Sokkin

Louis J

Fait le: 14 août 2019

Dernière mise à jour : September, 2019

Une copie de ce document est disponible en ligne sokkin.readthedocs.io

Table of contents

Index **18**

List of tables

Chapter 1

Installer son environnement de développement

1.1 Choix de l'éditeur de texte

Nous avons pris la décision de laisser le laisser aux développeurs le choix de leurs éditeurs de texte.

Cependant, nous pouvons recommander l'utilisation de deux éditeurs de textes.

1. `Visual Studio Code` est un éditeur développé par Microsoft pour Windows, Linux et OS X. Gratuit et open-source, il inclus des plugins pour le débogage, la prise en charge de git, et une aide à la saisie de texte "intélligente".

Note: Téléchargement: https://code.visualstudio.com/download

---

2. `WebStorm` est un outil open-source basé sur la plateforme intelliJ. C'est un choix particulièrement intéréssant pour la développement en React.

Note: Téléchargement: https://www.jetbrains.com/webstorm/

1.2 Installer android studio

Visual studio est un élément essentiel de votre environnement de travail si vous souhaitez tester utiliser l'application sur un système Android.

Vous pouvez télécharger Android Studio directement sur le site officiel: https://developer.android.com/studio

1.2.1 Créer un émulateur android

Une fois android studio installé, vous allez pouvoir installer votre premier émulateur Android. Un émulateur Android est tout simplement un appareil téléphone Android virtuel (AVD: Android Virtual Device).

Pour créer un émulateur android, utilisez la commande suivante dans un interpréteur de commande:

```
avdmanager create avd -n <nom_de_l_emulateur> -k <target>
```

`nom_de_l_emulateur`
: Vous pouvez remplacer ce champ par le nom donné à votre émulateur, par example `emulateur1`

`target`
: Target correspond à la version d'Android souhaitée. Vous pouvez lister les versions disponibles d'android sur votre machine avec la commande suivante:

```
avdmanager list targets
```

Cette commande vous affichera les target avec la version d'Android correspondante, par example:

```
Available Android targets:
id: 1 or "android-3"
     Name: Android 1.5
     Type: Platform
     API level: 3
     Revision: 4
     Skins: QVGA-L, HVGA-L, HVGA (default), HVGA-P, QVGA-P
id: 2 or "android-4"
     Name: Android 1.6
     Type: Platform
     API level: 4
     Revision: 3
     Skins: QVGA, HVGA (default), WVGA800, WVGA854
id: 3 or "android-7"
     Name: Android 2.1-update1
     Type: Platform
     API level: 7
     Revision: 2
     Skins: QVGA, WQVGA400, HVGA (default), WVGA854, WQVGA432, WVGA800
id: 4 or "android-8"
     Name: Android 2.2
     Type: Platform
     API level: 8
     Revision: 2
     Skins: WQVGA400, QVGA, WVGA854, HVGA (default), WVGA800, WQVGA432
```

Une fois votre émulateur crée, vous pouvez vérifier son status via la commande suivante:

```
avdmanager list avd
```

Note: L'ensemble de ces actions sont aussi possibles depuis l'interface graphique de `Android Studio`, cependant, l'éxecution en ligne de commande nous offre plus de contrôle et une plus grande richesse de paramètres d'exécution. Voir https://developer.android.com/studio/run/managing-avds.html pour la liste des options

---

1.3 Installer React-native

Pour installer React Native, nous avons besoin des prérequis suivants:

- Nodejs
- NPM
- Watchman

1.3.1 Installer Nodejs

React Native étant un framework de JavaScript, il requiert d'avoir Nodejs installé.

```
curl -sL https://deb.nodesource.com/setup_9.x | sudo -E bash -
```

```
sudo apt-get install -y nodejs
```

1.3.2 Installer NPM

Après avoir installer Nodejs, vous pouvez installer NPM, le gestionnaire de packets pour Nodejs.

```
curl http://npmjs.org/install.sh | sh
```

1.3.3 Installer Watchman

Watchman est un outil qui permet de "regarder" les fichiers de notre ordinateur. En d'autres termes, ils nous permet de recharger l'application dès qu'un fichier à été modifié. C'est un outil `facultatif` mais vivement conseillé pour gagner du temps.

```
git clone https://github.com/facebook/watchman.git
```

```
cd watchman ; git checkout v4.9.0
```

```
./autogen.sh ; ./configure ; make
```

```
sudo make install
```

1.3.4 Installer React Native

Finallement, vous allez pouvoir installer le fameux `React Native`

```
sudo npm install -g react-native-cli
```

Note: Vous pouvez noter l'utilisation de `sudo`. Nous sommes obligés d'utiliser `sudo` avec l'option `-g`, qui nous permet d'installer React Native globalemet sur votre machine. Vous pouvez décider d'installer React Native localement sur le projet avec la commande suivante:

```
npm install react-native-cli
```

1.4 Installer les outils de test d'intégration

Durant cette partie, nous allons procéder à l'installation des outils permettant l'exécution des tests d'intégration. Cette partie est facultative, cependant, ces outils peuvent vous permettre d'automatiser certaines de vos tâches, et donc d'optimiser votre temps de travail.

1.4.1 Installer appium

Appium est un système de d'automatisation pour applications natives ou hybrides.

```
sudo npm install -g appium@1.8.1 --unsafe-perm=true --allow-root
```

Pour vérifier que l'installation c'est correctement déroulée vous pouvez utiliser la commande suivante:

```
npm install -g appium-doctor
```

```
appium-doctor
```

1.4.2 Installer Python et pip

L'ensemble des scripts d'automatisation sont développés en python.

Si vous utilisez un système d'exploitation récent, vous avez des changes d'avoir d'ores et déjà ces deux outils sur votre machine.

Pour vérifier si c'est le cas:

```
python --version
```

```
pip --version || pip3 --version
```

Si ce n'est pas le cas, veuillez vous référer à la documentation officielle de pip (https://pip.pypa.io/en/stable/installing/)

Chapter 2

Démarrer l'application

Après ce long parcours, ous allez **enfin** pouvoir éxècuter du code React Native sur votre machine.

Pour ce faire, commencez par démarrer votre émulateur.

```
emulator -list-avds
```

Cette commande permet d'afficher l'ensemble des émulateurs déjà installés sur votre machine, il vous suffit d'en choisir un et de le lancer.

```
emulator -avd <nom_de_l_emulateur>
```

Dans le dossier de l'application, vous allez à présent pouvoir pouvoir lancer votre client React Native

```
react-native run-android
```

Lancez maintenant le serveur / packager React Native avec la commande suivante:

```
npm start
```

Vous devriez maintenant voir s'afficher l'application sur votre émulateur

Chapter 3

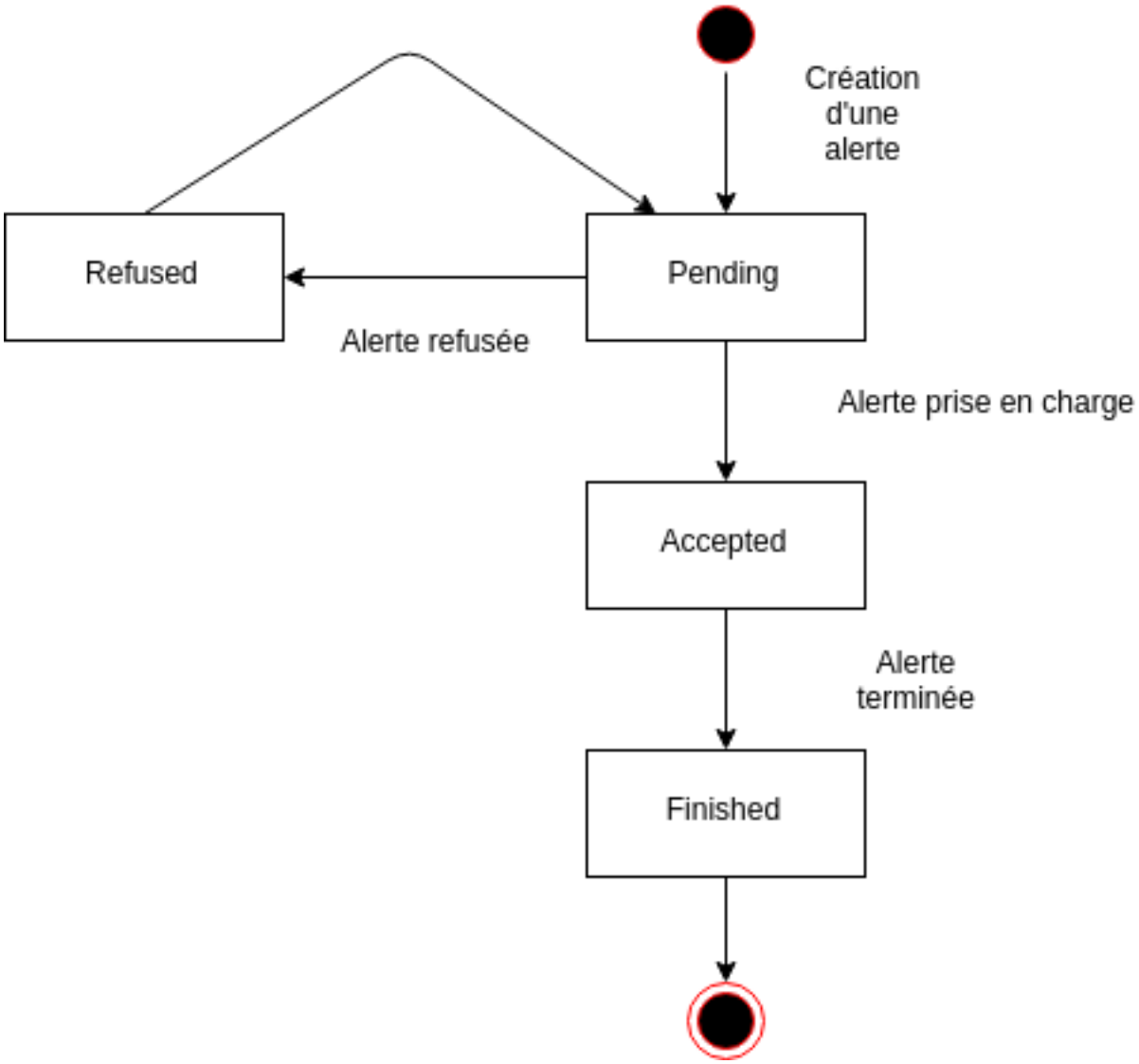
Concepts basiques

3.1 Les alertes

Les alertes est l'une des composantes principales de l'application Sokkin. Elles permettent à un membre de la famille d'envoyer une notification à son cercle familiale en cas de besoin.

Nous pouvons voir une alerte comme une machine à états ayant l'un des états suivants:

- Pending : L'alerte est en attente qu'un membre de la famille l'accepte ou la refuse.
- Refused : Un membre de la famille à refusé une alerte. Celle-ci retourne
- Accepted : Un membre de la famille à accepté une alerte. Celle-ci est donc en cours de réalisation.
- Finished : Une fois le processus finis, la personne termine l'alerte, celle-ci est supprimé.



3.2 Firebase, base de donnée en temps réel

Firebase est une plateforme “Backend as a Service” (BaaS) permettant de construire des applications web et mobiles.

Firebase est une base de données en temps réel. La plupart des bases de données sont basés sur le protocole HTTP. Firebase utilise des WebSockets. Les WebSockets sont bien plus rapides que HTTP. Les flux réseaux sont bien plus optimisés. (Vous pouvez trouver plus d'informations sur les WebSockets ici: https://tools.ietf.org/html/rfc6455)

Firebase est aussi un système de fichiers complexe. Firebase comporte une suite de règles de sécurité pour attribuer des privilèges aux utilisateurs authentifiés, protégent ainsi l'accès aux données.

Voici les services Firebase utilisés par Sokkin:

Table 1: Services Firebase.

Nom	Utilité	Lien de la documentation
Authentification	Gérer les identités des utilisateurs	https://firebase.google.com/docs/auth
Database	Stockage de données en temps réel	https://firebase.google.com/docs/database
Functions	Executer du code pour répondre aux évènements	https://firebase.google.com/docs/functions
Crashlytics	Monitoring et supervision des erreurs	https://firebase.google.com/docs/crashlytics

3.2.1 Se connecter à la console Firebase

Afin de vous connectez a la console de Firebase, vous devez avoir un compte google et les droits d'accès au projet Sokkin.

Vous pouvez ensuite acceder à la console ici:

- https://console.firebase.google.com/project/eip-359fe/overview

3.3 Les modèles de données

L'application Sokkin utilise un système de modélisation des modèles de données. Ce système à pour objectif de combler les lacunes que propose le service Firebase. (voir *Firebase, base de donnée en temps réel*)

L'objectif est multiple:

- Valider des objets JSON facilement.
- Décrire les objets utilisés.
- Offrir des messages d'erreur à l'interface utilisateur.

Pour ce faire, nous utilisons AJV (Another Json Validator)

Note: Lien de AJV: https://ajv.js.org/

Voici un example de modèle pour un `Utilisateur`

```
Utilisateur ={
   "type":"object",
   "required":[
      "email",
      "prenom"
   ],
   "properties":{
      "email":{
         "type":"string",
         "format":"email"
      },
      "prenom":{
         "type":"string"
      },
      "enfants":{
         "type":"array",
         "uniqueItems":true,
         "items":{
            "type":"string"
         }
```

(continues on next page)

(continued from previous page)

```
      }
   }
}
```

`type`
: *type* permet de définir le type (ou plusieurs types) pour une propriété. Les types de base sont les suivants: `number`, `integer`, `string`, `boolean`, `array`, `object` ou `null`

> **Warning:** Les autres mots clef s'appliquent souvent par rapport à un *type*. Si la donnée est de type différent, le mot clef ne sera pas appliqué et la donnée sera considérée valide.

`properties`
: Le mot clef *properties* est un objet de type clef-valeur. Chaque valeur doit être un schema JSON valide.

  L'objet `Utilisateur` devra, pour être valide, contenir toutes les propriétés citées. (un prenom et un email dans notre cas)

`required`
: Le mot clef *required* contient une liste de chaînes de charactères sans doublons.

  L'objet `Utilisateur` devra, pour être valide, contenir toutes les propriétés citées. (un prenom et un email dans notre cas)

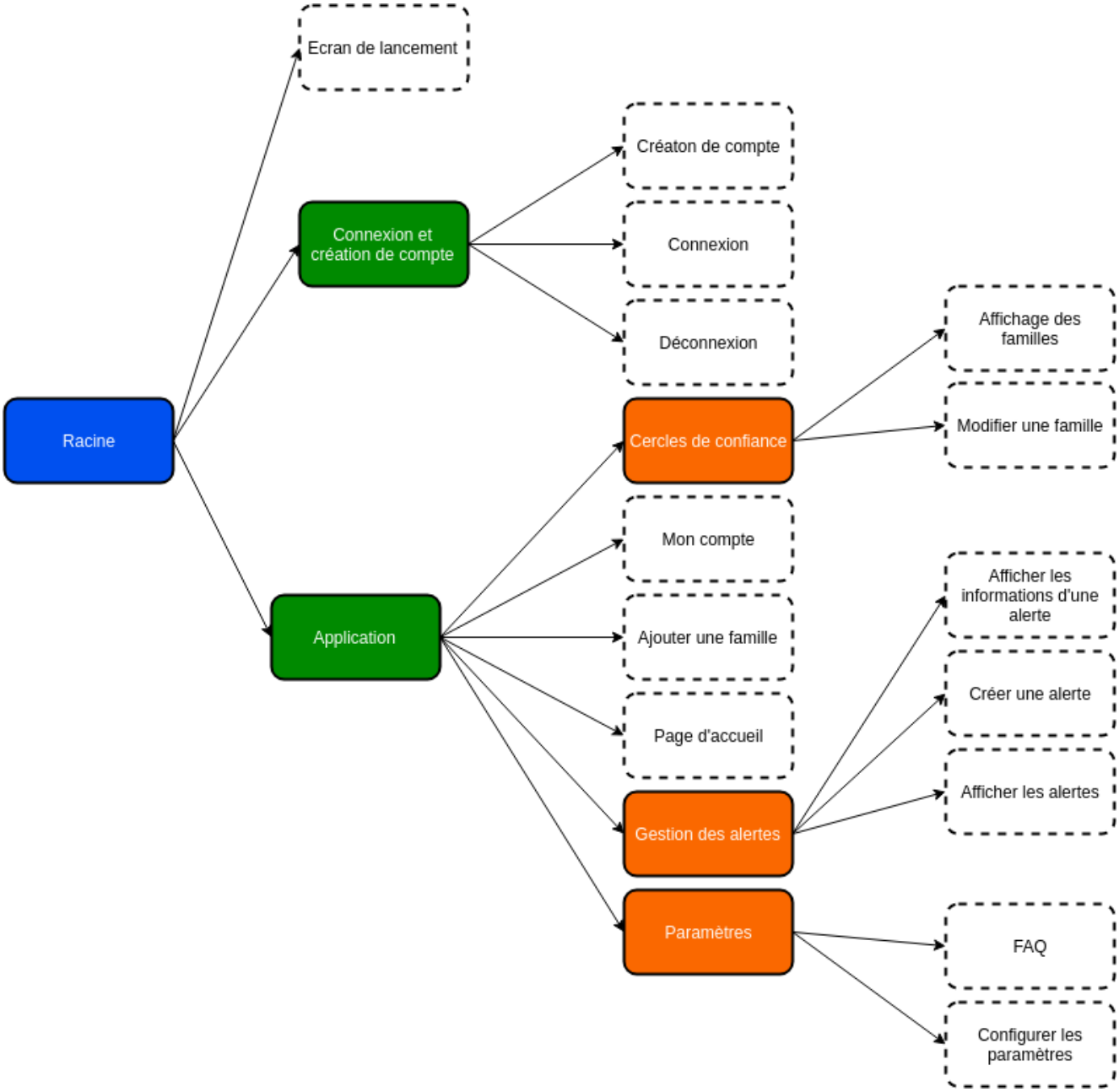
`format`
: Le *format* permet une granulosité accrue en enrichissant le *type*.

  Les formats accéptés sont les suivants: `date`, `date-time`, `uri`, `email`, `hostname`, `ipv4`, `ipv6` ou `regex`.

  > **Error:** Pour des raisons de sécurité, l'usage du format *regex* est déconseillé du à l'utilisation de la méthode 'eval'.

3.4 Le navigateur

L'application Sokkin est une service riche, composé de plusieurs pages. Pour diriger la présentation des pages et les transitions entre elles, nous utilisons un système de navigateur.


Ecran de lancement
Racine
Connexion et création de compte
Créaton de compte
Connexion
Déconnexion
Application
Cercles de confiance
Affichage des familles
Modifier une famille
Mon compte
Ajouter une famille
Page d'accueil
Gestion des alertes
Afficher les informations d'une alerte
Créer une alerte
Afficher les alertes
Paramètres
FAQ
Configurer les paramètres

Chapter 4

Documentation technique

4.1 Les services

yes

4.1.1 Authentification

Le service d'authentification permet de gérer les identitées, les droits et les sessions des utilisateurs. Il utilise le service `Authentification` de firebase (voir *Firebase, base de donnée en temps réel*)

`createUser(`*email*, *password*, *name*`)`
Crée un compte utilisateur avec les identifiants donnés.

Parameters
- `email` (*string*) – Addresse mail de l'utilisateur
- `password` (*string*) – mot de passe de l'utilisateur
- `name` (*object*) – Objet contenant le nom et le prenom de l'utilisateur

Return type `True` ou `False`

`login(`*email*, *password*`)`
Permet à l'utilisateur de se connecter avec le couple email:mot de passe donné.

Parameters
- `email` (*string*) – Addresse mail de l'utilisateur
- `password` (*string*) – mot de passe de l'utilisateur

Return type `True` ou `False`

`logout()`
Déconnecte l'utilisateur actuel.

Return type `True` ou `False`

4.1.2 User

Le service `User` met à disposition permettant de configurer le profile et le compte des utilisateurs.

`getCurrentUserProfile(`*id*`)`
Récupère les informations liées à l'utilisateur actuel.

Return type Objet contenant les données de l'utilisateur.

`updateCurrentUserProfile(`*data*`)`
Met à jour les informations de l'utilisateur actuel.

Parameters `data` (*object*) – Objet contenant les données a mettre à jour.

Return type Objet contenant les données de l'utilisateur.

4.1.3 Family

Le service `Family` offre une API permettant la gestion des cercles de confiance.

`getCurrentUserFamily()`
Permet de récupérer les familles liées à l'utilisateur actuellement connecté.

Return type Identifiant de la famille.

`getFamily(id)`
Récupère les informations liées à une famille à partir de son identifiant.

Parameters `id` (*string*) – identifiant de la famille
Return type Objet contenant les données de la famille.

`createFamily()`
Permet à l'utilisateur actuel de créer une famille. Il sera automatiquement ajouté à cette dernière.

Return type `True` ou `False`

`invitToFamily(email, role)`
Ajoute un membre dans une famille.

Parameters
- `email` (*string*) – Email du membre à ajouter.
- `role` (*string*) – Role du membre, parmis: `Aidant`, `helpers` ou `members`

Return type `True` ou `False`

`addChildrenToFamily(family, name, address)`
Ajoute un enfant dans une famille.

Parameters
- `family` (*string*) – Identifiant de la famille à laquelle l'enfant appartiens.
- `name` (*object*) – Objet contenant le nom et le prenom de l'enfant
- `address` (*string*) – Addresse de domicile de l'enfant.

Return type `True` ou `False`

4.1.4 Alert

Le service d'alerte offre une API permettant de manipuler le système d'alertes de l'application Sokkin. Voir *Les alertes* pour plus d'informations sur les alertes.

`getMyAlerts(status=None)`

Récupère la liste des alertes de l'utilisateur en cours.

Parameters `status` (*string*) – Paramètre facultatif permettant de filtrer la liste en fonction de leur status. Le status d'une alerte est une chaîne de charactères parmis : `Pending`, `Started` ou `Finished`
Return type Une liste contenant les identifiants des alertes.

`setAlert(status)`

Permet de changer le status d'une alerte.

Parameters `status` (*string*) – Le status d'une alerte est une chaîne de charactères parmis : `Pending`, `Started` ou `Finished`
Return type `True` ou `False`

4.1.5 Feedbacks

??

4.2 Structure du dépôt

Le dépôt Sokkin est organisé de la façon suivante:

- `__tests__/` : Fichiers de tests.
- `__tests__/samples/` : Fichiers contenant les jeux de données utilisables pour les tests unitaires.
- `__mocks__/` : Contient les mocks.
- `android/` : Code source et configuration de l'aplication Android.
- `ios/` : Code source et configuration de l'aplication iOS.
- `functions/` : Cloud functions pour Firebase.
- `gatling/` : Framework de tests d'intégration.
- `src/` : Code source de l'application.
- `src/assets/` : Fichiers statiques (images, logo, ...).
- `src/component/` : Components React Native.
- `src/font/` : Polices d'écriture.
- `src/pages/` : Les pages de l'application.
- `src/router/` : Définition de la navigation au sein de l'application via le routeur (*Le navigateur*).
- `src/services/` : Les services de Sokkin (*Les services*).
- `src/storages/` : Fonctions permettant d'utiliser le service `Storage` de firebase

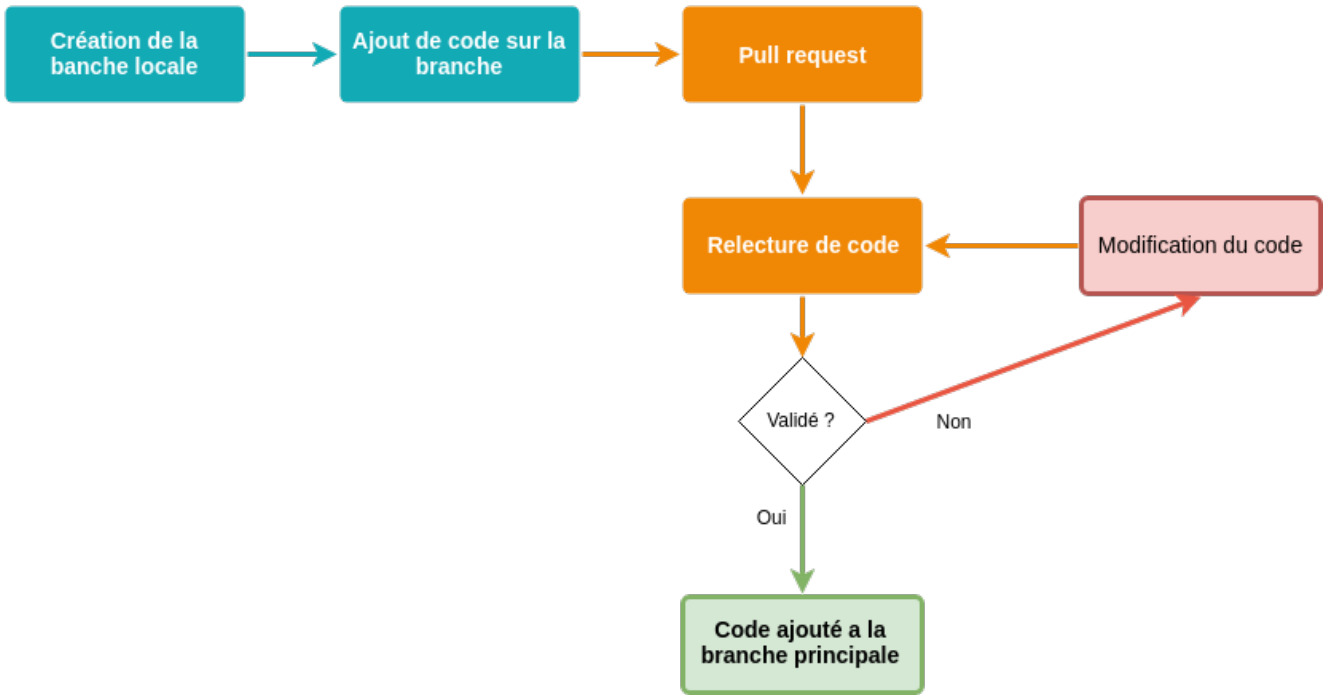
Chapter 5

Ajouter des fonctionnalités

5.1 Les pull requests

Sokkin utilise le processus de “pull requests” commun à de nombreux projets utilisant git. Ce processus devrait être familier aux contributeurs expérimentés de logiciels libres. L'idée est qu'un petit nombre de personne ait accés à la branche principale (communément appelée “master”).

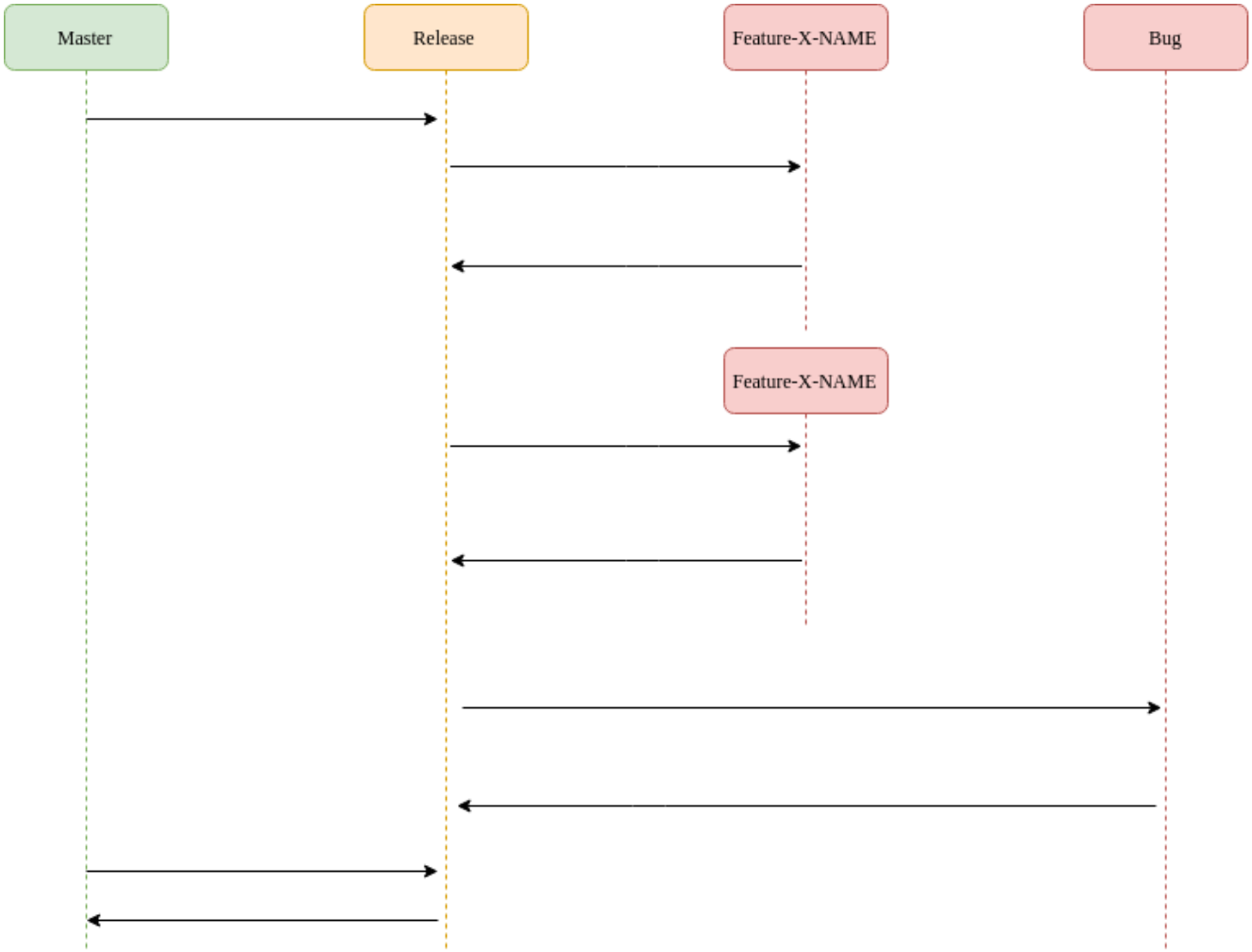
Pour ce faire, les contributeurs développent leurs fonctionnalités de la façon suivante:



1. Le développeur copie ou clone le dépot, puis crée une branche dédiée.
2. Le développeur crée une fonctionnalité sur le dépot local sur la branche dédiée.
3. Le développeur crée une pull request via l'interface GitLab.
4. Le reste de l'équipe procède à la relecture du code puis valide ou non la pull request.
5. Si la pull request est validée, le chef de projet procède à l'envois du code sur la branche principale.

5.1.1 Les branches

Chez Sokkin, nous utilisons les branches suivantes:



- `Master`: C'est la branche d'origine. Elle permet uniquement la création d'autres branches, et l'ajout de code via `Merge Request`
- `Release`: La branche release est la source des branches de développement. Elle permet de tester les fonctionnalitées avant de les ajouter sur la branche `Master` .
- `Feature-X-NAME`: Les branches de features sont crées par les développeurs, leur nom devra suivre la règle suivante: FEATURE-<nom_de_la_fonctionnalité>-<nom_du_developpeur>
  Par exemple: `Feature-UserAccount-John_d`
- `Bug`: Les branches de bug permettent de corriger une fonctionnalité ajoutée à une des deux branches réservés. (`Master` ou `Release`).

5.2 Tester son code

Le projet sokkin se base sur une méthodologie d'intégration continue. A chaque ajout de nouvelles fonctionalités, des séries de tests sont appliqués sur l'ensemble du projet.

Nous distinguons trois types de tests.

Table 1: Types de tests

Type de test	Definition
Tests unitaires	Valider l'execution d'une partie précise.
Tests de non régression	Empecher les effets de bord.
Tests d'intégration	Valider l'ensemble des parties.

Les tests unitaires sont sous la responsabilité du développeur lorsqu'il ajoute une nouvelle fonctionnalité.

5.2.1 Tests unitaires

Sokkin se base sur le framework jest pour la création de tests unitaires. Voir *Installer son environnement de développement* pour l'installation de jest.

Note: Pour plus d'information sur jest veuillez vous referer à la documentation officielle: https://jestjs.io/docs/en/getting-started

---

L'execution des tests unitaires se fais via la commande

```
npm run unittest
```

Les fichiers de tests sont dans le dossier ~/__tests__

Chapter 6

Questions fréquentes

6.1 Questions fréquentes

6.1.1 Je n'arrive pas à exécuter la commande avdmanager ou emulator

Avdmanager est installé automatiquement avec Android Studio.

Pour installer Android Studio, voir *Installer android studio*.

Une fois Android Studio installé, le chemin vers les commandes `avdmanager` et `emulator` ne sont pas forcément dans le path. Par défaut, ils sont installés dans ~/android/android_sdk/tools.

Vous pouvez ensuite ajouter ce chemin à votre variable d'environnement PATH.

```
export PATH="$PATH:~/android/android_sdk/tools"
```

6.1.2 Erreur "SDK location not found" lorsque je lance mon client React Native

Pour régler ce soucis, faites les étapes suivantes:

- Aller dans le dossier android/ de l'application
- Créez un fichier local.properties avec dedans

  ```
  sdk.dir = { chemin vers le SDK android }
  ```

  Par example, sur Linux le chemin ressemblera à celà :

  ```
  sdk.dir = /home/USER/Android/sdk
  ```

  Sur macOS:

  ```
  sdk.dir = /Users/USER/Library/Android/sdk
  ```

Index