Table des matières

Liste des figures

Liste des tableaux

Liste des abréviations

API :	Application Programming Interface / Interface de programmation applicative
BDD :	Base de données
CRUD :	Create (Créer), Read (Lire), Update (Mettre à jour), Delete (Effacer)
DOM :	Document Object Model /
EN :	Échelle Numérique
EVA :	Échelle Visuelle Analogique
EVS :	Échelle Verbale Simple
HIPAA :	Health Insurance Portability and Accountability Act,
IDE :	integrated development environment / Environnement de développement
JSON :	JavaScript Object Notation
NoSQL :	Not only SQL
PB :	Product Backlog
PO ;	Product Owner
REST :	Representational state transfer
RGPD :	Règlement général sur la protection des données
SaaS :	Software as a service
SP :	Story point
TB :	Travail de Bachelor
US :	User Story
VCS :	Version Control System / Logiciel de gestion des versions

Introduction

A chaque rendez-vous, la même question du physiothérapeute : « Comment allez-vous depuis la dernière fois ? »

En tant que patient, il faut plonger dans sa mémoire afin d'y trouver quelques informations, qui semble-t-il, pourraient être utiles au physiothérapeute dans son suivi médical.

L'idée principale de ce projet est d'offrir une interface simple qui permette au patient de saisir toutes les variations de douleurs sur la période de traitement.

Nous commencerons par définir les « use cases[1] » avec notre client Roger Hilfiker (professeur à l'HES-So Valais/Wallis Secteur santé). En découlera la modélisation des mockups. Pour le design de la partie mobile nous nous appuierons sur les guides des bonnes pratiques de Google.

Deuxièmement nous analyserons les différentes possibilités en termes de service cloud pour la base de données. Initialement l'utilisation de Firebase est prévue, mais cependant un problème technique nous contraindra à basculer vers Pryv.

Ce projet est composé de trois applications et nous expliquerons également comment nous avons structuré nos environnements de développement.

Nous enchaînerons avec une étape de faisabilité. Ce projet a pour but de valider un prototype fonctionnel. Il est donc nécessaire de faire quelques essais. Nous rencontrerons d'ailleurs un problème technique avec la smartwatch autonome qui sera la cause de l'abandon de Firebase. Dans cette partie nous commettons une erreur dans la méthodologie SCRUM qui nous ralentira dans la réalisation de notre travail et fera chuter notre vélocité.

[1] Use case : les cas d'utilisations permettent de décrire toutes les utilisations d'un système que l'utilisateur veut (Larson & Larson, 2004).

Ensuite, nous détaillerons Pryv qui est un service développé en Suisse et dont la protection des données est l'une des priorités. Nous vérifierons la faisabilité de ce nouveau paradigme qui s'avèrera concluante, pour ensuite finaliser le développement.

Avant de conclure, nous présenterons plus en détails l'architecture de notre application Web. Puis nous finirons ce rapport avec une présentation des améliorations possibles ainsi qu'une conclusion.

1 Présentation générale du besoin

1.1. Problématique

Le suivi d'un patient en physiothérapie est ponctué par des rendez-vous. Durant toute la phase de traitement l'état de santé du patient varie. Pour la majorité des patients il est difficile de quantifier ces variations et d'en définir les causes. De surcroît plus le délai entre les rendez-vous se prolonge, plus l'information est approximative. Dans ces conditions le physiothérapeute a une vision limitée de l'impact du traitement sur l'état général du patient.

1.2. L'idée principale côté patient

Offrir aux patients la possibilité de saisir en tout temps des informations sur leur état de santé. Afin de mesurer factuellement ces données, l'indication de l'intensité de douleur est le point d'entrée privilégié.

De plus, le patient doit pouvoir compléter une saisie, avec des indications personnelles et/ou selon une liste d'activités prédéfinies. Cela répond à deux besoins :

- Enrichir la donnée pour une analyse plus fine par le physiothérapeute
- Mémoriser des indications plus subjectives que le patient pourrait oublier lors du prochain rendez-vous.

Pour répondre à ces premières exigences, il est prévu d'utiliser deux « outils » :

- Une application smartwatch : pour la saisie du niveau de douleur et de l'activité
- Une application smartphone : pour donner plus de détails, compléments subjectifs

1.2.1. D'autres concepts

Dans la première phase d'analyse plusieurs autres idées ont été abordées :

- Enregistrement du complément par la voix
- Enregistrement des coordonnées GPS
- Information sur l'état émotionnel du patient

1.3. L'idée principale côté physiothérapeute

Une plateforme web qui permet au physiothérapeute d'analyser l'état général du patient. Cette application Web doit fournir les fonctionnalités suivantes :

- Afficher une liste de patients
- Afficher un graphique par patient des niveaux de douleur saisis durant une période donnée
- Permettre la gestion d'une liste d'activités prédéfinies

1.4. Technologies

L'essentiel du développement est orienté « front-end ». Il n'est pas envisagé de créer un serveur « back-end » mais plutôt d'exploiter des services disponibles en ligne. C'est également le cas pour la base de données.

1.4.1. Front-end, Mobile et Web

Pour répondre au besoin, trois applications « front-end » doivent être développées. La partie mobile Android est la solution privilégiée. D'une part parce que le kit de développement fourni par Google est gratuit (Google, 2016), d'autre part parce qu'une montre Huawei Watch 2, fonctionnant sur Android est mise à disposition par l'institut d'informatique de gestion pour la durée du TB.

L'application Web n'est pas soumise à condition et nous pouvons choisir librement la technologie.

1.4.2. Stockage

Selon le descriptif fourni en début de TB, deux pistes à analyser sont proposées.

- Firebase (Base de données en temps réel de Google)
- FHIR, une solution orientée pour le secteur médical

2 Déroulement du projet

2.1. Les étapes

Pour effectuer ce travail, trois phases sont identifiées.

1. La première étape a pour objectif de définir le besoin du client. Etablir les « use cases » et modéliser les mockups[2] des futures applications.
2. La 2e étape consiste à analyser les technologies adaptées pour la réalisation de ce travail. En particulier pour la base de données.
3. Ensuite, selon la méthodologie SCRUM[3], développer les trois applications.

Parallèlement, la gestion de projet et la rédaction du rapport doivent être assurées.

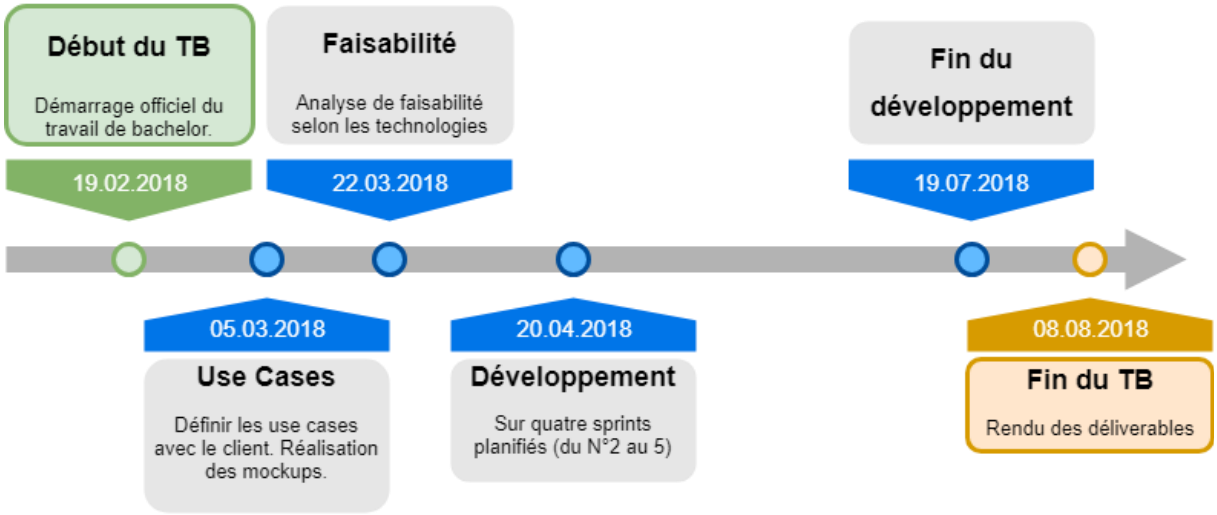


Figure 1 : Jalons principaux du TB

[2] Mockup : modéliser une maquette des différents écrans

[3] SCRUM : est un cadre de travail permettant de résoudre des problèmes complexes et changeants en apportant un maximum de valeur au client final (Arnaud, 2018).

3 Use cases & Mockups

Les use cases sont réalisés avec l'aide de Roger Hilfiker, professeur en physiothérapie à l'HES-So Valais/Wallis qui joue le rôle de client.

3.1. Définir un cadre

Avant de décrire tous les cas d'utilisations, il convient de définir certaines données de base. Tout d'abord l'ergonomie des smartwatches diffère des autres supports tels que smartphone ou ordinateur, en raison d'une taille d'écran restreinte.

Figure 2 : Taille des écrans des smartwatches (Shanklin, 2017)

Deuxièmement il convient de donner quelques précisions sur les échelles de mesure du niveau de douleur.

3.1.1. Ergonomie de Wear OS

Le système d'exploitation des smartwatches nommé « Android wear » depuis son lancement en Juin 2014 est renommé « Wear OS » en Mars 2018 (Maring, 2018). Dans le but

de présenter la meilleure expérience utilisateur possible, nous dessinons les mockups selon les bonnes pratiques conseillées par Google. Un site internet est d'ailleurs consacré à ce sujet. Ci-dessous une brève description des 3 règles élémentaires.

- Règle N°1 : Interface simple et facile à lire

Faire

Aider les utilisateurs à identifier le message de chaque écran.

Ne pas faire

Des interfaces trop chargées

Figure 3 : Vision créative, interface simple (Google, 2018)

- Règle N°2 : Facile à exploiter

Faire

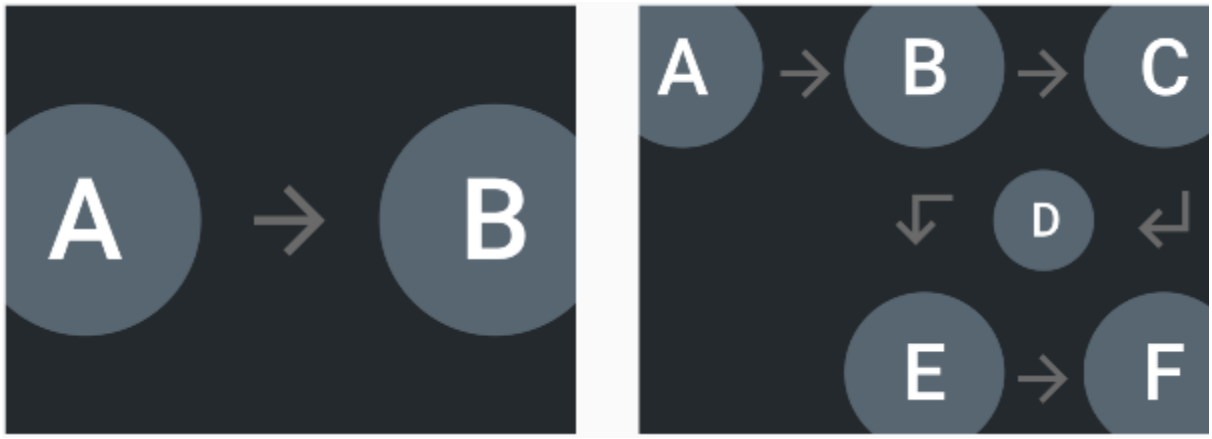
Utiliser de grands boutons

Ne pas faire

Des petits boutons augmentent la probabilité de cliquer sur le mauvais.

Figure 4 : Vision créative. Facile à exploiter (Google, 2018)

- Règle N°3 : Gain de temps



Faire

Réduire le nombre d'étapes pour accomplir une tâche, anticiper les actions utilisateur

Ne pas faire

Des interfaces complexes difficiles à lire et à manipuler

Figure 5 : Vision créative, gain de temps (Google, 2018)

3.1.2. L'échelle de douleur

Durant la séance la question sur la méthode d'évaluation du niveau de douleur est traitée. Il s'agit d'utiliser la méthode la plus précise et la plus adaptée à la smartwatch.

Il existe plusieurs méthodes pour évaluer la douleur. Comme la smartwatch est adaptée à des personnes sachant lire, nous excluons les échelles d'évaluation des nouveau-nés. Pour les adultes, trois échelles sont couramment utilisées (Elmer, 2015) :

- L'Échelle Numérique (EN) : noter la douleur de zéro à dix ou de zéro à 100. Zéro étant « aucune douleur » et dix ou 100 étant la « douleur maximale imaginable ».
- L'Échelle Visuelle Analogique (EVA) : Une réglette à deux facettes. Du côté patient une simple ligne dont une extrémité est notée « aucune douleur » et l'autre extrémité « douleur maximale imaginable », le patient déplace un curseur le long de la ligne. Puis sur l'autre face (côté soignant), on trouve une échelle numérique de zéro à dix ou de zéro à 100 (cf. Figure 6 : Échelle Visuelle Analogique recto-verso).

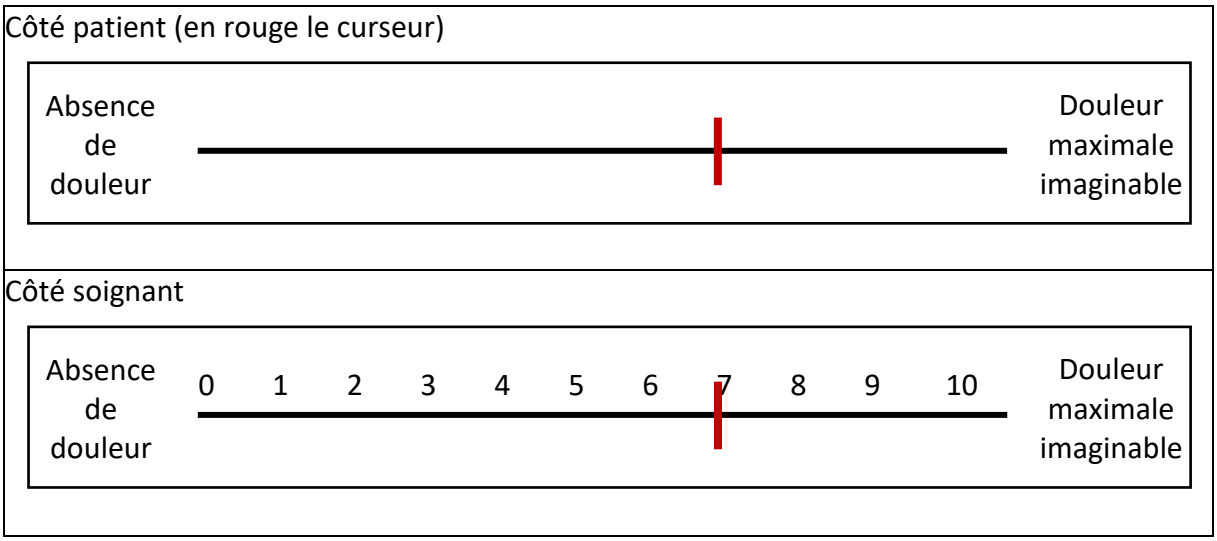


Figure 6 : Échelle Visuelle Analogique recto-verso

- L'Échelle Verbale Simple (EVS) : Le patient répond à trois questions et pour chacune d'elles le niveau de douleur est noté de 0 à 4.
    - Douleur au moment présent ?
    - Douleur habituelle sur les huit derniers jours ?
    - Douleur la plus intense sur les huit derniers jours ?

3.1.3. Choix de l'échelle de douleur

L'échelle verbale simple, n'est pas retenue, car afficher trois questions sur un écran ne répond pas à l'exigence de la règle N°1 qui spécifie « créer des interfaces simples ». De plus, afficher les trois questions, sur trois écrans différents augmente le temps nécessaire à accomplir une tâche (Règle N°3 cf. chapitre : 3.1.1).

L'échelle visuelle analogique pour être exploitée correctement doit avoir une dimension fixe et visible dans son intégralité. Nous étudions deux solutions mais aucune d'elles n'est satisfaisante. (cf. Figure 7 : Échelle Visuelle Analogique, pas exploitable sur le petit écran d'une smartwatch).

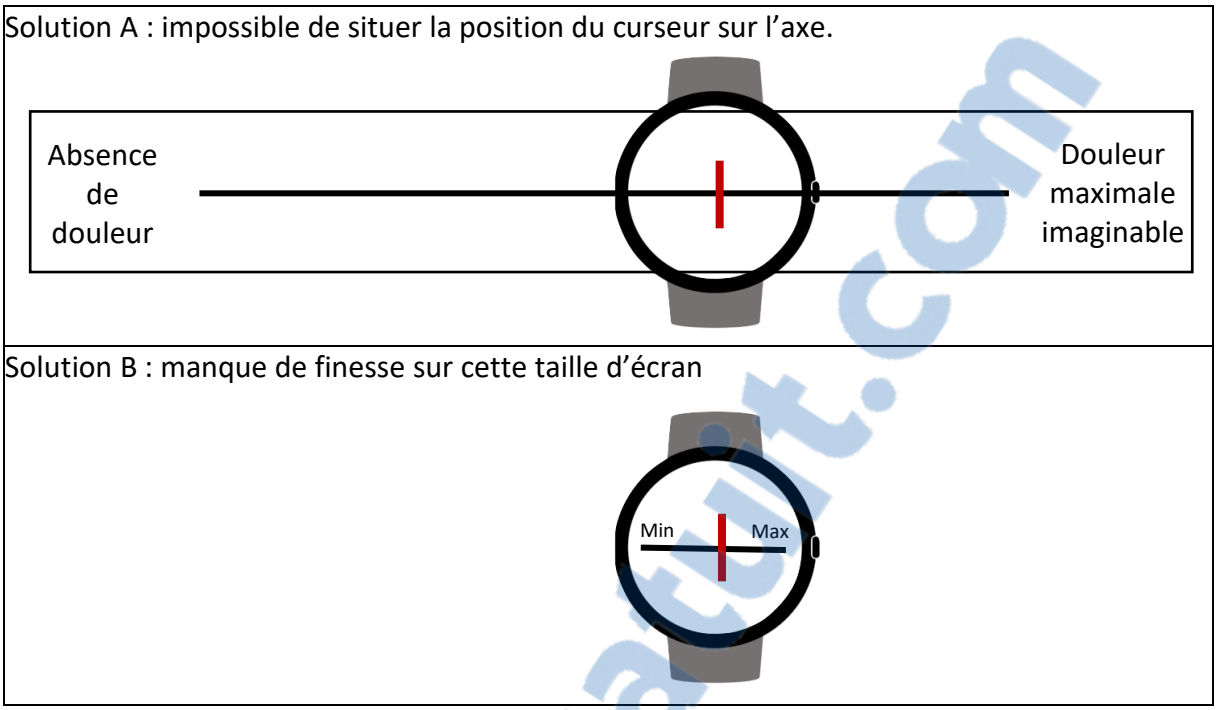


Figure 7 : Échelle Visuelle Analogique, pas exploitable sur le petit écran d'une smartwatch

L'échelle numérique, permet un affichage par liste, où chaque élément a une valeur définie. Dans le but d'obtenir une liste courte, nous arrêtons notre choix sur une échelle de zéro à dix.

3.2. Use cases : patient

L'objectif du TB étant de fournir un prototype fonctionnel, certains concepts évoqués en début de TB ont été exclus du périmètre. A savoir :

- Enregistrement audio
- Coordonnées GPS
- Etat émotionnel du patient (émoticône)

3.2.1. Smartwatch

En premier lieu le patient doit se connecter au service avec un compte utilisateur. Le choix du service et de la base de données n'étant pas établis à cet instant, la méthode d'authentification n'est pas traitée. Cependant le souhait de préserver l'anonymat des patients est demandé.

Avec une smartwatch, un patient saisit son niveau de douleur à chaque fois que celui-ci change. L'objectif étant d'obtenir une analyse graphique

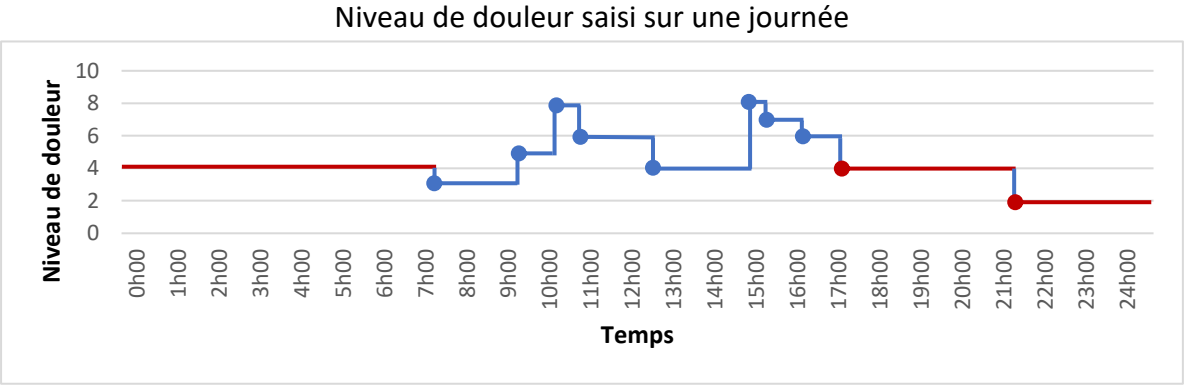


Figure 8 : Exemple de graphique, chaque point représente une saisie faite par le patient.

La question des longs intervalles entre les mesures est soulevée. Comment interpréter les lignes en rouge dans le graphique d'exemple ? Dans un premier temps nous décidons de résoudre la partie nuit en spécifiant une saisie au réveil et une au coucher.

Nous imaginons dans un premier temps quatre choix possibles pour sauver un nouveau niveau de douleur :

- Sauver comme première valeur de la journée (au réveil)
- Sauver comme dernière valeur de la journée (au coucher)
- Sauver sans autre information
- Associer au niveau de douleur choisi une activité prédéfinie et sauver.

Cependant, après une nouvelle étude du cas des longs intervalles, nous pensons qu'il est plus judicieux de traiter ça dans l'algorithme d'analyse. De surcroît le mode nuit n'est pas idéal dans le cas où le patient voudrait saisir des données alors qu'il a déjà préalablement activé le mode nuit (coucher). De ce fait, nous décidons de remplacer les deux premiers modes de sauvegarde par un seul :

- Sauver, ajouter un marqueur « flag » afin d'y ajouter un complément plus tard dans la journée et retour à la page d'accueil.

Ainsi trois modes de sauvegarde sont retenus.

3.2.2. Smartphone

Comme pour la smartwatch, le patient doit avant tout se connecter à son compte patient. L'historique de toutes les données saisies par la smartwatch est affiché. Cela peut être dans un graphique ou une liste voire les deux. Le patient peut modifier un niveau de douleur préalablement saisi et ajouter un complément d'information dans une zone de texte libre.

3.3. Use cases : physiothérapeute

Le physiothérapeute doit administrer les comptes patients : créer, modifier, effacer et les afficher dans une liste. Cette dernière contient une colonne « identifiant ». En effet pour des questions d'anonymisation on ne veut pas stocker les nom et prénom des patients.

Puis, quand on clique sur l'identifiant d'un patient depuis la liste on affiche toutes les données saisies par le patient, sous forme graphique. En termes d'analyse il convient de présenter les données selon différentes règles :

- Toutes les données : le graphique contient l'intégralité des données.
- Analyse par jour : on calcule la moyenne journalière des niveaux de douleur.

- Analyse par semaine : on calcule une moyenne hebdomadaire
- Analyse par mois : on calcule une moyenne mensuelle.

De plus le physiothérapeute gère une liste d'activités. Celles-ci sont regroupées par catégories. En plus de la création, il faut prévoir la modification et la suppression des catégories et des activités. Finalement, paramétrer des activités favorites patient par patient selon leurs besoins est une fonctionnalité désirée.

L'exportation en format CSV ou Excel est à prévoir.

3.4. Le « Product backlog »

Il est disponible à l'Annexe I. Tous les « use cases » décrit dans le chapitre précédent, ont été formatés en User stories et priorisés.

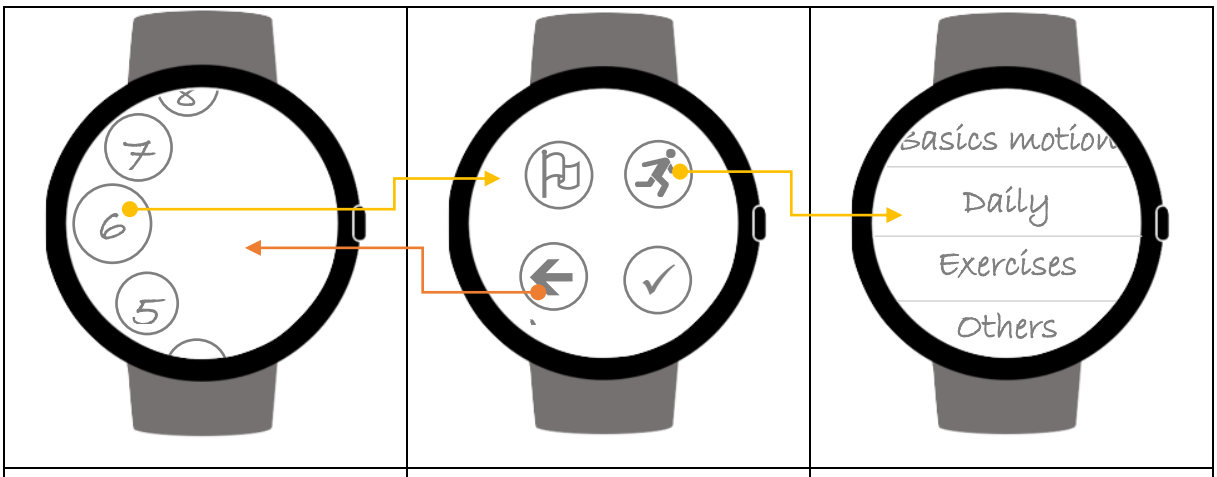
3.5. Mockups

Pour donner suite aux use cases précédemment décrits, nous modélisons les mockups des trois applications. L'objectif étant de les valider avant le début du développement, Jalon du 22.03.2018 (cf. Figure 1 : Jalons principaux du TB).

3.5.1. Mockups smartwatch

L'intégralité des mockups de la smartwatch est disponible à l'Annexe II. Voici néanmoins ci-dessous les écrans principaux (première version).

A noter que les flèches donnent des indications sur la navigation. Pour les boutons « Flag » et « Sauve », un message de confirmation informe le patient du résultat de la transaction. Si tout est correctement sauvegardé dans la base de données ou si une erreur est apparue.



Ecran principal : choix du niveau de douleur dans une liste	2e écran : mode de sauvegarde (Flag ; activité ; retour ; sauve)	3e écran : liste des catégories d'activités

Figure 9 : Mockups smartwatch : écrans principaux

3.5.2. Mockups smartphone

Tous les mockups relatifs au smartphone sont consultables à l'Annexe III. Pour cette application deux écrans principaux sont toutefois détaillés ci-dessous.

Ecran principal : historique des saisies dans une liste triée dans l'ordre du plus récent au plus ancien	2e écran : modifier le niveau de douleur et ajouter un complément sous forme de texte.

Figure 10 : Mockups smartphone : écrans principaux

3.5.3. Mockups de l'application web

Comme pour les deux applications précédentes les mockups complets sont disponible à l'Annexe IV.

4 Analyse des systèmes de base de données

Selon le descriptif du travail de TB. Il est prévu d'utiliser Firebase. Cette plateforme de Google offre plusieurs services :

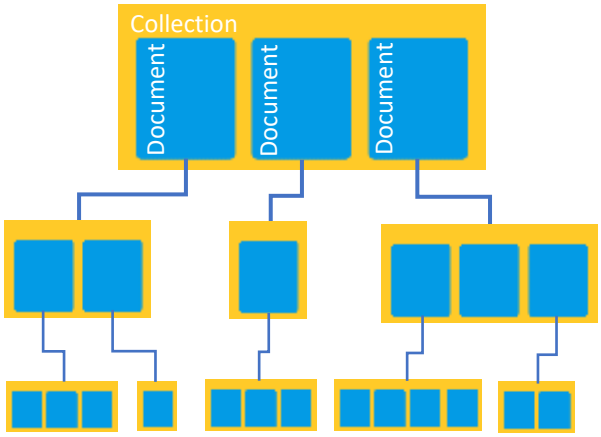
- Base de données
- Système d'authentification des utilisateurs
- Gestion des notifications
- Prise en charge des tests unitaires et des tests d'instrumentations
- Etc.

Effectivement dans l'idée de rendre un prototype fonctionnel, il est vrai que Firebase est particulièrement bien adapté. Néanmoins la question de la sécurisation des données personnelles reste un point sensible. En particulier, dans cette période où le RGPD entre en vigueur dans l'Union Européenne le 25 mai 2018 (Seydtaghia, 2018). En second lieu, nous ne maîtrisons pas l'emplacement géographique des serveurs de Firebase. Cela a plusieurs conséquences mais nous relevons en particulier celle-ci :

Perte de contrôle sur les données. *L'interconnexion mondiale et l'utilisation de la mémoire virtuelle font qu'il est souvent impossible de localiser les données, notamment lorsqu'on a recours à des nuages publics. Le maître des données ne sait donc pas où les données qu'il a déposées dans le nuage sont enregistrées et traitées alors qu'il répond de leur utilisation. Il ignore souvent si des sous-traitants interviennent et s'ils veillent à une protection adéquate des données. L'utilisateur d'un nuage ne peut donc pas assumer ses obligations en matière de protection des données (garantir la sécurité des données, accorder un droit d'accès, corriger ou effacer des données) ou uniquement en partie.* (Préposé fédéral à la protection des données et à la transparence (PFPDT), s.d.)

4.1. Realtime Database ou Cloud Firestore

Firebase propose deux solutions de base de données NoSQL. La version originale « Realtime Database », âgée de quatre ans et depuis octobre 2017, « Cloud Firestore ». Évidemment chacun de ces systèmes a des avantages et inconvénients. Cependant, la manière de stocker les données est très différente. Pour « Realtime Database » c'est un arbre JSON (cf. Figure 11). Pour « Cloud Firestore » le stockage est structuré en collections contenant des documents qui eux-mêmes contiennent d'autres sous-collections (cf. Figure 12).



Sources images : (Kerpelman, 2017)

Figure 11 : Ex. arbre JSON pour Realtime Database

Figure 12 : Ex. collections/documents pour Cloud Firestore

Toutefois quelle que soit la solution choisie nous structurons nos données par patient. Le document Patient contiendra l'intégralité des données le concernant.

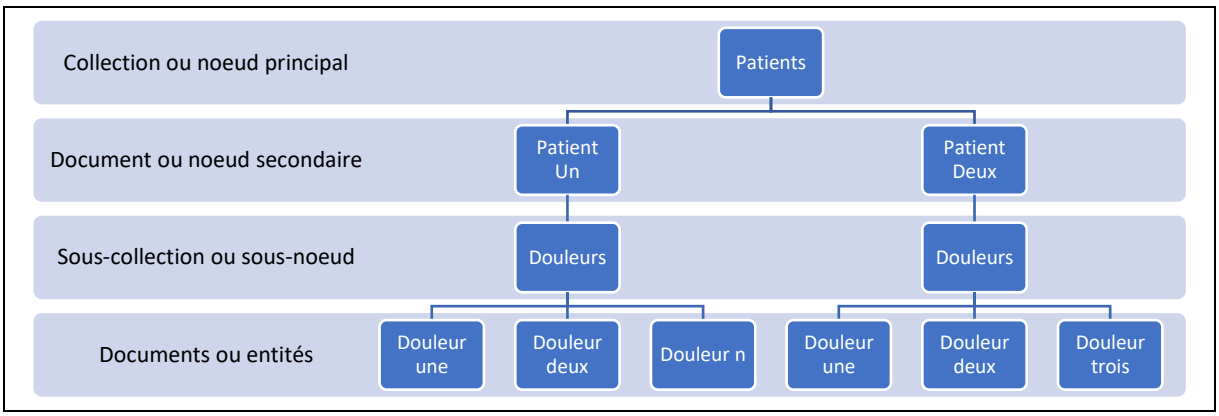


Figure 13 : Exemple de structure pour ce projet

4.1.1. Tableau comparatif

Pour effectuer le meilleur choix nous réalisons un tableau comparatif. Plusieurs critères sont analysés et une note entre zéro et quatre est attribuée pour chacun d'eux.

- Zéro : inexistant/non évaluable
- Un : insuffisant
- Deux : suffisant
- Trois : bon
- Quatre : très bon

Explication des critères évalués :

- **Types de données :** comment sont structurées les données dans le système.
- **Stockage des données simples :** c'est-à-dire sans relation avec d'autres données.
- **Stockage des données complexes :** par exemple des données hiérarchiques ou ayant des relations les unes avec les autres.
- **Requête simple :** retourner un résultat sans jointure, ni condition, ni filtre, ni tri.
- **Requête complexe :** effectuer des jointures, des conditions, des filtres ou des tris.
- **Transactions :** comment sont gérées les transactions et garantissent-elles l'atomicité ?
- **Évolutivité :** mesure la facilité d'évolution de la structure des données.
- **Sécurité :** comment la sécurité est-elle traitée.
- **Prix :** politique tarifaire

BDD Critères	Realtime Database		Cloud Firestore	
	Complément	Notes	Complément	Notes
Types (Données)	Arbre JSON	2	Structuré document organisé dans des collections	3
Stockage des données simples		4		4
Stockage des données complexes	Relativement difficile à grande échelle	2	Plutôt facile avec la gestion hiérarchique, sous-collections dans des documents	3
Requêtes simples	Renvoie toujours la totalité d'un arbre	2		3
Requêtes complexes	Limitées	1		4
Transaction	Implémentation de « Callback »	3	Transaction répétée jusqu'à ce qu'elle soit terminée	4
Evolutivité	Difficile	1	Adapté grâce à la structure	4
Sécurité	Uniquement sur la base de données	1	Deux niveaux : base de données et selon le SDK (Mobile & Web ou Server)	3
Prix	3 niveaux de prix à choisir	3	En fonction des opérations effectuées dans la BDD ; volume de stockage et utilisation de la bande passante	1
Totaux		19		29

Tableau 1 : Tableau comparatif entre Realtime Database et Cloud Firestore

4.1.2. Conclusion

À la suite du score obtenu par Cloud Firestore, nous arrêtons notre choix sur cette solution. A noter que la notion de requêtes complexes n'est pas déterminante dans notre contexte d'application. Car dans les deux cas nous avons choisi d'encapsuler les données par patient (cf. Figure 13 : Exemple de structure pour ce projet). Malgré tout, si on ignore ce critère, Cloud Firestore obtient toujours un meilleur score.

5 Architecture et mise en place de l'environnement de travail

5.1. Langage de développement des applications mobiles

Les applications mobiles sont développées en JAVA avec Android Studio version 3. Avant même de commencer le développement nous savons que certaines ressources (Class JAVA ; Images ; Couleurs…) seront communes au smartphone et à la smartwatch. De ce fait, nous décidons de travailler sur un projet unique qui contient les différents modules nécessaires :

- **Un module « phone » :** pour l'application du « smartphone »
- **Un module « wear » :** pour l'application de la smartwatch
- **Un module « smartphysiolibrary» :** qui contiendra toutes les ressources et entités communes

Pour déclarer le module « smartphysiolibrary » dans les autres modules, il suffit d'ajouter une implémentation dans le fichier build.gradle :

```
//Declare shared library
implementation project(':smartphysiolibrary')
```

5.2. Langage de développement de la web App

N'ayant aucune contrainte, il est convenu de développer en React.js. Nous n'étudions pas d'autres possibilités. Cependant cette bibliothèque Javascript est aujourd'hui une référence dans le monde du développement web et mobile.

5.2.1. React

React est une bibliothèque Javascript sortie en mai 2013 et développée par Facebook. Le principe est de structurer une application en assemblant des petits composants génériques et réutilisables. Ces composants ont l'objectif d'être autonomes. C'est-à-dire qu'ils contiennent l'aspect visuel (HTML et CSS) mais également la logique en Javascript. A noter que d'autres bibliothèques ou Framework comme Angular 2 et Vue.js reprennent le même principe.

5.3. VCS[4] avec Gitlab

Gitlab offre une solution SaaS dont les fonctionnalités sont similaires à Bitbucket ou à GitHub. Dans notre travail nous utilisons les fonctions suivantes :

- **VCS :** gestion initiale avec deux branches, « Dev » et « master ». Durant toute la durée d'un sprint nous travaillons sur la branche « Dev » et avant le « sprint review » nous créons un « merge request » vers « master ».
- **CI[5] :** en début de travail nous prévoyons également de mettre en place l'intégration continue. Ainsi chaque « commit » est vérifié.

5.4. Tests lab avec Firebase

Un avantage de travailler avec Firebase, c'est qu'il est possible tester l'interface utilisateur avec des tests d'instrumentations. Cela permet de valider le fonctionnement de notre application sur plusieurs modèles de smartphone. Cependant, si la création de tests d'instrumentations peut être développée sur Wear OS, leur exécution dans le Test Lab de Firebase n'est pas prise en charge.

Test d'instrumentation, 18/04/2018 23:23

Échec	Réussi	Résultats ignorés	Résultats non concluants
0	2	6	0

Exécution de test	Durée	Paramètres régionaux	Orientation	Problèmes
Pixel, niveau d'API 26	15 s	français	Portrait	–
Pixel, niveau d'API 25	13 s	français	Portrait	–
Galaxy S6 Edge, niveau d'API 25	–	français	Portrait	La combinaison appareil/niveau d'API est incompatible
OnePlus5, niveau d'API 25	–	français	Portrait	La combinaison appareil/niveau d'API est incompatible

Figure 14 : Résultat général des tests d'instrumentations dans Test Lab

[4] VCS : version control system. Logiciel de contrôle de version

[5] CI : continous integration. Intégration continue

5.5. Liste des applications utilisées

Pour mettre en œuvre l'ensemble de ce projet, nous utilisons plusieurs outils. Voici la liste exhaustive et une courte description.

Nom	Description	Licence	Version
Android Studio	Est une application développée par Google. Elle permet de développer des applications sur différents supports, comme : Smartphone, Tablette, Smartwatch, TV… Android studio prend en charge deux langages de programmation, Java et Kotlin.	Apache 2.0	3.1.X
Webstorm	Est fourni par JetBrains, cette IDE est idéale pour le développement Javascript et Typescript tant côté client que serveur.	Propriétaire	2017.3
Postman	Cet outil facilite le développement des requêtes http. En effet il permet d'envoyer sur un service distant des méthodes POST, GET, DELETE, PUT… puis d'en visualiser le résultat.	Libre	5.5.3
React DevTools	Une extension au navigateur « Google Chrome » qui permet de débuguer et d'analyser le contenu des « props » et du « state » (cf. Chapitre : 10.5, p. 64) des composants React.	Libre	3.2.4
Redux	Une extension au navigateur « Google Chrome » Qui permet l'analyse du « store » (cf. Chapitre 10.6, p. 65) de l'application Web	Libre	2.15.3

Tableau 2 : Liste des applications principales utilisées

6 Faisabilité

De manière à confirmer nos choix précédents, il convient d'en valider la faisabilité (cf. User Story N° 4 du PB). Nous réalisons en parallèle les trois applications avec pour chacune d'elles les objectifs suivants :

- **Smartwatch :** via une liste de zéro à dix, stocker dans Cloud Firestore la valeur choisie.
- **Smartphone :** s'authentifier dans Firebase et lire dans une liste l'historique des valeurs préalablement enregistrées avec la smartwatch.
- **Web App :** même fonction que pour le smartphone.

6.1. Paramétrage de Firebase

Premièrement, nous devons créer un projet dans Firebase. De manière à séparer les données de l'environnement de développement et celles du client, nous mettons en place un premier projet nommé : **SmartPhysioBeta**.

Comme l'administration des patients n'est pas prévue dans le sprint courant, nous générons un utilisateur directement dans l'interface de Firebase. Notre patient de référence est : **u10222@smartphysio.ch**

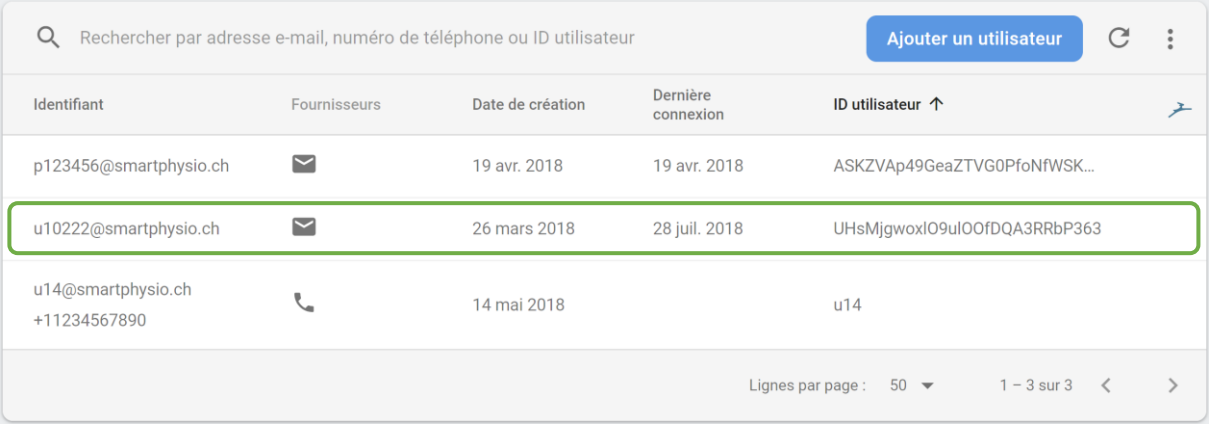


Figure 15 : Firebase, création d'un utilisateur

Nous paramétrons également les droits d'accès à la base de données. Nous limitons volontairement certains accès pour éviter les suppressions non volontaires, mais également pour tester les différents rôles, patient et physiothérapeute.

```
service cloud.firestore {
 match /databases/{database}/documents {
  match /users/UHsMjgwoxlO9ulOOfDQA3RRbP363/inputs/{document=**} {
   allow create, read, update, delete: if true;
  }
  match /users/wrongUID/inputs/{document=**} {
   allow create, read, update, delete: if false;
  }
  match /{document=**} {
   allow read: if true;
   allow write: if false;
  }
 }
```

Dans cette collection tout le monde peut :
lire, écrire, modifié et effacer

Dans cette collection on ne peut rien faire.

Dans toutes les autres collections, on ne peut que lire.

Figure 16 : Premier prototype de la smartwatch, liste des niveaux de douleurs

Finalement nous pouvons créer notre première collection « users » qui correspond aux « patients » et nos premiers documents dans Cloud Firestore. Les documents prennent un identifiant. Pour les patients, celui-ci correspond à « l'ID Utilisateur » (cf. Figure 15).

Légende

En orange : les collections

En bleu : les documents

En vert : un doc et ses attributs

Figure 17 : Premier prototype de la smartwatch, liste des niveaux de douleurs

6.2. Connexion de Firebase au projet Android

Notre projet Firebase étant créé, nous devons définir la connexion de celui-ci avec nos applications Android. Pour cette opération, Firebase propose un assistant qui permet de générer et de télécharger un fichier « google-services.json ». On place ce dernier dans le répertoire racine de l'application. L'intégralité du fichier « google-services.json » est disponible à l'Annexe V et il contient notamment les informations suivantes :

- Données d'accès au projet Firebase
- Données de l'application hôte, comme le nom du package
- Données d'authentification

6.3. Premier prototype de la smartwatch

Dans cette première phase de prototypage, nous codons certaines parties en « dur ». Notamment le chemin d'accès à la collection de notre patient « u10222 », défini par l'identifiant patient « uid ».

```
// Click on pain level element
private void onClick(int position) {

    String uid = "UHsMjgwox1O9ul0OfDQA3RRbP363";

    // get firestore instance
    FirebaseFirestore db = FirebaseFirestore.getInstance();

    // Create an new Pain Object and set some default values
    Pain pain = new Pain(painList.get(position));
    pain.setFlagged(false);
    pain.setActivity("none");
    pain.setTimestamp(new Date(currentTimeMillis()));

    //add a new pain document in Cloud Firestore. Path is hardcoded !
    db.collection("users").document(uid).collection("inputs").add(pain)
            .addOnSuccessListener(new OnSuccessListener<DocumentReference>(){
                // Log the result
                @Override
                public void onSuccess(DocumentReference documentReference) {
                    Log.d("Patient inputs", "ID: " + documentReference.getId());
                }
            })
            .addOnFailureListener(new OnFailureListener() {
                @Override
                public void onFailure(@NonNull Exception e) {
                    Log.w("Patient inputs", "Error writing document", e);
                }
            });
}
```

Pour la partie d'affichage nous utilisons la classe « `WearableRecyclerView` » spécifique à Wear OS pour la liste des niveaux de douleurs. Cette classe permet la gestion des écrans ronds.

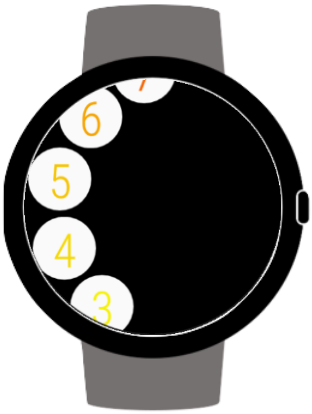


Figure 18 : Premier prototype de la smartwatch, liste des niveaux de douleurs

A chaque fois que le patient clique sur un élément de la liste, un document est généré dans le Cloud Firestore.

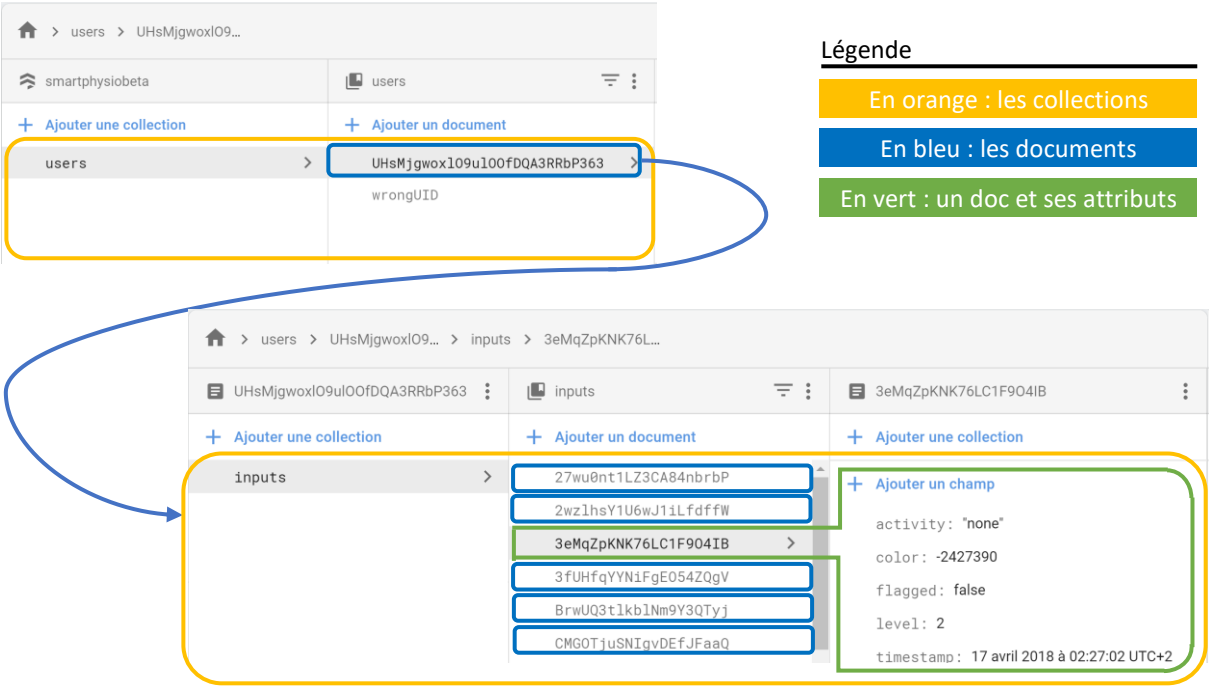


Figure 19 : Premier prototype de la smartwatch, liste des niveaux de douleurs

6.4. Premier prototype du smartphone

Les objectifs de ce premier prototype sont :

- Développer une page d'authentification, avec nom d'utilisateur et mot de passe
- Afficher dans une liste l'historique des niveaux de douleur saisis par le patient via la smartwatch

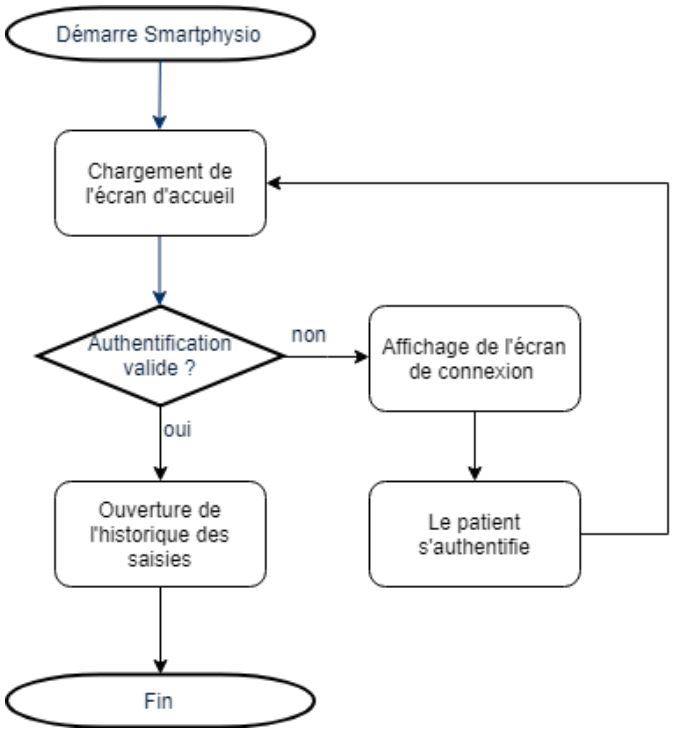


Figure 20 : Premier prototype du smartphone, diagramme de flux

La gestion du démarrage de l'application est traitée dans l'activité « MainActivity ». Selon l'état de l'instance utilisateur, le patient est dirigé vers les affichages correspondants.

```
// Get FirebaseAuthentication instance and get the current user.
mAuth = FirebaseAuth.getInstance();
FirebaseUser currentUser = mAuth.getCurrentUser();

// Test user object
if (user != null) {

    // if user is not null start HistoryActivity (Display all pains)
    startActivity(new Intent(this, HistoryActivity.class));
} else {
    // if user is null display Login layout (username;pwd and button)
    findViewById(R.id.email_password).setVisibility(View.VISIBLE);
}
```

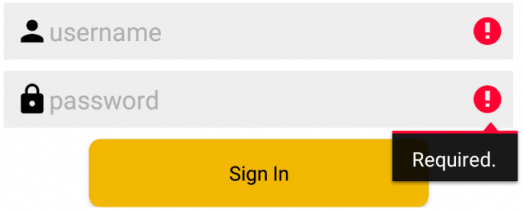
6.4.1. Tests d'instrumentations

De plus, notre volonté d'implémenter des tests d'instrumentations, nous amène à utiliser le Framework « Espresso » de Google. La mise en place est simple et correctement documentée[6]. Nous perfectionnons notre apprentissage avec YouTube sur la chaine de QA-Automated [7]

Le principe d'Espresso est de manipuler l'interface graphique. Une méthode « onView » permet d'accéder aux objets des « Layouts[8] ». Une fois l'objet désiré atteint, il est possible d'interagir avec celui-ci grâce à la méthode « perform ». Exemple avec le champ du mot de passe.

```
private void enterPassword(String password) {
// Get password field with onView (Use id and his parent to find it).
    ViewInteraction passwordField = onView(
            allOf(withId(R.id.password),
                    withParent(withId(R.id.email_password)),
                    isDisplayed()));
    // Change text value
    passwordField.perform(ViewActions.replaceText(password));
}
```

Dans la gestion des tests, la notion de tâche asynchrone ou multithread nécessite l'implémentation d'une classe d'attente « `IdlingResource` ». C'est notamment le cas pour le processus d'authentification. D'ailleurs nous déployons les tests uniquement sur ce processus, durant la phase de faisabilité.

Nom et description du test	Interface graphique
emptyUsernameAndPasswordFieldTest : vérification que le message « Required » s'affiche dans les deux champs vides.	

[6] À l'adresse : https://developer.android.com/training/testing/espresso/

[7] Playlist: Espresso Tutorial : https://www.youtube.com/watch?v=sDp8JNbITm4&list=PLx5ipGeoOO2hCbqjX784_I4Y4mDsVbVEf

[8] Fichier de ressource xml qui détermine l'affichage de l'interface graphique.

emptyPasswordFieldTest : vérification que le message « Required » soit visible dans le champ « password ».	u10222 password Sign In
emptyUsernameFieldTest : vérification que le message « Required » soit visible dans le champ « username ».	username ••••• Sign In
failedSignInTest : vérification que le message « Authentication failed » soit visible	wrong_username •••••••• Sign In Authentication failed
sucessfulSignInTest : vérification que le bouton « Sign out » soit visible.	SmartPhysio History 4 CMGOTjuSNIgvDEfJFaaQ Sun Jul 08 03:37:41 GMT+00:00 2018 4 27wu0nt1LZ3CA84nbrbP Thu Apr 26 13:46:51 GMT+00:00 2018 6 e0mfzuEK8gAn9vEDMn1i Thu Apr 26 13:46:42 GMT+00:00 2018

Tableau 3 : Description des tests (smartphone)

Finalement, ce travail sur les tests d'instrumentations donne plusieurs rapports de résultats. Cela va du simple booléen « passé » ou « échec », jusqu'à l'analyse complète de la couverture du code fourni, rapport fourni par « Jacoco »

Test Results ch.hevs.tb.jackh.smartphysio.HistoryTest recyclerPainLevel_ifOneExist_Test ch.hevs.tb.jackh.smartphysio.SignInTest emptyUsernameAndPasswordFieldTest emptyUsernameFieldTest emptyPasswordFieldTest failedSignInTest successfulSignInTest	debugAndroidTest **debugAndroidTest** Element / Missed Instructions / Cov. / Missed Branches ch.hevs.tb.jackh.smartphysio 68% ch.hevs.tb.jackh.smartphysio.util 65% Total 191 of 599 68% 14 of 24 Generated by the Android Gradle plugin 3.1.0
Passé ou échec	Analyse de la couverture du code

Figure 21 : Résultats des tests

Comme introduit dans le chapitre 5.4, nous exploitons le module « Tests Lab » de Firebase. L'objectif étant de tester notre application sur plusieurs périphériques, eux-mêmes installés avec des versions d'Android différentes. On peut simuler également les orientations et le choix des paramètres régionaux. Finalement on obtient un rapport détaillé du test. En particulier, la consommation de la mémoire, de la batterie et du CPU. Toutes ces données sont visibles sur la « timeline » de la vidéo des tests (cf. Figure 22).

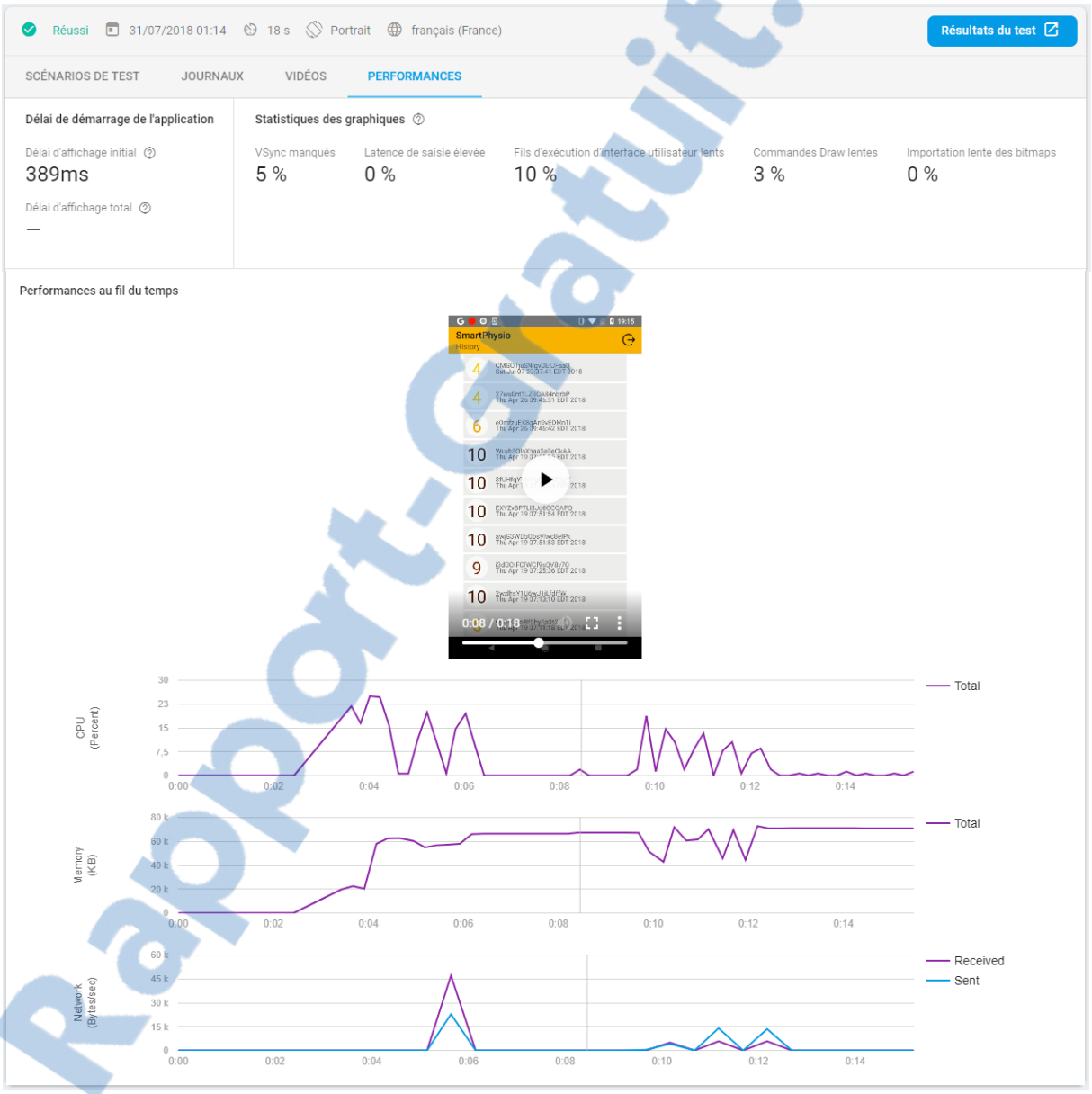
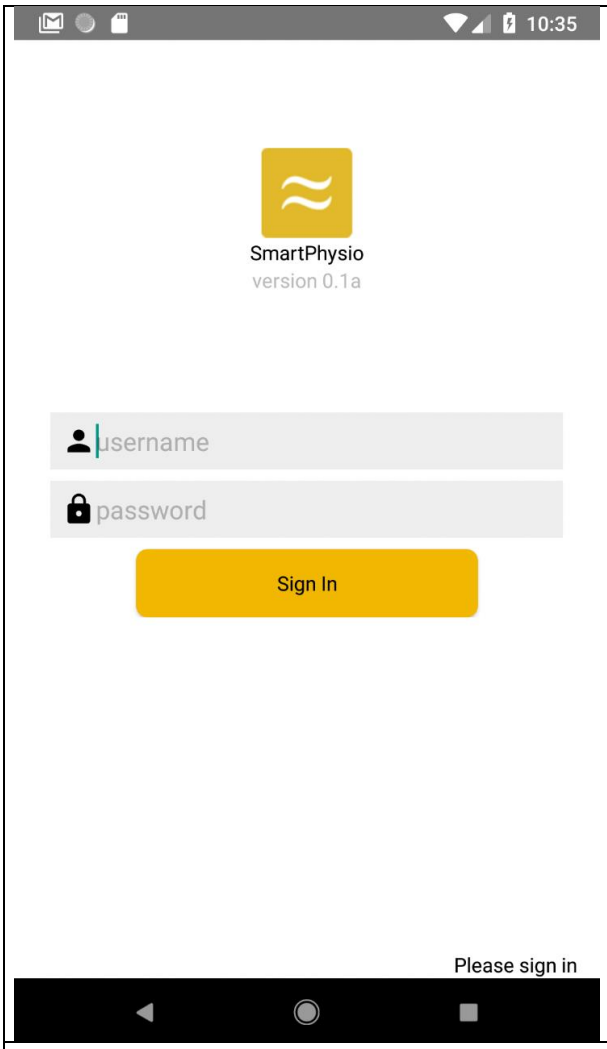
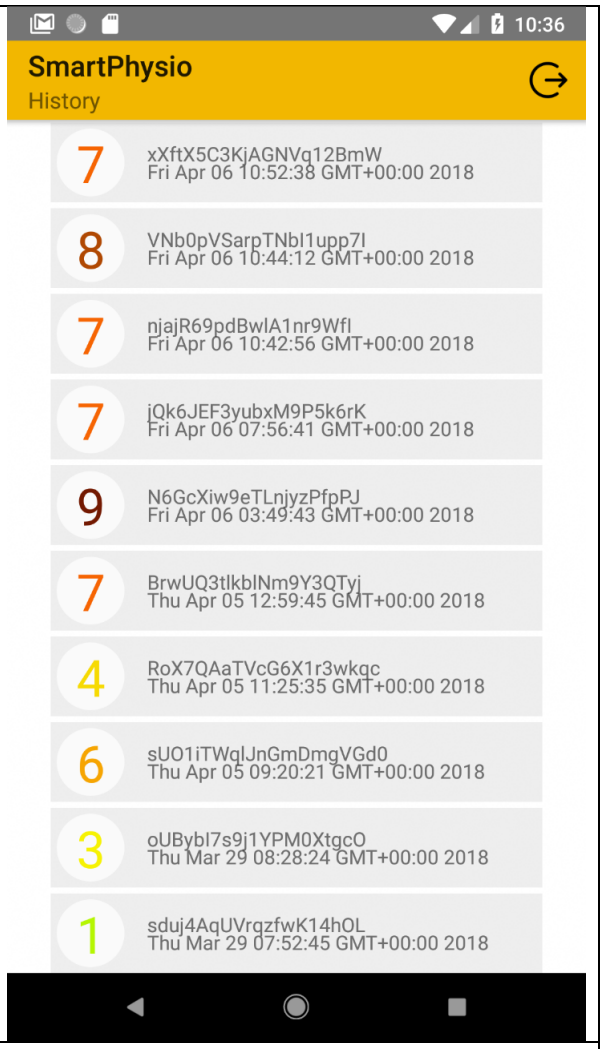


Figure 22 : Rapport détaillé des tests d'instrumentations exécutés dans Tests Lab de Firebase

6.4.2. Résultat du prototype

Notre prototype est terminé et il affiche, conformément à nos attentes, les données contenues dans la collection du patient authentifié.





1e écran : s'identifier Username : « u10222 » Password : « u10222 »	2e écran : affichage d'historique des saisies, dans une liste triée du plus récent au plus ancien

Figure 23 : Prototype du smartphone

6.5. Prototype de l'application web

La mise en place de l'application web a les mêmes objectifs que pour le smartphone :

- Développer une page d'authentification, avec nom d'utilisateur et mot de passe
- Afficher dans une liste l'historique des niveaux de douleur saisis par la smartwatch

Initialement la Web App est prévue pour le physiothérapeute, avec un compte utilisateur spécifique. Néanmoins dans cette phase de prototypage, nous voulons seulement valider la faisabilité de l'architecture. Nous utilisons donc notre compte patient « u10222 » pour tester le projet.

6.5.1. Création du projet

Nous utilisons l'outil « create-react-app » pour créer la base de l'application. Cela permet d'éviter certaines opérations de paramétrage. En particulier des outils « webpack » et « babel ».

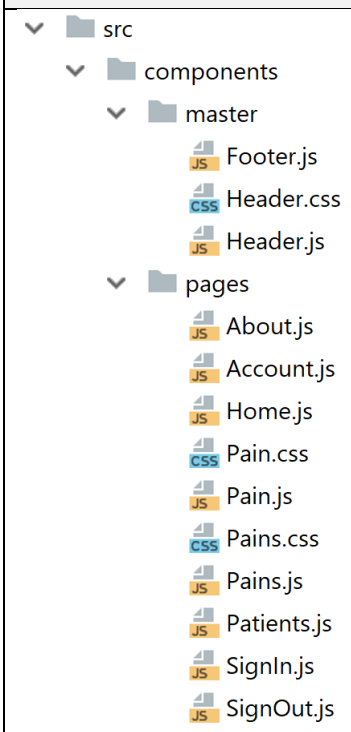
- **Webpack :** a différentes utilités, mais de manière générale il permet de simplifier l'organisation d'un projet. En regroupant des fichiers statiques (css ;svg…) en module. Mais aussi en facilitant leur chargement en tant que dépendance.
- **Babel :** il existe plusieurs versions normées du langage Javascript. Dans notre projet nous utilisons ES6. Néanmoins certaines versions de navigateur ne savent pas l'interpréter. Babel est un transcompilateur qui traduit du code Javascript d'une norme à l'autre.

6.5.2. Connexion de Firebase à notre projet React

Dans cette première phase de prototypage, nous devons également connecter notre projet React à Firebase. Comme pour Android nous utilisons l'assistant qui génère un fichier javascript contenant les informations nécessaires, voir Annexe VI.

6.5.3. Développement du prototype WebApp

Nous mettons en place une structure des fichiers qui facilite les développements futurs. Nous structurons donc notre projet en plusieurs composants. Certains sont indépendants et autonomes, d'autres sont de simples fichiers Javascript qui contiennent une certaine catégorie de fonctions ou de variables.

<table>
<tr><th>Structure</th><th>Description et exemple</th></tr>
<tr><td>


</td><td>
Tous ces composants étendent la classe « React.Component » qui impose l'implémentation de la méthode : « render » elle-même qui retourne le DOM.

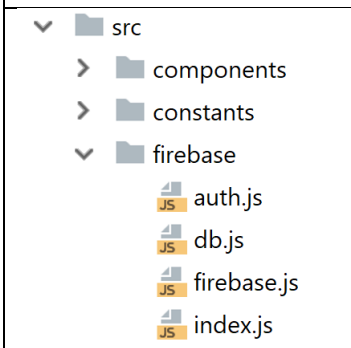
Exemple avec la classe « Footer » dont le contenu est statique.
<pre>
import React, {Component} from "react";
import "../App.css";

class Footer extends Component {

  render() {

    return (
      <footer className="Footer">
        hevs.ch - TB 2018 - Jacques Herren
      </footer>
    );

  }
}

export default Footer;
</pre>
</td></tr>
<tr><td>


</td><td>
Ces fichiers *.js décrivent des fonctions ou des variables uniquement.

Par exemple toutes les requêtes à Firestore dans « db.js »
<pre>
import {db} from './firebase';

// API

export const fetchPatientPains = (id) =>
  db.collection('users').doc(id).collection("inputs")
      .orderBy("timestamp", "desc").get();
</pre>
Cette méthode permet de récupérer toutes les saisies de douleurs selon l'identifiant du patient.
</td></tr>
</table>

Tableau 4 : Structure générale du projet React.js

6.5.4. Résultat du prototype de la Web App

Finalement, nous atteignons notre objectif en affichant toutes les saisies effectuées par le patient à l'aide de la smartwatch. La gestion des accès est également prise en charge. Tant que l'utilisateur de la plateforme n'est pas authentifié, il n'accède qu'aux pages « Sign in » et « About ».

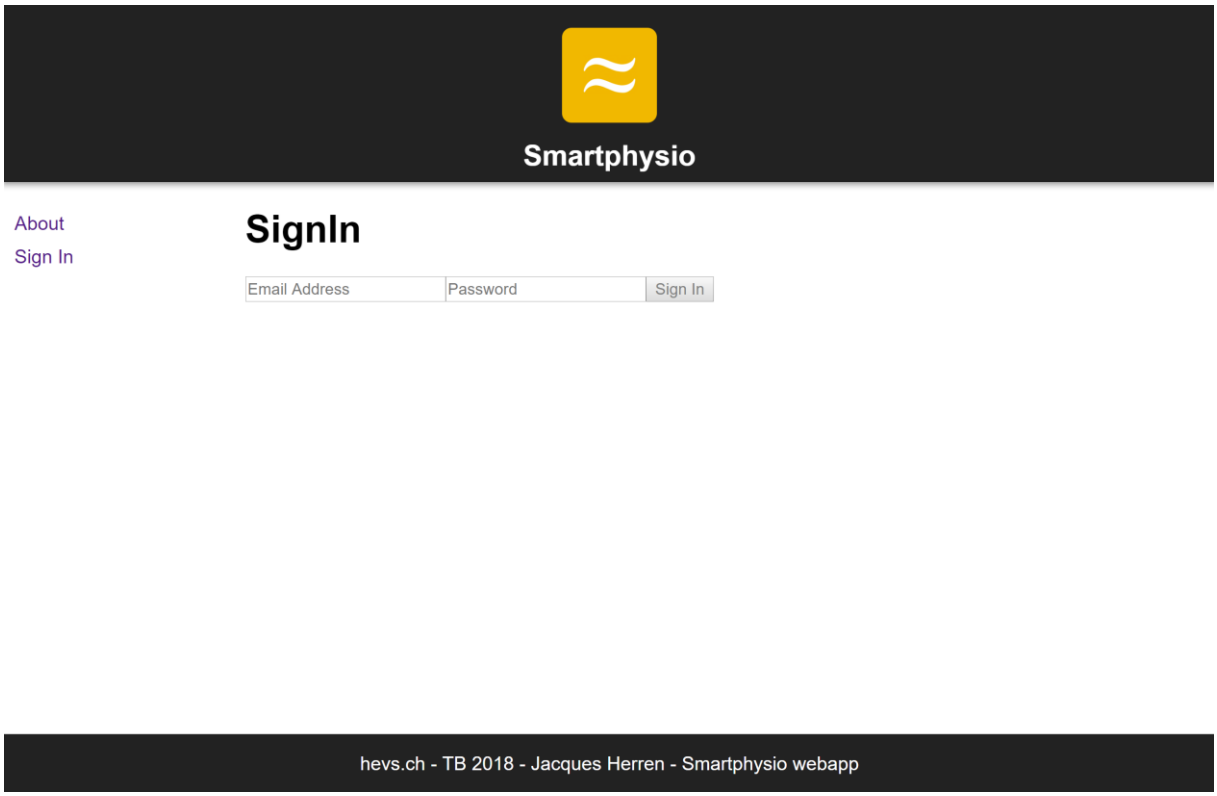


Figure 24 : Web App accès en tant que invité

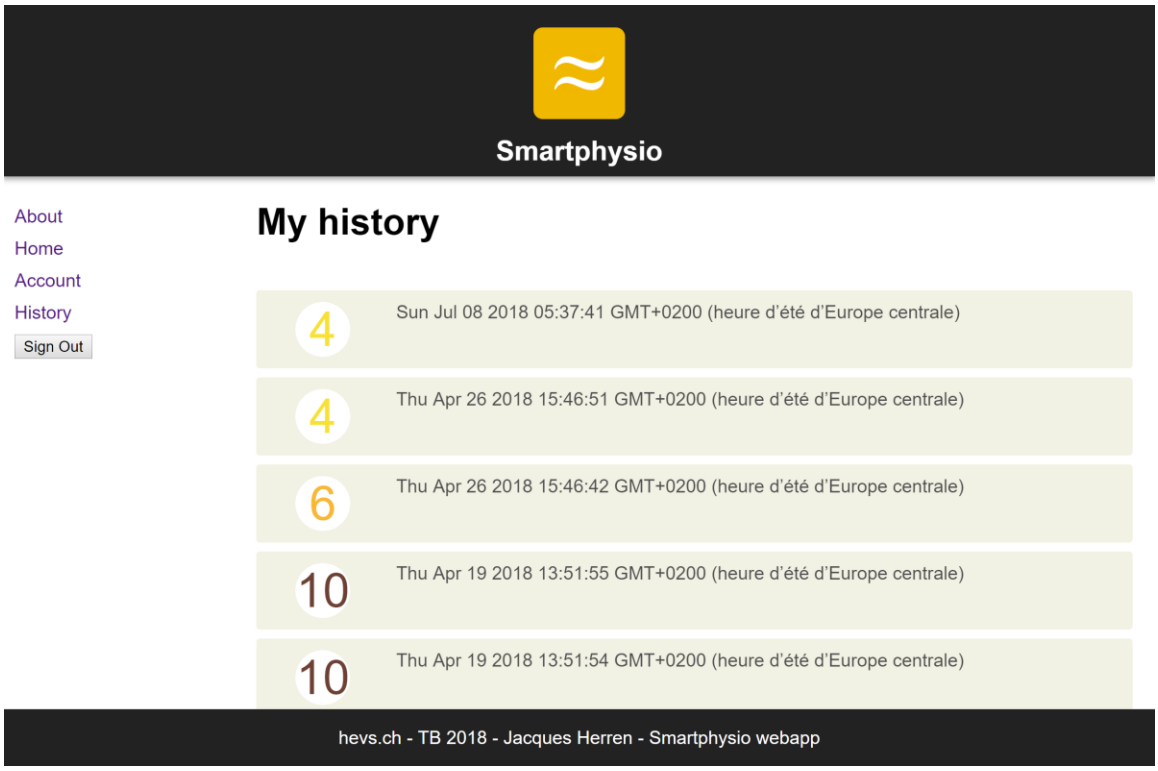


Figure 25 : Web App accès en tant que patient

6.6. Conclusion de la phase de faisabilité

Cette analyse sur la faisabilité correspond à la US N°4 du sprint N°1. Lors du « Sprint Review » le résultat de l'entier du prototype est validé. Toutefois une interrogation subsiste : ***comment s'identifier à la plateforme de Firebase avec la smartwatch ?*** Le fait de coupler la smartwatch avec le téléphone permettrait de saisir les données de connexion directement depuis le smartphone et ainsi connecter les deux appareils. Cette solution ne satisfait pas entièrement notre client. Car cela restreint l'accès à « Smartphysio[9] » uniquement aux possesseurs de smartphone Android.

Pour un cabinet de physiothérapie, l'idéal serait de prêter une smartwatch autonome, sur laquelle on installe l'application. Ainsi on ne connecte le compte du patient qu'une seule fois lors de la livraison. Nous acceptons ce nouveau « use case » et nous planifions la US correspondante pour le prochain sprint.

[9] Nom donné à ce projet de travail de bachelor.

7 Rendre la smartwatch autonome

Le sprint N°2 est composé de plusieurs US. Mais rendre la smartwatch 100 % autonome modifie notre modèle. En effet nous ne devons plus envisager une application mobile Android composée de deux modules « watch » et « phone ». Mais bien de deux applications indépendantes. Cela rend toujours possible le partage de librairie commune, comme « smartphysiolibrary » décrit dans le chapitre 5.1. Cependant nous ne pouvons pas utiliser le smartphone pour s'identifier à Firebase via les « DataLayer »[10] par exemple.

7.1. Analyse des possibilités de connexion

Nous étudions les bonnes pratiques d'authentification sur Wear OS. Google propose cinq possibilités :

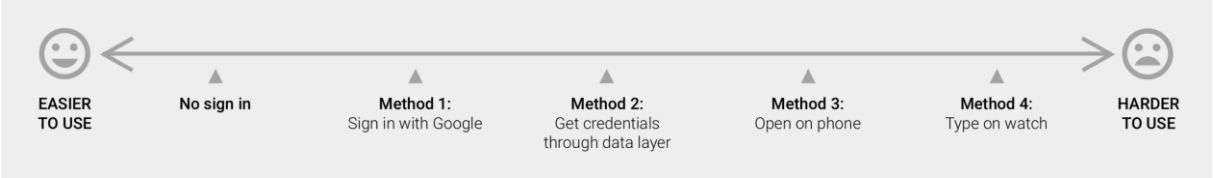


Figure 26 : Web App accès en tant que patient

Évaluation des différentes méthodes proposées :

- **No sign in :** non retenu, car notre application ne fait aucun sens si on ne peut pas identifier chacun des patients.
- **Sign in with Google :** à évaluer
- **Get credentials trough data layer :** non retenu, car nécessite un couplage smartwatch et smartphone Android.
- **Open on phone :** non retenu, pour les mêmes raisons que le cas précédent.
- **Type on watch :** à évaluer

[10] API de communication entre Wear OS et Android Phone

Comparaison des deux méthodes retenues.

Méthode 1 : Sign in with Google	
Avantages	Inconvénients
• Simple à mettre en place et à développer • Méthode d'authentification préconisée par Google. Simple pour l'utilisateur final	• Nécessite un compte Google • Protection des données • Pas de gestion multicomptes. Le patient ne peut pas utiliser sa propre montre

Tableau 5 : Comparaison des méthodes d'authentification (Méthode 1)

Méthode 4 : Type on watch	
Avantages	Inconvénients
• Simple à mettre en place et à développer • Permet de conserver notre architecture	• Méthode d'authentification difficile à utiliser. Les touches du clavier étant très petites

Tableau 6 : Comparaison des méthodes d'authentification (Méthode 4)

Nous privilégions la méthode 4 car l'unique inconvénient n'est pas déterminant pour l'usage quotidien du patient. L'objectif étant de ne saisir les informations de connexion qu'une seule fois lorsque le physiothérapeute remet la smartwatch au patient.

7.2. Compatibilité smartwatch avec Firebase

Malgré plusieurs tentatives lors de la mise en place d'une authentification autonome, nous rencontrons toujours la même erreur.

```
API: InternalFirebaseAuth.FIREBASE_AUTH_API is not available on this device.
```

Après, quelques recherches, nous découvrons que Wear OS n'est pas prévu pour être connecté de manière autonome sans un compte Google. (Lake, 2017). Même si le « post » date de plus d'une année, la non prise en charge de Firebase par Wear OS est toujours d'actualité (30.04.2018).

7.3. Proposition de solutions

Compte tenu des problèmes identifiés ci-dessus, nous cherchons des solutions afin de garantir l'autonomie de la smartwatch.

- **Solution N° 1 :** utiliser la méthode 1 d'authentification (cf. Chapitre 7.1).
- **Solution N° 2 :** développer une API indépendante qui fait l'interface entre la smartwatch et Firebase.
- **Solution N° 3 :** développer une API indépendante et une base de données totalement découplées de Firebase.

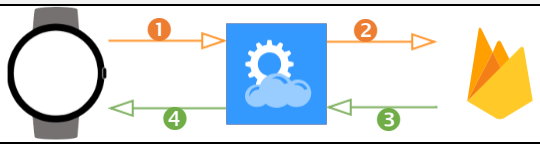
Solution N°2	
	
Smartwatch API Firebase	
Le principe consiste à développer un serveur API qui joue le rôle d'un intermédiaire entre la smartwatch et Firebase. Le patient saisit ses identifiants de connexion. Puis lorsqu'il les valide, une requête est envoyée au serveur ❶. Ensuite l'API transmet ❷ les données à Firebase qui les traite avant de renvoyer la réponse ❸. Cette dernière est ensuite retournée à la Smartwatch ❹.	
Avantages	Inconvénients
• Permet de conserver Firebase	• Nécessite de développer un serveur API • Ce serveur devra être hébergé et installé

Tableau 7 : Analyse API intermédiaire (Solution N°2)

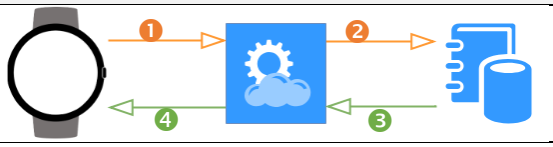
Solution N°3	
	
Smartwatch API SmartBDD	
Le principe est identique à la solution N°2. Mais au lieu d'utiliser Firebase on gère notre propre base de données.	
Avantages	Inconvénients
• Liberté de structurer nos données • Pas de dépendance à des services tiers • Meilleure protection des données	• Nécessite de développer un serveur API et une base de données • Ce serveur devra être hébergé et installé

Tableau 8 : Analyse API et BDD indépendantes (Solution N°3)

7.4. Décisions et erreurs

Nous soumettons la problématique à Michael Ignaz Schumacher et Fabien Dubosson avec les trois propositions de solutions envisagées. La réponse est : **Solution N°1**.

En dépit de cette réponse nous choisissons, malgré tout, d'implémenter la solution N°3. Ceci pour plusieurs raisons :

- La protection des données (indépendant de Google ou d'un autre service)
- Plus de souplesse pour la structure des données
- La notion d'infrastructure, notamment le serveur pour héberger l'API et la base de données, est mineure, car dans tous les cas pour l'application Web nous devons également mettre à disposition un serveur.
- Une surcharge de travail est à prévoir mais largement compensée par la protection des données.

A cet instant nous commettons deux erreurs dans notre projet :

- Premièrement le changement de technologie doit être décidé en accord avec le PO avec un arrêt de Sprint.
- Deuxièmement nous avons perdu deux jours de développement en cherchant des solutions annexes. Et nous perdons les quatre derniers jours prévus dans le sprint pour développer un serveur API qui en finalité ne sera pas exploité.

7.5. Présentation du serveur API

Dans les quatre jours restant du sprint N°2, nous développons une API REST basée sur les outils suivants :

- **Node.js :** permet de développer des serveurs de type http en langage Javascript.
- **MongoDB :** est un système de base de données de type NoSQL organisé comme Firestore en collections et documents.

Puis nous testons notre nouveau service avec la smartwatch. L'envoi des requêtes http est assuré par la librairie « RetroFit ».

7.5.1. Côté serveur

L'objectif étant de valider la faisabilité d'une authentification depuis une smartwatch via une API REST, nous mettons en place les services d'entrée suivants :

Méthode	URL	Description
POST	/users/signup	Création d'un nouveau patient
POST	/users/signin	Authentification du patient
GET	/users	Récupération de la liste de tous les patients

Tableau 9 : Analyse API et BDD indépendantes (Solution N°3)

Nous définissons un objet « user » avec trois propriétés (l'intégralité du code est disponible à l'Annexe VII) :

- Username : identifiant du patient
- Password : mot de passe
- Timestamps : gestion des dates de création et de modification

Quelques règles de bases sont définies sur notre objet « user » notamment la gestion du hachage du mot de passe, pour ne pas stocker des mots de passe en « clair » dans la base de données.

- Comparaison du mot de passe lors de l'opération de d'identification.

```
/** * @param password
 * @returns {hash password} */
UserSchema.methods.comparePassword = function (password) {
   return bcrypt.compareSync(password, this.password);
};
```

- Hachage du mot de passe effectué à la création du nouveau patient

```
/** * @param password
 * hash user's password */
UserSchema.methods.setPassword = function setPassword(password) {
   this.password = hashPassword(password);
};

/** * @param password
 * @returns hash string 'password' */
function hashPassword(password) {
   var salt = bcrypt.genSaltSync(10);
   return bcrypt.hashSync(password, salt);
}
```

- Garantir l'unicité des identifiants des patients

```
UserSchema.plugin(uniqueValidator, { message: "This username is already taken"});
```

7.5.2. Côté smartwatch

Grâce à l'application Postman, nous avons créé un nouveau patient dans notre base de données (cf. Figure ci-dessous).

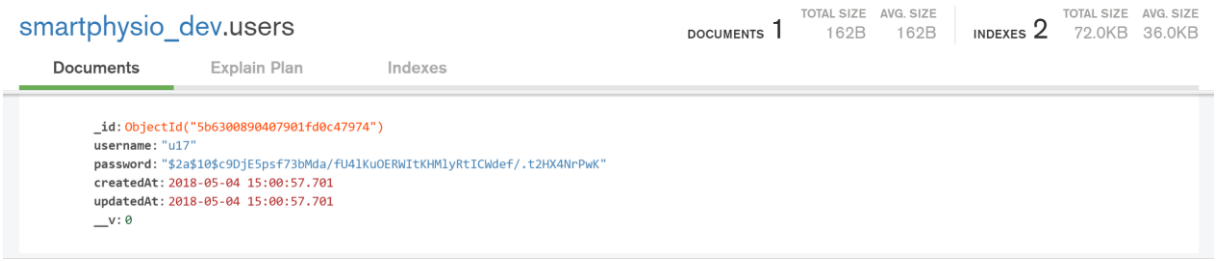


Figure 27 : Patient créé dans MongoDB

La consommation du service REST nécessite l'envoi d'une requête http depuis la smartwatch. Plusieurs librairies sont disponibles pour Android. En raison du délai court nous n'évaluons pas toutes les alternatives mais optons pour RetroFit. Si notre nouvelle architecture est validée en fin de sprint N°2, nous comparerons les outils disponibles plus tard.

- Log de la requête envoyée :

```
D/OkHttp: --> POST http://192.168.1.118:5000/api-dev/users/signin
http/1.1
          Content-Type: application/json; charset=UTF-8
          Content-Length: 35
          {"password":"pwd","username":"u17"}
          --> END POST
```

- Log de la réponse du serveur[11] :

```
D/OkHttp: <-- 200 OK http://192.168.1.118:5000/api-dev/users/signin
(274ms)
D/OkHttp: X-Powered-By: Express
          …
          Content-Length: 232
          ETag: W/"e8-91dZIhXVI6KindMzrgNNJXWNqVg"
          Date: Thu, 02 Aug 2018 09:30:47 GMT
          Connection: keep-alive
D/OkHttp:
{"success":true,"token":"eyJhbG…","_id":"5b6224a9ecda1167e0d73dff"}
          <-- END HTTP (232-byte body)
```

Figure 28 : Authentification de la smartwatch avec le compte du patient

[11] La réponse est volontairement tronquée ; l'intégralité est disponible en Annexe VIII

7.6. Fin du sprint N°2

La séance du sprint review, souligne les erreurs commises dans la gestion de ce sprint. Dans une situation où un problème technique est identifié et qu'un changement de technologie doit être envisagé, il est primordial d'organiser une séance de crise avec les différentes parties prenantes.

En conclusion, l'ensemble des US sont rejetées. De plus, l'utilisation d'un serveur API REST couplé à une base de données privée n'est pas retenu, car il est trop lourd à maintenir et il nécessite pour le client une infrastructure complexe.

Après discussion, l'idée d'utiliser « Pryv[12] » est évoquée. Les avantages sont multiples et parmi eux, les principaux que nous avons relevés :

- Protection et sécurité selon les normes RGPD et HIPAA.
- Lieu de stockage des données dans plusieurs pays à choix dont la Suisse
- API existante et documentée sur : http://api.pryv.com

Il est important d'ajouter que si le sprint N°2 semble être un échec, la manipulation des requêtes http, la gestion des tokens[13] et le stockage des identifiants sont relativement similaires à notre implémentation API REST de ce sprint.

A noter également que l'utilisation de Pryv dans ce travail, permet d'envisager une utilisation de Smartphysio au-delà du simple prototype initialement prévu.

[12] PrYv est un middleware conçu pour différentes applications garantissant une protection et une sécurité conforme aux normes RGPD et HIPAA.

[13] Token : jeton en français ; a pour objectif de garantir l'identité d'un utilisateur. On privilégie le terme token dans tout le document.

8 Implémentation de Pryv

Démarrage du sprint N°3, l'objectif étant de reprendre les US du sprint précédent (15 story points) et de les adapter à Pryv. De plus, nous devons revoir les procédures de connexion déjà développées pour le smartphone et la Web App. Au total 24 « story points » sont planifiés.

Nous prévoyons un temps de formation sur Pryv afin de tester et de se familiariser avec l'API.

8.1. Pryv en bref

L'entreprise Pryv SA basée à Ecublens (VD) a été créée en octobre 2012. Elle est active dans de multiples domaines informatiques (Office cantonal vaudois du registre du commerce, s.d.).

Dans le cadre de notre travail, nous nous intéressons en particulier aux services proposés suivants :

- La gestion et la sécurisation des données
- La distribution de logiciels et de services

8.1.1. Les données dans Pryv

MongoDB ou Firestore comme la plupart des systèmes actuels propose des types de données très génériques : entiers, chaînes, booléen, date… Pryv propose un dictionnaire de types de données spécifiques. Cela permet l'interopérabilité (Pryv, s.d.), c'est-à-dire la capacité d'échanger des données entre différents systèmes hétérogènes. On peut citer par exemple les types suivants :

- **frequency/bpm :** fréquence de battement par minute, peut être utilisé pour les pulsations cardiaques.
- **temperature/c :** température en degré Celsius
- **count/steps :** un nombre de pas

Il existe aujourd'hui (août 2018) un peu moins de 300 types de données différentes.

8.1.2. Structures des données

Comparée aux structures de Firestore et MongoDB qui proposent une hiérarchie en collections et documents (cf. Chapitre 4.1), Pryv s'en approche mais la terminologie change. On parle de :

- **Stream**[14] **:** Collection (dossier)
- **Event**[15] **:** Document (fichier)

Toutefois, en termes de hiérarchie on peut faire une analogie avec une arborescence de dossiers (Streams) et fichiers (Events). Un Stream peut donc contenir plusieurs sous-Streams, qui eux-mêmes regroupent des Events (Pryv, s.d.). Mais contrairement au document de Firestore qui peut contenir une sous-collection, cela n'est pas le cas des Events, exactement comme pour les fichiers.

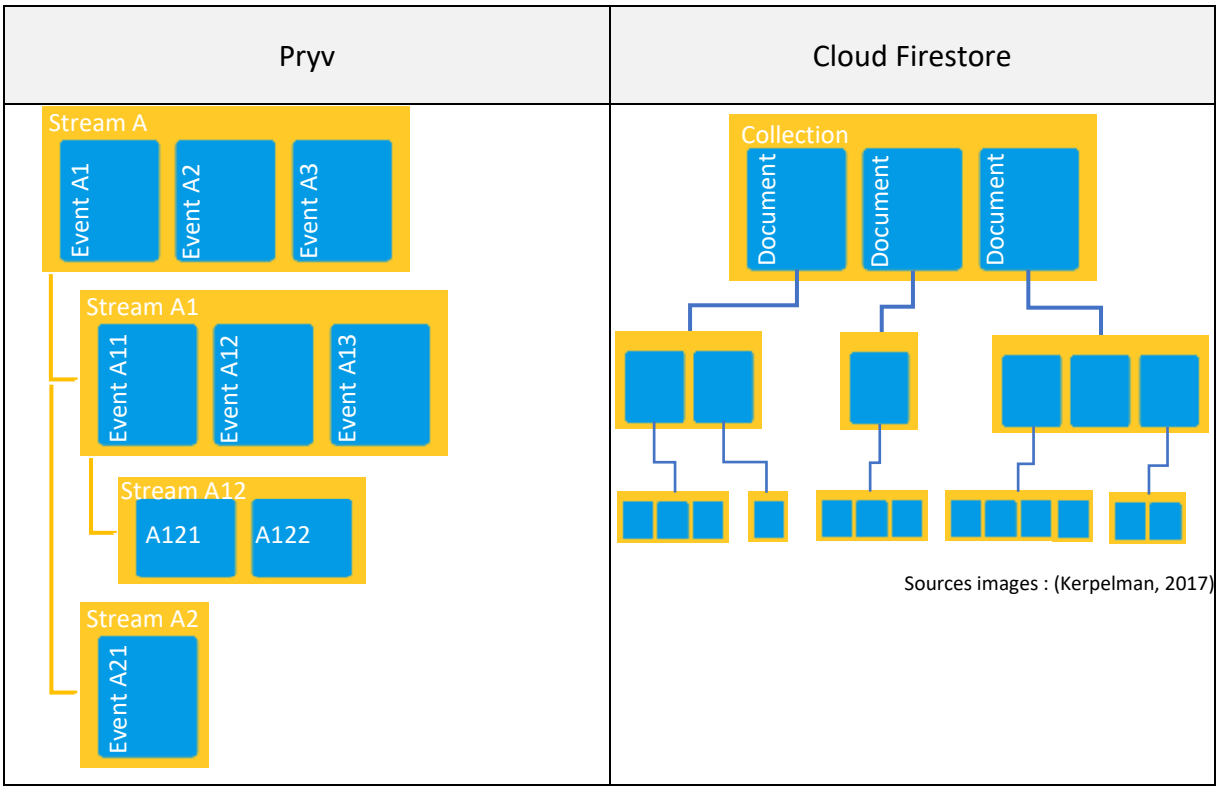


Figure 29 : Comparaison des structures entre Pryv et Cloud Firestore

[14] Stream : flux en français ; nous utiliserons le terme Stream dans tout le document

[15] Event : événement en français ; on conserve également le terme Event dans tout le document

8.1.3. Les Streams

Ils sont toujours composés de la même manière. Il faut savoir qu'il existe d'autres propriétés qui permettent de gérer la suppression d'un Stream ou d'un Event, non décrites ici.

- **Name :** le nom d'affichage
- **Id :** l'identifiant du Stream
- **Created, modified :** des indications temporelles de création et de modification
- **CreatedBy, modifedBy :** donne l'identifiant de l'opérateur
- **parentId :** si le Stream est un enfant on indique l'id du parent. Si le Stream est à la racine principale la valeur est 'null'
- **children :** est un tableau de Streams enfants
- **clientData :** peut contenir n'importe quelle structure JSON

```
{
    "name": "Heart",
    "parentId": null,
    "created": 1531981087.335,
    "createdBy": "cji5os3u11ntt0b40tg0xhfea",
    "modified": 1531981127.735,
    "modifiedBy": "cjhagb5up1b950b40xsbeh5yj",
    "clientData": {  },
    "id": "cjj9f7ixm005g0cd3abhfrblr",
    "children": [],
}
```

8.1.4. Les Events

Ils ont également une structure de base toujours identique. Certaines propriétés sont déjà décrites dans le Stream. Comme différence on peut noter :

- **Id :** l'identifiant de l'Event
- **StreamId :** correspond au Stream dans lequel l'Event est associé
- **Type :** définit le type de données (cf. Chapitre 8.1.2)
- **Content :** la valeur de l'Event pour une fréquence cardiaque, une valeur entière
- **Tags :** un tableau de mots-clés

```
{
    "streamId": "pulseOximeterApp",
    "type": "frequency/bpm",
    "content": 90,
    "tags": [],
    "id": "cji5pfumt1nu90b40chlpetyp"
}
```

Attention :
Les propriétés déjà décrites dans les Streams ne sont pas affichées volontairement

8.1.5. Compte utilisateur (patient)

C'est à ce niveau que Pryv modifie les paradigmes habituels des bases de données. Si on reprend notre analogie avec Firestore, voici une schématisation des architectures des BDD.

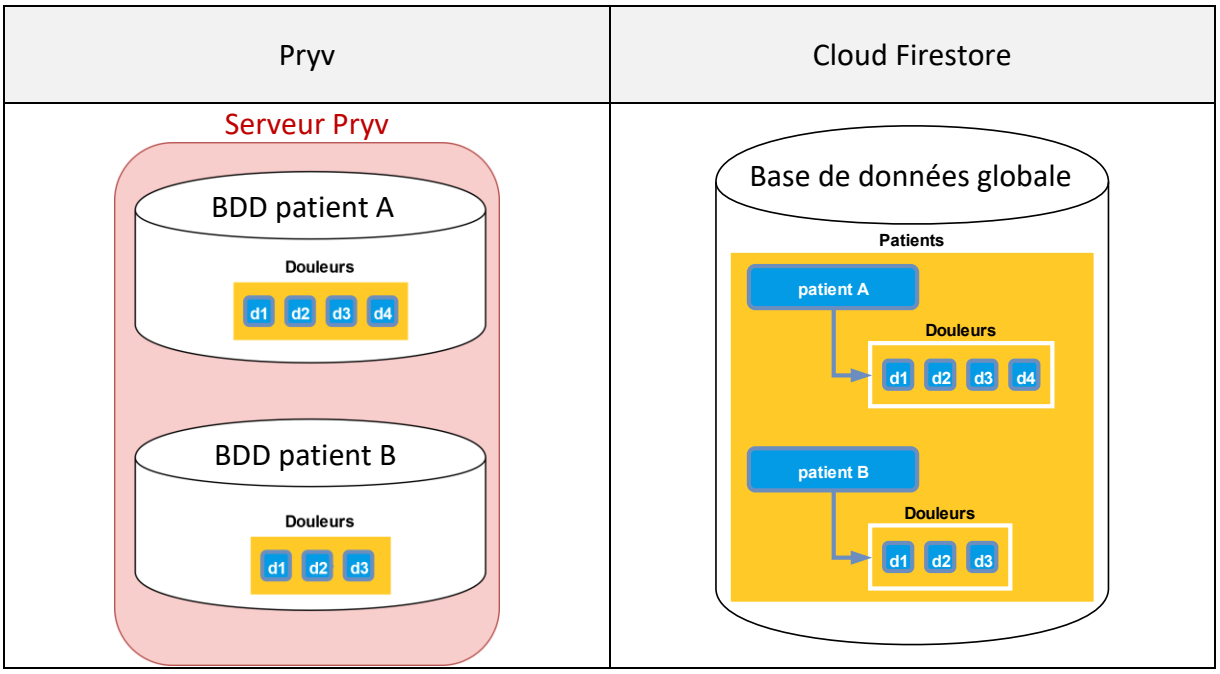


Figure 30 : Comparaison des structures de BDD entre Pryv et Cloud Firestore

Dans le cas de Pryv le patient est en possession de ses données. C'est lui qui choisit comment et avec qui il désire partager son contenu. Pour illustrer différemment ce modèle on peut analyser les requêtes envoyées aux différents services

	url pour récupérer la liste des douleurs d'un patient

Firestore	• Racine de l'url commune à tous les patients
	`https://firestore.googleapis.com/v1/projects/smartphysio/databases`
	• On passe l'identifiant du patient en paramètre pour cibler un patient
	`racine/patients/{id=id}/douleurs`

Pryv	• Chaque patient à sa propre url `https://patientA.pryv.me/events?limit=0&streamid=painlevel`

Tableau 10 : Comparaison des appels REST entre Pryv et Cloud Firestore

8.2. Gestion des autorisations dans Pryv

Les applications qui doivent accéder aux données de Pryv doivent obtenir une autorisation. Celle-ci est matérialisée par un nom d'utilisateur et un token (Pryv, s.d.). Un accès Pryv est défini par trois composantes : un nom d'application, un type et un ensemble de permissions. Dans l'Annexe IX un extrait du site de Pryv est disponible pour plus de détails. Procédure d'obtention de token :

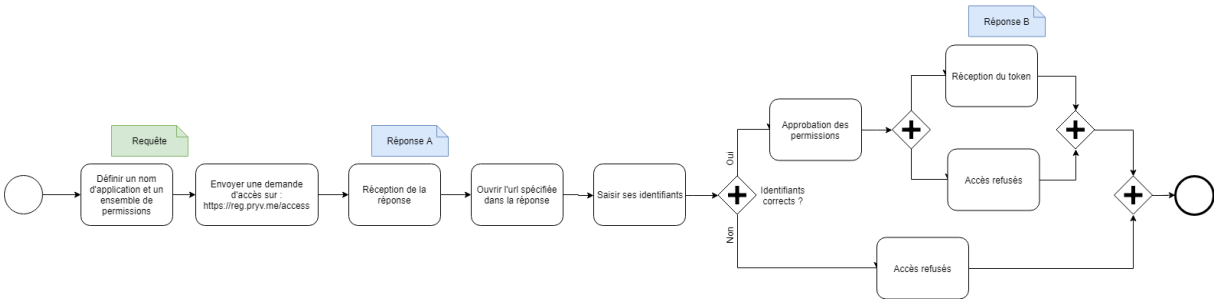


Figure 31 : Processus d'accès standard

- Contenu de la requête :

```
{   "requestingAppId": "smartphysio_exemple",
    "requestedPermissions": [
    {
        "streamId": "douleurs",
        "defaultName": "Douleurs",
        "level": "manage"
    }],
}
```

- Réponse A du serveur :

```
{   "status": "NEED_SIGNIN",
    "code": 201,
    "key": "JXGtk9SJLZGNWZeU",
    "requestingAppId": "smartphysio_exemple",
    "requestedPermissions": [
        {
            "streamId": "douleurs",
            "defaultName": "Douleurs",
            "level": "manage"
        }
    ],
    "url": "https://sw.pryv.me/access/access.html?lang=fr&
                    key=HnLqRfg2oCL2THQ6&
                    requestingAppId=smartphysio_exemple&
                    returnURL=false&domain=pryv.io&
                    registerURL=https%3A%2F%2Freg.pryv.me%3A443&
                    requestedPermissions=%5B%7B%22
                        streamId%22%3A%22douleurs%22%2C%22
                        defaultName%22%3A%22Douleurs%22%2C%22
                        level%22%3A%22read%22%7D%5D",
    "poll": "https://reg.pryv.me:443/access/ HnLqRfg2oCL2THQ6",
    "returnURL": "false",
    "poll_rate_ms": 1000
}
```

- Réponse B du serveur :

```
{   "status": "ACCEPTED",
    "username": "patientA",
    "token": "cjke0fcjm03sp0cd399b7pb5t",
    "code": 200
}
```

En conclusion, dans notre travail cela signifie que le physiothérapeute, doit être en possession du couple « username & token » pour chacun de ses patients et où le token définit :

- À quel Stream il peut accéder
- Quel est le niveau d'accès au Stream (read, manage ou contribute)

8.3. Structuration des données dans Pryv

8.3.1. Côté patient

La tâche principale du patient est de saisir et de modifier des « niveaux de douleurs ». Nous préparons l'arborescence des Streams selon la figure ci-dessous (Event en bleu) :

Schéma	Représentation JSON des Events

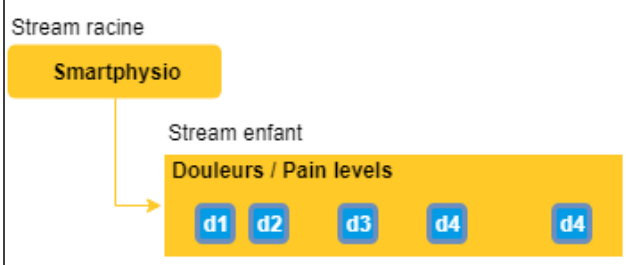
	{
	"clientData": {
	"spActivity": {
	"activity": "Swimming",
	"comment": null,
	"flag": false,
	"id": "cjj9f7ixm005g0cd3abhfrblr",
	"streamId": "cjjx0mic602f60cd39x…"
	}
	},
	"content": 6,
	"streamId": "levelPains",
	"type": "count/generic",
	"tags": [],
	"id": "cjjs609fd000fe6982t21y3xa"
	}

Figure 32 : Structure des données du patient

Pour les Event, le type de données retenu est « count/generic » qui permet de stocker des valeurs numériques. Pour le stockage de l'activité et des commentaires ajoutés par le patient nous décidons de les structurer en format JSON afin de les enregistrer dans la balise « clientData » des Events.

Une balise « spActivity » permet de garantir la pérennité de l'information. En effet on peut imaginer qu'une application tierce vienne également écrire dans « clientData », et il est donc recommandé de stocker dans un sous-niveau.

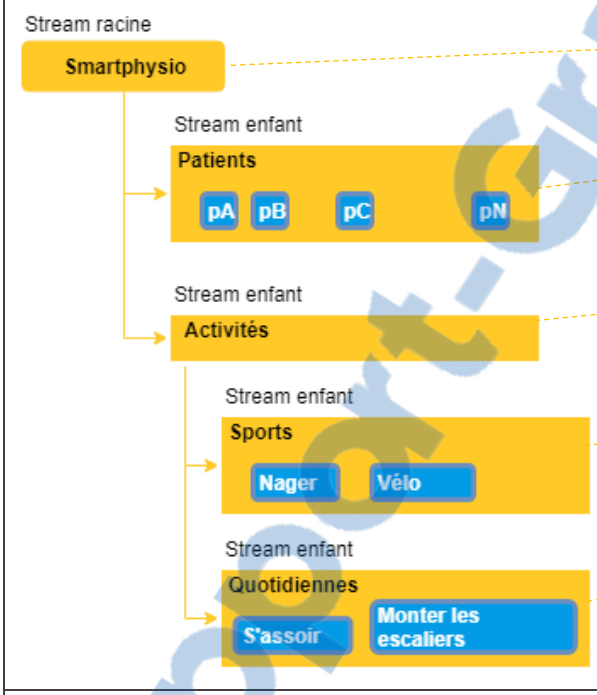
- **Activity :** donne le nom de l'activité
- **Comment :** commentaire complémentaire ajouté par le patient
- **Flag :** est un marqueur qui permet au patient d'ajouter un complément plus tard

- **Id :** correspond à l'identifiant de l'activité. N'est pas utilisé, mais permettrait la gestion multi-langue par exemple.
- **StreamId :** les activités étant classées par groupe, cela correspond donc à la catégorie de l'activité (cf. Chapitre 8.3.2)

8.3.2. Côté physiothérapeute

Nous avons déjà évoqué l'idée du stockage du couple d'accès « username & token » (cf. Chapitre 8.2). Ci-dessous une vision schématique de la structure des Streams du physiothérapeute.

Schéma	Représentation JSON des Streams

	{
	"name": "Smart physio",
	"id": "smartphysio",
	"children": [
	{
	"name": "Patients",
	"parentId": "smartphysio",
	"id": "patients",
	"children": []
	}
	{
	"name": "Activities",
	"parentId": "smartphysio",
	"id": "activities",
	"children": [
	{
	"name": " Sports ",
	"parentId": "activities",
	"id": "cjjxjkjrh02hh0cd…",
	"children": []
	},
	{
	"name": "Quotidiennes",
	"parentId": "activities",
	"id": "cjjx0o2vo02fb0cd…",
	"children": []
	},
	]
	}

Un patient :
• **Type :** « note/text »
• **Content :** on stocke le « username »
• **ClientData :** on sauve le token
 	
On trouve ainsi notre couple « username & token »
	{
	"streamId": "patients",
	"type": "note/text",
	"content": "patienta",
	"clientData": {
	"smartphysio_admin": {
	"auth": "cjjilo1pe014w0cd3letxyus9"
	},
	},
	"id": "cjjilo7rc014y0cd3tuu7s0f5"
	}

Une activité :
• **Type :** « note/text »
• **Content :** on stocke le nom de l'activité
	{
	"streamId": "cjjx0mic602f60cd39x1y3cgs",
	"type": "note/text",
	"content": "Running",
	"id": "cjjx0n9ig02f80cd3h3dijvzj"
	}

Tableau 11 : Structure des Streams et des Events côté physiothérapeute

9 Développement du sprint N°3

L'objectif principal de ce sprint est de connecter toutes les applications avec les utilisateurs adéquats :

- Web App avec un compte physiothérapeute
- Mobile avec un compte patient

9.1. Connexion de la Web App (US : 88)

Pryv fourni un exemple javascript (Pryv, s.d.) que nous implémentons dans notre projet React. Cela consiste à définir un fichier JSON similaire à celui des autorisations d'accès (cf. Chapitre 8.2). Il faut ajouter quelques propriétés afin de gérer l'interface graphique et les « callbacks ».

```
export let authSettings = {
   isAuthenticated:false,
   connection:null,
   requestingAppId: 'smartphysio_admin',
   requestedPermissions: [
      {
         streamId: 'smartphysio',
         defaultName: 'Smart physio',
         level: 'manage'
      }
   ],
   // set this if you don't want a popup
   returnURL: false,
   languageCode: 'fr',
   // use the built-in auth button
   spanButtonID: 'pryv-button',
   callbacks: {
      initialization: function () {
         //console.log('# Auth initialized');
      },
      // optional; triggered if the user isn't signed-in yet
      needSignin: function (){
         throw 'needSignin : Not implemented'
      },
      signedIn: function () {
         throw 'signedIn : Not implemented'
      },
      refused: function (reason) {
         //console.log('# Auth refused: ' + reason);
      },
      error: function (code, message) {
         //console.log('# Auth error: ' + ' ' + message);
      }
   }
};
```

Similaire à la gestion des accès

Bouton d'authentification

Les « callbacks » Doivent être implémentés dans la classe hôte

9.2. Connexion avec le smartphone (US : 85)

Pryv fournit également un projet d'exemple pour Android (Pryv, 2017). On modifie notre projet afin d'intégrer le système d'authentification. Concrètement on ne gère plus les champs texte pour la saisie « username » et « password » car Pryv utilise une « WebView[16] ». A noter que la troisième étape est proposée uniquement lors de la première connexion.

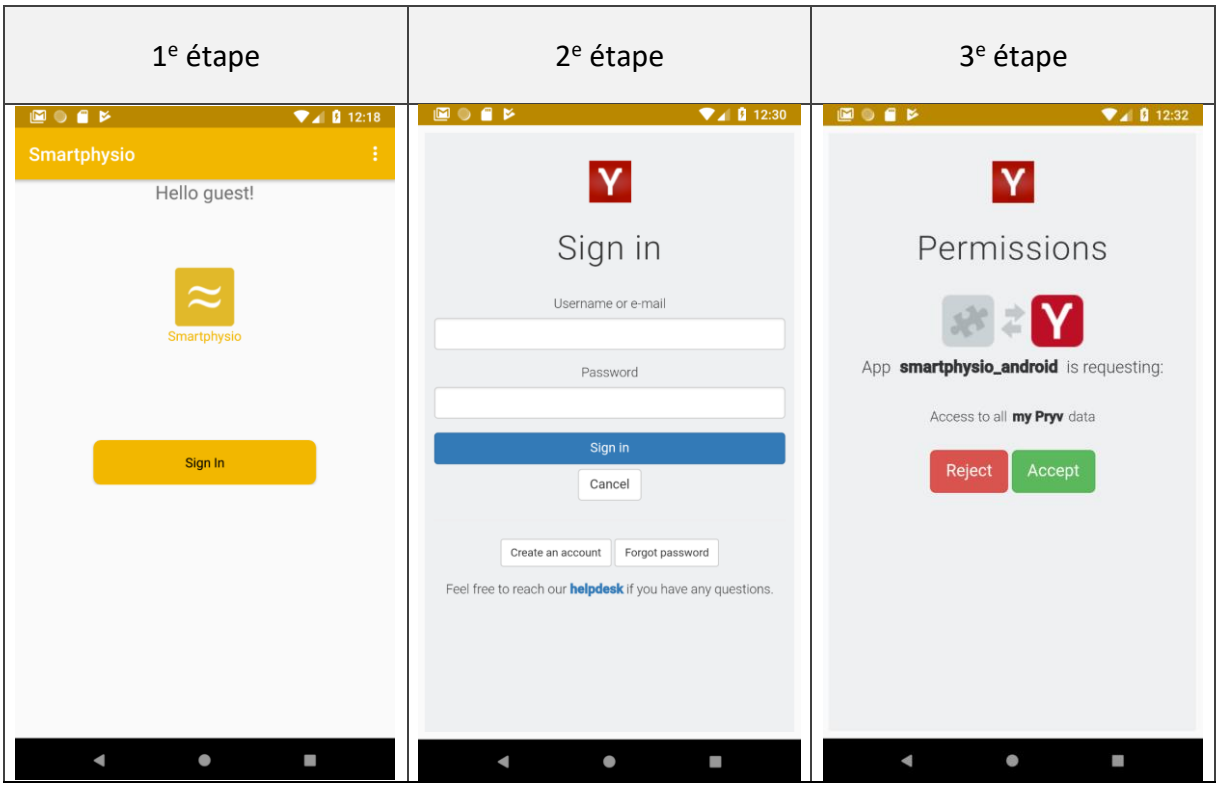


Figure 33 : Connexion depuis le smartphone sur Pryv

Si nous avons utilisé la classe « LoginActivity » fournie par Pryv telle quelle, nous avons pourtant ajouté une méthode qui crée automatiquement les Streams nécessaires au fonctionnement de l'application mobile (code disponible en Annexe X).

[16] WebView : est une fenêtre de type web qui est ouverte nativement dans l'application

9.3. Connexion avec la smartwatch (US : 82)

Pryv ne fournit pas d'exemple fonctionnant avec Wear OS. Nous déployons alors la même solution que pour le smartphone.

- Utilisation de la classe « LoginActivity » et « Credentials »
- Création d'un « layout » pour la gestion de la WebView

Contre toute attente nous obtenons un message d'erreur à propos de la WebView :

```
android.view.InflateException: Binary XML file line #14:
Error inflating class android.webkit.WebView…
```

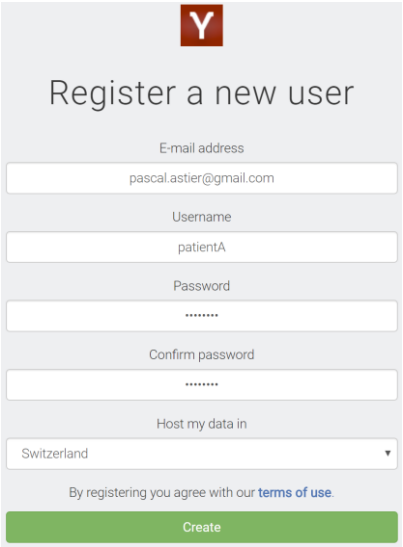
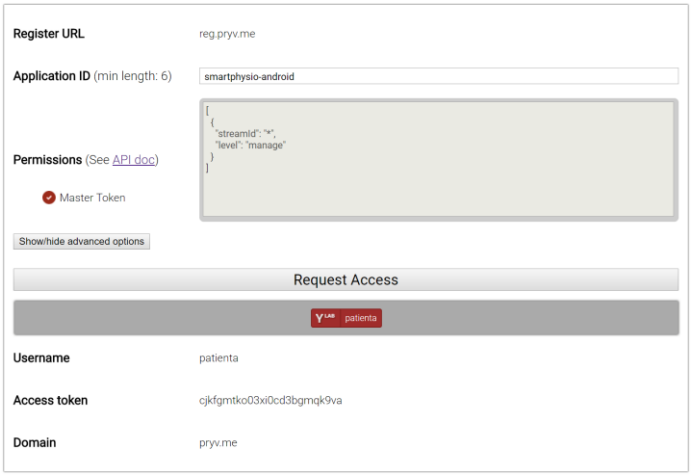
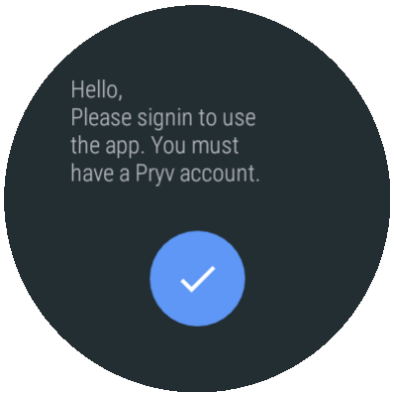
Nous trouvons sur le site de « developer.android.com » la cause du problème. Qui indique que : « The WebView.getCurrentWebViewPackage() method can return null if the device hasn't been set up correctly. It also returns null if you're running your app on a device that doesn't support WebView, such as a Wear OS device » (Google, s.d.).

Ce nouveau problème technique est rapidement transmis à Fabien Dubosson, qui propose de contourner le problème en saisissant sur la montre le couple « username & token ». Le résultat est concluant, et techniquement nous arrivons à poster des niveaux de douleurs dans Pryv. Toutefois, en termes d'expérience utilisateur, la saisie d'un token de 25 digits n'est pas idéale (exemple de token : « cjj689m010q2l0cd3q2h6lr54 »). Compte tenu du temps restant, nous approuvons néanmoins cette méthode.

- D'une part, car l'objectif principal de ce projet est de fournir un prototype fonctionnel.
- Deuxièmement, la saisie de couple « username & token » n'est effectuée qu'une seule fois par patient
- Finalement, on peut imaginer que Pryv propose, à terme, d'autres solutions de connexions à leurs services.

9.3.1. Premier processus de connexion avec une smartwatch

A noter que l'API de Pryv ne permet pas la création de nouveaux utilisateurs. Il faut impérativement passer par leur outil de connexion (cf. Figure 33 : Connexion depuis le smartphone sur Pryv) ou directement sur Pryv.me.

Etape N°1	Création d'un compte sur « pryv.me »	
Etape N°2	Obtenir un token. Pryv fournit un service de génération de token disponible à l'adresse : http://pryv.github.io/app-web-access/ Procédure en 6 étapes (voir annexe XII).	
Etape N°3	Démarrer l'aplication « Smartphysio » sur la smartwatch. Tant que le patient ne s'est pas identifié sur la smartwatch, il est invité à le faire.	

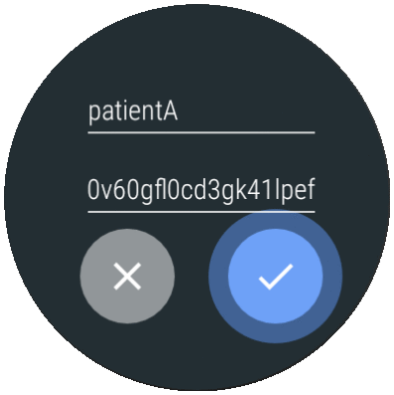
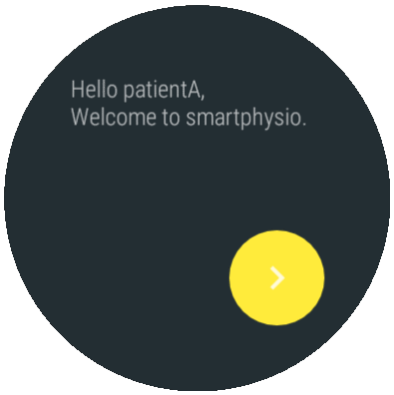
Etape N°4	On saisit ensuite les informations de connexion obtenues à l'étape N°2. Le « username » et le « token » sont automatiquement sauvegardés dans la mémoire interne de la smartwatch	
Etape N°5	Les identifiants sont enregistrés. Le patient peut démarrer la saisie des niveaux de douleurs en cliquant sur suivant A noter que cet écran n'apparaît plus lors des prochains lancements de l'application.	

Tableau 12 : Flux de connexion à Pryv avec la smartwatch

9.3.2. Conclusion sur le processus de connexion d'une smartwatch

Si le nombre d'étapes n'est pas compressible, l'étape 2 qui nécessite de naviguer sur un autre service rend l'opération relativement complexe. Notamment car l'outil de génération de token n'est pas adapté à un utilisateur lambda. Nous pensons automatiser cette étape dans le dernier sprint.

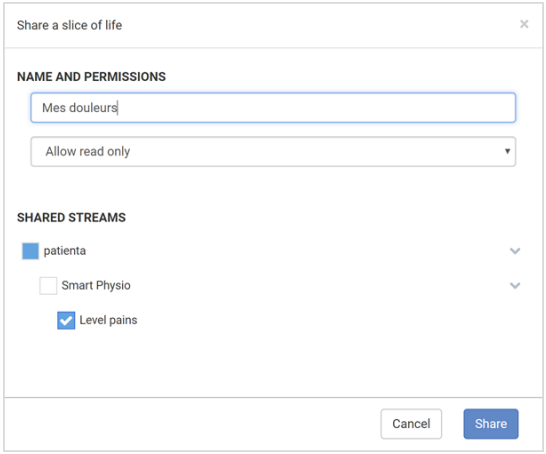
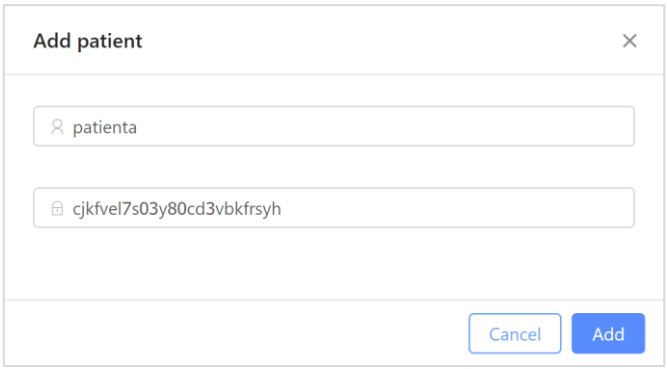
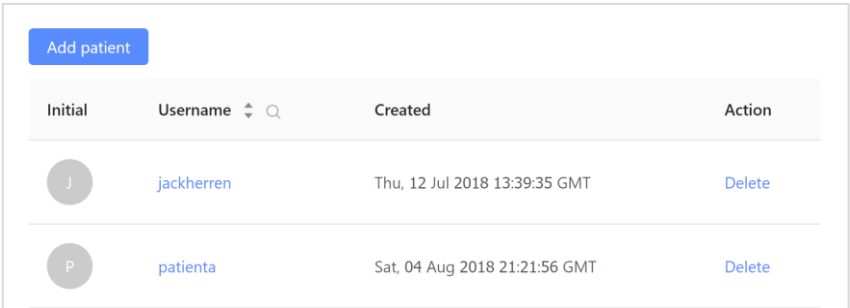
9.4. Ajout de patient dans la WebApp (US : 75)

C'est une phase importante de notre projet. Le physiothérapeute doit pouvoir ajouter des patients, afin d'analyser les fluctuations de leurs niveaux de douleurs. Nous savons comment stocker les patients dans notre structure Pryv. Cependant, comme pour la smartwatch, nous devons obtenir un couple « username & token ». Dans ce cas deux possibilités sont analysées :

- Utiliser le même outil de génération des token que pour la smartwatch
- Ou exploiter la notion de partage de Pryv

9.4.1. Partager un « slice » dans Pryv

Pryv fournit une application web qui permet manipuler les Streams et les Events. Depuis cette interface, il est possible d'envoyer un lien qui permet à quiconque le possédant d'accéder aux Streams partagés. Sauf, dans le cas où l'utilisateur décide de ne plus partager cette ressource.

Etape N°1	Partager un ou plusieurs Streams	
Etape N°2	Cela nous permet de récupérer le couple « username & token » `https://patienta.pryv.me/#/sharings/cjkfvel7s03y80cd3vbkfrsyh` **Username** **Token**	
Etape N°3	Dans la Web App de smartphysio, on peut ajouter le nouveau patient en saisissant ses identifiants.	
Résultat N°1	Résultat dans l'interface utilisateur	

Résultat N°2	Résultat dans la base de donnée Pryv	{
		"streamId": "patients",
		"type": "note/text",
		"content": "patienta",
		"clientData": {
		"smartphysio_admin": {
		"auth": "cjkfvel7s03y80cd3vbkfrsyh"
		}
		},
		}

Tableau 13 : Ajout d'un patient dans la WebApp

9.4.2. Comparaison des deux méthodes d'obtention des tokens

Méthode 1 : Outil de génération de token	
Avantages	Inconvénients
• Création de token « master[17] » • Création de token sur des Streams inexistants	• Procédure complexe en six sous-étapes • On doit saisir l'ensemble des permissions en format JSON

Tableau 14 : Comparaison des méthodes de génération des tokens (Méthode 1)

Méthode 2 : Partager un « slice »	
Avantages	Inconvénients
• Procédure simple • Plusieurs possibilités pour partager le lien généré (email, Facebook, Twitter, copier-coller)	• Les Streams doivent exister[18] • Pas de token « master »

Tableau 15 : Comparaison des méthodes de génération des tokens (Méthode 2)

Finalement, on constate que ces deux méthodes sont complémentaires.

[17] Un token master est un token qui donne un accès total à une application donnée.

[18] C'est la raison pour laquelle le partage des « slice » ne peut pas être utilisé pour la smartwatch. Car ce sont les applications mobiles qui créent la structure des Streams.

9.5. Fin du sprint N°3

L'apprentissage de Pryv, puis le problème de WebView, ajouté au fait d'avoir neuf story points supplémentaires nous ont quelque peu pénalisés. En effet la dernière US de ce sprint n'a pas été développée (US : 110, cinq SP). Cependant l'implémentation de Pryv est un succès. Nous sommes capables de créer des niveaux de douleurs depuis la smartwatch et de les lire via le smartphone et la Web App. Notre prototype est fonctionnel.

10 Sprint N°4 : Consolidation et ergonomie

Ce sprint est consacré à la WebApp et la smartwatch. Les deux périphériques principaux de notre projet. Le but est de développer les use cases suivants :

- Pour la Web App
  - Trier, rechercher les patients dans une liste
  - Voir sur un graphique les douleurs saisies par le patient
  - Création et gestion des activités
- Pour la smartwatch
  - Choisir une activité
  - Création d'une liste d'activités favorites

Nous profitons de ce sprint pour consolider l'architecture de la Web App. Avec une librairie graphique et Redux. Redux est une bibliothèque qui a pour fonction de centraliser les données dans un « magasin ». Dès qu'une modification est apportée dans le magasin elle est valable pour l'entier de l'application (Observer pattern).

10.1. Gestion des activités sur la smartwatch

Il y a deux types de liste d'activités :

- La liste « publique » : qui est gérée par le physiothérapeute (CRUD)
- La liste « favoris » : qui est stockée dans la mémoire interne de la smartwatch

Le patient doit pouvoir basculer d'une liste à l'autre. Puis chaque fois qu'il utilise une activité publique, elle est automatiquement ajoutée à la liste des activités favorites.

10.1.1. Récupération des activités publiques

Sur notre modèle de smartwatch, le temps de récupération des activités (environ une dizaine) prend dans certains cas de quatre à six secondes. Pour pallier ce problème nous chargeons les activités publiques dans un thread à chaque fois que le patient démarre l'application. Ceci pour autant que la smartwatch soit connectée à internet. Sinon seules les activités favorites seront disponibles.

Ci-dessous un extrait de code pour la récupération des activités publiques. A noter encore que pour y accéder, il est nécessaire d'avoir des identifiants « username & token ». En effet, la liste des activités publiques est stockée dans un Stream du physiothérapeute. Pour que tous les patients puissent y accéder, un token spécifique a été créé. Celui-ci est disponible dans notre classe « Global ». Nous abordons ce sujet dans le chapitre 12.3 (cf. p. 72).

```
public static void getPublicActivitiesOnPryv(final Context context) {
    // Initiate new connection to Pryv with connected account
    final Connection publicConnection = new
          Connection(Global.PUBLIC_ACTIVITY_USERNAME, Global.PUBLIC_ACTIVITY_AUTH,
          Global.DOMAIN);
    new Thread() {
        public void run() {
            try {
                // get all public activities
                List<Event> publicActivitiesOnPryv;
                Filter filter = new Filter().setParentId("activities");
                //limit = 0 > get all
                filter.setLimit(0);
                publicActivitiesOnPryv = publicConnection.events.get(filter);
                if (publicActivitiesOnPryv.size() > 0) {
                    //set global variable to keep it
                    Global.activitiesOnline = new
                                    ArrayList<>(publicActivitiesOnPryv);
                }
            } catch (IOException | ApiException e) {
                e.printStackTrace();
            }
        }
    }.start();
}
```

10.1.2. Gestion des activités favorites dans la smartwatch

Puis les favoris sont sauvegardés dans la mémoire interne de la smartwatch en format JSON et convertis en « Map » pour être manipulés. La clé de la « Map » étant l'identifiant de l'activité, nous garantissons ainsi l'unicité dans la liste de favoris.

```
public static Map<String, ClientData> loadFavorites(Context context) {
    // used for retrieving arraylist from json formatted string
    SharedPreferences settings;
    Map<String, ClientData> favorites;
    settings =
            context.getSharedPreferences(SHARED_PREFERENCES_FILE_CLIENT_DATA_LIST,
            Context.MODE_PRIVATE);
    //if patient has favorites
    if (settings.contains(SHARED_PREFERENCES_FAVORITES)) {
        String jsonFavorites = settings.getString(SHARED_PREFERENCES_FAVORITES,
        null);
        // JSON to Map
        Gson gson = new Gson();
        Type type = new TypeToken<Map<String, ClientData>>() {  }.getType();
        favorites = gson.fromJson(jsonFavorites, type);
        return favorites;
    }
    return null;
}
```

10.2. Liste des patients dans la WebApp

Trier et chercher des patients dans une liste fait partie des fonctionnalités demandées. Pour nous aider dans cette tâche, nous décidons d'utiliser une librairie et nous en testons deux parmi les plus populaires :

- Antd
- Material UI

Après quelques tests, l'ergonomie, la documentation et la facilitation d'utilisation de Antd nous motivent dans ce choix. De plus le composant table de Antd propose des exemples similaires à notre besoin de recherche.

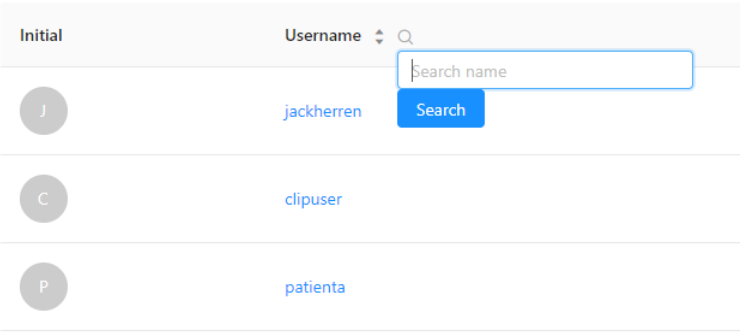


Figure 34 : Web App chercher un patient

10.3. Affichage des statistiques par patient

Pour ouvrir les statistiques d'un patient donné, nous devons lancer une requête qui contient notre couple « username & token » et construire l'url correspondante. Nous les passons en paramètre à notre fonction « api.pains ».

```
// Events > Pains
// connection : username & token (auth)
export const fetchPains = connection => dispatch =>
   api.pains
      .fetchAll(
         {
            username: connection.username,
            data: {
               headers: {'Authorization': connection.auth},
               params: {
                  limit: 0
               },
            }
         })
      .then(res => dispatch(painsFetched(res.data.events)));
```

Tous les appels à l'API de Pryv sont regroupés dans un seul fichier : ApiRequest.js qui est disponible en annexe XI. Ci-dessous le code pour la récupération des niveaux de douleurs. L'url est conçue dynamiquement ce qui permet d'accéder à tous les Events du patient. Le token contient la permission d'accéder au Stream des niveaux de douleurs (cf. 9.4.1 Partager un « slice » dans Pryv).

```
// Events > Pains
pains: {
   fetchAll: filter => axios.get("https://" + filter.username + ".pryv.me/events",
      filter.data).then(res => res),
}
```

On obtient un graphique qui affiche toutes les saisies effectuées par le patient. Nous avons utilisé Highcharts, la même librairie que Pryv pour générer le graphique. Un module de Highcharts est disponible pour une utilisation optimale avec React. Toutefois la librairie « highcharts-react-official » nous a posé passablement de problèmes. En particulier au niveau de l'affichage. Dans certains cas, seule la « timeline » (en bleu sous le graphique) était visible.

D'ailleurs les exemples fournis par highcharts et testables sur « CodePen ou jsfiddle »[19] posaient les mêmes types de problèmes. Finalement, nous avons trouvé une autre librairie « react-highcharts » qui fonctionnait selon nos attentes.

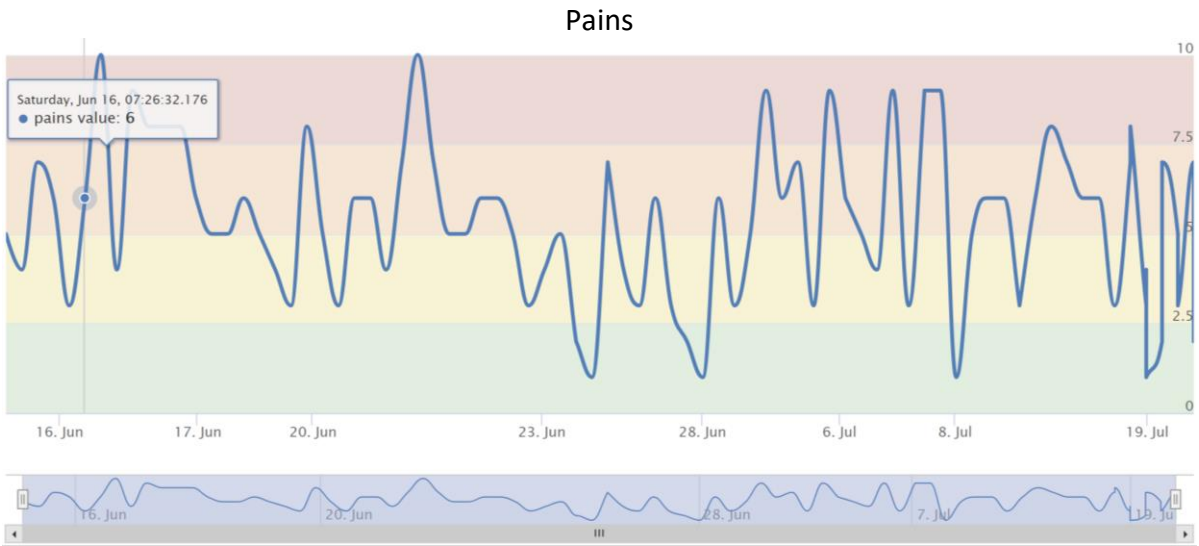


Figure 35 : Web App affichage des niveaux de douleurs

[19] Ce sont des outils en ligne qui permettent de comprendre en simulant des extraits de code.

10.4. Création des activités

Selon les pathologies traitées par le physiothérapeute ainsi que le niveau de détails désiré (cf. Tableau 16 : Niveau de détails des listes d'activités), le physiothérapeute gère les activités comme il le souhaite. L'idée de les classer par catégories permet d'orienter le patient lors du choix. A noter que le patient n'a pas la possibilité d'intervenir sur le contenu de la liste publique.

Liste des activités détaillées	Liste des activités plus générales
• Quotidiennes ○ S'asseoir ○ S'asseoir sur une chaise ○ S'asseoir sur un tabouret haut ○ Se lever ○ Se lever d'une chaise ○ Se lever d'un tabouret haut	• Quotidiennes ○ S'asseoir ○ Se lever

Tableau 16 : Niveau de détails des listes d'activités

Nous avons opté pour une représentation hiérarchique des activités dans leurs catégories respectives. Depuis cette interface le physiothérapeute peut : créer, renommer et supprimer des catégories (suppression uniquement si la catégorie n'a pas d'activité). Les activités peuvent être créées et supprimées. Cependant Il n'est pas possible de les renommer.

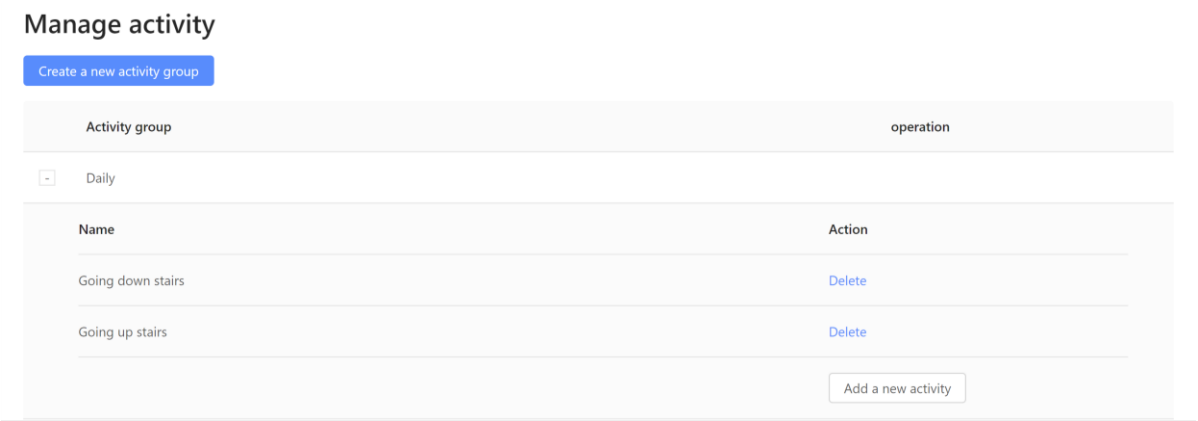


Figure 36 : Web App affichage des niveaux des activités

10.5. Consolidation et utilisation de Redux

React est structuré en composants. Un composant contient deux types de données (Belahcen, 2017) :

- **Le « state » :** est local au composant car il définit son comportement. On peut le modifier (accès en lecture/écriture) afin de changer l'état du composant.
- **Les « props » :** sont des propriétés que l'on passe d'un composant vers un autre. Ces propriétés ne sont pas modifiables.

Pourtant, un des défauts de React réside dans la difficulté à faire communiquer tous ces composants.

Dans cet exemple, si le « composant 3 » modifie une donnée (un ajout de patient, ou effacer une activité…) il ne peut transmettre cette modification qu'à ses enfants (4 et 5). Cependant les composants (1 et 2) ne sont pas informés.

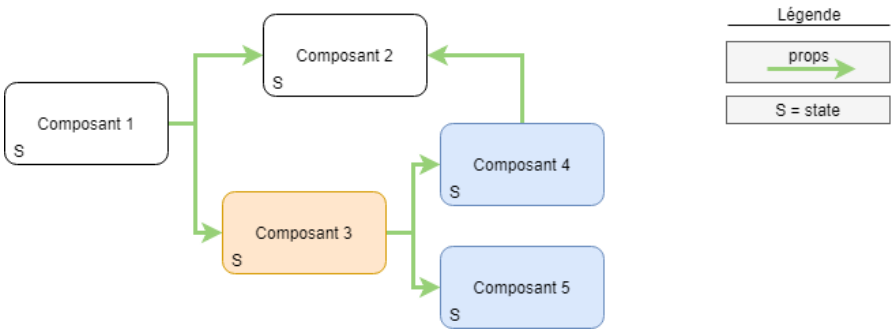


Figure 37 : Fonctionnement des « props » et « state » dans React

Redux permet de pallier cette problématique. Les composants s'abonnent au magasin de Redux et lorsqu'un composant modifie les données du magasin (dispatch), Redux notifie les abonnés qui se mettent à jour (Design pattern : Observer)[20].

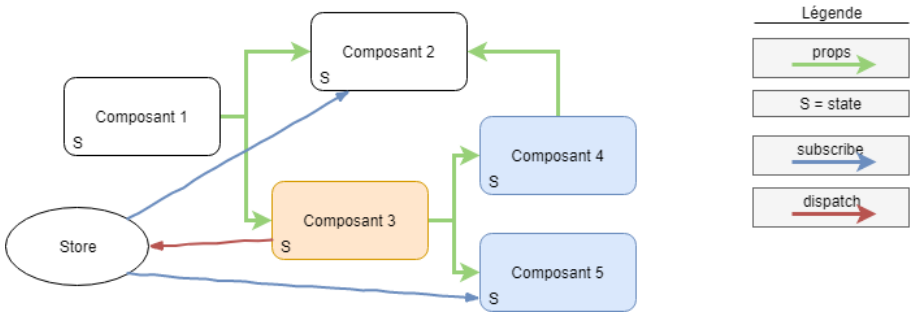
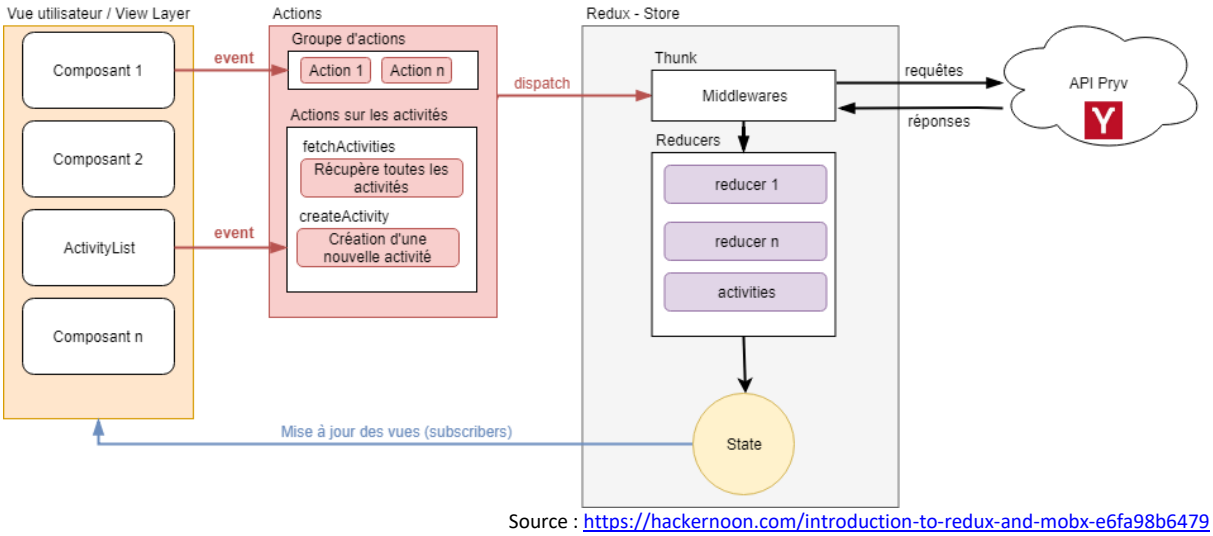


Figure 38 : Utilisation de React et de Redux

10.6. Architecture de notre application React

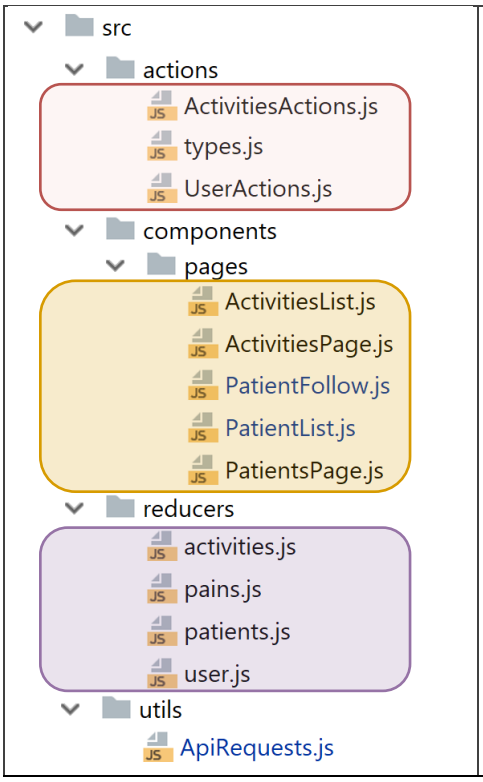
La mise en place de Redux oblige une certaine structure, qui offre l'avantage de bien séparer le code. Nous avons créé des dossiers qui correspondent au schéma de la structure ci-dessous.



Source : https://hackernoon.com/introduction-to-redux-and-mobx-e6fa98b6479

Figure 39 : Architecture de React + Redux

[20] Les patrons de conceptions sont des modèles de bonnes pratiques qui ont été définis par des développeurs afin de répondre à certaines problématiques dans le développement de logiciels. Le pattern observer est de type structurel et permet de notifier un ou plusieurs abonnés d'un changement afin qu'ils se mettent à jour.



Dans cette arborescence simplifiée, on trouve une structure de fichiers définie par l'architecture de React + Redux.

On distingue également le fichier « ApiRequest », qui regroupe tous les appels vers l'API de Pryv.

Figure 40 : Utilisation de React + Redux, séparation du code

10.7. Sprint Review N°4

Toutes les US sont validées. Toutefois, le processus d'authentification pour la smartwatch et l'ajout de patient dans la Web App doivent être optimisés. De plus, la terminologie « Add a new patient » n'est pas idéale, le terme « Follow a patient » convient mieux à la fonctionnalité.

11 Sprint N°5 : Finalisation du prototype

Dans cette dernière étape officielle, sprint court de 5 jours mais complet avec un « sprint review », le 19.07.2018, nous traitons trois sujets importants :

- Améliorer l'obtention des tokens :
  - Pour connecter la smartwatch
  - Pour suivre un patient dans la Web App
- Afficher dans le graphique des courbes qui permettent au physiothérapeute de mieux apprécier les tendances.
- Ajouter des informations complémentaires depuis le smartphone

11.1. Améliorer l'obtention des tokens (US : 155)

Le contexte est le suivant : nous devons générer depuis la Web App (Liste des patients) deux types de tokens :

- Un pour le suivi d'un patient : « username & token » sont sauvegardés (cf. 8.3.2), accès en lecture uniquement.
- Un pour connecter la smartwatch : ces identifiants ne sont pas sauvegardés, accès en mode « manage ».

Déroulement du processus de récupération du token de suivi :

1. Cliquer le bouton « Follow a patient »
2. Ouverture du « Popup » Pryv pour s'identifier, nom d'utilisateur et mot de passe
3. Accepter les permissions requises (lire le Stream des niveaux de douleurs)
4. Une fenêtre de confirmation affiche les identifiants, cliquer sur « Follow » pour ajouter le patient dans la liste.

Déroulement du processus de récupération du token de la smartwatch :

1. Cliquer le bouton « Show a watch token »
2. Ouverture du « Popup » Pryv pour s'identifier, nom d'utilisateur et mot de passe
3. Accepter les permissions requises (lire le Stream des niveaux de douleurs)

4. Une fenêtre de confirmation affiche les identifiants, cliquer sur « Close » pour terminer.

Nous constatons que les deux processus décrits ci-dessus sont très similaires. Nous développons donc un composant réutilisable, nommé « PatientFollow ». Nous lui passons les propriétés nécessaires qui permettent la gestion de son comportement.

```
renderShowToken = () => {
  return (
    <PatientFollow key={0} btnLabel={'Show watch token'} onAction={'SHOW_TOKEN'}/>
   )
};
renderFollowPatientByLogin = () => {
  return (
    <PatientFollow key={1} btnLabel={'Add patient'} onAction={'FOLLOW_LOGIN'}/>
   )
};
```

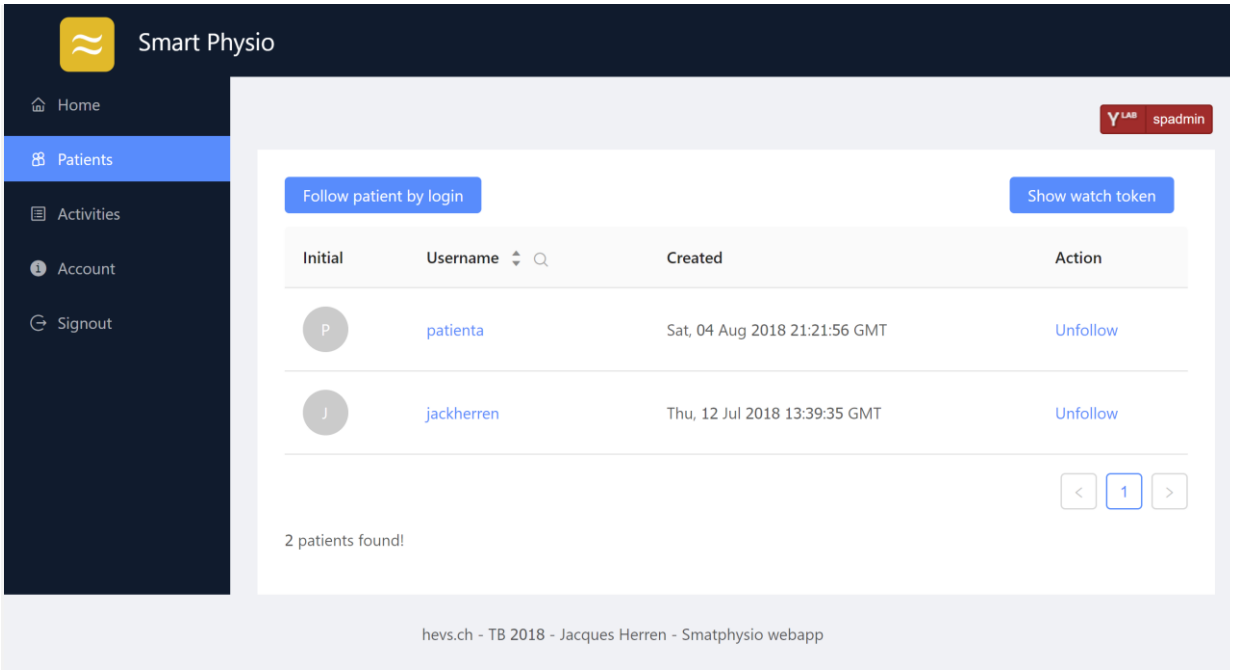


Figure 41 : Liste de patients et affichage des deux boutons générateur de tokens

Le composant « PatientFollow » gère l'affichage du « Popup » d'authentification de Pryv et l'appel de la fenêtre de confirmation. Grâce à la propriété « onAction » on connaît la source de l'événement et on affiche les informations correspondantes.

Dans l'extrait de code ci-dessous, nous pouvons distinguer la propriété « title » qui varie selon « onAction »

```
generateModal = (visible) => {
   switch (this.onAction) {
      case 'SHOW_TOKEN' :
         return (<WrappedPatientCreateForm
            visible={visible}
            title='Show token'   ❶
            username={this.state.patientAccess.username} ❷
            auth={this.state.patientAccess.token} ❸
            mode={'READ'} ❹
         />)

      case 'FOLLOW_LOGIN' :
         return (<WrappedPatientCreateForm
            visible={visible}
            title='Add patient'  ❶
            username={this.state.patientAccess.username} ❷
            auth={this.state.patientAccess.token} ❸
            mode={'EDIT'} ❹
         />);
   }
};
```

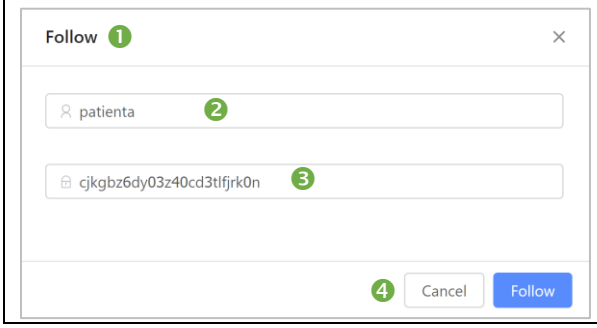
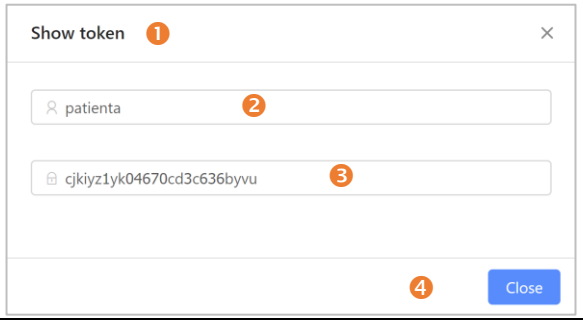
Identifiants pour le suivi du patient	Identifiants pour la smartwatch
	

Tableau 17 : Comparer les fenêtres de confirmation des tokens selon « onAction »

11.2. Moyenne mobile

On parle aussi de moyenne glissante. Cette méthode de calcul est une solution particulièrement bien adaptée aux séries temporelles (educatim, s.d.) car elle permet de lisser les valeurs extrêmes. On l'appelle « mobile » car elle est recalculée en continu sur une série de mesures successives.

Pour illustrer cette moyenne mobile, voici les saisies d'un patient sur une journée :

Heure de la saisie	Niveau de douleur	Moyenne mobile sur 3 valeurs
00h00	4	
07h00	3	
09h00	5	4.00
10h00	8	5.33
11h00	6	6.33
13h00	4	6.00
15h00	8	6.00
15h30	7	6.33
16h00	6	7.00
17h00	4	5.67
21h00	2	4.00
23h00	2	2.67

Tableau 18 : Exemple de calcule d'une moyenne mobile

Dans cet exemple, nous avons choisi arbitrairement de travailler avec **trois** valeurs successives. Le calcul est le suivant : (5 + 8 + 6) / **3** = **6.33.**

L'enjeu est de choisir la plage des valeurs successives. Plus on l'augmente plus la courbe à tendance à se lisser. Ci-dessous la moyenne mobile dans Smartphysio qui est affichée en « noir ». Elle est calculé sur 3 valeurs successives.

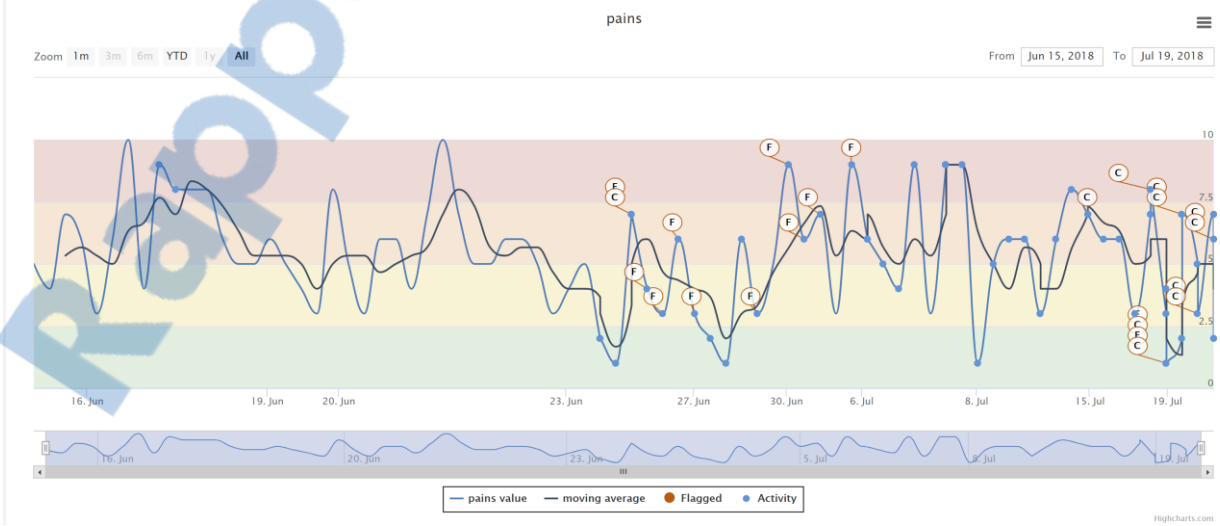


Figure 42 : Affichage de la courbe « moving average » en noir

12 Améliorations

12.1. Connecté à internet ?

Un des avantages majeurs de Android couplé à Firebase concerne la gestion des « callbacks ». En effet, si la smartwatch n'est pas connectée à internet, le patient peut l'utiliser pour saisir des données sans aucune restriction. Dès que la smartwatch retrouve une connexion elle passera les écritures dans la base de données. Avec Pryv ce comportement n'existe pas, il faudrait le développer. Par exemple en sauvant les saisies dans la mémoire interne de la smartwatch jusqu'à la prochaine connexion.

12.2. Multi langue

Dès le début du projet la notion de multi langue n'était pas une priorité. Dans Android la gestion des langues s'effectue par un module spécifique, idem pour React où il y a plusieurs librairies qui facilitent cette gestion. La partie technique se situe au niveau des catégories et des activités. Le physiothérapeute devrait saisir chacune d'elles dans les 'x' langues de l'application.

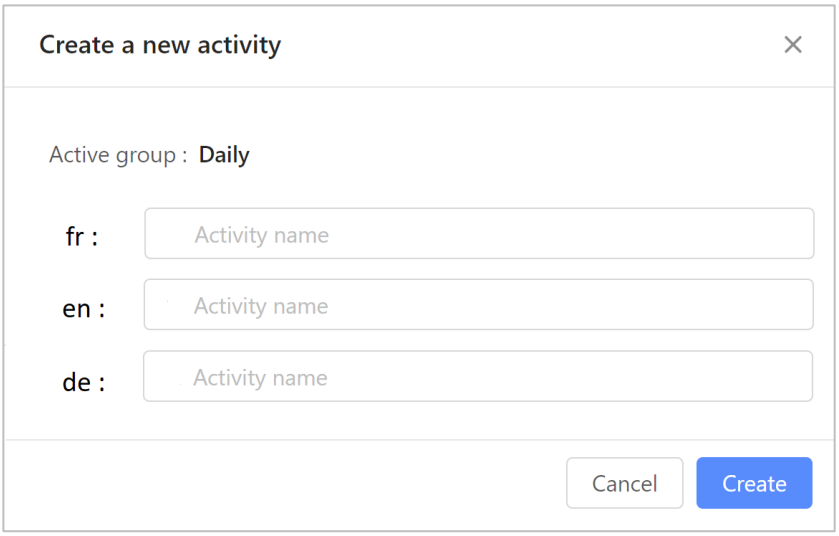


Figure 43 : Multi langue dans les activités

12.3. Gestion publique des catégories et des activités

Actuellement les catégories et les activités sont stockées dans le Stream « activities » du physiothérapeute « spadmin ». Pour accéder à ces données depuis la smartwatch, nous avons généré un couple « username & token » avec des permissions de lecture. Ces identifiants sont définis en « dur » dans la classe Global de « smartphysiolibrary ».

```
public static final String PUBLIC_ACTIVITY_USERNAME = "spadmin";
public static final String PUBLIC_ACTIVITY_AUTH = "cjj6vct480q6y0cd35r5fwon7";
```

Cela signifie qu'il n'est pas possible de créer un autre compte physiothérapeute (par exemple un autre cabinet) et que chacun gère ses activités comme il le souhaite.

Deux possibilités sont envisageables :

- Créer un Stream commun à tous les cabinets, lesquels collaborent pour alimenter la liste
- Lors de la mise en service de la smartwatch pour le patient, saisir deux identifiants :
    - un pour la gestion des niveaux de douleurs (déjà existant)
    - un pour l'accès aux activités publiques du cabinet

12.4. Obtention des tokens « follow & show watch token »

Dans notre version, le patient doit saisir deux fois de suite son nom d'utilisateur et son mot de passe. Une première fois pour être dans la liste de patients suivis, et une deuxième pour paramétrer sa smartwatch.

Nous pourrions effectuer une seule saisie avec une seule fenêtre de confirmation dans laquelle on afficherait tous les identifiants.

12.5. Moyenne mobile, choix des valeurs successives

Lors du calcul de la moyenne mobile, nous prenons en compte une plage de trois valeurs successives. Rendre cette valeur modifiable dynamiquement par le physiothérapeute permettrait de lisser davantage la courbe.

12.6. Exportation depuis le graphique avec les données de texte

Dans le graphique généré grâce à Highcharts, un module d'exportation est disponible. Il est possible d'exporter vers plusieurs formats différents (CSV ; Excel). Toutefois, les activités ne sont pas exportées, idem pour les « flag » et leurs commentaires associés. Il faudrait étudier les options d'exportation ou prévoir l'utilisation d'un autre outil.

12.7. Smartphone iOS

La smartwatch est autonome, c'est un avantage certain. Toutefois, si le patient n'est pas équipé d'un smartphone Android il ne peut pas ajouter de compléments ni modifier les saisies précédentes. Le développement du smartphone pourrait être fait avec React native et ainsi être utilisable par les utilisateurs Android et Apple.

13 Smartphysio en libre accès

Dans les annexes XIII & XIV nous avons mis les procédures permettant d'utiliser les applications développées durant ce travail de Bachelor.

Les applications mobiles sont publiées en version alpha sur « Google Play ». La Web App est disponible à l'adresse, suivante : http://212.47.240.226/ jusqu'à fin septembre 2018.

Conclusion

L'utilisation de la smartwatch est particulièrement bien adaptée à ce genre de besoins. Cependant l'ergonomie est une composante fondamentale dans ce type d'interface. Tout d'abord parce que l'écran est relativement petit, mais aussi parce qu'effectuer une tâche doit être aussi simple que possible.

Finalement, malgré deux limitations majeures de Wear OS, qui sont :

- Pas d'authentification possible à Firebase en mode autonome
- « Web view » non prise en charge pour la connexion à Pryv,

Nous avons développé une application fonctionnelle et aussi ergonomique. Entre deux et quatre « clics » sont nécessaires pour sauvegarder un niveau de douleur.

Du côté du smartphone, nous n'avons pas rencontré de problèmes particuliers. C'était également le périphérique sur lequel il y avait le moins de travail. Actuellement, si le patient est sous Android il peut grâce à son smartphone modifier un niveau de douleur et y ajouter un complément sous forme de texte.

La troisième application est dédiée au physiothérapeute. Elle est développée en React (une bibliothèque Javascript). Elle permet de gérer des patients, d'analyser des statistiques et de créer des listes d'activités. La majorité des problèmes rencontrés dans ce développement étaient probablement liés à l'apprentissage des subtilités de Javascript.

Ces trois applications « front-end » sont liées à un service de base de données cloud nommé Pryv. C'est un produit Suisse qui stocke les données en Suisse (mais également en France et aux USA). La documentation disponible en ligne est très complète. De plus la protection des données est l'une de leurs priorités en particulier dans le domaine médical.

A titre personnel, ce travail m'a permis de regrouper toutes (ou presque) les connaissances acquises durant cette formation de quatre ans. Cela va de la gestion de projet à l'industrialisation en passant par les algorithmes, l'expérience utilisateur ou encore la communication.

Références

Arnaud. (2018, 07 21). *Gérez votre projet avec une équipe Scrum*. Récupéré sur openclassrooms: https://openclassrooms.com/fr/courses/4511226-gerez-votre-projet-avec-une-equipe-scrum

Belahcen, M. (2017, 07 19). *React par la pratique – 3 : Utiliser le State et les props*. Récupéré sur http://apprendre-le-js.com: http://apprendre-le-js.com/react-pratique-3-utiliser-state-props/

educatim. (s.d.). *Moyenne glissante ou moyenne mobile*. Récupéré sur educatim.fr: http://www.educatim.fr/tq/co/Module_TQ_web/co/moyenne_glissante.html

Elmer, V. (2015, 06 08). *L'évaluation de la douleur*. Récupéré sur infirmiers: https://www.infirmiers.com/etudiants-en-ifsi/cours/l-evaluation-de-la-douleur.html

Garuda, G. (2017, 08 23). *Introduction to Redux and Mobx*. Récupéré sur hackernoon.com: https://hackernoon.com/introduction-to-redux-and-mobx-e6fa98b6479

Google. (2016, Décembre). *Terms and conditions*. Récupéré sur Developer Android: https://developer.android.com/studio/terms

Google. (2018, Mars). *Wear OS by Google– Creative vision*. Récupéré sur Design Guidelines: https://designguidelines.withgoogle.com/wearos/wear-os-by-google/creative-vision.html#

Google. (s.d.). *Managing Webview*. Récupéré sur developer.android.com: https://developer.android.com/guide/webapps/managing-webview

GrafiKart. (s.d.). *Comprendre Webpack*. Récupéré sur GrafiKart: https://www.grafikart.fr/formations/webpack

Kerpelman, T. (2017, 10 03). *Cloud Firestore for Realtime Database Developers*. Récupéré sur firebase.googleblog.com: https://firebase.googleblog.com/2017/10/cloud-firestore-for-rtdb-developers.html

Lake, I. (2017, 01 16). *Wear OS Developers*. Récupéré sur plus.google.com: https://plus.google.com/+LewisMcGeary/posts/BQzGE2G4RX6

Larson, E., & Larson, R. (2004). *Use cases*. Récupéré sur pmi.org: https://www.pmi.org/learning/library/use-cases-project-manager-know-8262

Maring, J. (2018, mars 19). *Android Wear needs way more than just a Wear OS rebrand*. Récupéré sur androidcentral: https://www.androidcentral.com/google-needs-more-just-re-brand-revive-android-wear

Office cantonal vaudois du registre du commerce. (s.d.). *Extrait internet*. Récupéré sur www.rc.vd.ch: https://www.rc2.vd.ch/registres/hrcintapp-pub/companyReport.action?rcentId=78971163000000055031025&lang=FR&showHeader=false

Préposé fédéral à la protection des données et à la transparence (PFPDT). (s.d.). *Explications concernant l'informatique en nuage (cloud computing)*. Récupéré sur admin.ch: https://www.edoeb.admin.ch/edoeb/fr/home/protection-des-donnees/Internet_und_Computer/cloud-computing/explications-concernant-l-informatique-en-nuage--cloud-computing.html

Pryv. (2017, 11 28). *App java examples - BasicExample*. Récupéré sur github.com: https://github.com/pryv/app-java-examples/tree/master/BasicExample

Pryv. (s.d.). *Getting started - Authorize your app*. Récupéré sur api.pryv.com: http://api.pryv.com/getting-started/javascript/#authorize-your-app

Pryv. (s.d.). *Getting started - Authorize your application*. Récupéré sur api.pryv.com: http://api.pryv.com/getting-started/pryvme/#authorize-your-application

Pryv. (s.d.). *Getting started - Data Modelling Tips*. Récupéré sur api.priv.com: http://api.pryv.com/getting-started/pryvme/#data-modelling-tips

Pryv. (s.d.). *Pryv product*. Récupéré sur pryv.com: https://pryv.com/product/

Seydtaghia, A. (2018, 05 22). *Données personnelles: ce qui va changer en Suisse aussi avec le RGPD*. Récupéré sur Le Temps: https://www.letemps.ch/economie/donnees-personnelles-va-changer-suisse-rgpd

Shanklin, W. (2017, 05 17). *2017 Smartwatch Comparison Guide*. Récupéré sur newatlas: https://newatlas.com/smartwatch-comparison-2017-specs/49553/

Vignier, N., & Gougeon, E. (2016). *Mesurer la douleur*. Récupéré sur brainberry: http://www.brainberry.fr/mesure-douleur/

Annexe I : Product Backlog

US Nr.	Theme	As an/a …	I want to …	in order to	Acceptance criteria	Priority	Status	Story Points	moscow	Initial Sprint	Current Sprint	US accepted (done done)
10	Manage project	bachelor	study android watch capabilities	get a great user experience		**100**	●	2	M		0	
20	Manage project	bachelor	draw mockups of all frontend applications	be clear and simple		**98**	●	5	M		0	22/03/18
30	Manage project	bachelor	make a choice between firebase and cloud firestore	find the best solution for this project		**96**	●	2	M		0	19/04/18
40	Manage project	bachelor	add tests datas in the cloud from the smartwatch and read them from others frontends	validate the architecture	Read datas from SP and WebApp	**94**	●	5	M		1	19/04/18
50	Smartwatch	Patient	choose my pain level in a list	save it		**93**	●	5	M		1	19/04/18
60	Manage project	bachelor	configure project environment	to manage and share as a real project	VCS & CI Configured	**92**	●	5	M		1	19/04/18
70	WebApp	bachelor	Create a physio account	connect to the web app		**90**	●	2	M	2	3	06/06/18
~~**80**~~	~~Smartwatch~~	~~Patient~~	~~connect a smartwatch to my firebase account~~	~~get access from firebase~~		~~**88**~~	●	~~2~~	~~M~~		~~2~~	
90	Smartphone	Patient	read all input in a list	view my history about pain level.	get the list	**86**	●	3	M	2	3	06/06/18
100	Smartwatch	Patient	choose how to save my pain	give more details to the physiotherapist	as mockup	**84**	●	3	M	2	3	06/06/18
110	Smartwatch	Patient	choose an activity	give more details to the physiotherapist	from list	**82**	●	5	M	2	4	06/07/18
75	WebApp	Physio	add/follow new patients in the webapp	follow them (pryv)	display new patient in the list	**80**	●	3	M		3	06/06/18
82	Smartwatch	Patient	set username and token on the smartwatch	give an authenticate access to Pryv	be able to save in Pryv	**78**	●	3	M		3	06/06/18
85	Smartphone	Patient	Connect a smartphone with my Pryv account	give an authenticate access to Pryv		**76**	●	2	M		3	06/06/18
88	WebApp	Physio	Connect as a Physio to the Web App	access to the plateforme and analyze stats	see when connected	**74**	●	5	M		3	06/06/18
120	WebApp	Physio	view all patients in a list	select one to browse to the history page	listing sans filtre et recherche	**72**	●	2	M		4	06/07/18
130	WebApp	Physio	view in the history page all pains level saved by the patient	analyze them	see a graph	**68**	●	8	M		4	19/07/18

Annexe I (suite) : Product Backlog

US Nr.	Theme	As an/a …	I want to …	in order to	Acceptance criteria	Priority	Status	Story Points	moscow	Initial Sprint	Current Sprint	US accepted (done done)
140	Smartwatch	Patient	select an activity outside my favorites	choose the good one	get public activities in list	**66**	●	2	M		4	06/07/18
145	Logic/Algo	Physio	see a moving average curve	obtain results as precise as possible	see a new series in the graph	**64**	●	2	M		5	19/07/18
150	WebApp	Physio	export datas	do other analyses outside the platforme	open exported data in excel	**62**	●	2	M		5	19/07/18
155	WebApp	Physio	Improve how to follow patient	make everything in the web app		**61**	●	5	M		5	19/07/18
160	WebApp	Physio	create activity type	classified them	see change in the list	**58**	●	5	M		4	06/07/18
170	WebApp	Physio	create a new activity	analyse more precisly what my patient do and how activty influence pain	see change in the list	**56**	●	3	M		4	06/07/18
180	SmartPhone	Patient	add comment on flagged pains	give more details to the physiotherapist	Be able to read them in the history list	**50**	●	3	S		5	19/07/18
190	WebApp	Physio	change time period and granularity in the curve	analyze average value and tendance	as mockup	**46**	●	3	S		5	19/07/18
200	SmartPhone	Patient	modify past inputs	change it in case of mistake	see change in the history list	**44**	●	5	S		5	19/07/18
205	WebApp	Physio	zoom by windows in the curve	go at desired period		**43**	●	1	S		5	19/07/18
210	WebApp	Physio	show comment on the curve	have more details	Be able to read them in the graph	**42**	●	1	S		6	
220	Smartwatch	Bachelor	Changer icone http	be more user friendly		**40**	●	1	S		6	
230	WebApp	Bachelor	Changer home page	have some general informations		**38**	●	1	S		6	
240	WebApp	Physio	have comments and activity name in exporting datas	have all informations		**36**	●	3	S		6	
400	WebApp	Physio	improve the algo average calculation	have better analyze		**30**	●	3	S			
500	SmartPhone	Patient	visualize all datas on a curve chart	see my evolution		**28**	●	2	S			
550	WebApp	All	set my prefered language	feel confortable with the service		**26**	●	3	S			
600	Smartwatch	Patient	be notified if I haven't set new pain level for a while	don't forget any change		**24**	●	2	S			
650	Smartwatch	Patient	to change interval notifications	set it in my convenience		**10**	●	3	W			
750	WebApp	Administrator	create (CRUD)* a new physiotherapist	manage access to the platform		**6**	●	8	W			

Annexe II : Mockups de la smartwatch

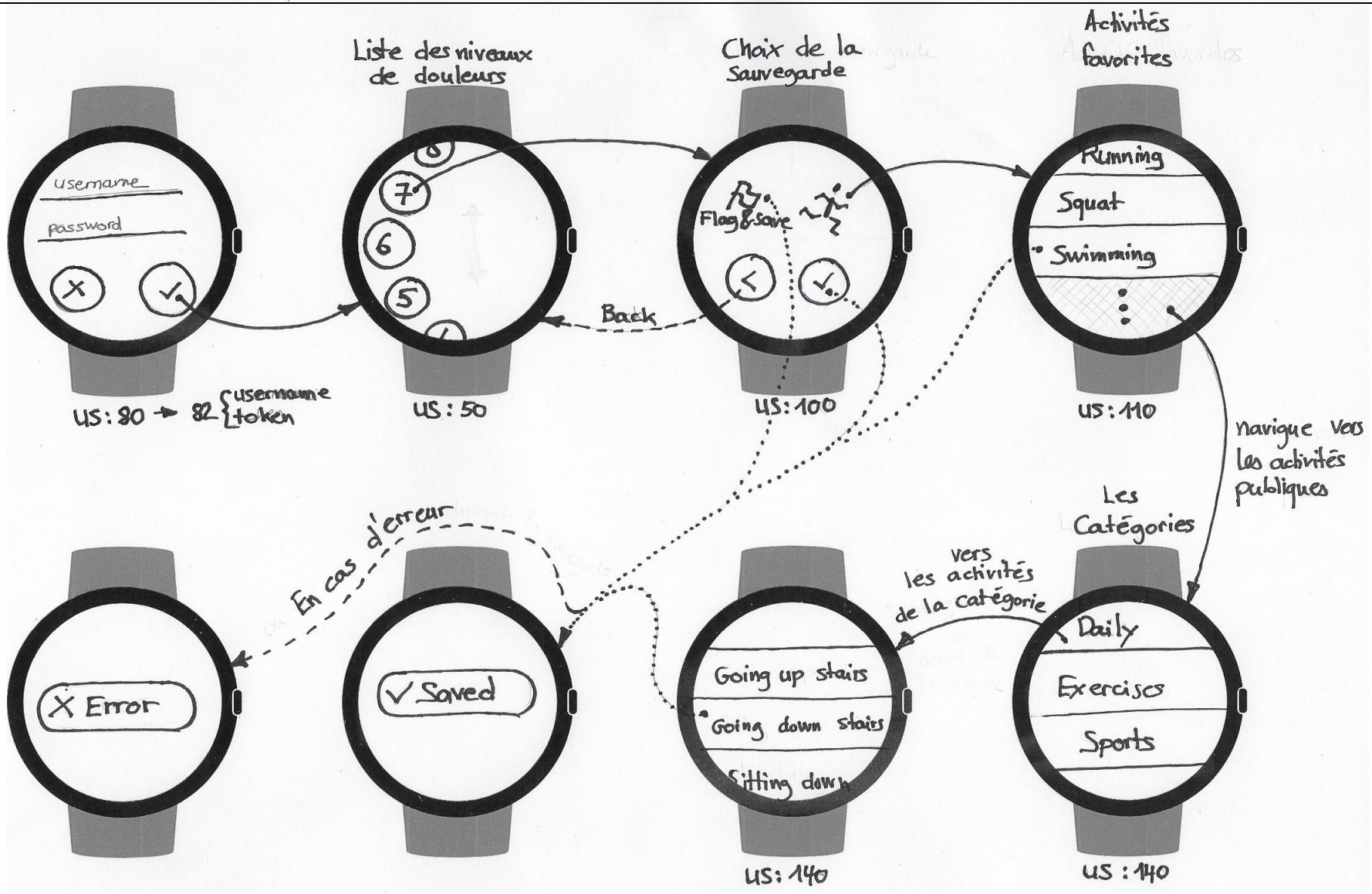


Figure 44 : Mockups de la smartwatch

Annexe III : Mockups du smartphone

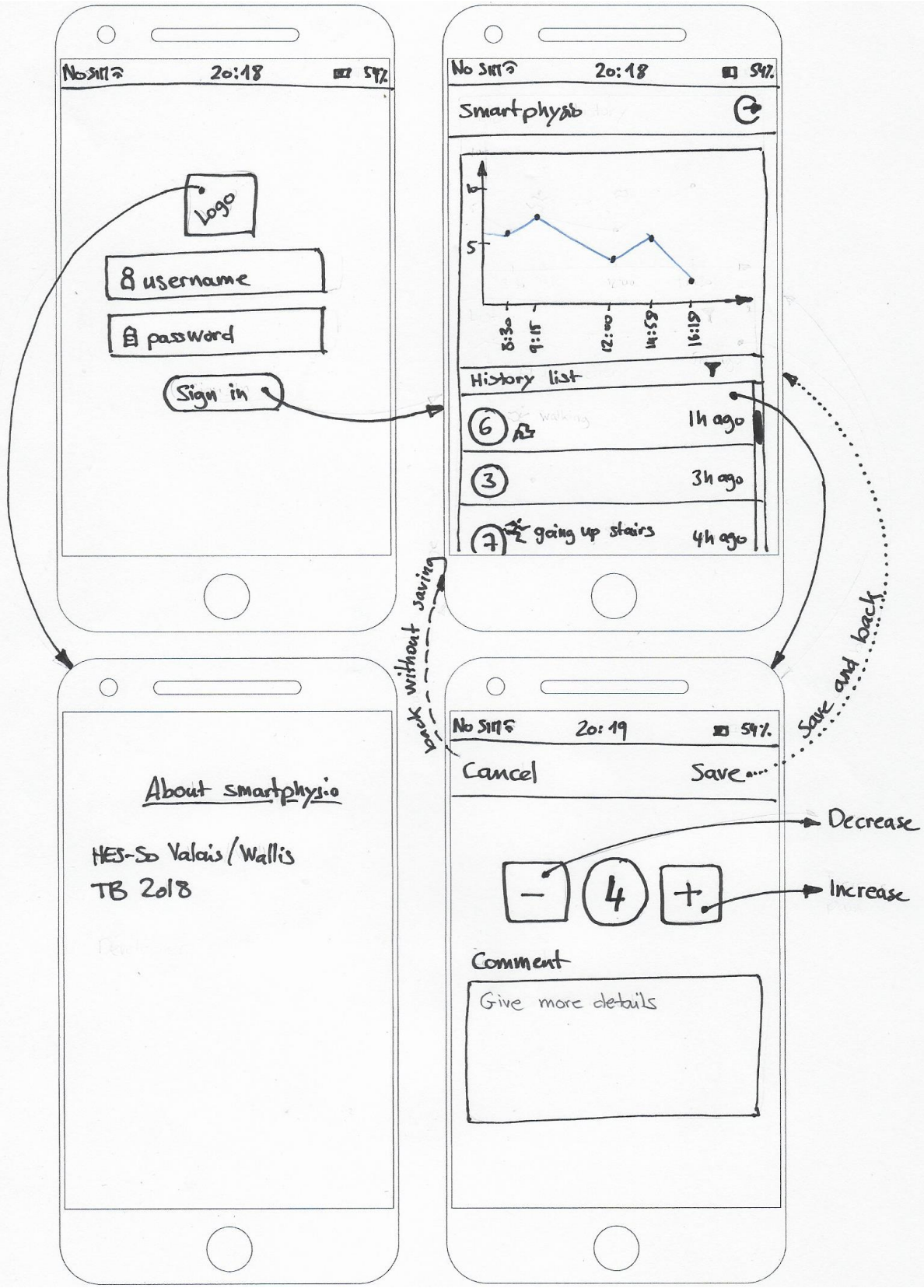


Figure 45 : Mockups du smartphone

Annexe IV : Mockups de l'application web

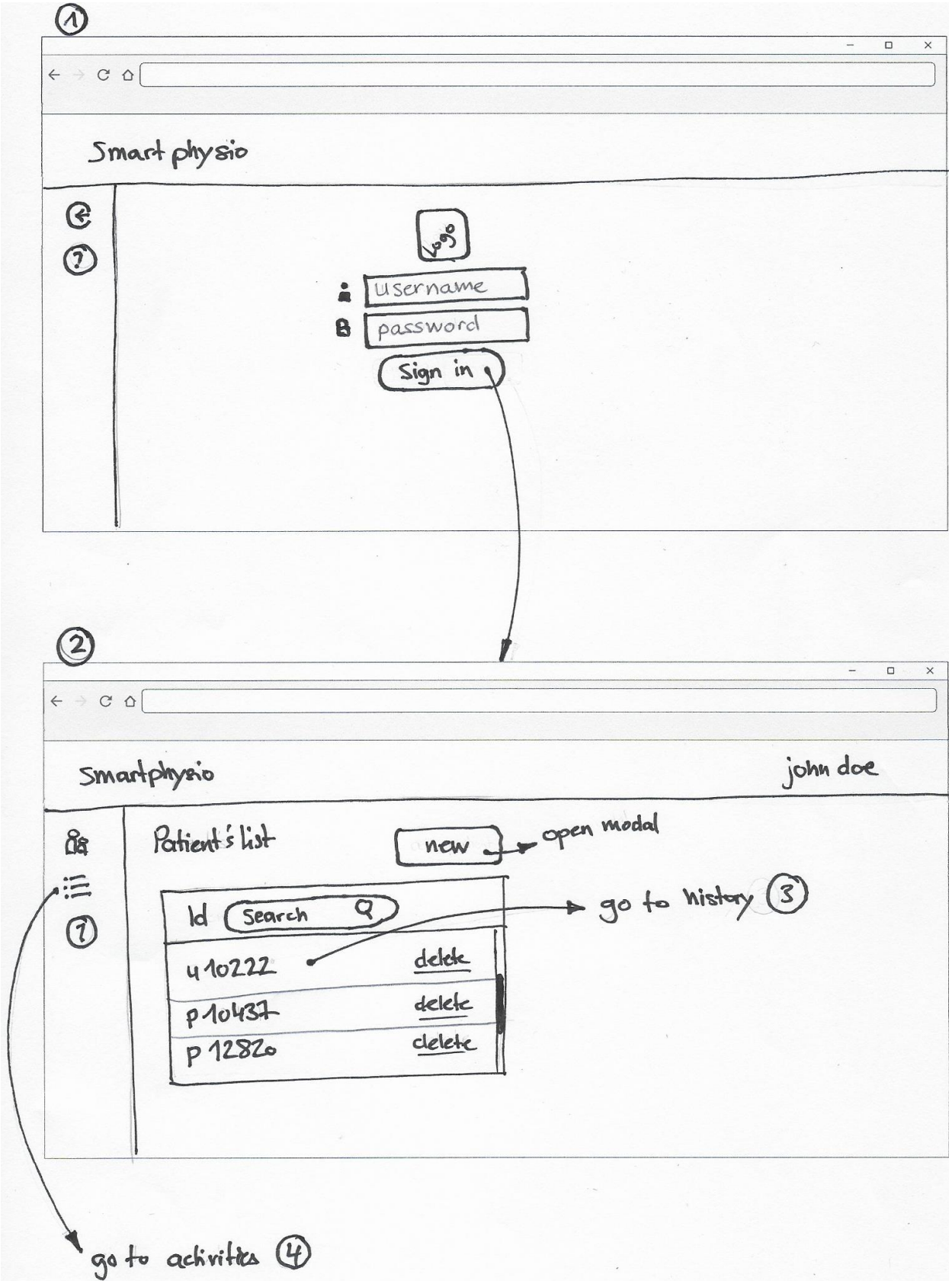


Figure 46 : Mockups de l'application web (a)

Annexe IV (suite) : Mockups de l'application web

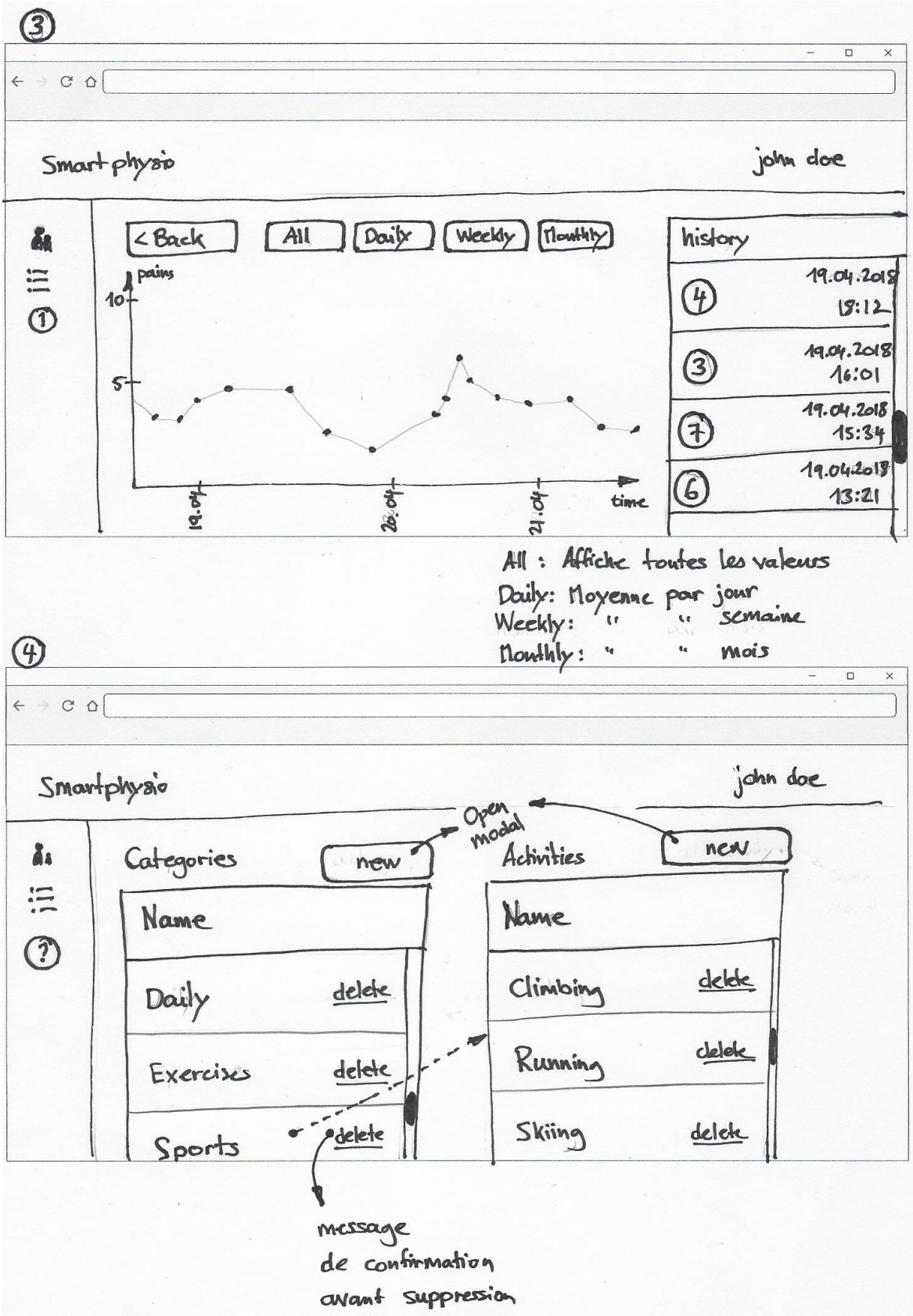


Figure 47 : Mockups de l'application web (suite et fin)

Annexe V : Google-service.json


```
{
  "project_info": {
    "project_number": "849987662045",
    "firebase_url": "https://smartphysiobeta.firebaseio.com",
    "project_id": "smartphysiobeta",
    "storage_bucket": "smartphysiobeta.appspot.com"
  },
  "client": [
    {
      "client_info": {
        "mobilesdk_app_id": "1:849987662045:android:7b253f3c39a6ccf3",
        "android_client_info": {
          "package_name": "ch.hevs.tb.jackh.smartphysio"
        }
      },
      "oauth_client": [
        {
          "client_id": "849987662045-
410aik5shoospt0utnbt803h046bh5t9.apps.googleusercontent.com",
          "client_type": 1,
          "android_info": {
            "package_name": "ch.hevs.tb.jackh.smartphysio",
            "certificate_hash": "876afd1d2527c07a5cd47014d0eb7521d01439f3"
          }
        },
        {
          "client_id": "849987662045-
40jd4fgh4m3o2vl7d4jdg79j75srlj75.apps.googleusercontent.com",
          "client_type": 3
        }
      ],
      "api_key": [
        {
          "current_key": "AIzaSyAfo1hd4VbQCsrx-ZBTaf31jvtaBXvUHUk"
        }
      ],
      "services": {
        "analytics_service": {
          "status": 1
        },
        "appinvite_service": {
          "status": 2,
          "other_platform_oauth_client": [
            {
              "client_id": "849987662045-
40jd4fgh4m3o2vl7d4jdg79j75srlj75.apps.googleusercontent.com",
              "client_type": 3
            }
          ]
        },
        "ads_service": {
          "status": 2
        }
      }
    }
  ],
  "configuration_version": "1"
}
```

Annexe VI : firebase.js

```
import "firebase/auth";
import "firebase/firestore";

var config = {
   apiKey: "AIzaSyDteuvZE_YhXRu3MCsr1NCI-egQu8NsBZg",
   authDomain: "smartphysiobeta.firebaseapp.com",
   databaseURL: "https://smartphysiobeta.firebaseio.com",
   projectId: "smartphysiobeta",
   storageBucket: "smartphysiobeta.appspot.com",
   messagingSenderId: "849987662045"
};


if (!firebase.apps.length) {
   firebase.initializeApp(config);
}

const auth = firebase.auth();
const db = firebase.firestore();

export {
   auth,
   db,
};
```

Annexe VII : ModelUser.js

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
var bcrypt = require('bcrypt-nodejs');
var uniqueValidator = require('mongoose-unique-validator');

var UserSchema = new Schema(
    {
        _id: { type: Schema.ObjectId, auto: true },
        username: {
            type: String,
            unique:true,
            required:true
        },
            password: {
                type: String,
                required:true
        },

    },
    {
    timestamps: true
    }
);

/**
 * jackh 04.05.2018
 * @param password
 * @returns {hash password}
 */
UserSchema.methods.comparePassword = function (password) {
    return bcrypt.compareSync(password, this.password);
};

/**
 * jackh 04.05.2018
 * @param password
 * hash user's password
 */
UserSchema.methods.setPassword = function setPassword(password) {
    this.password = hashPassword(password);

};

/**
 * jackh 04.05.2018
 * @param password
 * @returns hash string 'password'
 */
function hashPassword(password) {
    var salt = bcrypt.genSaltSync(10);
    return bcrypt.hashSync(password, salt);
}


UserSchema.plugin(uniqueValidator, { message: "This username is already taken" });


module.exports = mongoose.model("User", UserSchema);
```

Annexe VIII : Réponse http du serveur REST lors de l'authentification


```
D/OkHttp: <-- 200 OK http://192.168.1.118:5000/api-dev/users/signin (274ms)
D/OkHttp: X-Powered-By: Express
    Access-Control-Allow-Origin: *
    Access-Control-Request-Headers: access-control-allow-origin,content-type
    Access-Control-Allow-Methods: POST, GET
    Access-Control-Max-Age: 3600
    Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers,
Authorization, X-Requested-With
    Content-Type: application/json; charset=utf-8
    Content-Length: 232
    ETag: W/"e8-91dZIhXVI6KindMzrgNNJXWNqVg"
    Date: Thu, 02 Aug 2018 09:30:47 GMT
    Connection: keep-alive
D/OkHttp:
{"success":true,"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI1YjYyMjRhO
WVjZGExMTY3ZTBkNzNkZmYiLCJ1c2VybmFtZSI6InUxNyIsImlhdCI6MTUzMzIwMjI0N30.SZu2aH85Px1D
pO9QhWJ9wgWSzaD9Pebsdl4g4tlDBV0","_id":"5b6224a9ecda1167e0d73dff"}
    <-- END HTTP (232-byte body)
```

Annexe IX : Gestion des accès dans Pryv

Source : Pryv. (s.d.). *Hetting started - Authorize your application*. Récupéré sur api.pryv.com: http://api.pryv.com/getting-started/pryvme/#authorize-your-application

In our previous examples, we used an app token corresponding to a new access we generated at the end of the Authorization flow.

Each access is defined by a 'name', a 'type' and a set of 'permissions'.

Pryv.IO distinguishes between these access types:

- **Shared**: used for person-to-person sharing. They grant access to a specific set of data and/or with limited permission levels, depending on the sharing user's choice. You will not encounter this access type in your applications.
- **App**: used by applications which don't need full, unrestricted access to the user's data. They grant access to a specific set of data and/or with limited permission levels (e.g. read-only), according to the app's needs. This is the type of access we used for our Pulse Oximeter application.
- **Personal**: used by applications that need to access to the entirety of the user's data and/or manage account settings.

Each permission specifies a 'streamId', the id of the stream to which we want to give access, and an access 'level', which can be one of the following:

- 'read': Enable users to view the stream and its contents (sub-streams and events).
- 'contribute': Enable users to contribute to one or multiple events of the stream. Cannot create, update, delete and move streams.
- 'manage': Enable users to fully control the stream. Can create, update, delete and move the stream.

Finally, note that an existing access can be used to create other accesses, but only if the new access has lower permissions (Shared < App < Personal and Read < Contribute < Manage). Also, an access can create other accesses only in the same scope, namely with permissions on the same streams and their childrens.

Annexe X : Création de la structure des Streams du patient

```
/**
 * Create default stream structure
 */
private void initPryvConnection() {
    // Initiate new connection to Pryv with connected account
    connection = new Connection(credentials.getUsername(),
credentials.getToken(), Global.DOMAIN);

    // Initiate two streams : "Smartphysio" containing "Painslevel"
    painStream = new Stream()
            .setId(Global.PAIN_STREAM_ID)
            .setName(Global.PAIN_STREAM_NAME);

    smartPhysioStream = new Stream()
            .setId(Global.PAIN_STREAM_PARENT_ID)
            .setName(Global.PAIN_STREAM_PARENT_NAME)
            .addChildStream(painStream);

    // start a new thread
    new Thread() {
        public void run() {
            try {
                Stream createdSmartPhysioStream =
connection.streams.create(smartPhysioStream);
                updateStatusText("Stream created with id: " +
createdSmartPhysioStream.getId());
                Stream createdPainStream =
connection.streams.create(painStream);
                updateStatusText("Stream created with id: " +
createdPainStream.getId());
            } catch (IOException e) {
                updateStatusText(e.toString());
            } catch (ApiException e) {
                e.printStackTrace();
                updateStatusText("painStream : " + e.toString());
            }
        }
    }.start();
}
```

Annexe XI : ApiRequest.js

```
export default {
   pains: {
      fetchAll: filter => axios.get("https://" + filter.username +
            ".pryv.me/events", filter.data).then(res => res),
   },
   patients: {
      fetchAll: () => axios.get("/events", fetchAllPatientsByStream).then(res =>
            res.data.events), //tested
      create: data => axios.post("/events", data, adminHeader).then(res =>
            res.data.event),
      delete: id => axios.delete("/events/" + id, adminHeader).then(res =>
            res.data.event),
   },
   activities: {
      fetchAll: () => axios.get("/events", fetchAllActivitiesByStream).then(res =>
            res.data.events),  //tested
      create: data => axios.post("/events", data, adminHeader).then(res =>
            res.data.event),
      delete: id => axios.delete("/events/" + id, adminHeader).then(res =>
            res.data.event),
      update: id => axios.put("/events/" + id, adminHeader).then(res =>
            res.data.event),
   },
   streams: {
      fetchAll: () => axios.get("/streams", parentStream).then(res =>
            res.data.streams),  //tested
      create: data => axios.post("/streams", data, adminHeader).then(res =>
            res.data.stream),
      delete: id => axios.delete("/streams/" + id, adminHeader).then(res =>
            res.data.stream),
      update: data => axios.put("/streams/" + data.id, data.update,
            adminHeader).then(res => res.data.stream),
   },
   access: {
      fetchAccess: data => axios.post("https://reg.pryv.me/access", data).then(res
            => res.data),
      updateAccess: urlPoll => axios.get(urlPoll).then(res => res.data),
   },

   users: {
      signIn: data => {
         return dispatch => {
            axios
               .post('https://reg.pryv.me/access', {data}).then((res) => {
                  dispatch({ type: USER_SIGNED_IN, payload: res });
               })
               .catch((err) => {
                  console.error(err);
               });
         };
      },
   }
}
```

Annexe XII : Générateur de tokens pour Pryv

Pryv Access Token Generation

How to

1. Enter the **Application ID** that you will use in your app
2. Setup the streams you will request access to in the **Permissions** box
3. Press the **Request Access** button
4. **Sign in** with your username and password
5. Authorize the application access to your account by pressing on **Accept**
6. Copy the **Access token** and use it to run your tests

Permissions

This defines the streams that your access token will be allowed to manipulate.
If you request permissions for streams that don't exist yet, they will be created on the user's account upon user authorization.
The value provided in the **defaultName** property will be used as its name.

The **level** must be one of the following values:

- **read**: Allows to get the stream's values.
- **contribute**: Read and generate new content.
- **manage**: Contribute, update and delete.

Edit the fields **StreamId**, **Level** and **defaultName** in the text box containing a JSON array where each item represents a Permission that will be requested upon successful authentication.

If the **Master Token** option is checked, it will set the level **manage** on all streams for the selected application.

Register URL reg.pryv.me

Application ID (min length: 6) app-web-access-test

Permissions (See API doc)

```
[
  {
    "streamId": "diary",
    "defaultName": "Journal",
    "level": "read"
  }
]
```

Master Token

Show/hide advanced options

Request Access

Username

Access token

Domain

Source : Pryv.com - http://pryv.github.io/app-web-access/

Annexe XIII : Accès à la Web App Smartphysio

Un libre accès à Smartphysio est à disposition sur : http://212.47.240.226/

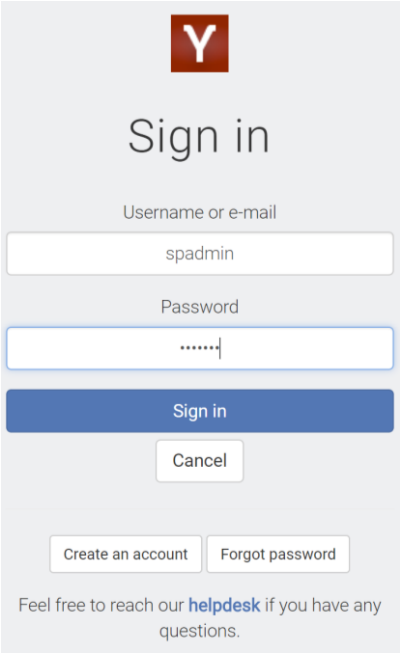
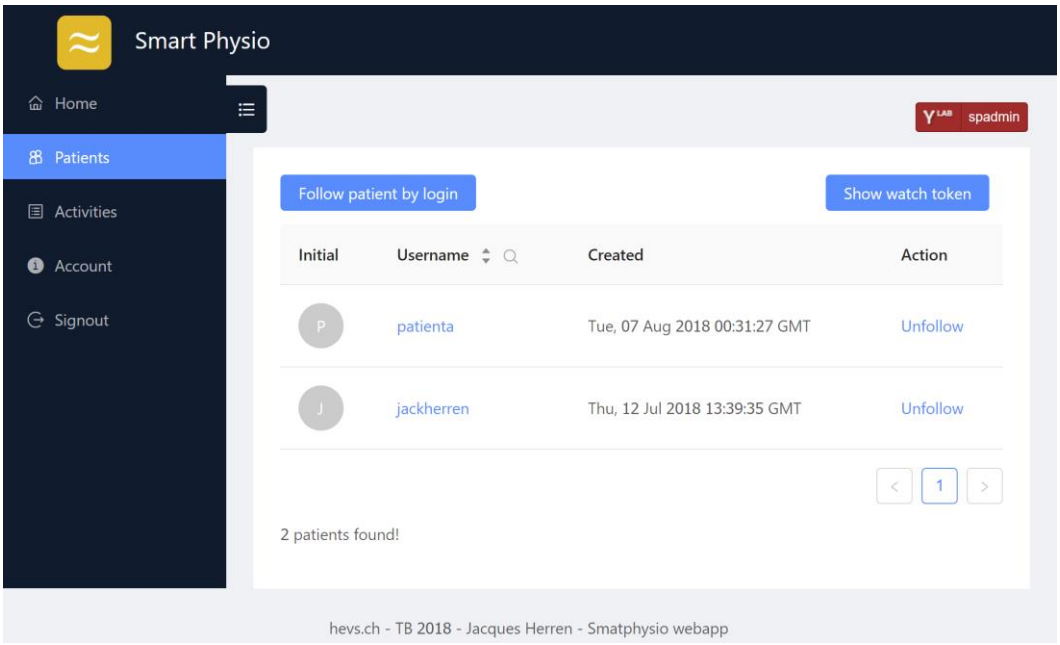
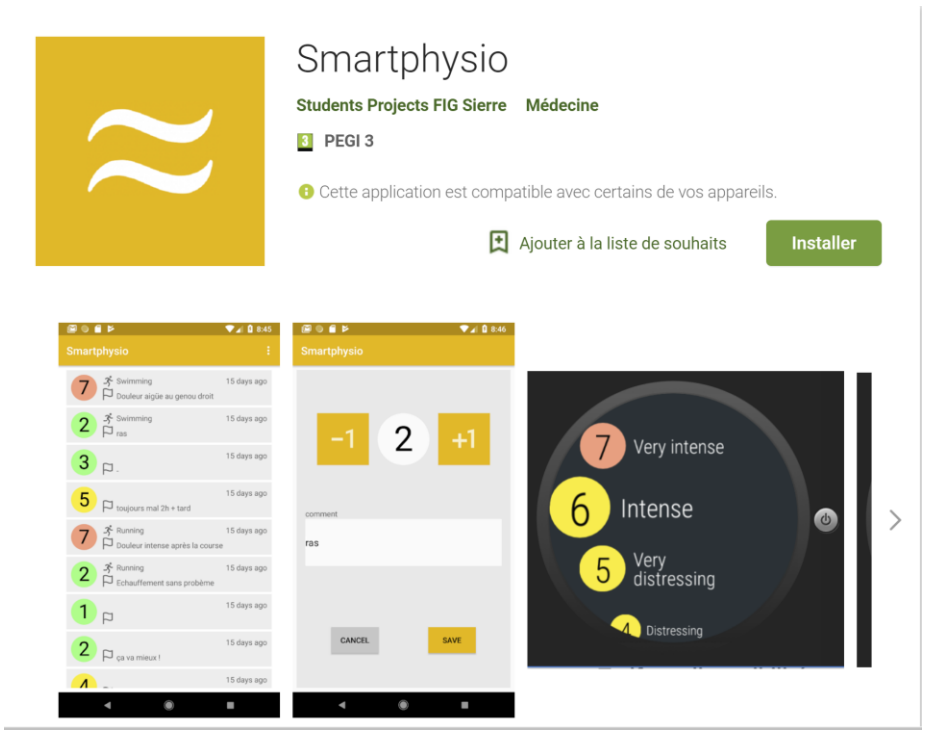
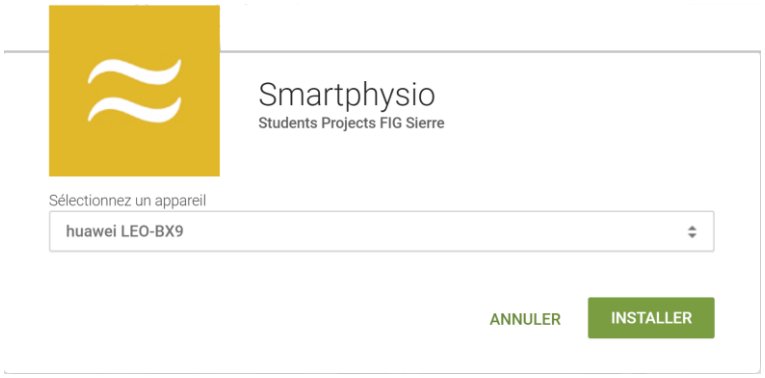
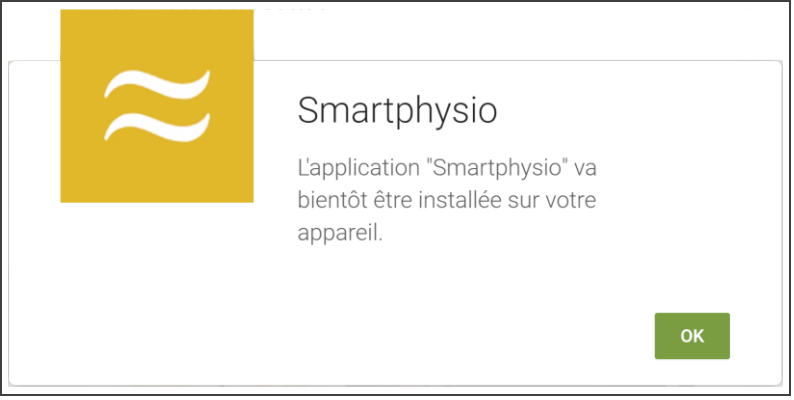
Etape N°1	Pour se connecter : cliquer sur le bouton « Pryv » en haut à droite de la page.	
Etape N°2	Utiliser les informations de connexion suivante : username : **spadmin** Password : **spadmin**	
Etape N°3		

Tableau 19 : Connexion à Smartphysio en libre accès.

Annexe XIV : Installer Smartphysio sur une smartwatch

Smartphysio est disponible sur Google Play à l'adresse suivante :

https://play.google.com/store/apps/details?id=ch.hevs.tb.jackh.smartphysio_android

Etape N°1	Cliquer sur installer	
Etape N°2	Choisir l'appareil et cliquer sur installer.	
Etape N°3	Cliquer sur OK	

Etape N°4a	S'identifier sur la smartwatch
	
	Deux possibilités :
	• Création de votre propre compte sur : https://sw.pryv.me/access/signinhub.html
	  ○ Puis suivre les opérations du chapitre 11.1 en page 67. (show token)
	• Ou utiliser le patient générique : « PatientA » avec les identifiants suivants :
	  ○ Username : **patientA**
	  ○ Token : **cjkiyz1yk04670cd3c636byvu**

Etape N°4b	S'identifier sur un smartphone
	
	Deux possibilités :
	• Avec vos propres identifiants Pryv
	• Ou utiliser le patient générique : « PatientA » avec les identifiants suivants :
	  ○ Username : **patientA**
	  ○ Password : **patientA**

Tableau 20 : Installer Smartphysio sur une smartwatch et un smartphone depuis le Google Play.

Déclaration de l'auteur

Je déclare, par ce document, que j'ai effectué le travail de Bachelor ci-annexé seul, sans autres aides que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de Bachelor, y compris au partenaire de recherche appliquée avec lequel j'ai collaboré, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail et que je cite ci-après :

- Michael Schumacher, professeur HES et collaborateur à l'Institut Informatique de gestion.
- Fabien Dubosson, collaborateur à l'institut d'informatique de gestion de la HES-SO Valais-Wallis

Jacques Herren, 08.08.2018