

# STR

## SIForms : Application de saisie de formulaires

Auteurs :

**Adèle DESMAZIERES**

**Léa EL ABOUD**

**Kevin XU**

Destinataires :

**Paul GUYOT**

**Julien KERLIDOU**

**Cédric BESSE**

**Lamia LARAQUI**

Responsable : **Kevin XU**

Type : **STR**

Date : **13/12/2024**

Nombre de pages : **28**

Statut : **Initial**

Adresse : [typst.app/PISTL](https://typst.app/PISTL)

## Historique

Version	Date	Modification	Rédacteur
v.0.1	14/11/2024	Plan, objets métiers, services	Kevin XU
v.0.2	21/11/2024	Diagrammes d'états des objets métiers	Adèle DESMAZIERES
v.0.3	22/11/2024	IHM détaillé	Léa EL ABOUD
v.0.4	22/11/2024	Plan d'intégration et tests d'intégration	Kevin XU
v.0.5	22/11/2024	Sous-systèmes, services, Workflow, déploiement, conclusion	Adèle DESMAZIERES
v.0.6	22/11/2024	Diagramme de classes	Léa EL ABOUD
v.0.7	27/11/2024	Lexique	Adèle DESMAZIERES
v.0.8	29/11/2024	Objets métiers	Adèle DESMAZIERES
v.0.9	01/12/2024	Services et objets métiers	Adèle DESMAZIERES
v.1.0	01/12/2024	Diagramme de classes	Léa EL ABOUD
v.1.1	06/12/2024	Objets métiers et diagramme de classes	Adèle DESMAZIERES, Léa EL ABOUD
v.1.2	09/12/2024	Services, workflow, technologies	Adèle DESMAZIERES
v.1.3	13/12/2024	Objets métiers, services, versions des techno	Adèle DESMAZIERES
v.1.4	13/12/2024	Comparaison des techno	Kevin XU
v.1.5	13/12/2024	Objets métiers et diagramme de classes	Léa EL ABOUD

## Table des matières

<b>1. Objectif .....</b>	<b>3</b>
<b>2. Objets métiers .....</b>	<b>3</b>
2.1. Liste détaillée des objets métiers .....	3
2.1.1. Objet métier : Modèle de formulaire .....	3
2.1.2. Objet métier : Formulaire .....	3
2.1.3. Objet métier : Document attaché à un formulaire .....	5
2.1.4. Objet métier : Compte utilisateur .....	6
2.1.5. Objet métier : Session .....	6
2.2. Identification des sous-systèmes .....	6
2.3. Diagramme de classes .....	7
2.3.1. Diagramme de classe base de données et serveur. ....	7
2.3.2. Diagramme de classe application. ....	8
<b>3. Services .....</b>	<b>8</b>
3.1. Services du sous-système : Application mobile .....	8
3.1.1. Gestion de compte .....	8
3.1.2. Gestion de session .....	8
3.1.3. Accès à un nouveau formulaire pré-rempli .....	8
3.1.4. Accès à un nouveau formulaire vierge .....	9
3.1.5. Remplissage de formulaires .....	9
3.1.6. Gestion des formulaires .....	9
3.1.7. Envoi asynchrone .....	9
3.1.8. Protocole d'échange et d'acquittement des données .....	9

<b>4. Workflow .....</b>	<b>9</b>
4.1. Workflow de la gestion de ses formulaires et entrées .....	9
<b>5. Architecture technique .....</b>	<b>10</b>
5.1. Découpe en composants .....	10
5.2. Choix des technologies .....	11
5.2.1. Solution pour le frontend .....	11
5.2.2. Technologie de l'application mobile .....	11
5.2.3. Technologie du serveur .....	11
5.2.4. Technologie des données .....	12
5.2.5. Format des entrées des formulaires .....	12
5.3. Synthèse des choix des technologies .....	12
<b>6. Plan de déploiement et mise en exploitation .....</b>	<b>13</b>
6.1. Initialisation de la configuration .....	13
6.2. Pérennisation .....	13
6.3. Sécurisation .....	13
<b>7. Nomenclature de l'IHM .....</b>	<b>13</b>
7.1. Aspect de l'IHM .....	13
7.2. Ergonomie de l'IHM et navigation .....	14
<b>8. Plan d'intégration .....</b>	<b>19</b>
8.1. Dépendance entre composants .....	19
8.2. Tests d'intégration .....	20
8.2.1. Test d'envoi des formulaires .....	20
8.2.2. Test sur le téléchargement des formulaires .....	21
8.2.3. Test de création de compte et login .....	22
8.2.4. Test de gestion des formulaires .....	23
<b>9. Plannification .....</b>	<b>24</b>
9.1. Work Breakdown Structure (WBS) .....	24
9.2. Tâches .....	24
9.2.1. Conception .....	25
9.2.2. Réalisation .....	25
9.2.3. Tests .....	25
9.2.4. Déploiement .....	26
9.2.5. Gestion .....	26
<b>10. Conclusion .....</b>	<b>26</b>
<b>11. Annexes .....</b>	<b>27</b>
11.1. Lexique .....	27
11.1.1. Termes fonctionnels .....	27
11.1.2. Termes techniques .....	27
11.1.3. Technologies .....	27
11.2. Bibliographie .....	28
11.2.1. Sources pour la comparaison des technologies .....	28
11.2.2. Documentation des technologies .....	28

## 1. Objectif

Ce projet vise à développer un outil permettant aux techniciens de saisir et de soumettre des réponses à des formulaires de manière asynchrone. Les Spécifications Techniques de Réalisation (STR) fixent les principaux choix architecturaux et techniques, et détaillent la nomenclature des composants à réaliser et leurs interfaces.

## 2. Objets métiers

### 2.1. Liste détaillée des objets métiers

#### 2.1.1. Objet métier : Modèle de formulaire

**Nom :** Model

**Description :** Un modèle de formulaire est un objet représentant un ensemble de questions d'un formulaire. Deux formulaires du même modèle ont les mêmes questions et les mêmes règles qui régissent l'apparition de ces questions. Il existe pour le moment 3 modèles de formulaires :

- le formulaire de visite technique,
- le formulaire d'installation,
- le formulaire d'informations du site d'intervention.

L'outil doit pouvoir intégrer de nouveaux modèles de formulaire aisément, c'est pourquoi on ne fait pas une classe par modèle, mais une classe qui représente les modèles en général.

#### Attributs :

- id: integer : Identifiant unique du modèle.
- name: string : Nom du modèle, sert d'identifiant pour réunir les différentes versions d'un même modèle (exemple : "visite technique"). La paire (name, version) est unique.
- version: integer : La version du modèle.
- date: datetime : La date de création de cette version.
- data: string : Les données des questions en format JSON.

#### 2.1.2. Objet métier : Formulaire

**Nom :** Form

**Description :** Un formulaire est un ensemble de réponses écrites par un technicien, suivant les questions d'un modèle de formulaire. C'est un document conçu pour collecter des informations auprès des utilisateurs. Il peut contenir des documents attachés à la réponse. Il a un lien vers son Model de questions et vers son auteur Account. Du côté application, il est dans un état Status spécifique représentant l'avancement de son envoi au serveur. Du côté serveur, il peut-être pré-rempli par un employé de Smart Impulse ou rempli complètement par un technicien.

#### Attributs :

- id: integer : Identifiant unique de l'entrée de formulaire du côté backend.
- id\_client: integer : Identifiant unique de l'entrée de formulaire dans l'application.
- data: string : Données des réponses. Peut contenir des identifiants de documents attachés au formulaire.

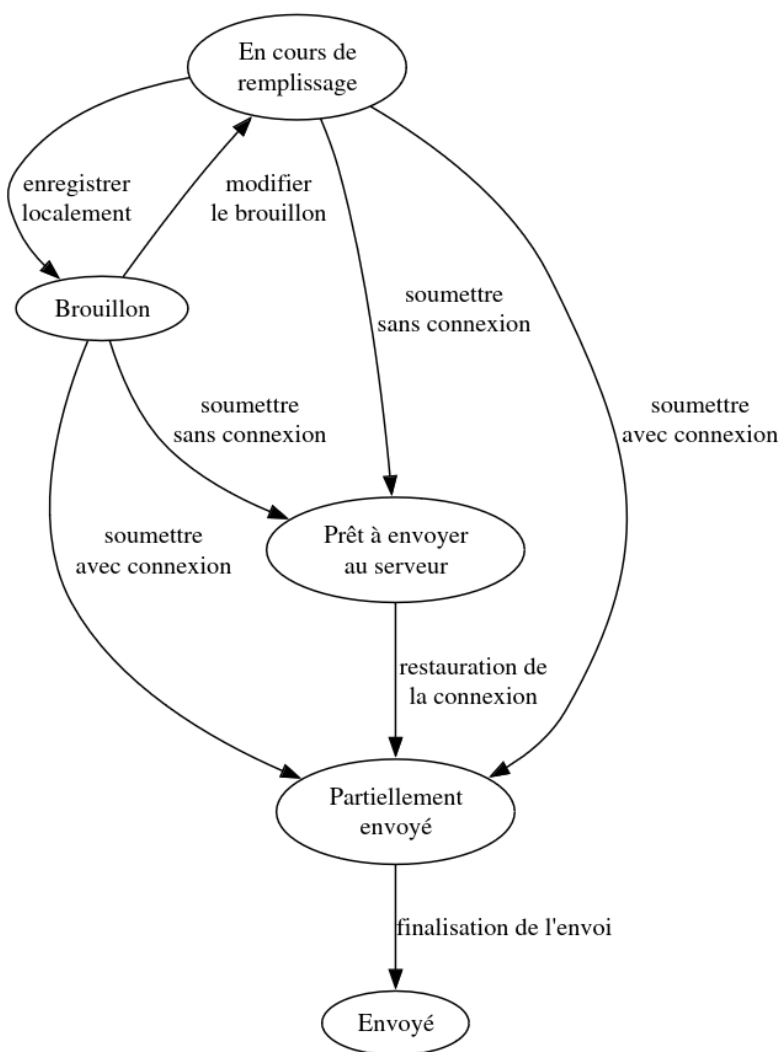
- `created_at`: `datetime` : Date de création de l'entrée.
- `sent_at`: `datetime` : Date de réception du formulaire par le serveur. Attribut présent seulement du côté backend.
- `status`: `Status` : État d'envoi de l'entrée de formulaire. Cet attribut est implémenté uniquement dans l'application, car le backend sait déjà quel est l'état de l'envoi en fonction de ce qu'il a reçu.
- `is_complete`: `boolean` : Indique si le formulaire a été pré-rempli par Smart Impulse avec `false`, ou complété et renvoyé par un technicien avec `true`. Attribut présent uniquement dans la partie backend.

De plus dans les données brutes en JSON d'entrée de formulaire, une réponse peut pointer vers un identifiant de `Document`, pour représenter que ce document est attaché à cette réponse du formulaire.

#### Attributs relationnels :

- `model`: `integer` : Identifiant du modèle suivi par le formulaire.
- `author`: `integer` : Identifiant du compte de l'auteur de la réponse. Cet attribut est implémenté uniquement dans la partie backend (le serveur et la base de données) et pas dans l'application mobile, car dans celle-ci les formulaires accessibles appartiennent tous à l'utilisateur actuellement authentifié. Lorsqu'un formulaire est pré-rempli par un employé de Smart Impulse, l'auteur du formulaire est vide.

Diagramme d'état d'un formulaire du côté application mobile. Les processus d'échanges de messages entre l'application et le serveur sont décrits dans la [Partie 3.1.8](#).



**Status :** Une énumération des états possibles d'un formulaire. Présent uniquement dans l'application mobile. Elle peut prendre une des valeurs `string` suivantes :

- `draft` : Le technicien a reçu un formulaire soit par un lien, soit par sélection parmi la liste des modèles. Qu'il ait saisi une réponse ou non, l'instance du formulaire sera enregistrée dans sa liste de formulaires.
- `waiting` : Après que le technicien a cliqué sur « Submit », le formulaire est envoyé quand la connexion est rétablie sans autre interaction du technicien.
- `sending` : Le formulaire a été partiellement envoyé au serveur et continue son envoi tant que la connexion le permet.
- `sent` : Le formulaire a été entièrement envoyé au serveur.

### 2.1.3. Objet métier : Document attaché à un formulaire

**Nom :** Document

**Description :** Des documents peuvent être attachés à une réponse de formulaire, pour répondre à une question spécifique. Cet objet métier ne change pas d'état, c'est pourquoi nous n'avons pas représenté son diagramme d'état.

**Attributs :**

- `id: integer` : Identifiant du document unique dans le serveur.

- `id_client: integer` : Identifiant du document unique au sein de son formulaire.
- `path: string` : Chemin vers les données brutes du document. C'est un chemin vers le fichier enregistré localement dans l'appareil de l'utilisateur, attribut présent uniquement dans l'application.
- `data: binary large object` : Contenu du document, uniquement dans le serveur.

#### 2.1.4. Objet métier : Compte utilisateur

**Nom :** Account

**Description :** Le compte d'un utilisateur rassemble les données d'un technicien sous une seule identité, associée à une adresse email unique.

**Attributs :**

- `id: integer` : Identifiant unique du compte.
- `first_name: string` : Prénom l'utilisateur.
- `last_name: string` : Nom de famille de l'utilisateur.
- `email: string` : Adresse email de l'utilisateur.
- `phone: string` : Numéro de téléphone de l'utilisateur.
- `password: string` : Mot de passe chiffré, présent uniquement dans la partie backend.
- `lang: string` : Langue de préférence pour l'affichage des formulaires.

#### 2.1.5. Objet métier : Session

**Nom :** Session

**Description :** Représente une session d'utilisation de l'application par un utilisateur. La session est créée quand l'utilisateur s'authentifie, et est détruite quand l'utilisateur se déconnecte de son compte. Cette session ne sera représentée que dans la partie backend. Par défaut, la session expire 30 jours après la dernière activité.

**Attributs :**

- `token: string` : Token de session permettant de s'assurer de l'identité de l'envoyeur lors des échanges du backend avec le frontend.
- `created_at: datetime` : Date de début de session.
- `expires_at: datetime` : Date de fin de session.

**Attribut relationnel :**

- `user: account` : Utilisateur qui a démarré cette session.

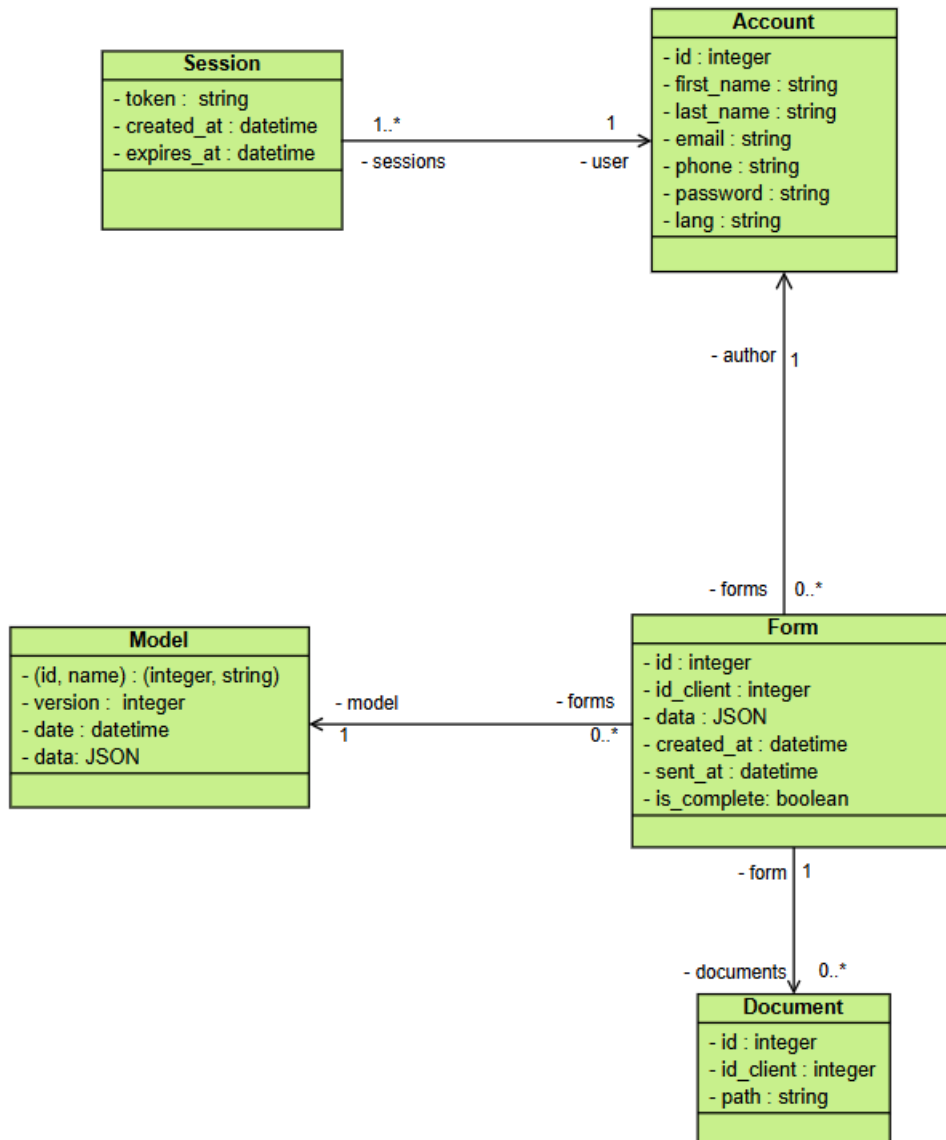
## 2.2. Identification des sous-systèmes

Nous avons identifié les sous-systèmes suivants :

- **L'application mobile** : Elle englobe tous les objets métiers sauf `Session`. Elle a accès aux données locales de SIForms dans l'appareil mobile du technicien. Elle communique avec le serveur par réseau Internet.
- **Le serveur** : Il englobe tous les objets métiers sauf `Status`. Il a accès aux données de la base de données du backend. Le composant backend est un composant « bouchon » qui sera remplacé à terme par un serveur de Smart Impulse.

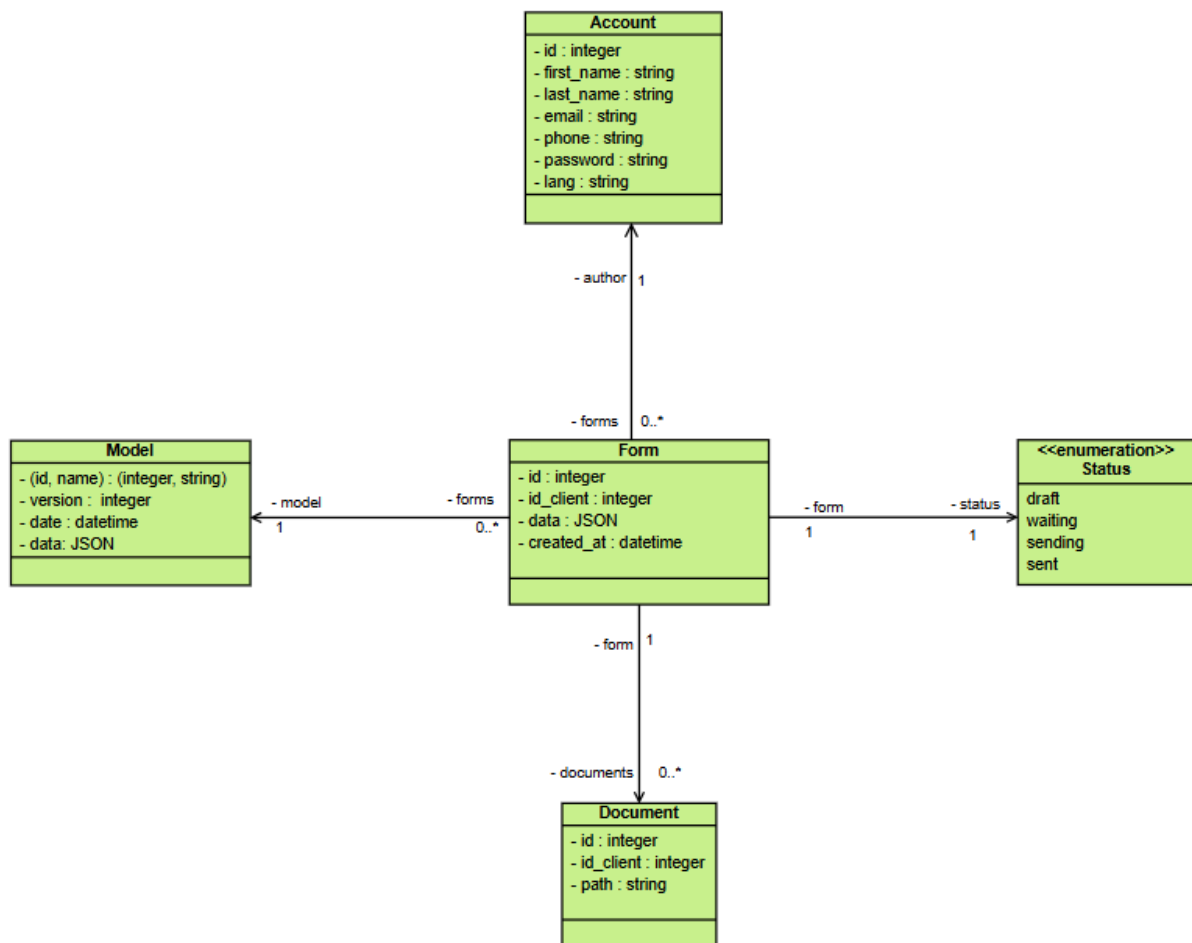
## 2.3. Diagramme de classes

### 2.3.1. Diagramme de classe base de données et serveur.





### 2.3.2. Diagramme de classe application.



## 3. Services

### 3.1. Services du sous-système : Application mobile

#### 3.1.1. Gestion de compte

Ce service permet aux techniciens de créer un compte sur l'application et de s'y authentifier pour accéder aux fonctionnalités.

- **Création de compte** : crée un objet `Account`.
- **Modification de la langue des formulaires** : modifie l'attribut `lang` de l'objet `Account`.

#### 3.1.2. Gestion de session

- **Authentification** : crée un objet `Session`.
- **Déconnexion** : supprime un objet `Session`.

#### 3.1.3. Accès à un nouveau formulaire pré-rempli

Ce service permet d'ajouter un nouveau formulaire pré-rempli à sa liste en suivant un lien ou un QRcode. Cela récupère du serveur vers l'application un objet `Form` et éventuellement des `Document`, en ajoutant le formulaire à la liste de l'utilisateur et en indiquant l'utilisateur courant comme auteur de `Form`.

### 3.1.4. Accès à un nouveau formulaire vierge

Ce service permet de générer un formulaire vierge à partir d'un modèle. L'application lit le modèle enregistré localement, et crée un objet métier `Form` à partir de ce modèle, en indiquant l'utilisateur courant comme auteur du `Form`.

### 3.1.5. Remplissage de formulaires

Ce service permet au technicien de compléter les formulaires en saisissant les informations requises et en y attachant des fichiers multimédias. Ce service modifie l'attribut `data` de l'objet métier `Form`.

### 3.1.6. Gestion des formulaires

- **Consulter une entrée de formulaire** : Permet de consulter en lecture seule une réponse à un formulaire précédemment envoyé. Ne modifie pas d'objet métier.
- **Modifier une entrée de formulaire** : Modifie les données d'un `Form` en brouillon.
- **Supprimer un formulaire envoyé** : Supprime localement un formulaire envoyé de l'appareil mobile du technicien. Permet de libérer de la mémoire.

### 3.1.7. Envoi asynchrone

Ce service permet, lorsque l'utilisateur est hors-ligne, de **demandeur l'envoi différé** des formulaires. Ils seront ensuite automatiquement envoyés une fois la connexion rétablie, sans autre action du technicien. Ce service modifie l'état `status` d'un objet métier `Form` en le passant de `draft` à `waiting` quand le technicien vient de soumettre le formulaire, puis à `sending` quand l'échange de données avec le serveur a débuté, puis à `sent` quand l'envoi est totalement complété.

### 3.1.8. Protocole d'échange et d'acquittement des données

Les méta-données du formulaire et de son modèle seront d'abord envoyées, puis l'entrée textuelle du formulaire, et enfin les documents attachés au formulaire. L'envoi suit le protocole TCP avec une connexion HTTPS.

L'entrée textuelle et les données brutes des documents seront découpées en plusieurs blocs appelés pour faciliter l'envoi et éviter la perte de données, tandis que les documents seront envoyés en un seul bloc sérialisé.

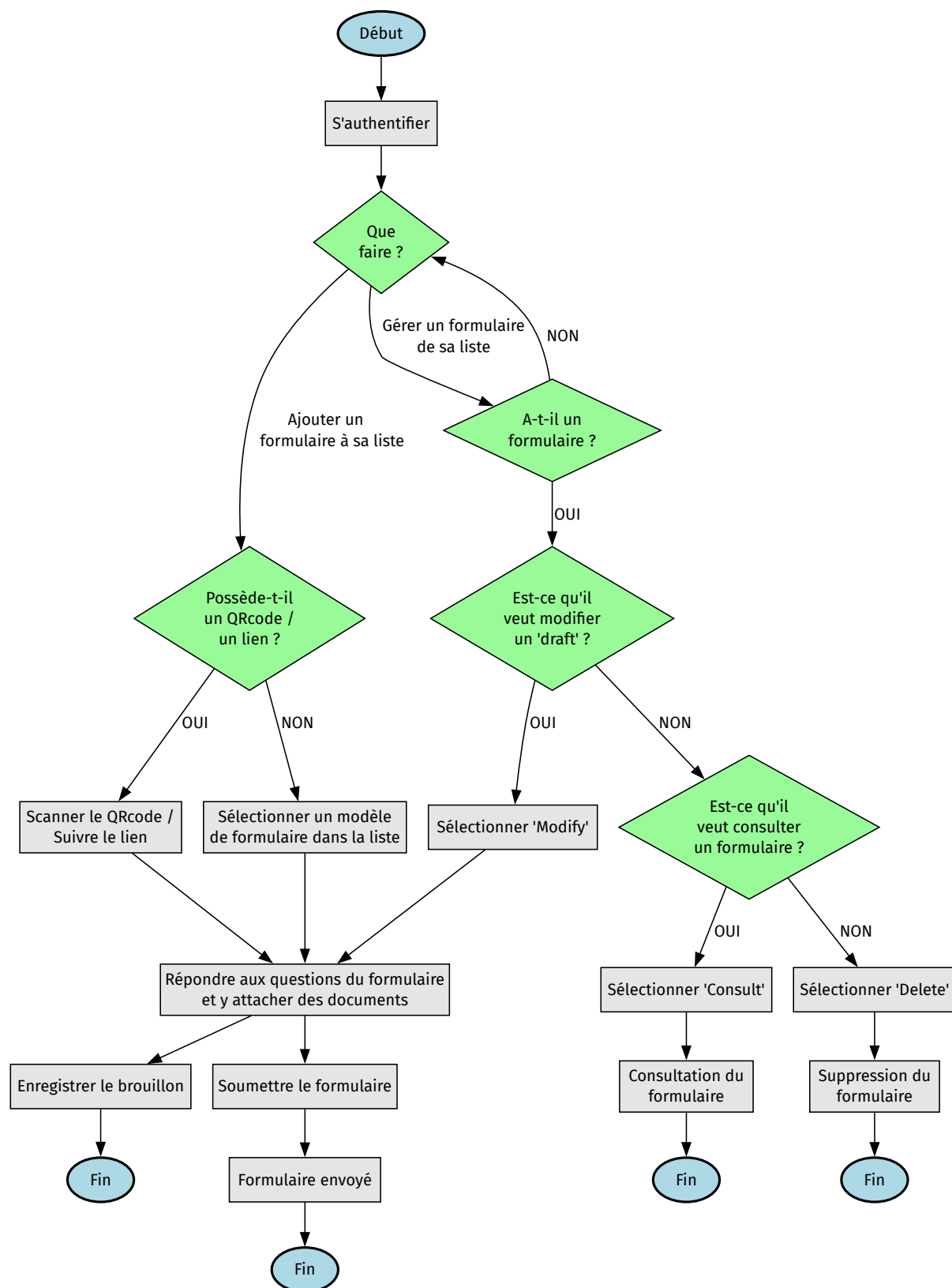
Si un bloc n'est pas reçu par le serveur à cause de connexion instable, il demande son nouvel envoi à l'application mobile. Si la connexion est interrompue au cours de l'envoi, le serveur conservera les données déjà reçues et attendra un rétablissement de la connexion pour que l'application mobile lui envoie le reste.

Au cours des échanges, l'objet métier `Form` du côté application aura un statut `sending`, qui sera changé en `sent` quand toutes les trames auront été reçues par le serveur. De plus, le serveur mettra à jour l'attribut `sent_at` de `Form` pour conserver la date de réception du formulaire.

## 4. Workflow

### 4.1. Workflow de la gestion de ses formulaires et entrées

- **Acteur** : Un technicien.
- **Début** : Le technicien a un compte créé dans SIForms mais n'est pas encore authentifié. L'application est ouverte.
- **Fin** : Les fins renvoient l'utilisateur à l'étape « Que faire ? ».



## 5. Architecture technique

### 5.1. Découpe en composants

Nous avons isolé les composants suivants, nécessaires à la conception de l'outil :

- **Le composant frontend** d'application mobile, permettant à un technicien de saisir et envoyer ses réponses à des formulaires.

- **Le composant backend**, composé d'un serveur permettant d'envoyer des formulaires vierges ou pré-remplis et d'enregistrer les réponses dans une base de données accessible par Smart Impulse. Le composant backend est un composant « bouchon » qui sera remplacé à terme par un serveur de Smart Impulse.

## 5.2. Choix des technologies

### 5.2.1. Solution pour le frontend

Application mobile ✓	Progressive Web App ✗	Site web ✗
<ul style="list-style-type: none"> <li>• permet l'accès hors-ligne</li> <li>• simplifie la soumission asynchrone</li> <li>• simplifie la mise à disposition des formulaires précédents</li> </ul>	<ul style="list-style-type: none"> <li>• rien à installer pour les techniciens</li> <li>• fonctionne sur n'importe quelle plateforme avec un navigateur</li> <li>• permet de consulter des formulaires hors-ligne grâce au cache</li> <li>• envoi des données en asynchrone compliqué</li> </ul>	<ul style="list-style-type: none"> <li>• rien à installer pour les techniciens</li> <li>• fonctionne sur n'importe quelle plateforme avec un navigateur</li> <li>• envoi des données en asynchrone compliqué</li> <li>• pas d'accès hors ligne</li> </ul>

Pour le composant frontend, nous avons donc choisi de développer une application mobile, qui simplifiera la gestion de l'historique des réponses et l'envoi asynchrone.

### 5.2.2. Technologie de l'application mobile

**Contraintes** : L'application doit être cross-plateforme pour Android et iOS. Nous avons donc ignoré le développement natif, pour ne pas devoir programmer deux l'application avec les langages natifs.

ReactNative ✓	Flutter ✗
<ul style="list-style-type: none"> <li>• langages : React et JavaScript</li> <li>• beaucoup de bibliothèques existantes</li> <li>• grande communauté et plein de tutoriels</li> <li>• moins performant mais probablement suffisant</li> <li>• déjà utilisé chez Smart Impulse</li> <li>• on souhaite se former à cette technologie</li> <li>• gestion compliquée des dépendances mais le module expo simplifie cela</li> </ul>	<ul style="list-style-type: none"> <li>• langage : Dart</li> <li>• plus performant</li> <li>• moins de ressources car communauté moins développée mais en croissance</li> <li>• verbeux quand on souhaite rentrer dans les détails</li> </ul>

### 5.2.3. Technologie du serveur

**Contraintes** : Serveur en Python imposé par Smart Impulse.

Django ✓	Flask ✗	Ruby on Rails ✗
<ul style="list-style-type: none"> <li>framework adapté aux gros projets</li> <li>meilleure sécurité</li> <li>meilleure maintenance</li> <li>format pour les formulaires</li> <li>utilisé à Smart Impulse</li> <li>adapté à PostgreSQL</li> <li>Paul Guyot aimerait réduire l'utilisation de Django à Smart Impulse mais nous l'a quand même conseillé</li> </ul>	<ul style="list-style-type: none"> <li>framework minimaliste</li> <li>prise en main plus rapide</li> <li>environnement flexible (bibliothèques)</li> <li>plus performant</li> <li>moins sécurisé</li> </ul>	<ul style="list-style-type: none"> <li>facile à développer</li> <li>bon environnement test</li> <li>difficile de créer une API avec</li> <li>manque de documentation</li> </ul>

### 5.2.4. Technologie des données

#### Données relationnelles : PostgreSQL

Nous avons sélectionné PostgreSQL car c'est une base de données (BD) relationnelle déjà utilisée à Smart Impulse, et recommandée par les professionnels. De plus, Django communique très bien avec cette BD.

#### Données brutes : enregistrées sur serveur local

Les données brutes (contenu des formulaires et documents attachés) seront enregistrées en dur dans le serveur. En cas d'avancement suffisant du projet, nous envisageons une migration des données dans un cloud (comme AWS), offrant une plus grande place de stockage.

### 5.2.5. Format des entrées des formulaires

JSON ✓	XML ✗	YAML ✗
<ul style="list-style-type: none"> <li>utilisé actuellement par Smart Impulse</li> <li>très répandu</li> <li>lisible par un humain</li> <li>rapide pour requêter un champ spécifique</li> <li>souvent utilisé pour l'échange de données</li> </ul>	<ul style="list-style-type: none"> <li>très répandu</li> <li>souvent utilisé pour l'échange de données</li> <li>très verbeux</li> <li>syntaxe complexe</li> </ul>	<ul style="list-style-type: none"> <li>très lisible par un humain</li> <li>plus souvent utilisé pour de la configuration</li> </ul>

Le format des modèles de formulaires est imposé par Smart Impulse comme étant en JSON Schema légèrement modifié pour s'accorder à leurs besoins.

## 5.3. Synthèse des choix des technologies

- **Application mobile** : ReactNative 0.75 (émulateur : Android Studio)
- **Serveur backend** :
  - Python 3.12
  - Django 4.2
- **Base de données** : PostgreSQL 16
- **Données brutes** :
  - Modèles de formulaire (JSON Schema)

- Entrées de formulaires (JSON)
- Documents attachés (PNG, JPEG/JPG, BMP...)

## 6. Plan de déploiement et mise en exploitation

L'application sera déployée sur les magasins d'application de iOS et Android. Le serveur maquette sera déployé en tant que composant bouchon chez Smart Impulse.

### 6.1. Initialisation de la configuration

- **Application mobile** : Déployée via un fichier APK pour Android et via un profil de déploiement pour iOS.
- **Backend serveur** : Hébergé sur un serveur local (ex : une machine sous Ubuntu 22.04) et comprenant :
  - API backend (Python/Django)
  - Base de données (PostgreSQL)

### 6.2. Pérennisation

- **Monitoring** : Utilisation d'outils légers comme **htop** et journaux système pour surveiller la charge et détecter les erreurs.
- **Sauvegardes** : Planification de sauvegardes locales automatisées via **cron** (hebdomadaires) avec export des bases de données et synchronisation vers un disque externe.
- **Mises à jour** : Processus manuel pour appliquer les correctifs de sécurité et les nouvelles versions.
- **Modularité** : Nous veillons à produire du code propre et modulaire, séparés en composants distincts, pour faciliter le remplacement ou la migration de parties du projet, notamment dans l'optique de remplacer le serveur et la base de données peu de temps après sa livraison.

### 6.3. Sécurisation

- **Transferts sécurisés** : HTTPS activé et données cryptées.
- **Authentification des utilisateurs** : Stockage des mots de passe hashés et sessions sécurisées via tokens.

## 7. Nomenclature de l'IHM

L'**Interface Homme-Machine (IHM)** de l'application SiForms est conçue pour faciliter la saisie et la gestion de formulaires par les techniciens, et pour permettre son utilisation hors ligne. L'application mobile sera déployée sur les systèmes Android et iOS, et sera accessible exclusivement depuis des appareils mobiles (personnels ou de travail) des techniciens.

### 7.1. Aspect de l'IHM

L'IHM est conçue pour être intuitive, réactive et accessible sur mobile (Android et IOS). Elle permet aux techniciens de :

- **Naviguer facilement entre les différents modules** grâce à des onglets clairs pour les formulaires soumis, en attente, ou non soumis.
- **Accéder aux formulaires même hors ligne**, avec des statuts clairs pour chaque étape (brouillon, attente de connexion, envoi partiel).
- **Paramétrer leur expérience utilisateur** en sélectionnant la langue des formulaires.

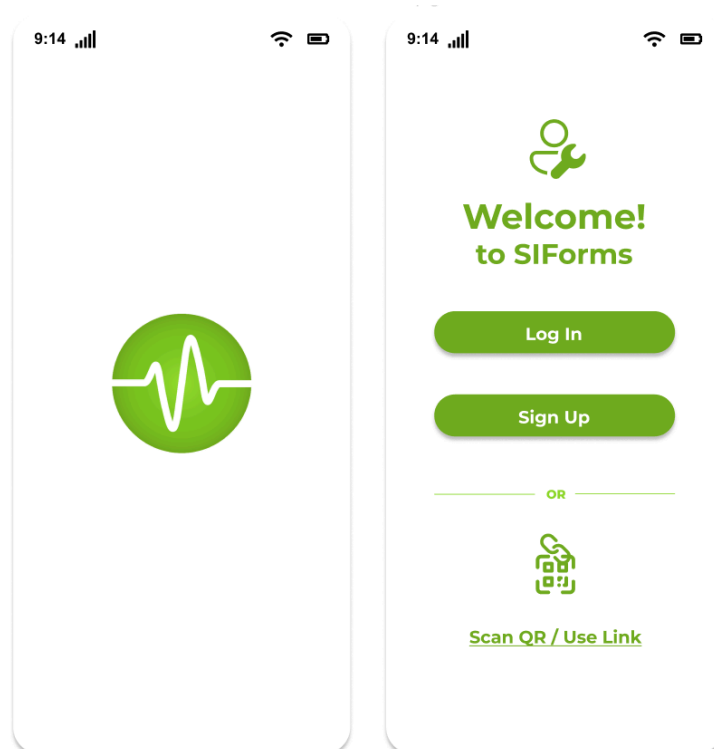
## 7.2. Ergonomie de l'IHM et navigation

L'IHM de **SIForms** suit des principes d'ergonomie mobile, assurant une expérience utilisateur fluide :

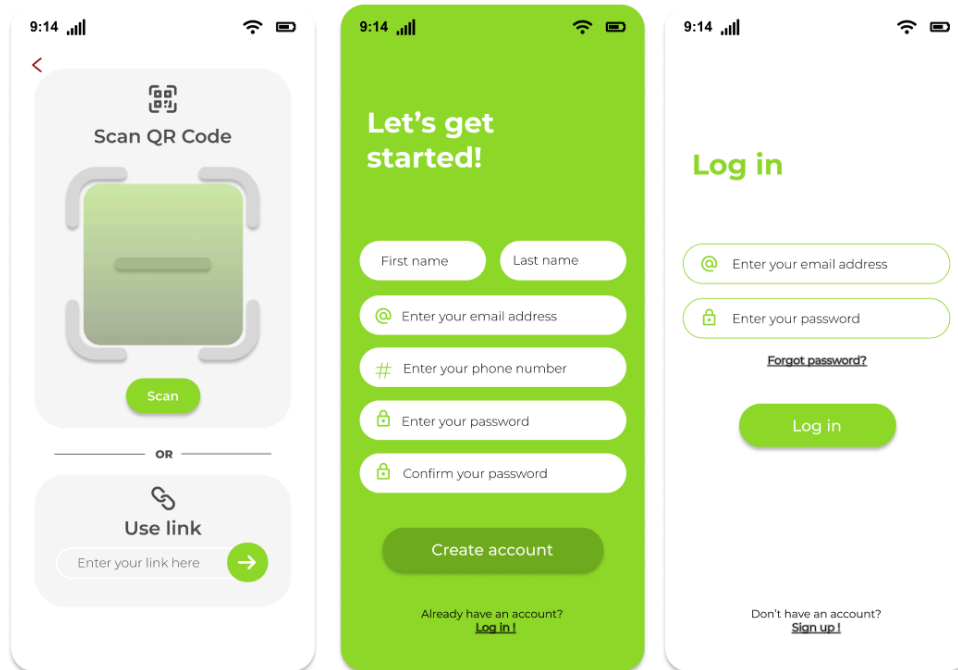
1. **Navigation simple** : La structure de l'application se base sur un design en onglets et en écrans distincts, avec des boutons d'action facilement identifiables (par exemple, « Nouveau site », « Envoyer », « Brouillon »).
2. **Retours d'état en temps réel** : L'application affiche des notifications (snackbar ou pop-up) pour confirmer chaque action importante (ex. sauvegarde en brouillon, envoi réussi).
3. **Accessibilité hors ligne** : L'IHM offre des alertes pour informer l'utilisateur de l'état de connexion et assure l'accès aux formulaires déjà sauvegardés.
4. **Guidage utilisateur** : Les boutons et icônes sont clairs et le parcours utilisateur minimise le nombre d'actions nécessaires pour remplir et soumettre un formulaire.

Notre IHM est composée des écrans suivants:

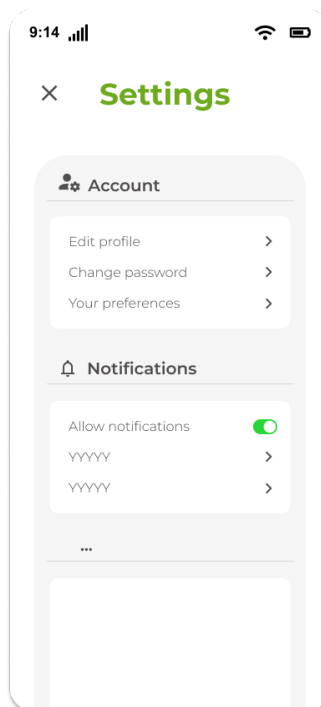
- **Écran splash puis écran d'accueil** : Permet l'accès aux options de connexion ou d'inscription.



- **Écran de connexion** : Permet de se connecter si un compte existe, en entrant son adresse e-mail et son mot de passe. Inclut une option de récupération pour mot de passe oublié ainsi qu'une option de création de compte pour les nouveaux utilisateurs.
- **Écran de création de compte**: Pour créer un compte en fournissant un prénom, nom, adresse e-mail, mot de passe et numéro de téléphone.
- **Écran de scan QR**: Permet de scanner un code QR ou de saisir manuellement un lien pour accéder à un formulaire pré-rempli. Une création de compte ou une authentification sera requise pour être redirigé vers le formulaire spécifique.

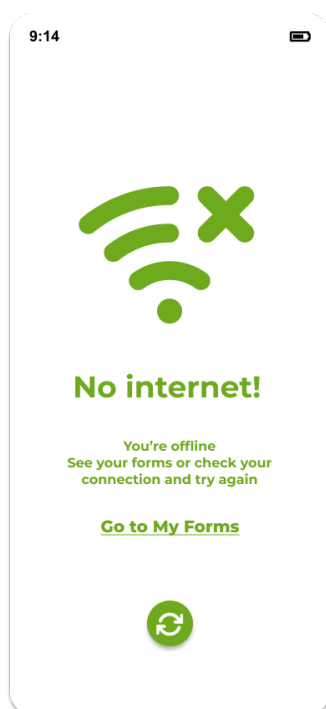


- **Écran de paramètres:** Pour gérer les préférences utilisateurs, telles que les notifications et la langue des formulaires.

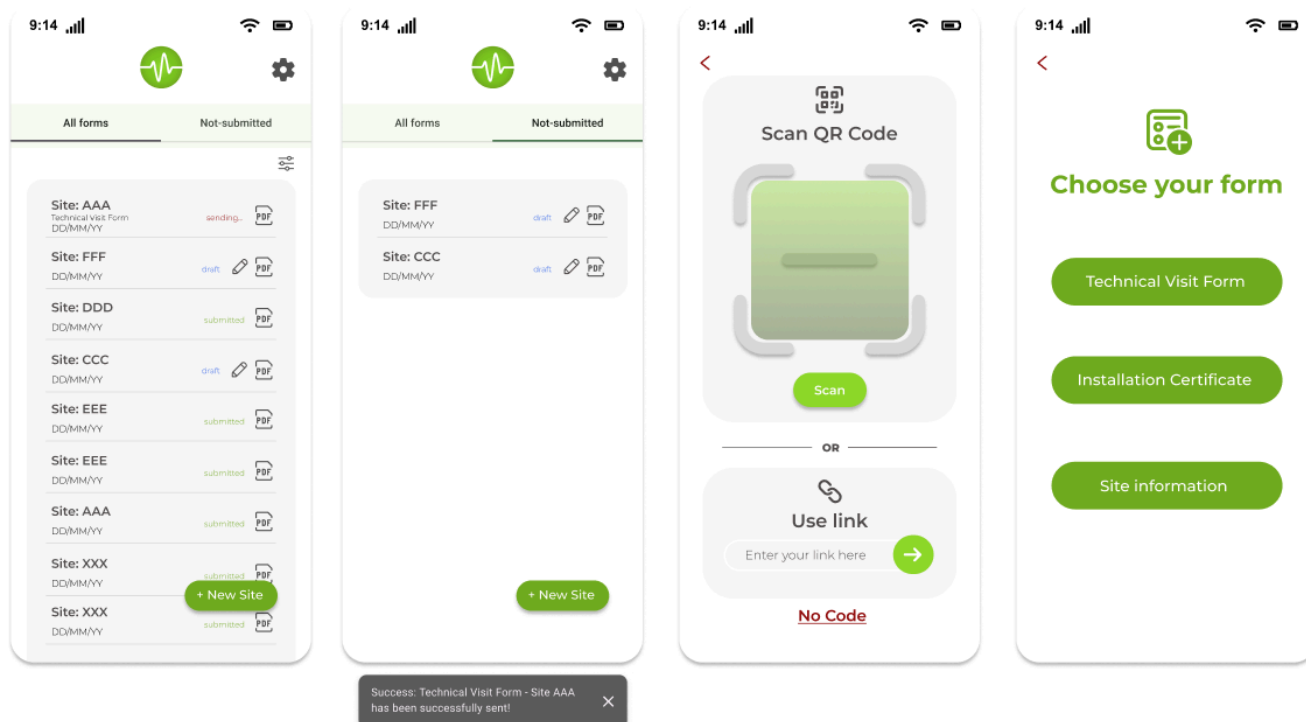


- **Écran hors-ligne :** Pour informer l'utilisateur qu'il est hors ligne tout en lui permettant d'accéder aux formulaires déjà enregistrés ou de réessayer la connexion une fois celle-ci rétablie.

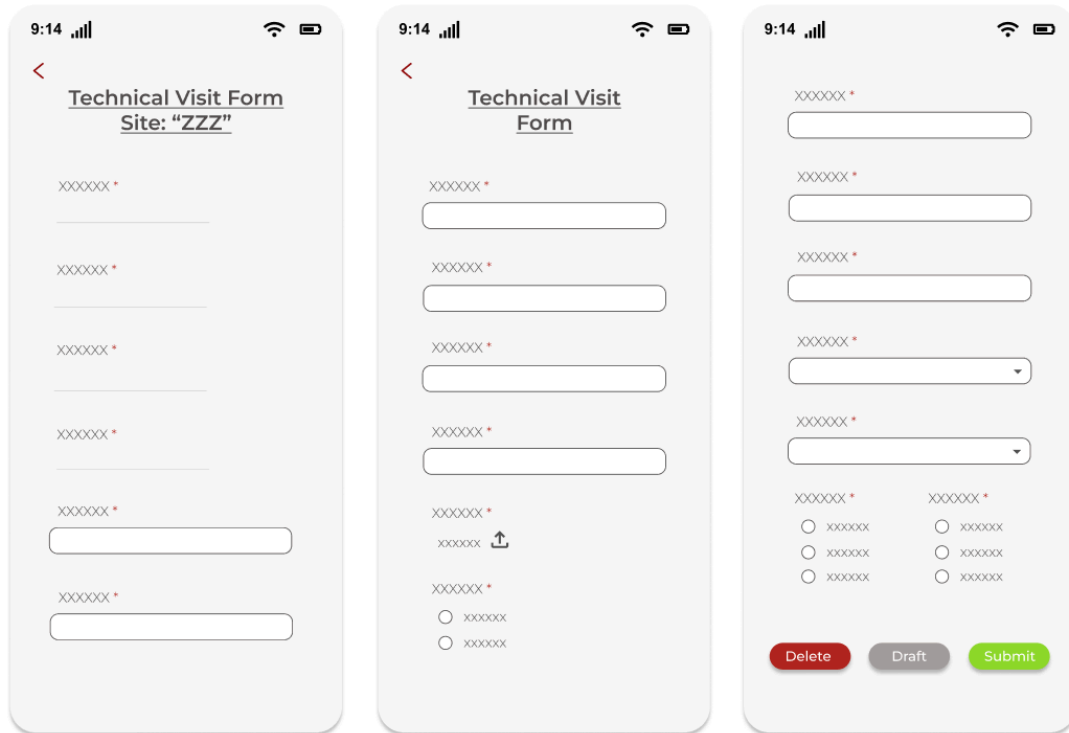




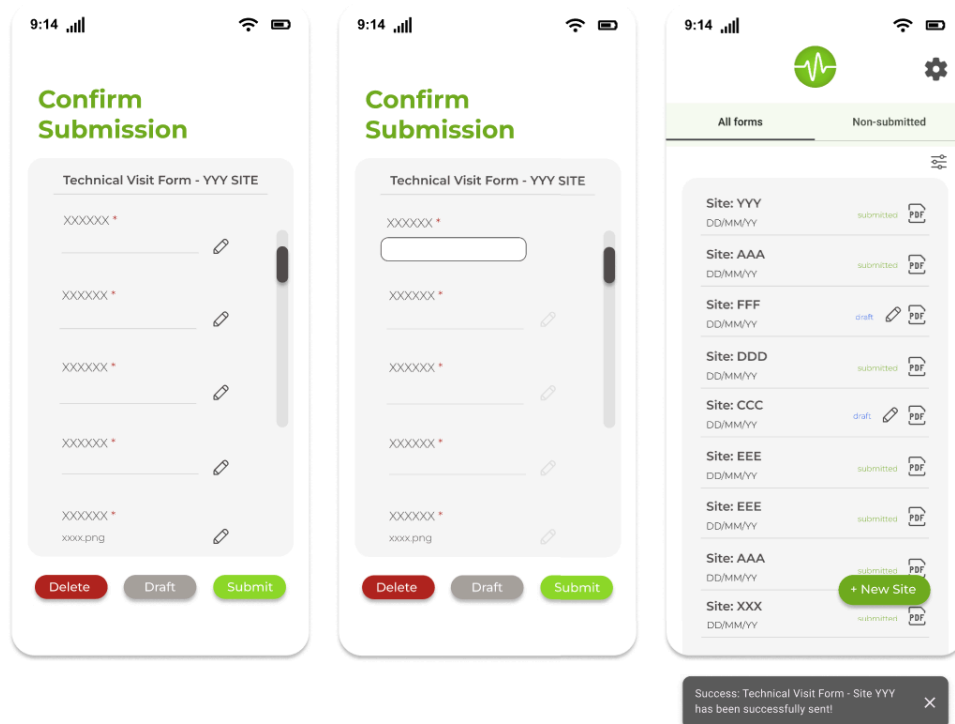
- **Écrans des formulaires** : La page principale présente deux onglets : « All forms » pour tous les formulaires et « Not submitted » pour ceux qui n'ont pas été soumis. En sélectionnant « New site », l'utilisateur accède à une page permettant de scanner un code QR ou de saisir un code, avec une option « No Code » pour accéder directement à la sélection du type de formulaire.



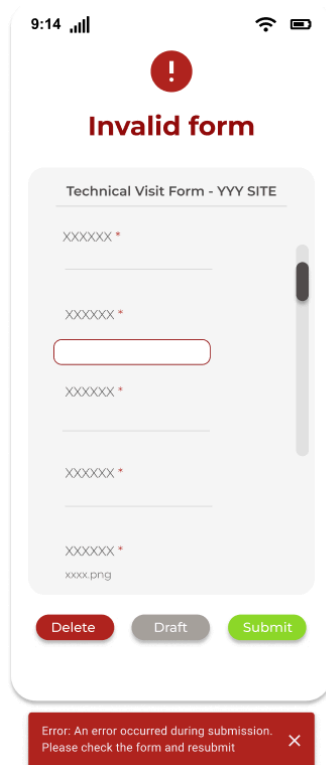
- **Écrans de remplissage d'un formulaire** : Pour répondre à un formulaire (pré-rempli pour un site ou vierge). La réponse peut être sauvegardée en tant que brouillon, soumise, ou supprimée.



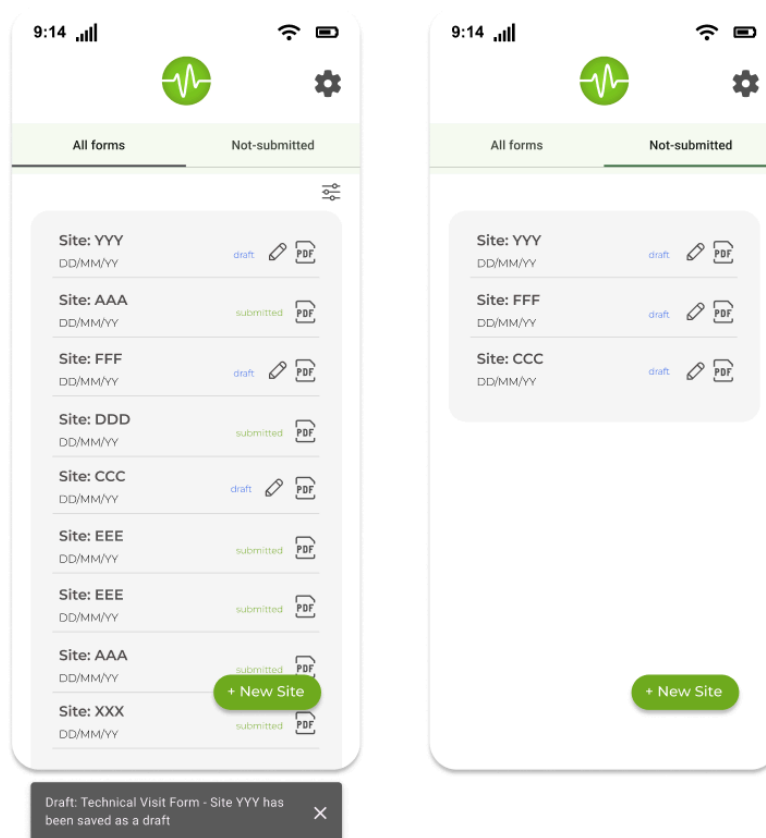
- **Écrans de confirmation et soumission d'une réponse :**



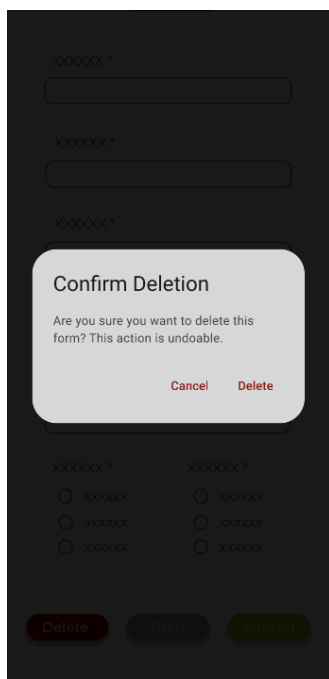
- **Écran de formulaire invalide :** Pour corriger les erreurs afin de pouvoir soumettre la réponse.



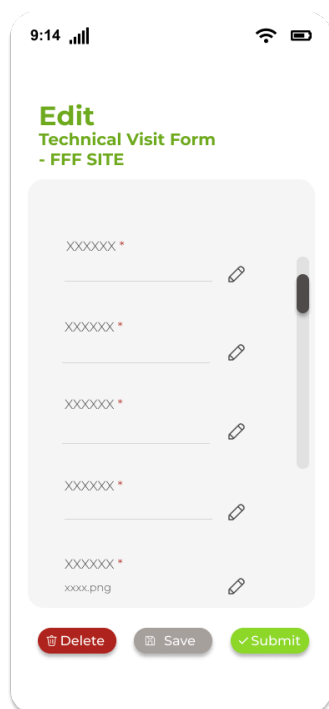
- **Écrans d'enregistrement d'une réponse** : Ajout de la nouvelle réponse sauvegardée dans les deux onglets.



- **Écran de suppression d'une réponse** : Pour supprimer une réponse enregistrée ou en cours de remplissage.



- **Écran de modification d'une réponse enregistrée:** Pour modifier un ou plusieurs champs d'une réponse enregistrée.



Cette structure garantit une navigation intuitive et une facilité d'utilisation adaptée aux techniciens, même dans des environnements à faible connexion.

## 8. Plan d'intégration

### 8.1. Dépendance entre composants

L'application mobile reçoit et envoie des données au serveur. Notamment l'application peut recevoir des modèles de formulaires, ainsi que des formulaires pré-remplis et elle renvoie au serveur des formulaires. Les deux composants s'échangent également des informations relatives aux comptes utilisateurs.

L'application mobile enregistre des entrées de formulaires et documents attachés dans l'appareil mobile de l'utilisateur, dans un dossier créé à cet effet.

Le serveur enregistre les données relationnelles et données brutes dans une base de données PostgreSQL.

## 8.2. Tests d'intégration

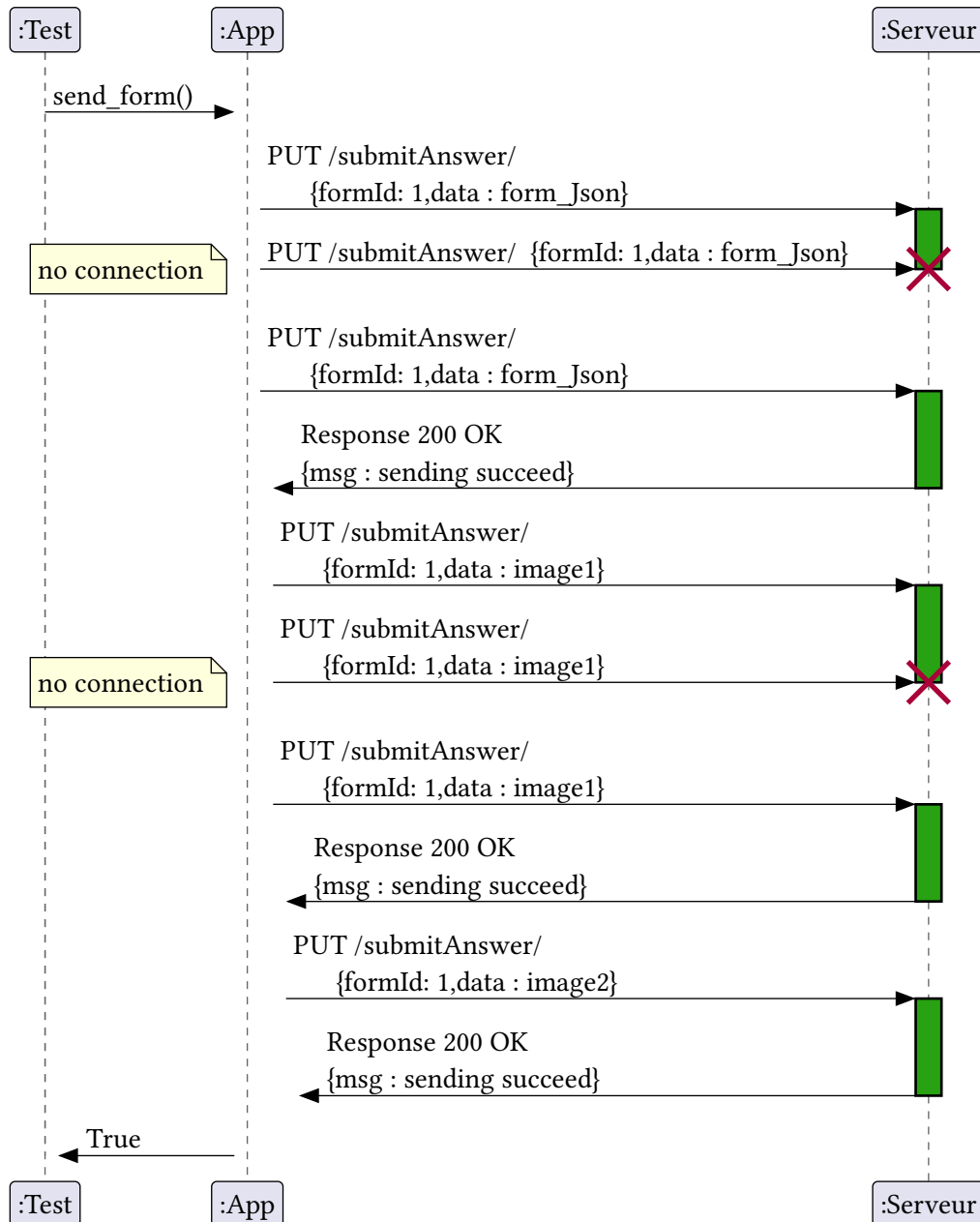
Les tests d'intégration vérifient que les différents composants du système fonctionnent ensemble comme prévu. Voici les tests à mettre en place :

### 8.2.1. Test d'envoi des formulaires

**Objectif :** Vérifier que les formulaires sont envoyés correctement et dans les délais.

**Scénario de test :**

- Soumettre un formulaire a complet.
- on essaie de couper la connexion dans certains moment pour voir si l'envoie se re-effectue de la bonne façon



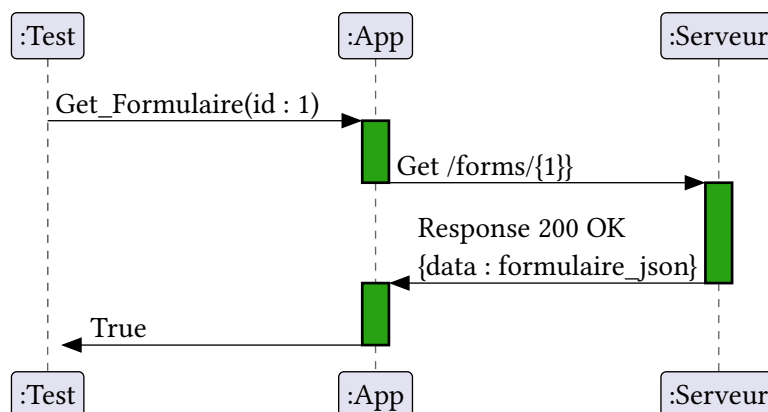
**Fig. 1.** Diagramme de séquences d'envoi des formulaires.

### 8.2.2. Test sur le téléchargement des formulaires

**Objectif :** Tester le temps de téléchargement et l'affichage des formulaires dans l'application.

**Scénario de test :**

- Test si le téléchargement est bien effectuer



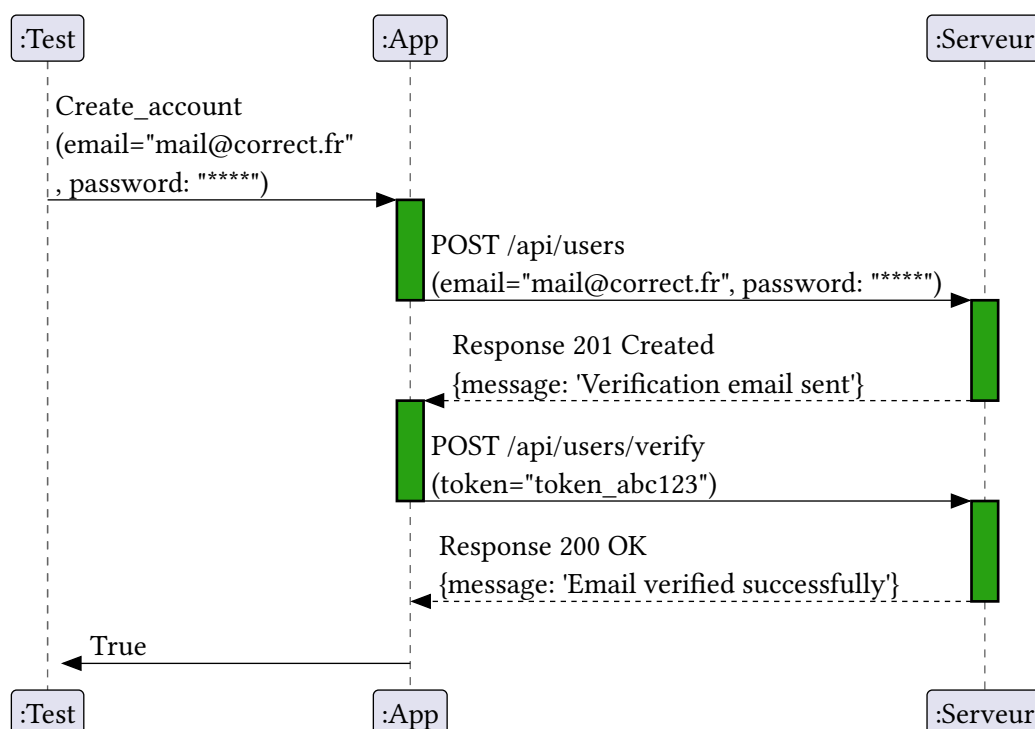
**Fig. 2.** Diagramme de séquences du téléchargement des formulaires.

### 8.2.3. Test de création de compte et login

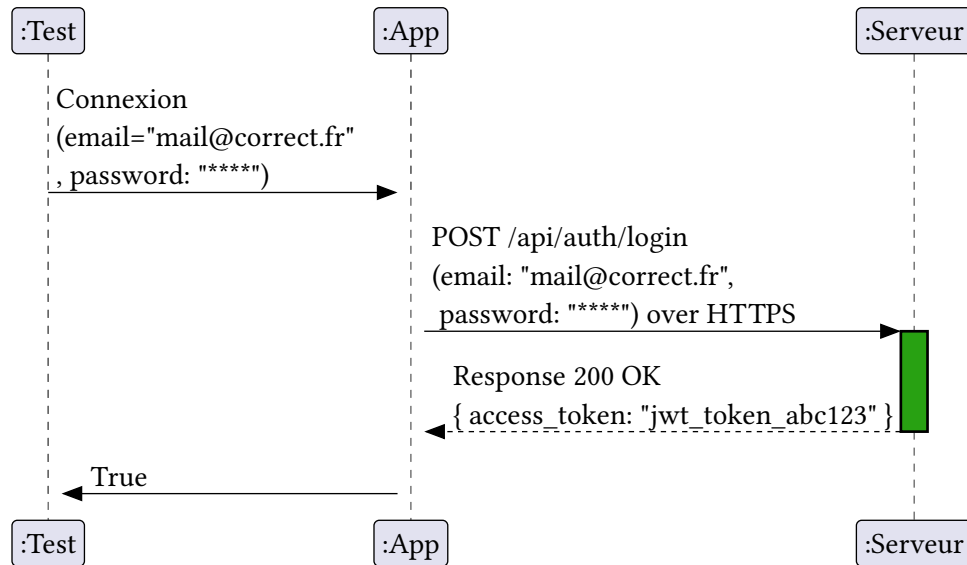
**Objectif :** Tester la création de compte utilisateur, le processus de login et les erreurs potentielles.

**Scénario de test :**

- Créer un compte avec des données valides et invalides.
- Tester les mécanismes de validation (mots de passe, email valide, etc.).
- Vérifier le comportement du système en cas de login incorrect (ex. : mauvais mot de passe).



**Fig. 3.** Diagramme de séquences pour s'inscrire.



**Fig. 4.** Diagramme de séquences pour se connecter.

### 8.2.4. Test de gestion des formulaires

**Objectif :** Vérifier que le système peut gérer plusieurs formulaires simultanément, les stocker et les récupérer sans erreurs.

**Scénario de test :**

- Créer/Enregistrer un formulaire
- Écrire dans un formulaire
- Tester les cas d'erreurs (par exemple, un formulaire incomplet).

Un model crée un fonction a partir de ses propres paramètres et les renvoie

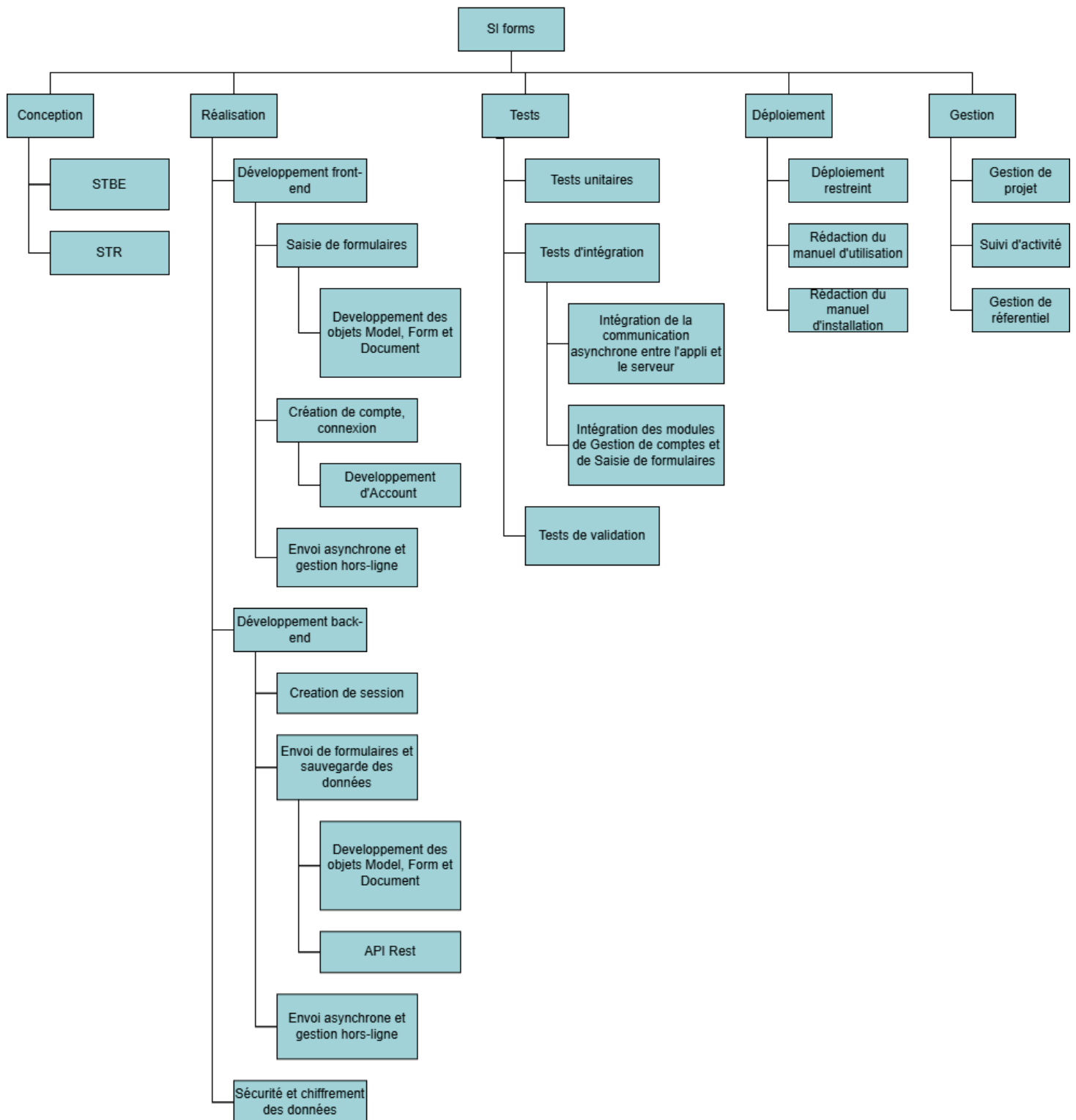
Écrit dans un formulaire et regarder si les champs sont modifié via l'affichage

Regarder si les champs modifier restent présent après enregistrement puis ouvrir et fermer le formulaire



## 9. Plannification

### 9.1. Work Breakdown Structure (WBS)



### 9.2. Tâches

Détail des tâches à réaliser pour le projet. Le temps estimé est en jours par personne.

### 9.2.1. Conception

Nom de la tâche	Respon- sable	Description	Temps estimé	Ressources
STBE	Adèle	Documenter les exigences fonctionnelles et non fonctionnelles du système.	34 j/p	Typst, Figma, LibreOffice, GoogleSheets, Graphviz
STR	Léa	Décrire l'architecture, les technologies et les méthodes d'implémentation du système.	7 j/p	Typst

### 9.2.2. Réalisation

Nom de la tâche	Respon- sable	Description	Temps estimé	Ressources
Saisie de formulaires	Léa	Frontend : Implémenter des formulaires pour saisir les informations des utilisateurs.	5 j/p	React Native, JSON
Création de compte, connexion	Adèle	Frontend : Ajouter des fonctionnalités de création de compte et de connexion.	4 j/p	React Native, base de données
Envoi asynchrone et gestion hors-ligne	Kevin	Permettre l'envoi asynchrone des formulaires du serveur vers l'application, et une utilisation hors-ligne.	9 j/p	React Native
Sauvegarde des données	Léa	Backend : Gérer la sauvegarde des réponses aux formulaires.	3 j/p	SQL
Sécurité et chiffrement des données	Kevin	Implémenter des mesures de sécurité pour protéger les données.	3 j/p	Bibliothèque Django

### 9.2.3. Tests

Nom de la tâche	Respon- sable	Description	Temps estimé	Ressources
Tests unitaires	Léa	Tests de chaque module individuellement.	4 j/p	/
Tests d'intégration	Kevin	Vérifier l'interaction entre les différents modules	5 j/p	/
Tests de validation	Adèle	Valider que le produit final respecte les exigences utilisateurs	2 j/p	/

#### 9.2.4. Déploiement

Nom de la tâche	Respon- sable	Description	Temps estimé	Ressources
Déploiement restreint	Léa	Déploiement limité pour les tests finaux en conditions réelles.	4 j/p	/

#### 9.2.5. Gestion

Nom de la tâche	Respon- sable	Description	Temps estimé	Ressources
Gestion de projet	Adèle	Suivi de l'avancement du projet et gestion des équipes.	/	Discord
Suivi d'activité	Kevin	Assurer le suivi régulier des activités de développement.	/	/
Gestion de référentiel	Léa	Gérer les versions des documents et des codes sources.	/	Google Drive, Typst, Github

## 10. Conclusion

Ce document spécifie le développement de **SiForms**, une application mobile pour iOS et Android permettant la saisie asynchrone de formulaires dynamiques. L'architecture repose sur un serveur local sécurisé (Python, Django, PostgreSQL) assurant le stockage et la synchronisation différée des données.

L'IHM, optimisée pour une utilisation hors ligne, se concentre sur la simplicité et la robustesse, avec des fonctionnalités clés : accès, modification, gestion des formulaires et suivi des soumissions. Cette solution répond aux contraintes terrain des techniciens tout en garantissant la fiabilité et la sécurité des données.

## 11. Annexes

### 11.1. Lexique

#### 11.1.1. Termes fonctionnels

- **entrée de formulaire** : Ensemble de réponses à un formulaire.
- **formulaire** : Ensemble de questions à réponses textuelles ou cases à cocher (à choix multiples ou unique) et des champs d'insertion de documents photos.
- **formulaire dynamique** : Formulaire dont le nombre et le contenu des questions change en fonction des réponses aux questions précédentes. Par exemples :
  - La question "Quelle sont les caractéristiques de la prise électrique ?" est affichée si l'utilisateur a répondu "Oui" à la question "Y a-t-il une prise électrique ?".
  - La question "Quel est la taille de l'armoire électrique ?" est affiché 5 fois quand l'utilisateur a répondu "5" à la question "Combien y a-t-il d'armoires électriques ?".
  - Le bloc de questions 12 à 15 s'affiche 3 fois si l'utilisateur a répondu « 3 » à la question 11.
- **localisation de l'application** : Processus d'adaptation de l'application à une langue spécifique.
- **modèle de formulaire** : Un modèle de formulaire correspond à un ensemble spécifique de questions, formant un formulaire. Deux formulaires du même modèle ont les mêmes questions et les mêmes règles qui régissent l'apparition de ces questions (cf. "formulaire dynamique"). Il existe pour le moment 3 modèles de formulaires :
  - le formulaire de visite technique,
  - le formulaire d'installation,
  - le formulaire d'informations du site d'intervention.
- **site d'intervention** : C'est le lieu d'installation du compteur électrique Smart X. Parfois raccourci en « site ».

#### 11.1.2. Termes techniques

- **authentifier** : Démarrer une session d'utilisation avec son compte utilisateur.
- **backend** : La partie d'une application qui gère la logique métier, les bases de données et les interactions côté serveur, invisibles pour l'utilisateur.
- **framework** : Ensemble de composants logiciels permettant de simplifier le développement logiciel en définissant une base d'architecture.
- **frontend** : La partie visible d'une application ou d'un site web avec laquelle l'utilisateur interagit directement, comprenant l'interface utilisateur.
- **snackbar** : Élément d'interface graphique informant l'utilisateur. Court message affiché dans l'application, dans une pop-up temporaire au premier plan.

#### 11.1.3. Technologies

- **Dart** : Un langage de programmation développé par Google, principalement utilisé pour le développement d'applications avec le framework Flutter. C'est un langage orienté objet, optimisé pour la performance, qui supporte la compilation en code natif et en JavaScript pour les applications web.
- **Django** : Un framework web Python complet et structuré, conçu pour faciliter le développement rapide et sécurisé d'applications web, avec des fonctionnalités intégrées pour la gestion des bases de données, l'authentification, et le routage.

- **Flask** : Un framework Python très léger pour créer des applications web, connu pour sa simplicité et sa flexibilité, offrant des fonctionnalités de base pour gérer les requêtes HTTP et le routage sans imposer de structure rigide.
- **Flutter** : Un framework de Google permettant de créer des applications natives multiplateformes (iOS, Android, web, desktop) en Dart, offrant une interface utilisateur très réactive.
- **JavaScript** : Un langage de programmation principalement utilisé pour ajouter des fonctionnalités dynamiques et interactives aux pages web, exécuté côté client ou côté serveur.
- **PostgreSQL** : Un système de gestion de base de données relationnelle open-source, supportant les transactions, les requêtes complexes, et les types de données avancés.
- **React** : Une bibliothèque JavaScript développée par Meta pour construire des interfaces utilisateur pour des applications web à pages dynamiques, en utilisant des composants réutilisables.
- **ReactNative** : Un framework de Meta permettant de développer des applications mobiles natives pour iOS et Android en utilisant JavaScript et React.

## 11.2. Bibliographie

### 11.2.1. Sources pour la comparaison des technologies

1. [ReactNativ at Airbnb](#), blog-post sur Medium, 19/06/2018
2. [Flutter ou React Native pour développer une application ?](#), blog-post sur Citronnoir, 2021
3. [Flask vs Django: Let's Choose Your Next Python Framework](#), blog-post sur Kinsta, 05/06/2023
4. [10 Advantages of Using Django for Web Development](#), blog-post sur Inoxoft, 10/07/2024
5. [What is Flask? Benefits and uses](#), blog-post sur Mytaskpanel
6. [Une API REST, qu'est-ce que c'est ?](#), blog-post sur RedHat, 02/02/2024
7. [Go-Back-N ARQ](#), Wikipédia FR, article web visité le 09/12/2024
8. [Ruby on rails vs Django](#), Wikipédia FR, article web visité le 13/12/2024

### 11.2.2. Documentation des technologies

1. [Django documentation](#)
2. [React Native documentation](#)
3. [Python documentation](#)
4. [PostgreSQL documentation](#)
5. [Expo documentation](#)