélection d'utilisateurs devait se limiter à nos camarades de formation, parmi lesquels ne se trouvait aucun skateur, donc nous avons ciblé ceux dont l'âge se rapprochait le plus de notre cible majoritaire, en nous basant également sur le look (lien avec la culture urbaine et hip-hop). Cela nous a permis d'établir malgré tout un panel d'utilisateurs susceptibles d'utiliser l'application s'ils décidaient de pratiquer le skate.

Testeur	Âge	Amateur de culture urbaine et underground	Activité sportive			Any Other Business
			Fréquence	Intensité	Contexte	
Mehdi	31	Oui	3-4 fois par semaine	Modérée	Solo, parfois avec des amis	Pense se réinscrire dans un club début septembre. Aime sortir de sa routine et testant de nouveaux sports de glisse, souvent hors contexte urbain (surf, ski, etc.)
Andy	28	Oui	Ne pratique pas	Ne pratique pas	Ne pratique pas	Amateur de culture skate mais n'en fait pas lui-même
Clovis	36	Oui	3-4 fois par semaine	Maintien en forme et intensif	Solo et sport de combat	A beaucoup pratiqué les sports de combat (boxe thaï et MMA) en salle de sport
Jules	28	Oui	1 fois par semaine	Décrassage	Solo	A déjà pratiqué le skate au skatepark, vision positive de ce sport
Morgan	27	Oui	1 fois par mois	Maintien en forme	Solo	A déjà pratiqué le skate

I.4 User journeys

Les utilisateurs de notre panel ont alors accepté de participer à une simulation de parcours de notre application, sur la base de nos storyboards, ce qui nous a permis d'établir cinq user journeys (ou tests utilisateurs). Mehdi et Andy ont expérimenté un scénario se basant sur le storyboard affiché en premier ci-dessus, tandis que Clovis, Jules et Morgan ont testé le second. Cette section présente un user journey pour chacun des deux storyboards.

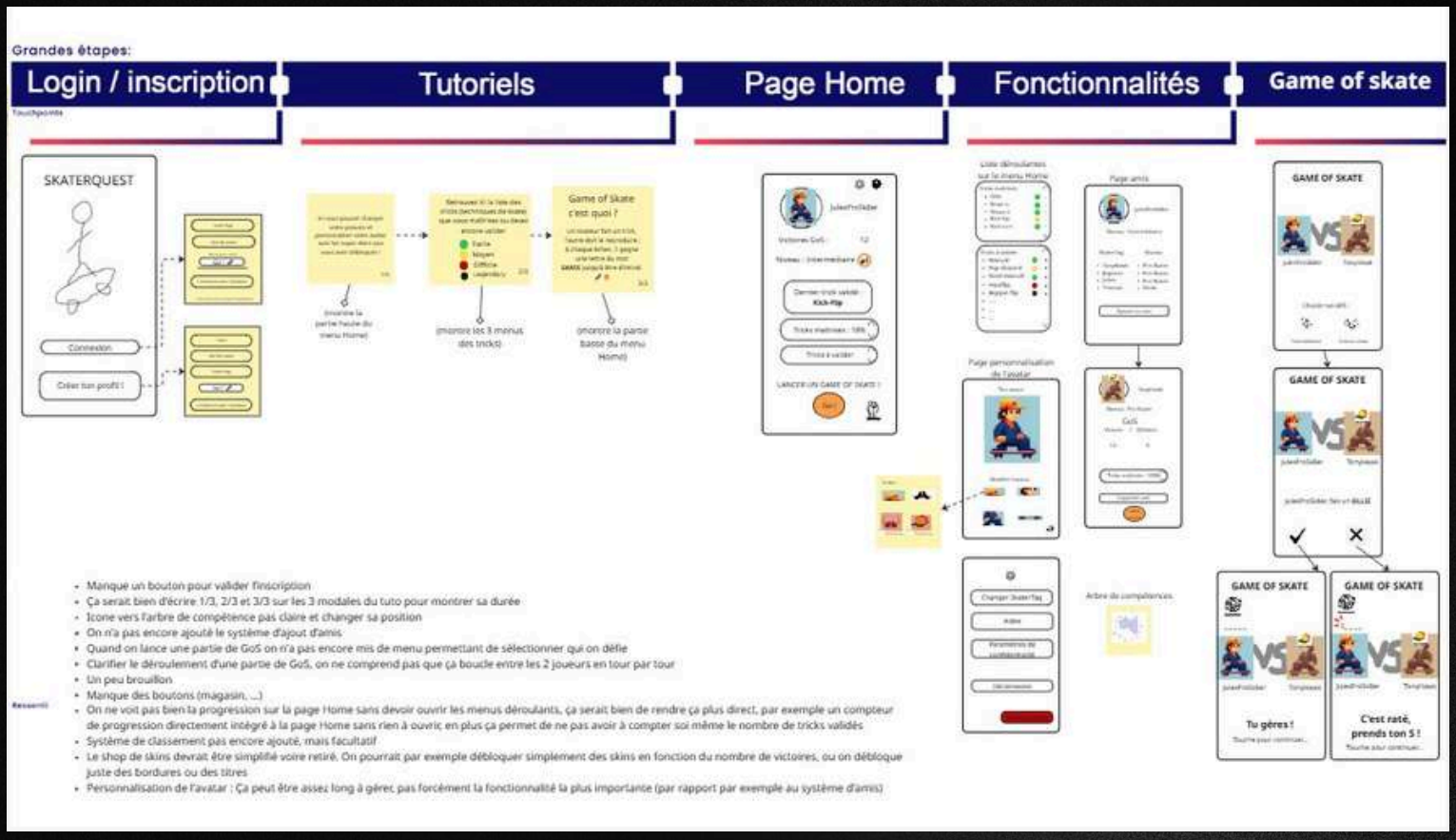
Premier storyboard : User journey de Mehdi



Retranscription des retours de Mehdi :

- Réticence à la création d'un compte. Peut-être commencer plutôt par un tutoriel
- Le système de classement est très intéressant. On pourrait mettre en place un système de ligues (débutant/amateur/professionnel) et ainsi renforcer l'importance des équipes (crews) tout en rendant ce système plus ludique
- L'application devrait s'attacher à reprendre les codes culturels du milieu
- Peut-être donner un côté "lifestyle" à l'interface
- On pourrait lier la localisation des spots à une Apple Watch
- On pourrait utiliser un système de couleurs pour indiquer l'affluence sur les spots
- Rendre plus ludique le fait de filmer des tricks
- Attention à rendre l'application accessible aux débutants aussi

Second storyboard : User journey de Clovis



Retranscription des retours de Clovis :

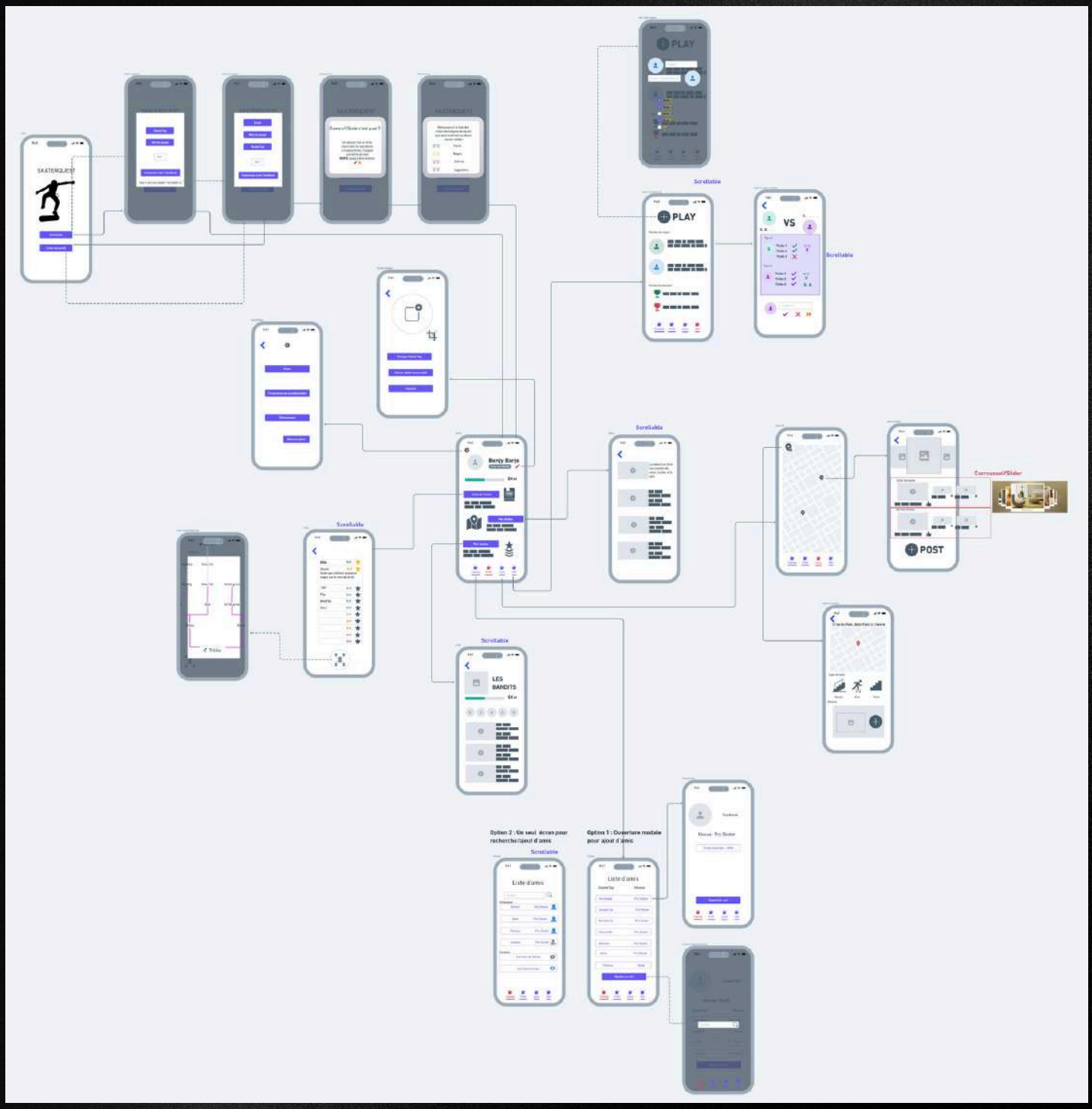
- Il manque un bouton pour valider l'inscription
- Il serait mieux d'indiquer la durée du tutoriel en ajoutant “1/3”, “2/3” et “3/3” sur les trois fenêtres modales successives présentant l'application lors de l'inscription
- L'icône vers l'arbre de compétence n'est pas claire, ainsi que sa position
- Il manque le système d'ajout d'amis
- Quand on lance une partie de *game of skate*, il manque un menu pour sélectionner l'adversaire à défier
- Le déroulement d'une partie de *game of skate* n'est pas très clair, on ne comprend pas que ça boucle entre les deux joueurs, en tour par tour
- Le tout est un peu brouillon
- On ne voit pas bien la progression des tricks sur la page d'accueil sans devoir ouvrir les menus déroulants, il serait mieux par exemple d'avoir un compteur de progression directement intégré à la page d'accueil sans rien avoir à ouvrir. Et avoir un indicateur de progression globale permettrait de ne pas devoir compter soi-même le nombre de tricks validés
- Il manque des boutons, comme celui pour accéder à la boutique de skins pour personnaliser l'avatar

- Le système de classement n’est pas encore présent, mais il est facultatif
- La boutique de skins de personnalisation devrait être simplifiée, voire supprimée. On pourrait par exemple débloquer simplement des skins en fonction du nombre de victoires, ou simplement débloquer des bordures pour l’avatar, ou des titres
- La personnalisation de l’avatar semble une fonctionnalité longue à développer alors qu’elle semble moins importante que d’autres, comme le système d’amis

I.5 Wireframe

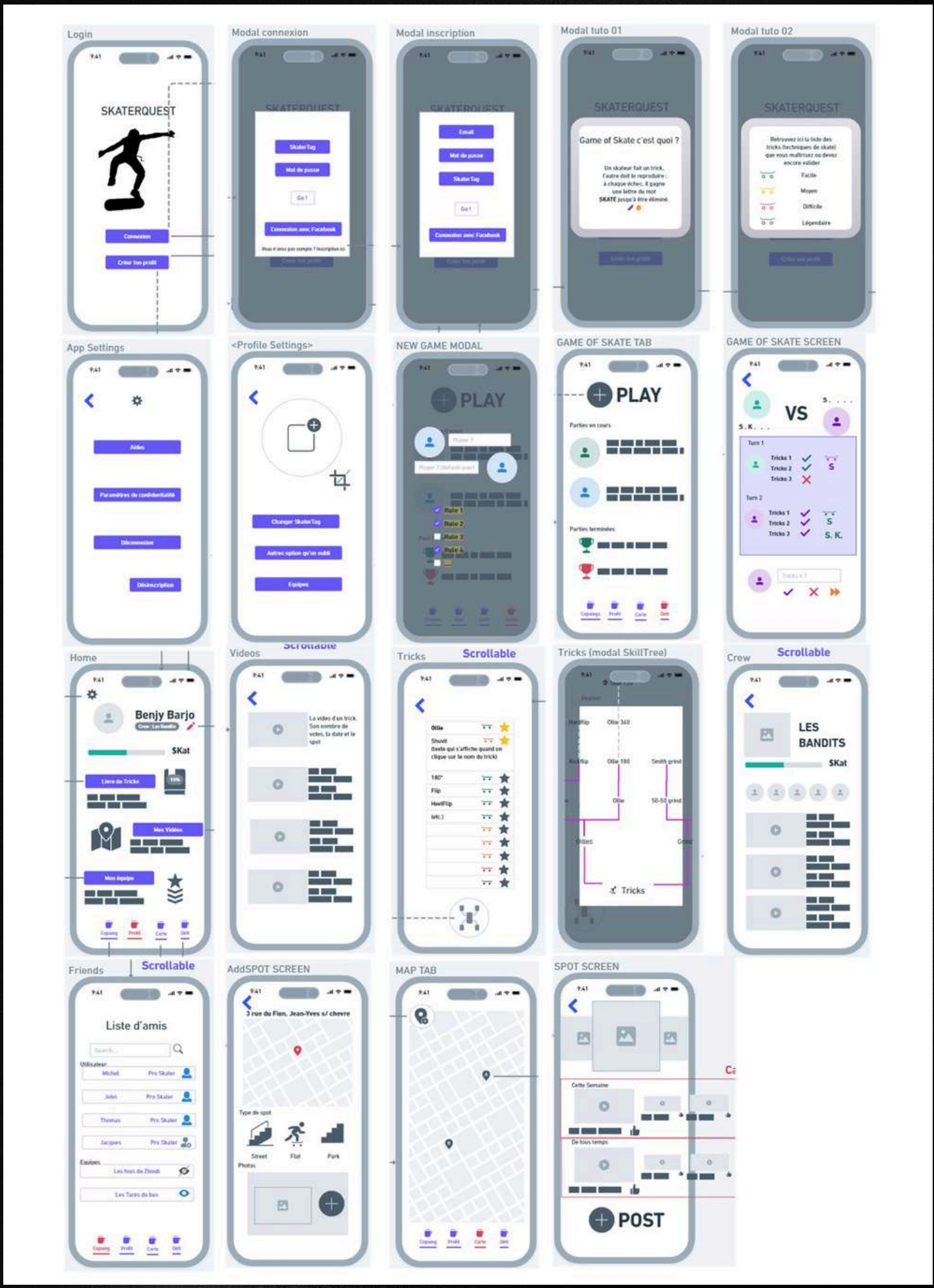
Avoir établi et testé nos storyboards nous a permis de préciser notre vision de SkaterQuest. Nous avons ainsi mis en commun les résultats des précédentes étapes afin d'en dégager une version plus aboutie, illustrée ci-après sous la forme d'un wireframe, réalisé avec Whimsical. En comparaison avec les storyboards présentés plus tôt, nous avons par exemple décidé de supprimer la personnalisation de l'avatar, trop longue à développer pour une fonctionnalité secondaire, simplifié le *game of skate* et retiré certains compteurs de progression qui rendaient l'application trop chargée.

Aperçu global :



SKATERQUEST

Aperçu rapproché :

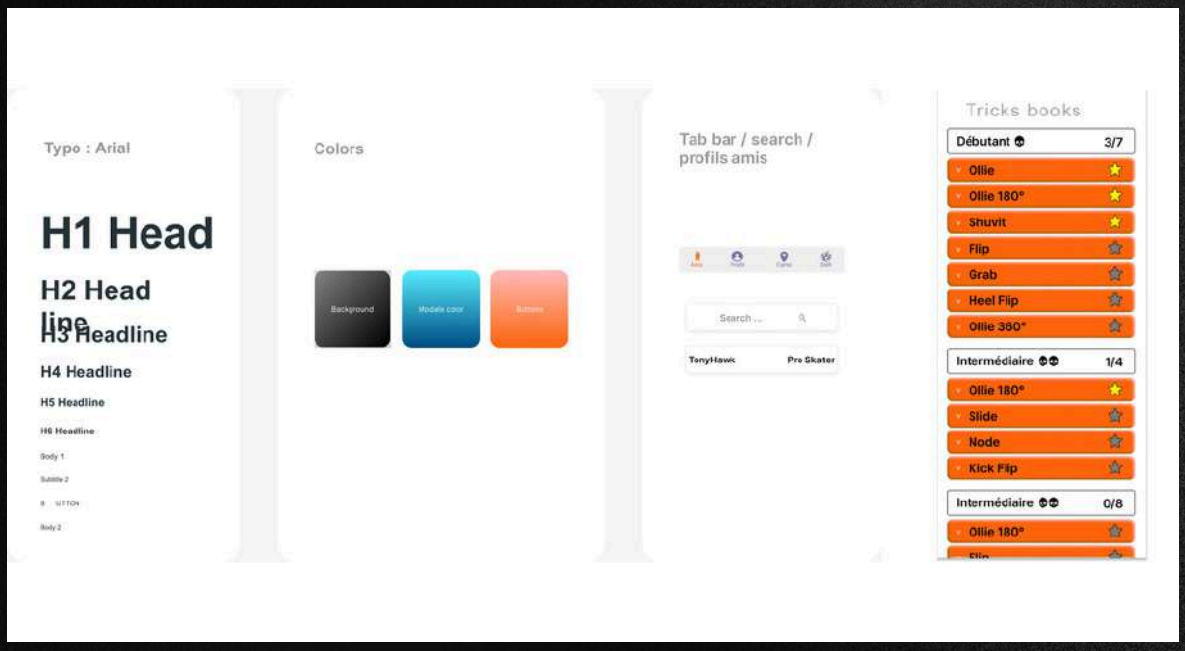


I.6 UI kit

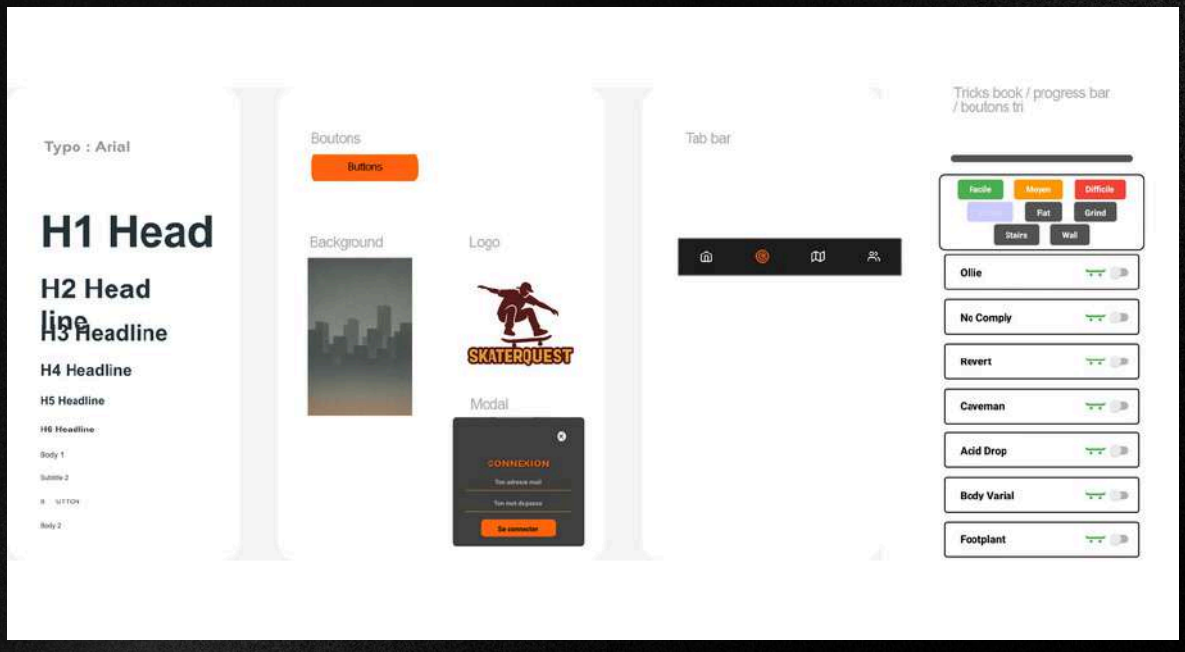
Nous avons ensuite défini un style pour SkaterQuest, en réalisant l'UI kit suivant à l'aide de Figma.

Nous souhaitions évoquer sobrement le monde urbain. Nous avons ainsi opté pour un style minimaliste, et une monochromie de orange (évoque la vitesse et la signalisation routière) combinée à des nuances de gris-noir (évoquent le goudron et les grandes villes). Nous avions originellement pensé utiliser du bleu mais la monochromie nous a finalement paru plus élégante.

UI kit - Version de test :

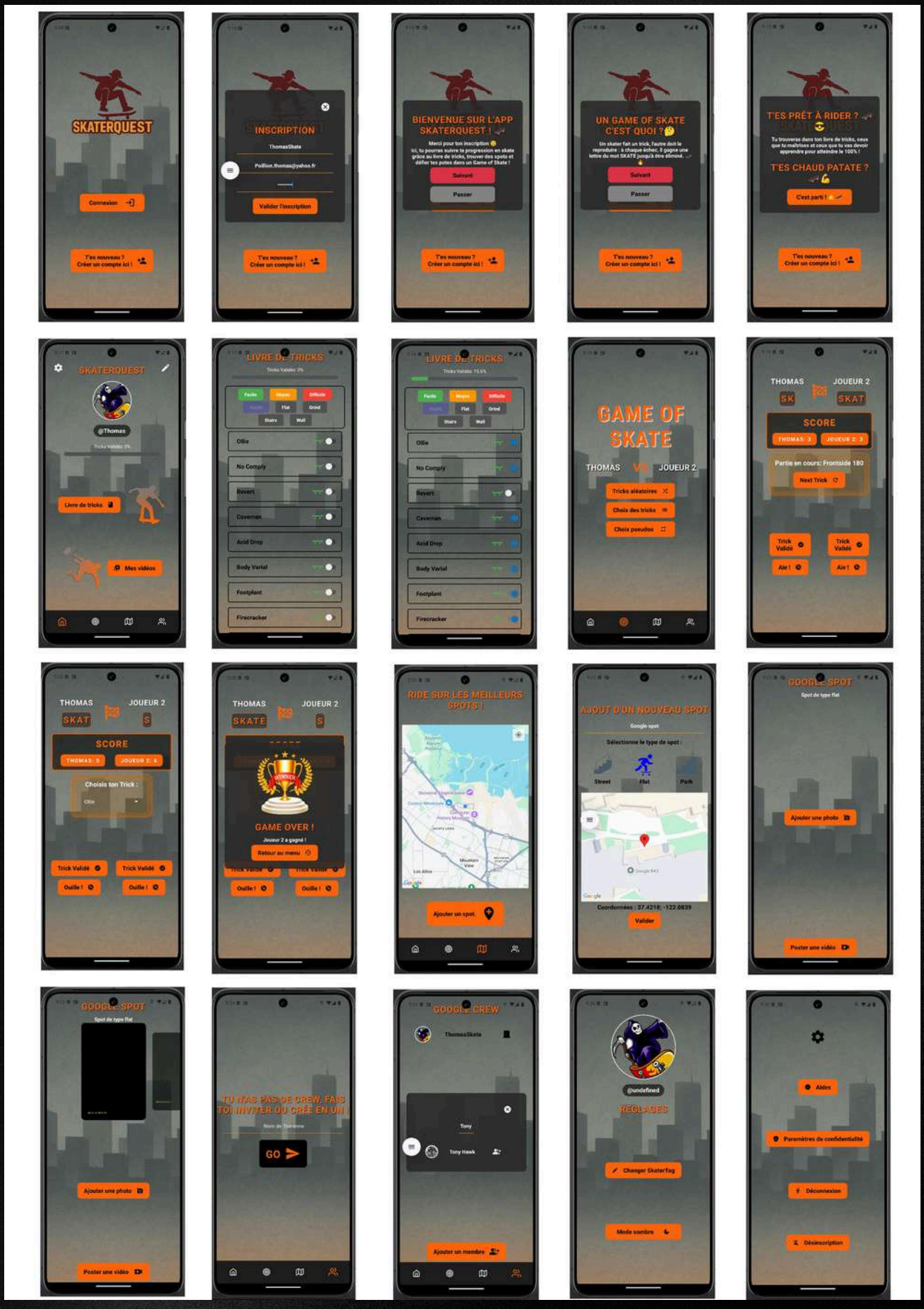


UI kit - Version finale :



I.7 Maquette finale

Toutes ces étapes préparatoires nous ont permis d'élaborer la maquette suivante :



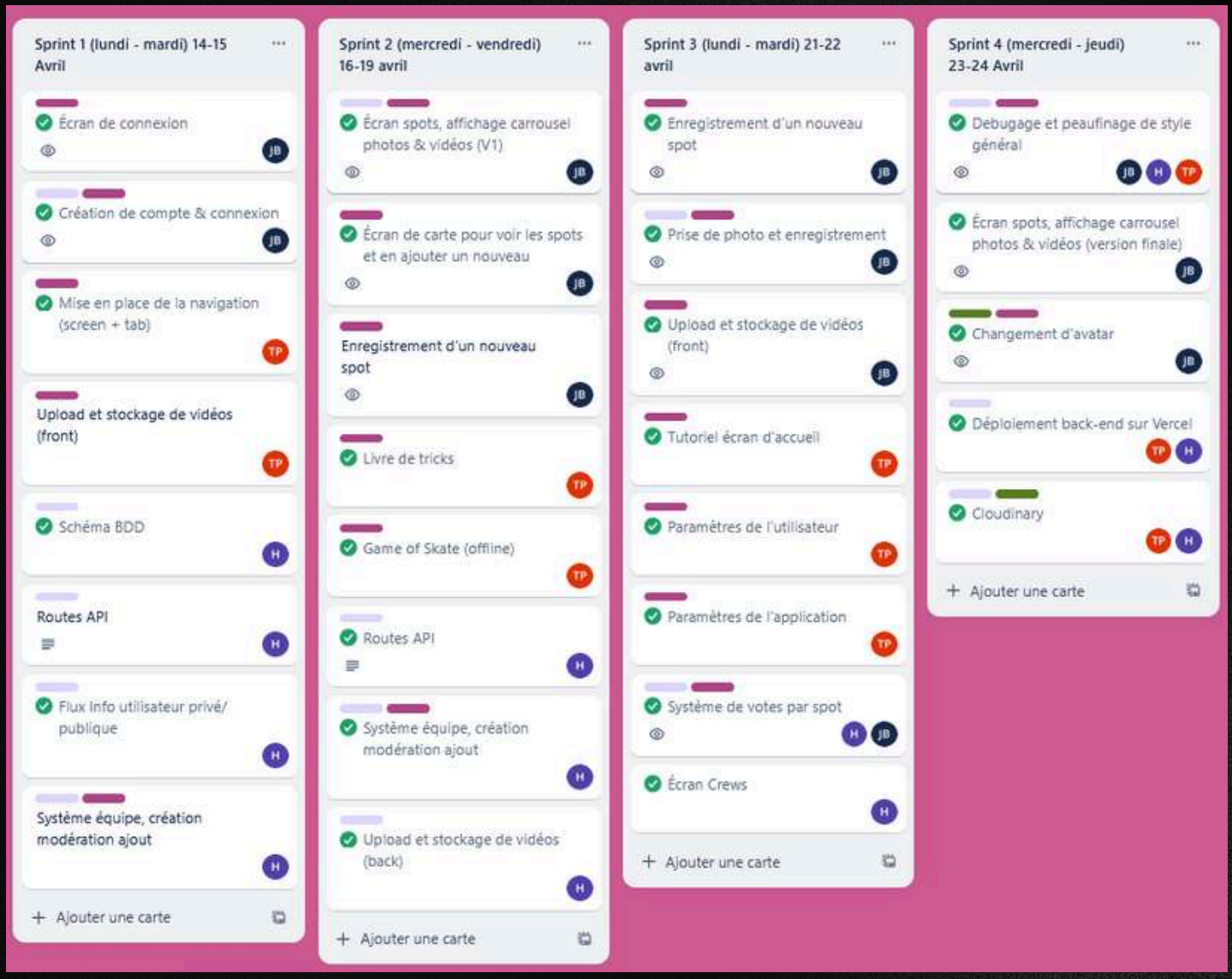
PARTIE II · PRÉPARATION DU PROJET

II.1 Préparation du sprint de développement

Nous avons préparé notre sprint de développement en établissant sur Trello des user stories réparties en 4 sprints de 2 jours (3 jours pour le second sprint).

Nous avons commencé par lister les fonctionnalités à développer, dans la colonne “backlog” du tableau kanban ci-dessous, qui montre la préparation de nos user stories.

Sprints de développement :



Lien vers le tableau Kanban sur Trello : https://trello.com/b/qQXDCKtU/skaterquest

II.2 Analyse des compétences requises

Voici la liste des compétences nécessaires au développement de SkaterQuest :

- Maîtrise des langages, frameworks et outils suivants :
  - JavaScript et React Native
  - CSS (au moins les propriétés fondamentales)
  - Express JS
  - Redux
  - Expo

- Création d'un back-end avec Node JS et Express JS
- Architecture MVC du projet

- Création de base de données avec MongoDB
- Système CRUD

- Requêtes vers les API, et du front-end vers le back-end
- Cryptage de données
- Navigation au sein d'une application mobile
- Contrôle, test et debugage du code

- Travail collaboratif sur GitHub, avec gestion des branches et des conflits
- Déploiement de l'application sur Vercel

- Esprit d'équipe avec communication efficace
- Exploitation de la documentation en ligne
- Connaissances suffisantes sur l'univers et la pratique du skate

II.3 Schéma de base de données

Nous avions d'abord pensé la modélisation de notre base de données comme suit :

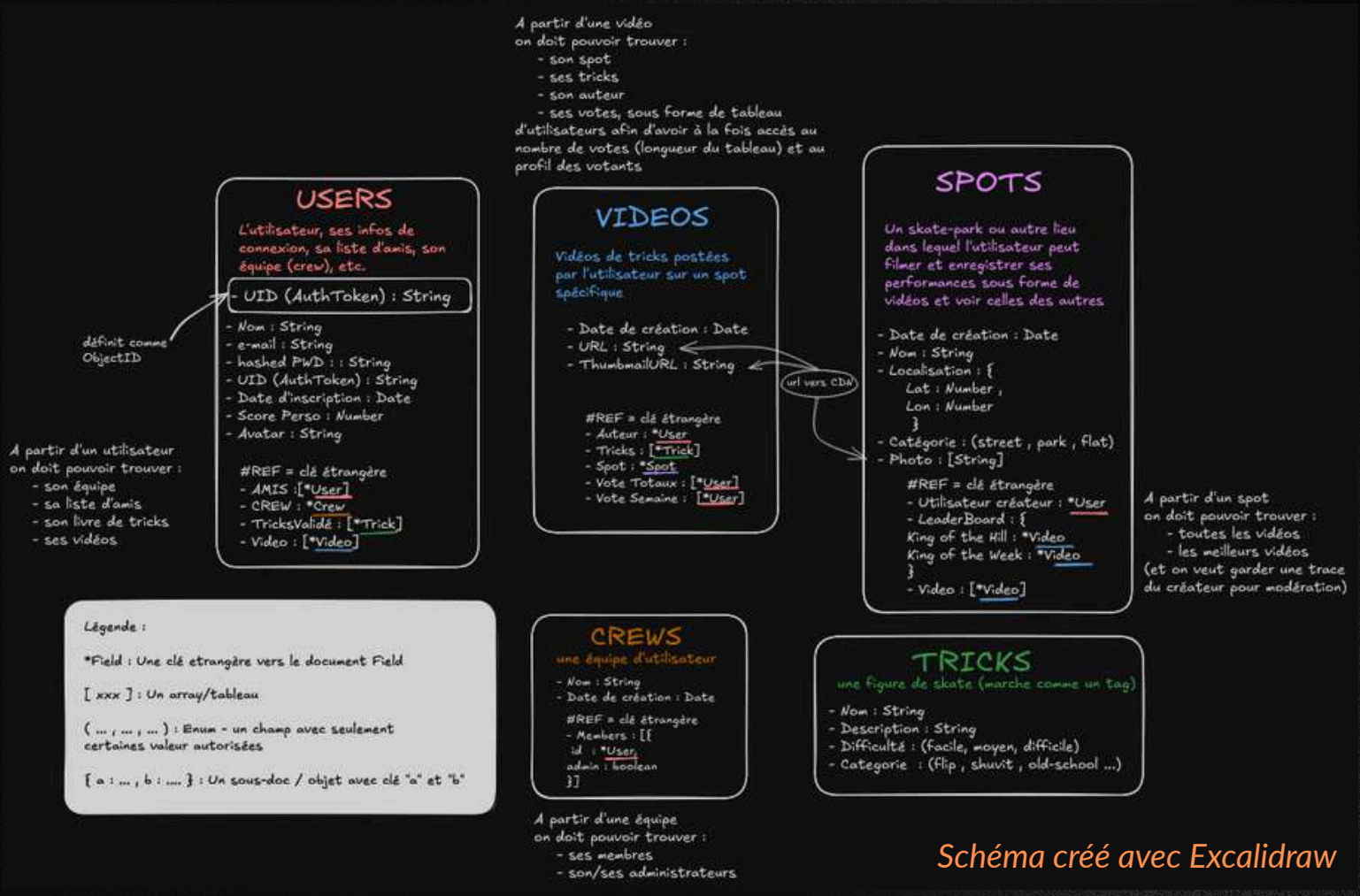


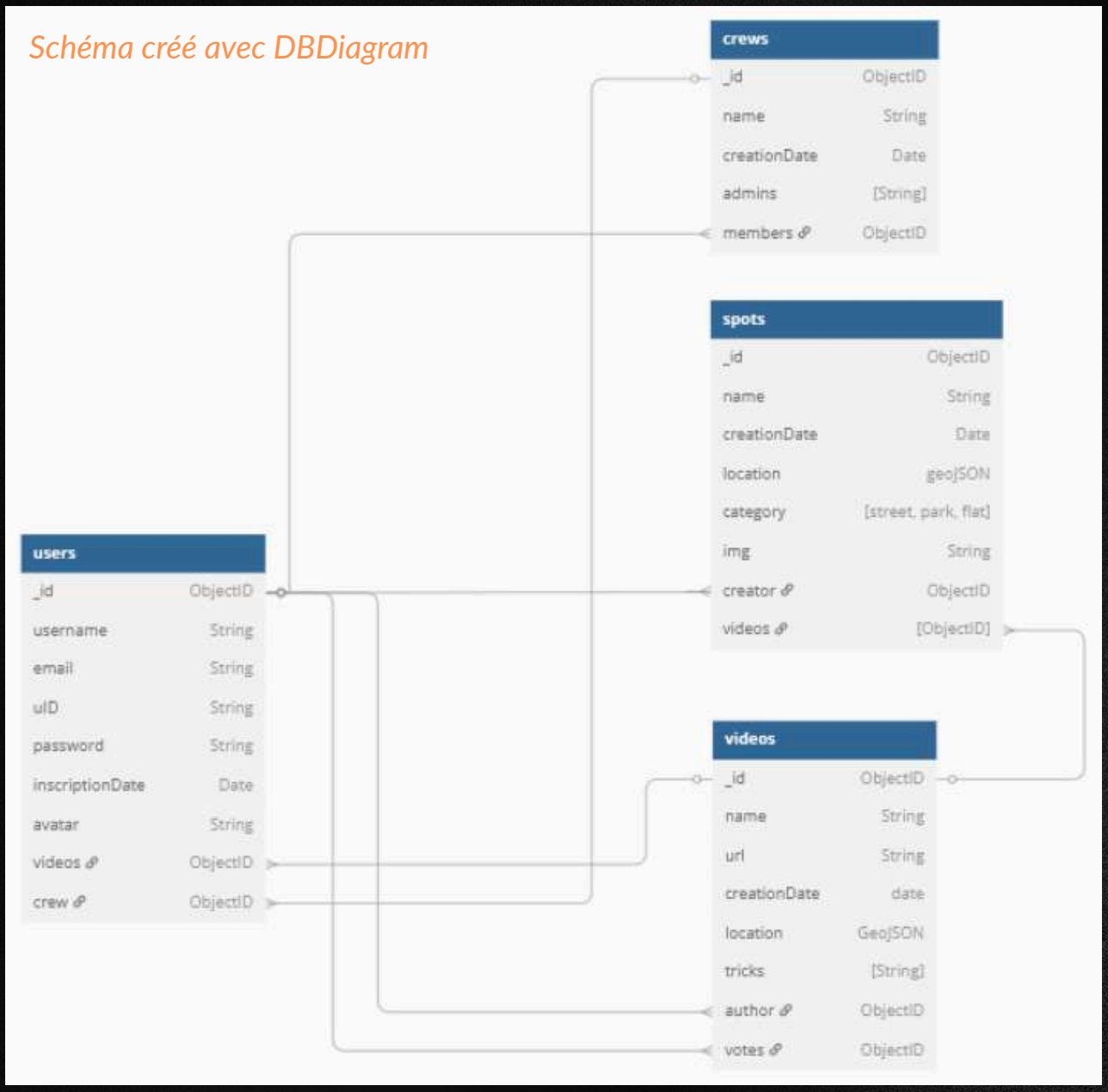
Schéma créé avec Excalidraw

Mais en début de développement, nous avons réalisé qu'il était plus pertinent d'enregistrer les tricks en store Redux uniquement, et ajouté quelques autres modifications. Un schéma actualisé et plus lisible est disponible sur la page suivante.

Nous avons ainsi eu besoin des quatre collections suivantes :

- "users" : Données des utilisateurs, comme le pseudonyme (username), le mot de passe (crypté) et l'URL de l'avatar
- "crews" : Données des équipes, comme la liste des administrateurs et la date de création
- "videos" : Données des vidéos, comme l'URL, le spot associé et les tricks effectués dessus
- "spots" : Données des spots, comme les coordonnées (location) et la catégorie

Ce qui donne le schéma actualisé ci-dessous :



Concernant les relations entre les données, nous avons utilisé des clés étrangères, représentées par les lignes grises dans l'illustration ci-dessus. Elles nous ont notamment servi à lier chaque utilisateur à son crew, ses vidéos publiées et ses spot ajoutés, et à lier chaque vidéo au spot correspondant.

La manière dont l'application utilise ses données rendait les relations en sous-documents moins pertinentes, nous n'en avons donc pas utilisées. Par exemple, chaque utilisateur peut visualiser les vidéos liées à un spot même si elles n'ont pas été publiées par lui-même, donc il valait mieux relier la collection des vidéos à celle des utilisateurs et des spots avec des clés étrangères.

II.4 API du back-end

Le back-end de SkaterQuest contient des routes liées aux quatre collections de notre base de données. La présente section décrit ces routes.

Préfixe de la route	Nom de la route	Type	Rôle	Entrée	Sortie
/user	/	GET	Récupération des informations de l'utilisateur connecté		{ result: true, data: user }
	/:uID	GET	Récupération des données d'un utilisateur à l'aide de l'uID	Params : { uID }	{ result: true, data: user }
	/search/:searchTerm	GET	Recherche d'un utilisateur à l'aide de son nom	Params : { username }	{ result: true, data: user }
	/extend	GET	Renouvellement du token		{ result: true, data: { token } }
	/signup	POST	Inscription d'un nouvel utilisateur	Body : { email, username, password }	Succès : { result: true, data: { token, uID, username, email } } Échec : • 400 (erreur base de données) • 401 (utilisateur existant)
	/signin	POST	Connexion d'un utilisateur déjà inscrit	Body : { username, password }	Succès : { result: true, data: { token, uID, username, email } } Échec : • 400 (utilisateur inexistant) • 401 (mot de passe invalide)
	/avatar	POST	Mise à jour de l'image de profil	Body : { token } Files : { photoFile }	Succès : { result: true } Échec : • 400 (erreur dans l'envoi d'image) • 500 (erreur de serveur (Cloudinary))
	/skaterTag	PUT	Mise à jour du nom d'utilisateur	Body : { username }	Succès : { result: true } Échec : • 400 (champ manquant) • 500 (erreur de serveur)

SKATERQUEST

Préfixe de la route	Nom de la route	Type	Rôle	Entrée	Sortie
/user	/	DELETE	Suppression du compte de l'utilisateur		Succès : { result: true, message: "Compte supprimé avec succès" } Échec : • 404 (utilisateur introuvable) • 500 (erreur de serveur)
/crew	/:crewID	GET	Récupération des données d'un crew	Params : { crewID }	Succès : { result: true, data: crew } Échec : • 404 (crew introuvable)
	/	POST	Création d'un nouveau crew en entrant son nom	Body : { name }	Succès : { result: true, data: newCrew } Échec : • 400 (utilisateur déjà dans un crew)
	/promote/:targetUserID	PUT	Promotion d'un membre du crew en admin.	Params: { targetUserID }	Succès : { result: true } Échec : • 400 (échec lors de la promotion)
	/demote/:targetUserID	PUT	Retrait du grade d'admin. à un membre du crew	Params: { targetUserID }	Succès : { result: true } Échec : • 400 (échec de la rétrogradation)
	/add/:targetUserID	PUT	Ajout d'un utilisateur dans le crew	Params: { targetUserID }	Succès : { result: true } Échec : • 400 (utilisateur déjà dans un crew ou erreur)
	/remove/:targetUserID	PUT	Retrait d'un utilisateur d'un crew	Params: { targetUserID }	Succès : { result: true } Échec : • 400 (échec lors de la suppression)
	/leave	PUT	Sortie de l'utilisateur du crew		Succès : { result: true } Échec : • 400 (non membre ou erreur)

SKATERQUEST

Préfixe de la route	Nom de la route	Type	Rôle	Entrée	Sortie
/spot	/loc/:lon/:lat/:limit	GET	Récupération des spots proches de la localisation de l'utilisateur	Params : { lon, lat, limit }	Succès : { result: true, data: [spots] } Échec : • 400 (aucun résultat)
	/:spotID	GET	Récupération d'un spot par son ID	Params : { spotID }	{ result: Boolean(data), data: spot }
	/	POST	Création d'un nouveau spot	Body : { name, lon, lat, category }	Succès : { result: true, data: spot } Échec : • 400 (erreur lors de l'ajout du spot) • 406 (spot trop proche) * * On empêche la création de deux spots au même endroit
	/picture/:spotID	POST	Ajout d'une photo au spot	Params : { spotID } Files : { photoFile }	Succès : { result: true } Échec : • 400 (erreur lors de l'ajout de l'image) • 500 (erreur de serveur (Cloudinary))
/video	/	POST	Publication d'une vidéo, avec saisie des tricks exécutés dessus	Body : { tricks, spot }	Succès : { result: true, data: video } Échec : • 400 (erreur de l'ajout en BDD) • 500 (erreur de serveur (Cloudinary))
	/upvote/:videoID	PUT	Incrémentation du compteur de votes de la vidéo	Params : { videoID }	Succès : { result: true } Échec : • 400 (ID de la vidéo incorrect)
	/unvote/:videoID	PUT	Décrémentation du compteur de votes de la vidéo	Params : { videoID }	Succès : { result: true } Échec : • 400 (ID de la vidéo incorrect)
	/:videoID	DELETE	Suppression de la vidéo par son propriétaire	Params : { videoID }	Succès : { result: true } Échec : • 400 (vidéo inexistante ou utilisateur non propriétaire)

Nous avons également mis en place des middlewares, qu'une grande partie des routes décrites ci-dessus utilisent. Ils nous ont permis de cloisonner et optimiser davantage le code, et de simplifier la gestion des retours d'erreurs.

Ces middlewares sont les suivants :

Nom du middleware	Rôle	Exemples de routes utilisant le middleware
checkBody	Vérifie que chaque champ passé en paramètre a été rempli, afin de permettre ou non l'exécution d'une route nécessitant des entrées de l'utilisateur dans le corps de la requête	• POST /user/signup • POST /video (car l'utilisateur doit entrer manuellement les tricks de la vidéo)
trimFields	Supprime les espaces de tous les champs passés en paramètres, afin de "nettoyer" les entrées de l'utilisateur	
getUserData	Renvoie un corps de requête contenant toutes les informations de l'utilisateur	• POST /crew • POST /spot
isUserCrewAdmin	Vérifie si le membre d'un crew est admin. ou non	• PUT crew/promote/:targetUserID • PUT crew/add/:targetUserID
tokenAuth	Possède deux fonctions, une qui génère un token et une qui vérifie que la requête contient un token valide dans son header	• POST /user/signup (génération de token) • PUT /video/upvote/:videoID (vérification du token)

PARTIE III · PILOTAGE DU DÉVELOPPEMENT

III.1 Déroulé des sprints de développement

Cette section résume le déroulé de chacun de nos quatre sprints de développement, anticipés sur Trello et montrés précédemment (partie II.1). Nous avions l'habitude de faire un point le matin et en début d'après-midi sur la répartition des tâches de chacun, puis d'adapter cette organisation selon les éventuelles difficultés rencontrées. Nous communiquions aussi beaucoup et n'hésitions pas à envoyer des commits fréquents, à la fois pour limiter au maximum le risque de conflits de code et pour avoir des feedbacks réguliers sur les modifications et ajouts effectués.

Sprint 1 :

Pour ce premier sprint, Thomas et moi avons notamment commencé par créer le squelette de l'application dans le back-end en mettant rapidement en place les différents écrans et la navigation entre eux, tandis que Baptiste s'est concentré sur le back-end et les interactions entre le back et le front.

Thomas s'est principalement occupé de la navigation et de la structure de certains écrans et composants. Il a ensuite commencé à développer les fonctionnalités liées à la publication et au stockage des vidéos mais a rencontré des difficultés liées à l'upload, et nous avons décidé de remettre leur développement à plus tard.

Me concernant, j'ai légèrement touché au back-end en créant le schéma des utilisateurs (users) afin de faire gagner du temps à Baptiste. Puis après avoir préparé quelques écrans en complément de Thomas, je me suis chargé des fonctions d'inscription et de connexion de l'utilisateur, ainsi que de l'affichage dans l'écran de connexion (fenêtres modales conditionnelles, inputs, messages d'erreur, etc.). L'écran et ses fonctionnalités étaient opérationnelles en fin de sprint, mais je n'étais pas totalement satisfait de l'affichage que j'avais mis en place, l'écran de connexion manquant d'ergonomie. Nous avons ultérieurement amélioré ça.

Côté back-end, Baptiste a mis en place les schémas de notre base de données, puis a commencé à créer les nombreuses routes décrites dans la section précédente. Il en a aussi profité pour travailler sur les mécaniques de sécurité (flux des informations, authentification par token, cryptage des mots de passe). En complément des routes, Baptiste a aussi commencé à mettre en place les middlewares présentés plus tôt, et s'est également lancé dans la création de fonctions faisant le pont entre le front-end et le back-end, placées dans un fichier *request.js* dans le front. Ces fonctions avaient pour but de faciliter et optimiser le développement des fonctionnalités du front, en se chargeant de l'envoi des requêtes. Nous avons ainsi pu réunir nos *fetch* du front-end dans un même fichier, qui sera par la suite importé dans les différents composants de l'application. Baptiste a eu affaire à quelques bugs dans le back, que nous avons mis pour certains du temps à corriger, mais tout était réglé en fin de sprint. Il a aussi entrepris de mettre en place des tests (TDD), dont la méthodologie et la justification sera détaillée dans la partie IV.3 de ce dossier. Baptiste a enfin entamé une partie de la fonctionnalité du système de crews (création de l'équipe, ajout et modération des membres), qu'il terminera lors du sprint suivant.

Le bilan de ce premier sprint fut globalement positif. Les écrans mis en place et la navigation opérationnelle nous permettaient désormais de parcourir l'application afin d'avoir un bon aperçu de ses évolutions. Nous avions de plus suffisamment de routes fonctionnelles pour nous permettre de développer et tester rapidement plusieurs fonctionnalités dans le front-end ces prochains jours, le tout avec une méthodologie de cloisonnement et d'optimisation qui nous a fait démarrer sur des bases solides. Nous avions pu respecter les tâches prévues dans notre planning des user stories, et n'avons donc pas significativement changé nos plans pour le sprint suivant.

Sprint 2 :

Ce second sprint a occupé la deuxième moitié de cette première semaine de développement. Nous l'avons mis a profit pour terminer le back-end et coder les premières "vraies" fonctionnalités de SkaterQuest.

Baptiste a poursuivi son travail sur le back-end et la communication depuis le front-end. Il a fini la création des routes et des middlewares, et a avancé sur les fonctions de requêtes depuis le front-end (fichier *request.js*). Il a également terminé le développement des fonctionnalités liées aux crews, commencées lors du sprint précédent.

Thomas s'est attaqué à deux fonctionnalités très importantes de l'application, décrites en introduction : le *livre de tricks* et le *game of skate*. Pour le livre de tricks, il a mis en place un système de filtres très pratique pour permettre à l'utilisateur d'afficher facilement les figures selon leur difficulté, leur catégorie et s'il les connaît ou non. Concernant le *game of skate*, Thomas a pu au terme de ce sprint aboutir à une version fonctionnelle malgré quelques bugs, essentiellement visuels. Comme nous l'avions décidé durant la préparation des user stories, il s'agissait d'une version hors-ligne, sans enregistrement des victoires des utilisateurs. Une version en ligne correcte du *game of skate* prendrait beaucoup de temps à développer et nous l'avions ainsi envisagée comme une fonctionnalité bonus, que nous n'avons finalement pas décidé de prioriser dans le cadre de ce MVP (*minimum viable product).*

De mon côté, je me suis concentré sur les fonctionnalités liées aux spots. J'ai commencé par mettre en place la carte affichant la localisation de l'utilisateur et des spots proches de lui. Je me suis ensuite chargé de l'affichage d'un spot, accessible en touchant son icône sur la carte (ou lors de la création d'un nouveau spot), ce qui m'a pris beaucoup de temps. En effet, l'écran d'un spot devait montrer les photos du spot et les vidéos filmées dessus, et nous avions envie d'un affichage de type carrousel, pour un défilement dynamique et fluide des images. N'étant encore ni totalement à l'aise avec l'affichage précis et le responsive, et n'ayant jamais expérimenté les options d'animation en React Native, j'ai du étudier le fonctionnement de plusieurs composants et attributs afin de pouvoir mettre en place le carrousel de la manière que j'avais en tête. Le résultat était plutôt convainquant, malgré quelques défauts que je décidais de régler plus tard, lorsque le chargement des photos et des vidéos serait opérationnel (je testais alors mon travail avec des images temporaires). J'ai ensuite commencé à ajouter la possibilité d'ajouter un nouveau spot à l'endroit où l'on se trouve, sans avoir le temps de terminer. La finalisation de cette fonctionnalité est reporté au prochain sprint.

Durant ce sprint et les suivants, nous avons parfois rencontré un problème lié à l'affichage dynamique des composants. En effet, leur structure et notre manière de gérer les données et les états ont causé certaines fois des soucis d'actualisation des composants, qui auraient contraint l'utilisateur à quitter puis revenir sur l'écran pour afficher certaines modifications. Baptiste a réglé le problème en créant dans les écrans concernés un état dédié à la mise à jour forcée du composant, en utilisant le hook *useReducer* de React. Il s'agit cependant d'une solution non optimale, que nous aimerions améliorer plus tard, hors du cadre de ce MVP.

Sprint 3 :

Notre objectif pour cet avant-dernier sprint était d'aboutir à une application globalement fonctionnelle, afin de réserver au maximum pour le sprint suivant les options mineures et l'amélioration du style.

J'ai commencé par terminer la fonctionnalité d'ajout d'un nouveau spot. Cet ajout se fait dans un écran dédié, permettant à l'utilisateur de définir les informations du spot. Durant la création du spot, nous avions prévu que l'utilisateur prenne également des photos du spot, j'ai donc développé la possibilité de prendre des photos sur cet écran. Cependant, un retour de notre formateur lors de ce sprint a relevé que l'écran d'ajout de spot semblait trop chargé et peu intuitif. Nous avons donc décidé de retirer la prise de photo du composant chargé de l'ajout du spot, et que la prise de photo se ferait depuis un nouvel écran réservé à cette fonctionnalité, et accessible depuis l'écran du spot (celui qui affiche les photos et vidéos en carrousel). J'ai donc ensuite travaillé sur ce nouveau composant, qui m'a permis de développer la prise et l'ajout de photos de manière plus avancée que prévu. J'ai par exemple étudié comment afficher les photos sous la forme de miniatures que l'utilisateur peut toucher longuement afin d'en supprimer certaines avant de valider leur publication.

J'ai également rencontré un problème, dont la résolution m'a servi pour améliorer plusieurs fonctionnalités déjà codées ou à venir : Ayant chez moi un très mauvais débit internet, certains chargements qui semblaient instantanés chez Thomas et Baptiste pouvaient durer plusieurs secondes voire minutes chez moi, ce qui donnait l'impression que certaines fonctionnalités échouaient, puisque l'application ne donnait aucun feedback pour indiquer qu'une requête avait bien été prise en compte mais était en cours de traitement. SkaterQuest étant une application pensée pour être utilisée en extérieur, il était particulièrement important d'anticiper qu'elle serait parfois employée dans des endroits avec une faible couverture réseau. J'ai réglé le problème en utilisant le composant *ActivityIndicator* de React Native, qui à l'aide d'un état dédié m'a permis d'activer un indicateur visuel de chargement juste avant l'envoi d'une requête, puis de le masquer une fois les données chargées. Cette étude de la documentation de React Native m'a aussi permis de découvrir le composant *Alert*, qui permet également d'améliorer le feedback de l'application, cette fois en affichant des messages sous la forme de fenêtres modales paramétrables.

J'ai terminé ce sprint en m'occupant de la publication des vidéos. Pour des raisons dépendant de la manière dont nous prévoyons l'utilisation de SkaterQuest, les photos se prennent directement depuis l'application grâce aux composants d'Expo-camera tandis que les vidéos sont à charger depuis la galerie du téléphone. Baptiste ayant déjà mis en place les routes permettant de stocker les vidéos en base de données et d'y accéder, cette fonctionnalité fut rapide à mettre en place.

Il fallait aussi mettre en place le système de votes sur les vidéos. J'ai commencé à m'en occuper mais Baptiste a finalement pris le relais pour des raisons de temps et parce qu'il avait un meilleur aperçu de la manière dont il avait géré cette fonctionnalité dans le back-end (la mise à jour visuelle des votes concernait le problème d'affichage dynamique mentionné du tôt). Baptiste s'est aussi occupé durant ce sprint de l'écran des crews que nous avions laissé vide jusqu'alors. Il a ainsi développé, en complément des routes déjà mises en place, toutes les fonctionnalités liées aux équipes côté front-end : création et nommage d'un nouveau crew, recherche et ajout de nouveaux membres, ajout et retrait du grade d'administrateur, suppression de membres, et conditionnement de certaines de ces fonctionnalités selon la possession ou non du grade d'administrateur.

Thomas s'est chargé de compléter l'écran d'accueil, en y ajoutant notamment un tutoriel sous forme de trois fenêtres modales successives, s'affichant lors de l'inscription d'un nouvel utilisateur ou en touchant une icône d'aide en haut de cet écran. Il a ensuite créé les deux écrans d'options accessibles depuis l'écran d'accueil : un dédié aux paramètres de l'utilisateur (changement d'avatar, changement de pseudonyme) et un concernant ceux de l'application (déconnexion, désinscription, ...). Les différents boutons de ces écrans d'options n'étaient pas encore fonctionnels, et certains ne le sont toujours pas car ils correspondent à des fonctionnalités que nous avons jugées facultatives pour ce MVP, à l'instar d'un bouton permettant d'alterner entre un affichage en mode clair ou sombre. Nous avons cependant préféré laisser ces boutons dans l'application, afin de montrer l'aspect final de ces écrans, en attendant leur éventuel développement ultérieur, hors du cadre de notre formation.

Thomas et Baptiste ont ensuite pu travailler sur le style de l'application, que nous avions laissé au second plan lors des sprints précédents.

Sprint 4 :

En atteignant ce dernier sprint, nous avions globalement réussi à respecter les prévisions de nos user stories. La plupart des ajouts bonus que nous avions envisagés n'auraient pas leur place dans le contexte de ce MVP, mais nous étions satisfaits des fonctionnalités développées et des idées ajoutées en cours de route, sans avoir dévié de notre vision initiale.

L'objectif de ces deux derniers jours était la finalisation de certaines fonctionnalités, le peaufinage de l'affichage et du style, le test et le débugage échéant de toute le code, l'ajout de la sauvegarde des fichiers sur Cloudinary et le déploiement de l'application.

Baptiste et Thomas ont souvent travaillé ensemble lors de ces deux jours. Ils se sont occupé de finir d'appliquer le style de notre UI kit sur toute l'application. Ils se sont ensuite chargé de l'enregistrement des fichiers sur les serveurs de Cloudinary, et du déploiement du back-end sur Vercel.

Pendant ce temps, j'ai pris le temps de tester toute l'application afin de repérer tous les bugs restants, que nous avons ensuite corrigé. Il y en a eu plusieurs, mais souvent assez mineurs. En général, le développeur qui s'occupait d'un bug était celui qui avait développée la fonctionnalité concernée.

L'option de changement d'avatar n'avait pas encore été codée côté front, je m'en suis occupé. Baptiste avait déjà mis en place la route correspondante côté back et dans notre fichier passerelle *request.js* expliqué plus tôt.

J'ai aussi considérablement amélioré les carrousels affichant les photos et vidéos d'un spot, afin d'aboutir enfin à un résultat satisfaisant. Baptiste, qui utilisait une tablette plutôt qu'un smartphone pour tester l'application, a découvert un problème : l'affichage responsive fonctionnait bien sur téléphone mais s'est révélé inadapté pour l'écran d'une tablette. Baptiste et moi avons rapidement corrigé ça, bien que nous jugeons assez faible le taux d'utilisateurs qui pratiquent le skate équipés d'une tablette.

Nous avons manqué de temps à la fin et n'avons pas pu mettre le front-end en ligne sur Expo. Malgré tout, ce dernier sprint s'est bien déroulé, SkaterQuest est fonctionnel et sans bugs identifiés ! Nous avons dédié le lendemain du sprint à la présentation de l'application devant nos formateurs et nos camarades, à l'occasion du *demo day* de La Capsule, et avons obtenu de très bons retours.

III.2 Lien vers le code hébergé sur GitHub

Nous avons utilisé GitHub pour héberger le code de SkaterQuest et le partager entre nous trois durant le développement.

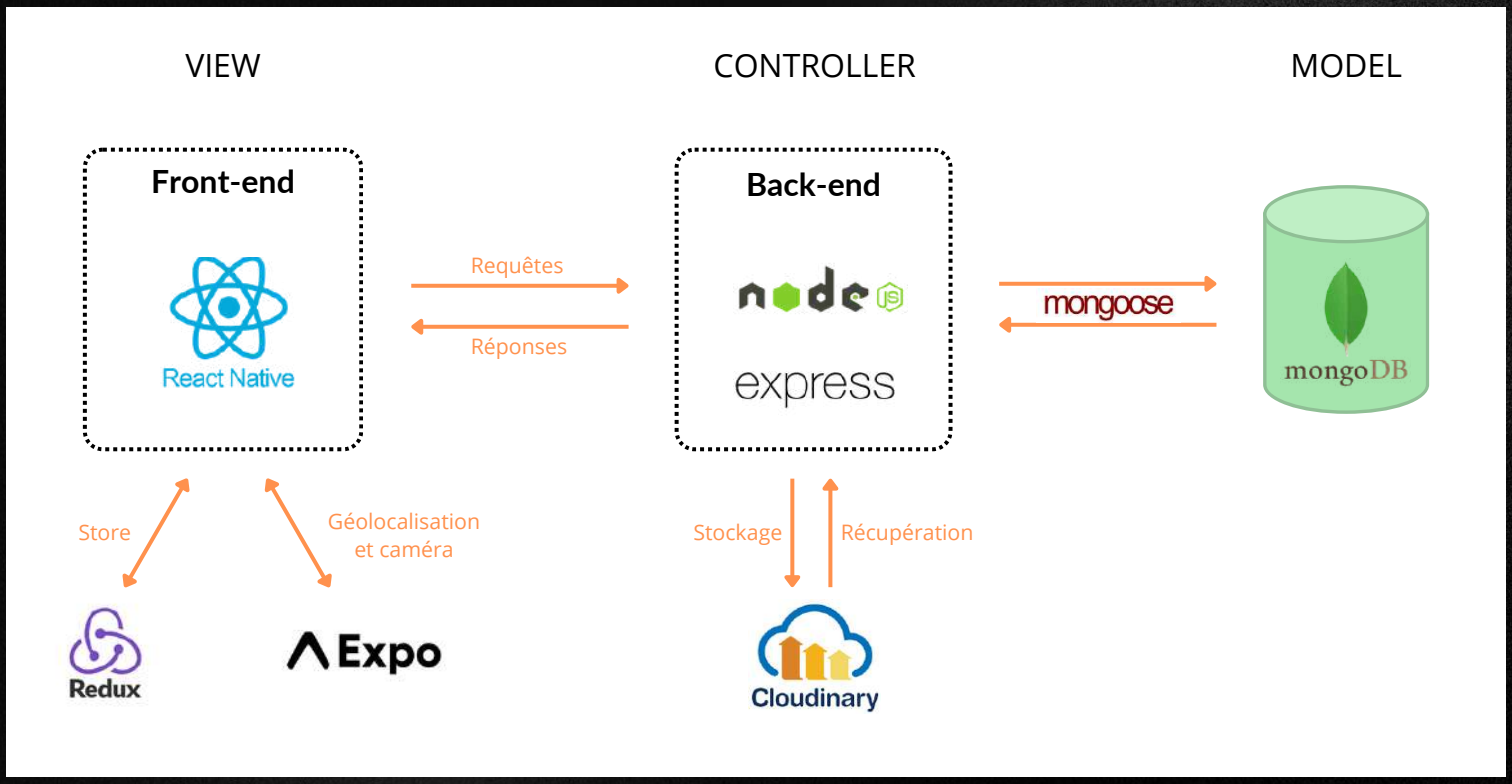
Nous avons préféré travailler majoritairement sur la branche principale en nous assurant une communication explicite et une répartition astucieuse des tâches afin d'éviter les conflits. Cette stratégie nous a permis de gagner du temps, d'autant plus que nous effectuions des commits fréquents.

Voici les liens vers les répertoires du back-end et du front-end :

- SkaterQuest-backend : https://github.com/ThomasPoillion/SkaterQuest-backend
- SkaterQuest-frontend : https://github.com/ThomasPoillion/SkaterQuest-frontend

III.3 Schéma de l'architecture de l'application

Nous avons structuré le projet selon une logique MVC (Model - View - Controller) :



III.4 Liste des composants utilisés

Écrans de l'application :

Nom	Description
LoginScreen.js	Écran d'arrivée sur l'application pour un utilisateur non connecté, permet de s'inscrire et de se connecter
HomeScreen.js	Écran d'accueil depuis lequel accéder au livre des tricks, aux options ou au tutoriel. Affiche la progression de l'utilisateur, son nom et son avatar
TrickScreen.js	Livre des tricks, permet de visualiser les tricks à l'aide de filtres
MapScreen.js	Affiche la carte et les spots enregistrés, permet d'en ajouter un
SpotScreen.js	Affiche les informations, photos et vidéos d'un spot, permet d'en publier de nouvelles
AddSpotScreen.js	Permet d'enregistrer un nouveau spot après avoir saisi ses informations (nom et catégorie)
AddPhotoScreen.js	Permet à l'utilisateur de prendre des photos d'un spot, de supprimer celles qu'il souhaite puis de les publier sur un spot
VideoScreen.js	Affiche les vidéos postées par l'utilisateur sur tous les spots
CrewScreen.js	Permet de créer et gérer son crew
GosPlayScreen.js	Permet de lancer une partie de *game of skate* et de modifier les pseudonymes des joueurs
GosVersusScreenChoices.js	Écran d'une partie de *game of skate* en mode sélection de tricks
GosVersusScreenRandom.js	Écran d'une partie de game of skate en mode tricks aléatoires
SettingScreen.js	Permet de modifier les paramètres de l'utilisateur (changement de pseudonyme et d'avatar)
AppSettingScreen.js	Permet de modifier les paramètres de l'application (déconnexion et désinscription)

Composants importés dans les écrans :

Nom	Description
ItemCarrousel.js	Galerie en affichage carrousel utilisée pour les photos et vidéos sur l'écran d'un spot
VideoPlayer.js	Lecteur de vidéo
ProgressBar.js	Jauge de progression des tricks, affichée dans le livre de tricks et sur l'écran d'accueil
BackgroundWrapper.js	Composant réutilisé dans chaque écran pour unifier l'application avec un fond d'écran personnalisé et une mise en page flexible
Buttons.js	Bouton orange présent sur la plupart des écrans
ModalContent.js	Fenêtre modale personnalisée avec une animation d'apparition/disparition, permettant d'y insérer du contenu JSX librement. S'affiche par-dessus tout le contenu de l'écran, avec un bouton pour fermer cette fenêtre
ConfirmModal.js	Fenêtre modale demandant une validation à l'utilisateur
ErrorModal.js	Fenêtre modale indiquant une erreur

III.5 Liste des dépendances utilisées

Back-end :

Catégorie	Nom	Version	Description
Sécurité et authentification	cookie-parser	1.4.4	Lecture des cookies envoyés dans les requêtes HTTP
	cors	2.8.5	Permet la communication entre le back et le front
	bcrypt	5.1.1	Hachage des mots de passe
	jsonwebtoken	9.0.2	Génération de tokens JWT et méthodes pour agir dessus
Services externes et uploads	mongoose	8.13.2	Interface entre le back-end et la base de données (MongoDB)
	cloudinary	2.6.0	Service cloud pour héberger et gérer des fichiers
	express-fileupload	1.5.1	Middleware pour gérer les fichiers uploadés via des requêtes HTTP
	form-data	4.0.2	Construction de formulaires (notamment pour les uploads)
Outils Express et de back-End	express	4.16.1	Framework pour Node.js
	dotenv	16.5.0	Charge les variables d'environnement depuis le fichier .env
	debug	2.6.9	Affichage de messages de debugage dans la console
	morgan	1.9.1	Affichage des détails des requêtes dans la console
	fs	0.0.1	Module de Node.js pour interagir avec le système de fichiers

Catégorie	Nom	Version	Description
Tests	jest	29.7.0	Framework de tests (TDD)
	babel-jest	29.7.0	Intègre Babel à Jest (permet à Jest de comprendre le code ES6+/JSX)
	@testing-library/ react-native	13.2.0	Bibliothèque de tests simulant le comportement d'un utilisateur
	@testing-library/ jest-native	5.4.3	Ajoute des matchers (toBeVisible, toHaveTextContent, ...) pour les tests avec Jest
	react-test-renderer	19.1.0	Rend les composants React pour les tests sans DOM
	redux-mock-store	1.5.5	Simule un store Redux pour tester les actions et reducers
Utilitaires et développement	node-fetch	2	Permet de fetch côté back-end
	uid2	1.0.0	Génère des identifiants uniques sécurisé (pour les tokens)
	uniqid	5.4.0	Génère des identifiants uniques simples
	nodemon	3.1.9	Redémarre automatiquement le serveur Node.js en cas de modification des fichiers (gain de temps)

Front-end :

Catégorie	Nom	Version	Description
Base	react	18.3.1	Bibliothèque principale
	react-native	0.76.9	Framework de création d'applications mobile avec React
	expo	52.0.43	Complément à React Native pour faciliter le développement d'applications mobile
UI & Composants visuels	@expo/vector-icons	14.0.2	Icônes intégrées dans Expo (FontAwesome, MaterialIcons, ...)
	react-native-vector-icons	10.2.0	Icônes vectorielles pour React Native
	react-native-paper	5.13.1	Composants UI stylés basés sur Material Design
	react-native-linear-gradient	2.8.3	Affichage de dégradés en fond
	expo-linear-gradient	14.0.2	Idem (version Expo)
	react-native-animatable	1.4.0	Animations pour les composants
	react-native-confetti-cannon	1.5.2	Animation de confettis (victoire game of skate)
	expo-status-bar	17.0.6	Contrôle de la barre de statut sur iOS et Android
Carte	react-native-maps	1.18.0	Affichage d'une carte
	expo-location	18.0.10	Accès à la géolocalisation de l'appareil

Catégorie	Nom	Version	Description
Médias	expo-media-library	17.0.6	Accès aux fichiers média de l'utilisateur
	expo-camera	16.0.18	Accès à la caméra de l'appareil
	expo-av	15.0.2	Lecture audio et vidéo
	expo-video	2.0.6	Lecture de vidéos dans Expo
	expo-video-thumbnails	9.0.3	Génère les miniatures d'une vidéo
	expo-image-picker	16.0.6	Ouvre la galerie pour importer une image
Navigation	@react-navigation/native	6	Cœur de la navigation React Navigation
	@react-navigation/bottom-tabs	6	Navigation par onglets en bas de l'écran
	@react-navigation/native-stack	6	Navigation par menus
	react-native-screens	4.4.0	Optimisation des transitions écran sur mobile
	react-native-safe-area-context	4.12.0	Gestion des zones non interactives (notch, status bar...)
Stockage et états	react-redux	9.2.0	Connecte Redux à React
	redux-persist	6.0.0	Rend le store Redux persistant
	@react-native-async-storage/async-storage	2.1.2	Stockage local clé-valeur (persistant)

Catégorie	Nom	Version	Description
Stockage et états	@react-native-community/hooks	100.1.0	Petits hooks utilitaires prêts à l'emploi
	@reduxjs/toolkit	2.6.1	Simplifie la configuration et l'utilisation de Redux
	@react-native-picker/picker	2.11.0	Composant natif pour créer des menus déroulants
Tests	jest	29.7.0	Framework de tests (TDD)
	jest-expo	52.0.6	Adapte Jest à l'environnement Expo
	@testing-library/react-native	13.2.0	Bibliothèque de tests simulant le comportement d'un utilisateur
	@testing-library/jest-native	5.4.3	Ajoute des matchers (toBeVisible, toHaveTextContent, ...) pour les tests avec Jest
	react-test-renderer	19.1.0	Rend les composants React pour les tests sans DOM
Babel	@babel/core	7.26.10	Moteur de compilation JavaScript (base de Babel)
	babel-preset-expo	12.0.11	Preset Babel spécifique à Expo
	@babel/preset-env	7.26.9	Préréglage pour cibler les environnements JavaScript récents
	@babel/preset-flow	7.25.9	Support pour le typage statique avec Flow
	@babel/preset-react	7.26.3	Support JSX pour Babel (React)
	babel-jest	29.7.0	Intègre Babel à Jest (permet à Jest de comprendre le code ES6+/JSX)

PARTIE IV · MISE EN PRODUCTION

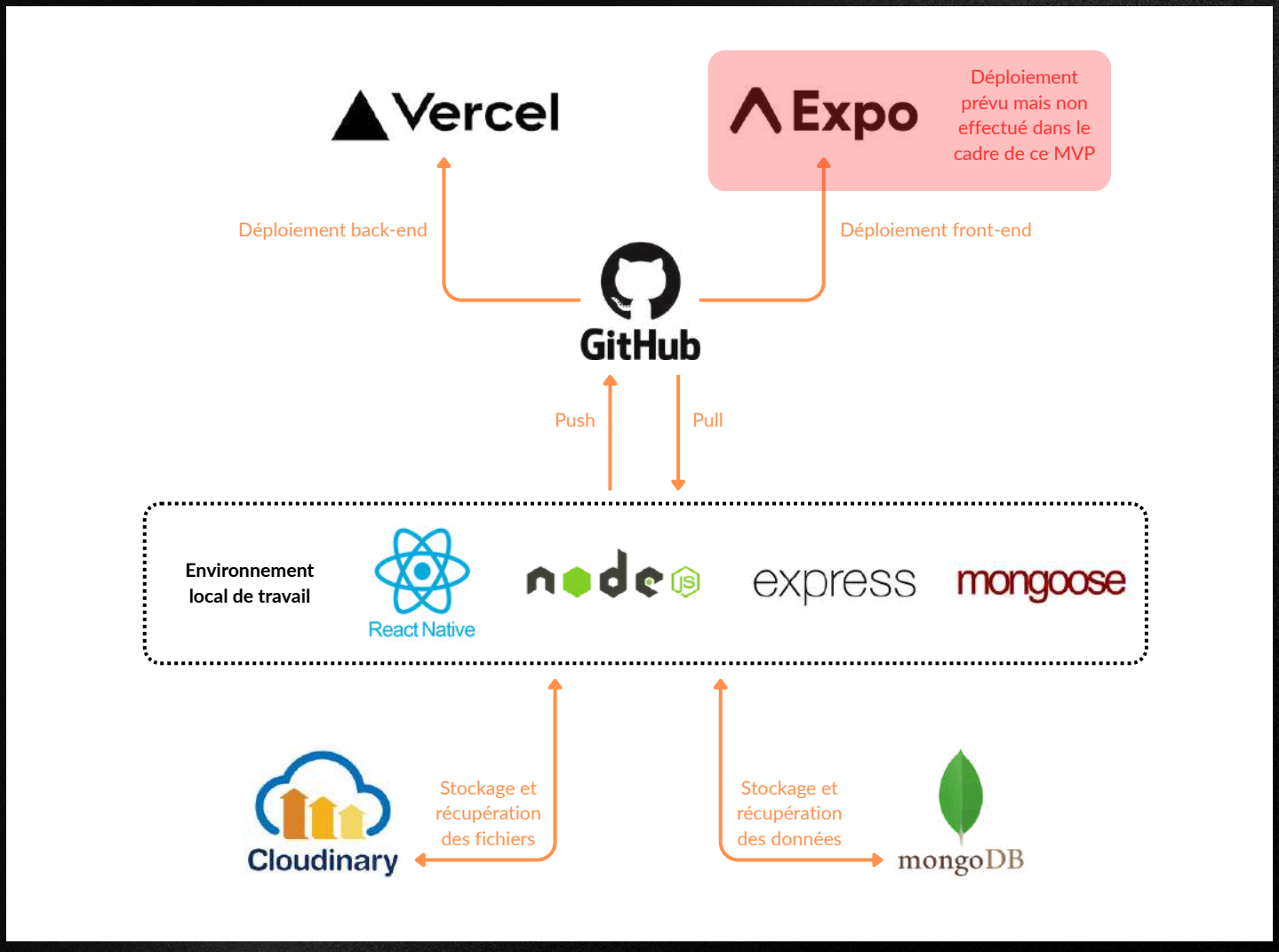
IV.1 Déploiement de l'application

Nous avons utilisé Vercel pour mettre le back-end de SkaterQuest en ligne.

URL : https://skater-quest-backend.vercel.app/

Nous avons prévu de mettre en ligne le front-end avec Expo, mais avons manqué de temps lors du dernier sprint, ce déploiement n'a donc pas été effectué pour ce MVP.

IV.2 Schéma de l'environnement de déploiement



IV.3 Mise en place d'un environnement de TDD

Le TDD (Test Driven Development) est une méthode de développement consistant à tester le code de manière automatisée, afin de rapidement repérer quand une fonctionnalité est opérationnelle ou si une erreur est survenue.

Nous avons utilisé ce système et le framework Jest afin de tester les requêtes du front-end vers le back-end. Cela était particulièrement pratique en début de développement, car ça nous a permis de tester la communication entre le front et le back et le bon fonctionnement des routes à un stade où nous n'avions pas encore codé la plupart des fonctionnalités du front-end. Nous avons ainsi utilisé un total de 11 tests, réunis dans un même fichier *request.test.js* évaluant certaines des fonctions du fichier *request.js*. Des variables temporaires déclarées au début de ce fichier permettaient de tester les requêtes sans avoir encore accès à toutes les données (comme les informations saisies de l'utilisateur dans des fonctions pas encore créées).

Plus le développement progressait et plus nous étions en mesure de tester le code de manière directe, donc nous avons de moins en moins utilisé le TDD, notre temps étant très limité.

Exemple de test contrôlant la fonction *signInRequest()*, dont le rôle est d'envoyer une requête au back-end pour appeler la route gérant la connexion d'un utilisateur :

```
const dummyEmail = "text0@test.test";
const dummyPassword = "test";

it("Sign In", async () => {
  const { result, data } = await signInRequest(dummyEmail, dummyPassword);
  expect(result).toBe(true);
  expect(data.email).toBe(dummyEmail);
});
```

IV.4 Système d'authentification

Cette partie décrit les mécaniques d'authentification que nous avons mis en place pour sécuriser les données des utilisateurs de l'application.

Système de token :

Nous avons utilisé la mécanique de token afin de sécuriser le front-end, démarche importante compte tenu de la faible opacité des requêtes y transitant. Le token est un nombre long et généré de manière aléatoire, que l'on peut stocker en base de données avec les autres informations d'un utilisateur, pour ensuite pouvoir s'en servir pour l'identifier durant sa connexion.

Concernant la génération du token, elle est effectuée dans notre back-end par la fonction *generateToken()* du middleware contenu dans le fichier *tokenAuth.js*. Cette fonction utilise le module *uid2* afin de créer des chaînes de 32 caractères aléatoires, et la bibliothèque *jsonwebtoken* afin de créer des tokens JWT, plus sécurisés et complexes, générés à l'aide de la méthode *sign()* de cette bibliothèque. Cette génération de token est ensuite utilisée dans les routes suivantes :

- *user/signup* : lors de l'inscription, génération du token
- *user/signin* : lors de la connexion, renvoi du token enregistré
- *user/extend* : génération d'un nouveau token

Au niveau du contrôle du token, elle est gérée par la fonction *tokenVerifierMW()* de ce même middleware, qui utilise cette fois la méthode *verify()* de la bibliothèque *jsonwebtoken* afin de vérifier la validité du token JWT (signature correcte, token non expiré, etc.). Cette fonction est ensuite appelée dans toutes les routes dans lesquelles il était préférable de contrôler l'identification de l'utilisateur.

Cryptage du mot de passe :

Au niveau de la protection des mots de passe des utilisateurs, nous avons utilisé le module *bcrypt* pour les chiffrer de manière irréversible et complexe grâce à la mécanique de hachage et de salage de cet outil.

Nous avons ainsi utilisé les méthodes *hashSync()* et *compareSync()* de *bcrypt*, respectivement pour chiffrer le mot de passe (lors de l'inscription - route *user/signup*) et pour comparer celui saisi avec sa version cryptée (lors de la connexion - route *user/signin*).

Conclusion

Concevoir et développer une application fonctionnelle avec un temps et un effectif si réduit fut un excellent moyen pour me permettre de consolider mes acquis de formation et de gagner en autonomie. Ce projet m'a permis d'approfondir mes connaissances en développement, mais aussi de me confronter à la résolution de problématiques diverses, à la gestion du temps et au travail collaboratif. Les différentes étapes décrites dans ce dossier témoignent de l'expérimentation à petite échelle de l'ensemble d'un pipeline de production, de la préparation initiale du projet à la finalisation d'un MVP.

Le travail en équipe fut le centre névralgique de ce projet. Assurer une communication et une entraide efficace, partager nos connaissances, anticiper méthodologiquement les potentiels problèmes liés à un environnement de développement partagé (conflits, redondances, incohérences inter-fichiers, ...), veiller à maintenir un code explicite et rigoureux (clean code, commentaires, consistance des choix syntaxiques, ...) sont tout autant d'enjeux que nous avons expérimentés durant ces deux semaines, et qui me semblent être fondamentaux dans un contexte professionnel.

Je suis reconnaissant envers Thomas et Baptiste, envers mes formateurs de la Capsule et envers tous les camarades nous ayant fait de précieux retours durant cette aventure.

Ce projet m'a ainsi permis de beaucoup évoluer au niveau technique, méthodique et collaboratif (contrairement à ma progression de tricks encore à 0% dans SkaterQuest), et a renforcé mon envie de m'investir pleinement dans le développement web et mobile.