Développement d'une application mobile hybride pour le recueil de données sur le terrain

Naïma Dambrine[1]

Résumé. Quelque soit le thème de recherche et le projet d'étude abordé, les données de recherche, issues de l'observation, de l'expérimentation ou dérivées de sources existantes sont une part importante des travaux de recherche. Dans ce cadre, je présenterai une application mobile hybride capable de recueillir les données sur le terrain. Les données stockées localement sur l'appareil seront envoyées vers un serveur une fois la connexion internet rétablie. Le système est bâti sur un ensemble de technologie opérant ensemble dans une relation client-serveur. Sur la partie serveur sont hébergés la base de données et les web services, la partie cliente est l'application mobile.
Mots clés : Architecture n-tiers, base de données, Web Service, application mobile hybride

Abstract. Whatever the research problem or study project is, research data from observations, experiments or derived from existing sources are an important part of scientific research. In this scope, we will present an hybrid mobile application able to collect data in the field. Data stored locally on the application are sent towards a server once connected through internet. The system is then based on a client-server technology operating together. The server hosts database and web-services whereas the client party is the mobile application.
Keywords : n-third party architecture, database, web service, hybrid mobile application

1 INRAE, UMR Epi-a, Vetagro-sup Lyon, France. Naima.dambrine@inrae.fr

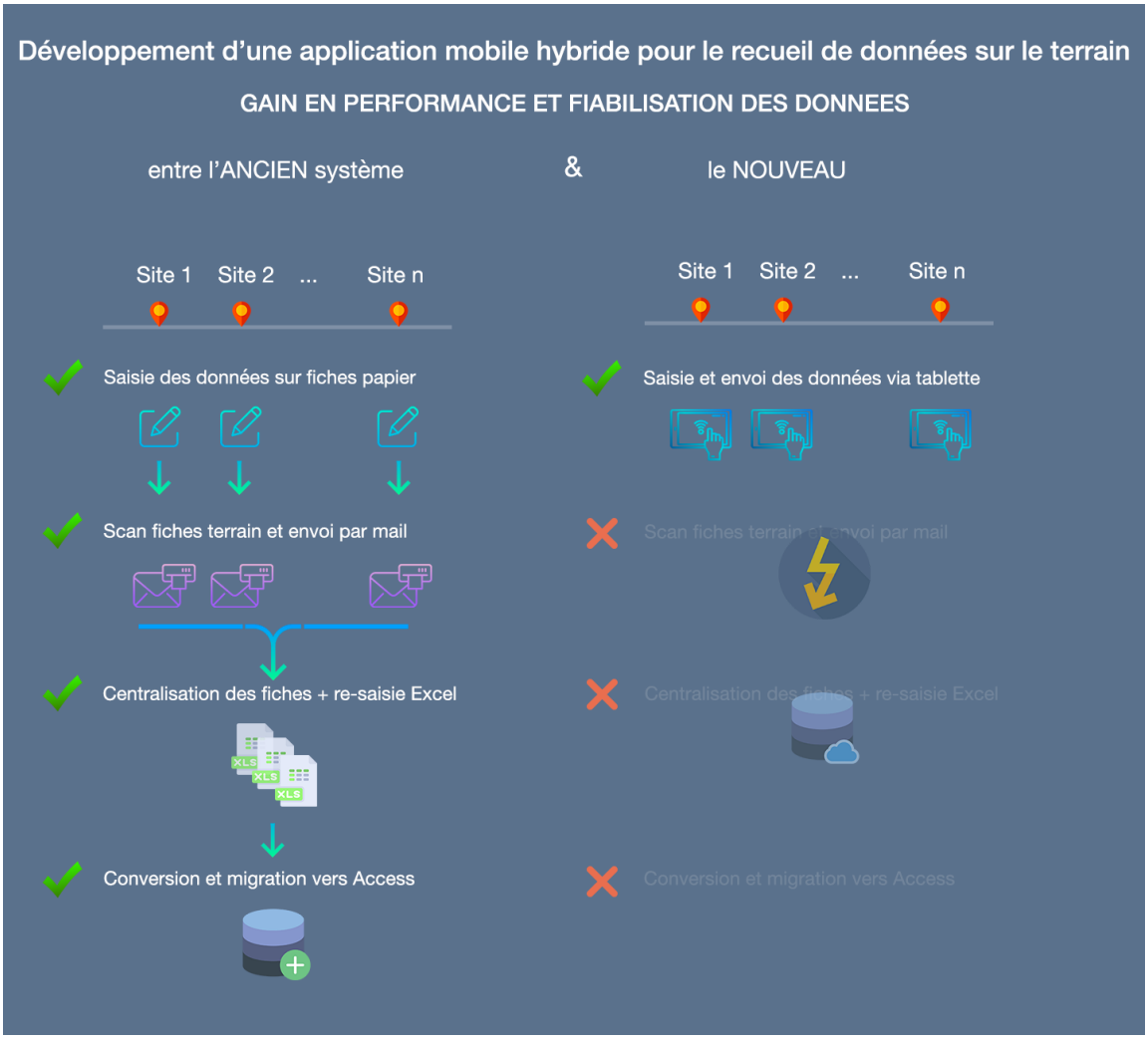


Figure 1. Développement d'une application mobile hybride pour le recueil de données sur le terrain

Introduction

Dans de nombreux laboratoires, des agents récoltent des données sur le terrain et les intègrent dans des bases de données, pas toujours de manière uniforme et souvent en multipliant les saisies. On peut simplifier et sécuriser cette phase à l'aide de saisie sur tablette pour une intégration directe en base de données.

Afin d'illustrer mon propos, j'exploiterai le cas d'une étude portant sur les tiques et le climat comme indicateur de prédiction sur l'abondance des tiques.

Des campagnes mensuelles de collecte et d'échantillonnage de tiques sont orchestrées sur plusieurs sites en France.

Voici la liste, pour chaque site, des tâches mensuelles effectuées pour la collecte.

1. Saisir sur le terrain les données observées à l'aide d'une trame papier,
2. Scanner et envoyer par mail la trame saisie au centre Inra qui centralise le recueil des données,
3. Une personne ressource saisit chaque scan reçu dans un fichier Excel,
4. Le fichier Excel est repris dans une base de données Access par le doctorant travaillant sur le projet,

5. La base de données Access est mise à jour au fil de l'eau et constitue la base d'échange entre les chercheurs.

Constat

On peut aisément déterminer les points faibles d'une telle organisation :

- risque d'erreur accru puisque une même donnée est saisie plusieurs fois par différentes personnes ;
- création d'un goulot d'étranglement au niveau du centre Inra centralisateur puisqu'une seule personne ressource est affectée à la saisie de toutes les fiches terrain pour tous les sites ;
- l'emploi d'une base de données Access ne permet pas le partage et il faut « s'envoyer la dernière version à jour », cela entraîne là aussi une lourdeur et le risque de se tromper de version.

Proposition

Notre objectif dans le cas présent est de supprimer toutes les étapes mentionnées ci-dessus en développant une application mobile qui aura la charge de récolter les données sur le terrain en mode offline, c'est-à-dire, sans couverture réseau puisque les données seront stockées dans l'appareil. Au retour d'un signal internet, les données pourront être envoyées sur le serveur de données.

Méthodes

Plusieurs technologies sont mises en œuvre dans ce projet et forment un environnement permettant à l'application de fonctionner. Ces concepts de base sont expliqués en soulignant les aspects directement couverts dans ce travail.

Qu'est-ce qu'une application mobile hybride ?

L'ensemble des applications mobile peut être classé en trois catégories : web, hybride et native.
Les applications web utilisent du code web, HTML, CSS et JAVASCRIPT et les applications natives utilisent un langage natif ciblant un appareil ou un système spécifique, Objective C pour iOS, Java pour Android par exemple.
Une application hybride est une combinaison des deux approches.
Evidemment, les applications web peuvent être exécutées depuis n'importe quel navigateur (sur PC, tablette ou mobile). Cependant, lorsqu'il s'agit de mobile, les applications natives offrent des fonctionnalités que les applications web n'ont pas, comme :

- la possibilité d'être distribuée en ligne depuis les stores Apple et Google
- la possibilité d'accéder aux API natives de l'appareil (Wifi, GPS ou camera)

L'inconvénient majeur est qu'il faudra utiliser pour chaque type de mobile (plateforme) un langage dédié.

C'est ici qu'entre en jeu l'intérêt d'une application hybride : un seul langage pour toutes les plateformes. Cela est rendu possible grâce aux Framework Cordova ou Capacitor, ces API vont envelopper l'application dans un Shell natif (une Webview) et agir comme une passerelle entre l'application et l'appareil.

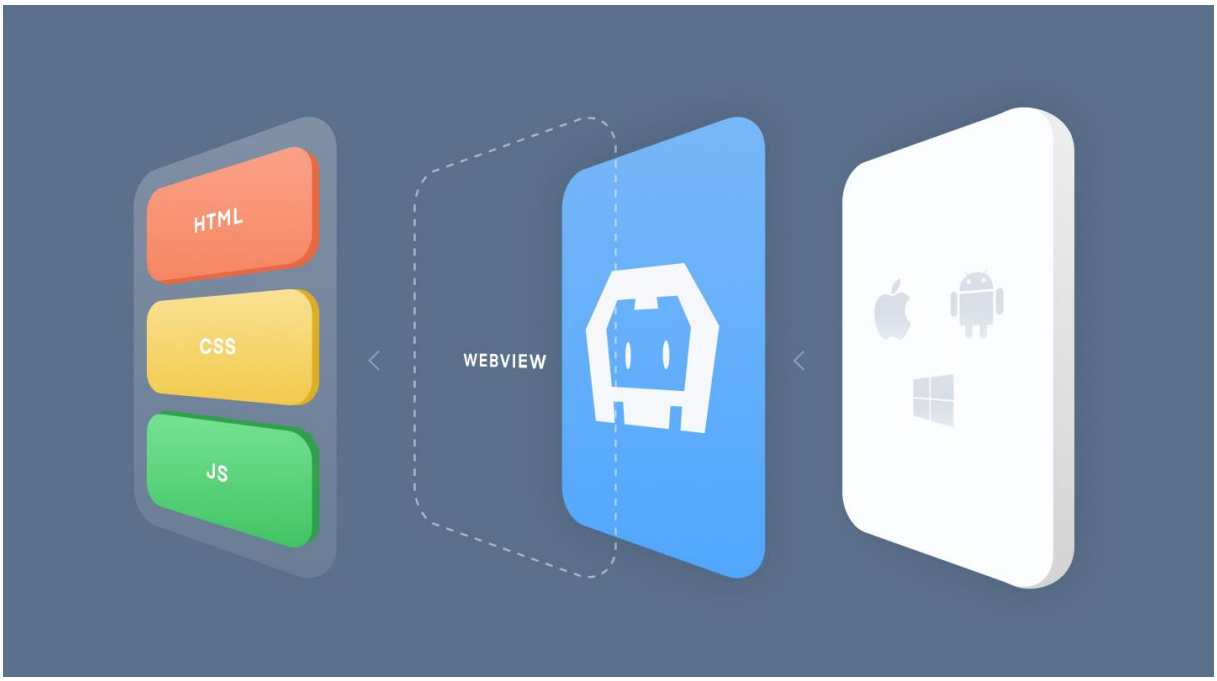


Figure 2. Framework Cordova : interface entre le code web et les différentes plateformes mobiles

Source : https://ionicframework.com/resources/articles/what-is-apache-cordova

Modélisation des données

Une définition de la modélisation tirée de Wikipédia (Source : https://fr.wikipedia.org/wiki/Modèle_de_données) est donnée ci-après :

> *La modélisation des données est le processus par lequel on crée un modèle de données instance en appliquant une théorie de modèle de données. On emploie cette méthode pour se conformer à des exigences ou des attentes d'entreprises ou d'organismes publics.*
>
> *Les besoins sont généralement exprimés à travers un modèle conceptuel de données et/ou un modèle logique de données. Ces modèles sont ensuite transformés en un modèle de données physique, qui décrit les bases de données physiques employées.*
>
> *Pour une application spécifique, on définit les tables (objets, relations... les conventions de nommage dépendent du modèle général). Par exemple, on décrit "client", "commande", "article", ainsi que les relations entre eux ("un client commande des articles").*
>
> *Si on utilise un modèle relationnel, on doit définir des ensembles de contraintes spécifiques (clé primaire, clé candidate, clé étrangère), en utilisant le langage approprié conformément au modèle général (par exemple SQL).*

Cette étape est l'une des toutes premières. Elle va permettre de concevoir la base de données et vise à représenter fidèlement les caractéristiques des données de l'application ainsi que leur gestion.

Voici, à titre d'exemple, le modèle conceptuel obtenu pour notre cas d'étude :

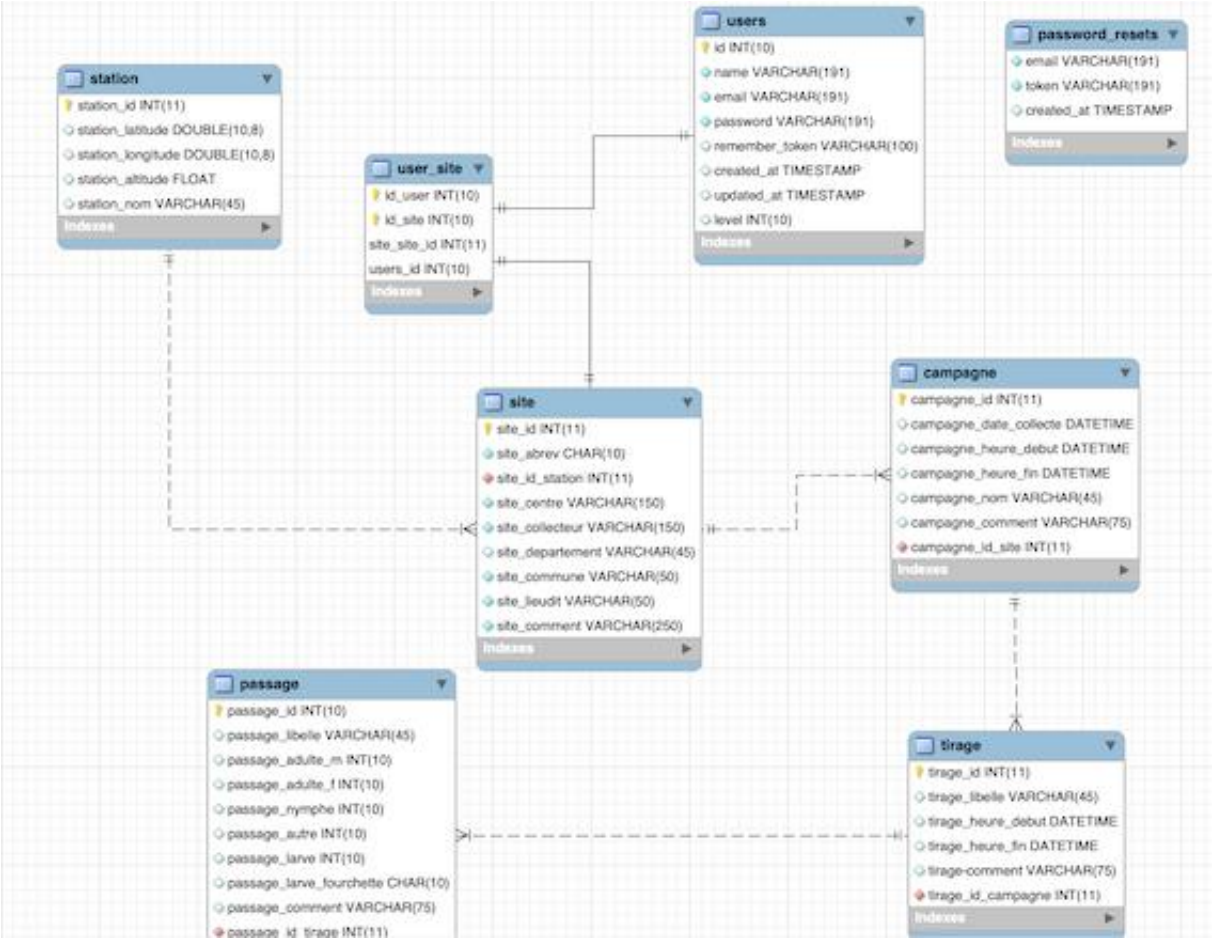


Figure 3. Schéma modèle conceptuel des données

Le modèle conceptuel est transformé en modèle physique des données et constitue une coque vide prête à recevoir les données.

Dans notre cas d'étude, des données existantes sont déjà présentes sous un autre format (ACCESS). Une opération de migration des données de l'ancien système vers le nouveau a été mise en œuvre. Il s'agit de pure technique SQL et cela ne sera pas abordé ici. On peut toutefois souligner l'importance d'une bonne modélisation afin d'être en mesure non seulement d'importer les données existantes mais aussi d'intégrer à ce moment-là les améliorations souhaitées ainsi que les données spécifiques de l'application qui seront utilisées pour en assurer la gestion.

Architecture 3-tiers

On peut représenter comment s'articulent les différentes technologies entrant en jeu dans notre projet par le schéma suivant :

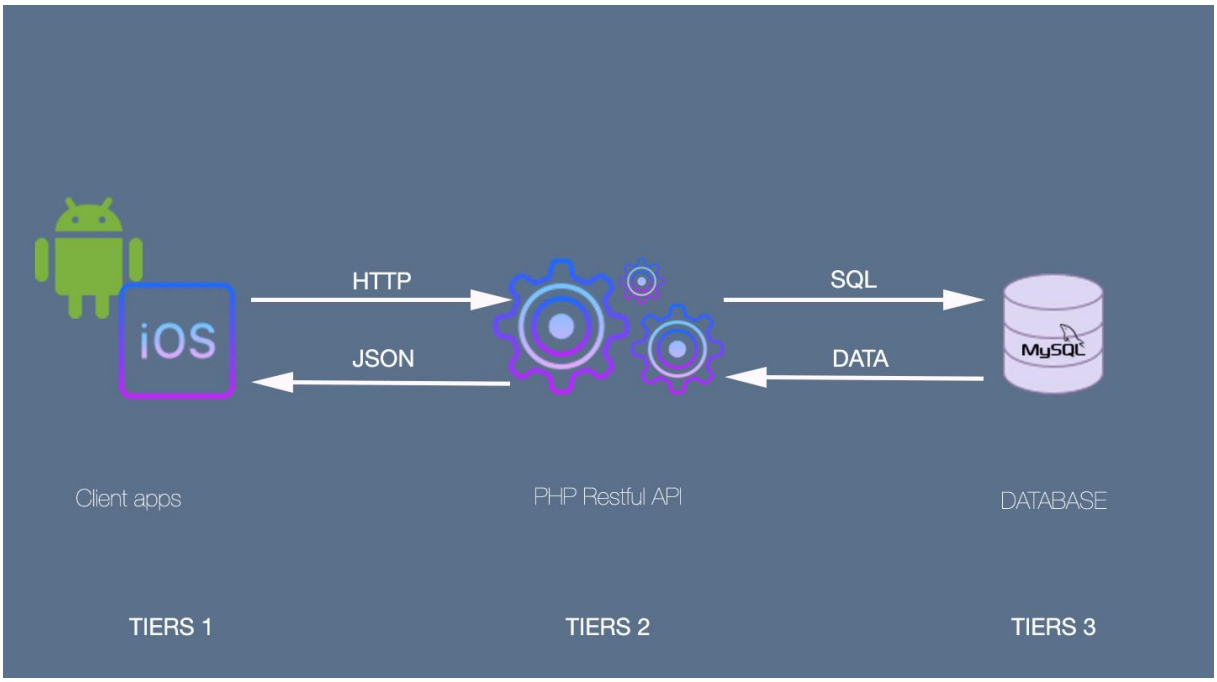


Figure 4. Schéma architecture 3-tiers

Dans un tel modèle, on identifie le client comme l'utilisateur final (Tiers 1), l'agent sur le terrain qui, à l'aide d'une tablette ou d'un mobile (Android/iOS), va rentrer ses données d'observation.
Ces données seront envoyées, plus tard, via une connexion internet (Http), sur la partie serveur pour traitement. Ce traitement est opéré entre deux tiers, une api ou web service, en charge de traiter la donnée (Tiers 2, authentification, validation des données) et la base de données (Tiers 3, stockage des données).

Web Service

Par web service, on veut souligner le fait qu'il n'y aura pas d'intervention humaine mais une interaction entre machines. Dans notre cas, l'application mobile va dialoguer avec notre API Rest qui elle-même va dialoguer avec la base de données. Ces échanges « machine » repose sur le protocole Http, le langage du web. Il sera donc régi par les spécifications de ce protocole.
Le fonctionnement de ce type d'architecture REST se caractérise par deux notions :

- les ressources : le but de l'API étant de rendre accessible des ressources « resources » qui correspondent aux données de l'application ;
- les en-têtes : GET/PUT/POST/DELETE.

La combinaison des ressources et des en-têtes permet de construire une URL de type Rest :

- /resources, GET, récupère la liste des ressources,
- /resources/ :id, GET, récupère la ressource selon l'ID,
- /resources, POST, crée une nouvelle ressource,
- /resources/ :id, PUT/PATCH, met à jour une ressource,
- /resources/ :id, DELETE, supprime une ressource.

Imaginons que nous souhaitions la liste des sites impliqués dans notre projet. Il faudrait interroger notre API en faisant une requête Http :

```
Request : GET http://epia.clermont.inra.fr/vas/apiclimatick/public/api/sites
Response :   "sites": [
                {
                   "site_id": 1,
                   "site_abrev": "S1",
                   "site_centre": "VetAgroSup Lyon",
                   "site_collecteur": "EpiA INRA",
                   "site_departement": "69",
                   "site_commune": "La-Tour-de-Salvagny",
                   "site_lieudit": "Route de Lozanne"
                },
                {
                   "site_id": 2,
                   "site_abrev": "S2",
                   "site_centre": "INRA Theix",
                   "site_collecteur": "EpiA INRA",
                   "site_departement": "63",
                   "site_commune": "Saint Genès",
                   "site_lieudit": "Route des Volcans"
                }, ...
```

Figure 5. Requête HTTP

Authentification JWT

La question que l'on est en droit de se poser est : est-ce que tout le monde peut accéder à toutes les ressources disponibles ? Evidemment, non. Les ressources doivent être protégées et cela nécessite la mise en place d'une

authentification. Nous utiliserons le standard ouvert JWT Json Web Token (RFC 7519), voici comment ce dernier fonctionne :

on s'enregistre une première fois auprès de notre api en monnayant un couple identifiant/mot de passe. Si tout se passe bien (i.e. une vérification en base de données confirme l'existence de l'identifiant et du mot de passe) alors l'api nous renvoie un jeton ou Token. C'est une chaîne de caractère qui équivaut à une signature numérique et garantit ainsi l'échange des données entre les parties (tiers) de manière compacte et autonome. Ce Token a une durée de vie limitée mais tant qu'il n'est pas périmé, il sera transmis dans toutes les requêtes que nous souhaitons faire au travers des en-têtes Http.

Si on reprend l'exemple donné précédemment, on transmet le Token reçu dans les en-têtes de la requête :

```
curl      -H 'Accept: application/json'
          -H "Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC9lcGlhLmNsZXJtb250LmlucmEuZnJcL3Zhc
1wvYXBpY2xpbWF0aWNrXC9wdWJsaWNcL2FwaVwvbG9naW4iLCJpYXQiOjE1Njk4NTc5NTEsImV4cCI6MTU2
OTg2MTU1MSwibmJmIjoxNTY5ODU3OTUxLCJqdGkiOiJVRmVONWxJaTJ1Y2dNNUJ5Iiwic3ViIjo5LCJwcnYiOiI4
N2UwYWYxZWY5ZmQxNTgxMmZkZWM5NzE1M2ExNGUwYjA0NzU0NmFhIn0.s3VZAmgN1Q-
iKW3sePW9LEowEfeoTIQcQ6Ek5o-1VJ9" http://epia.clermont.inra.fr/vas/apiclimatick/public/api/sites
```

Figure 6. Type de Token

Les en-têtes (-H) spécifient le type de données (mime) qui vont transiter durant l'échange : Json et le type d'authentification (Bearer) utilisé suivi du Token.

Chaque fois que nous aurons besoin d'accéder à une ressource, l'application injectera le Token dans l'en-tête Http. Côté serveur, l'API va vérifier que le Token est bien présent et de fait autorisera l'utilisateur à accéder aux ressources protégées.

Description de l'App

L'application est développée avec le Framework Ionic dans sa version 4, Angular 7 et Capacitor pour l'accès aux API natives, caméra, système de fichier, etc.

Le découpage de l'App se fait comme suit :

Les pages sont les suivantes :

- Welcome : première page à l'ouverture de l'App,
- Login : connexion à l'App,
- Signup : création d'un profil,
- Profile : modification du profil,
- Site : synchronisation des sites,
- Campagne : liste des campagnes saisies,

- Detail : détail de la campagne,
- Tirage de 1 à 10 : liste des tirages.

Les services sont les suivants :

- Auth.service : gestion de l'authentification,
- Campagne.service : gestion des campagnes,
- Network.service : gestion de la connectivité,
- Photo.service : prendre et gérer les photos,
- Site.service : synchronisation des sites,
- User.service : gestion des utilisateurs.

Le menu de configuration est Setting-pop-over.

Les interfaces sont les suivantes :

- Campagne,
- Site,
- User.

Workflow

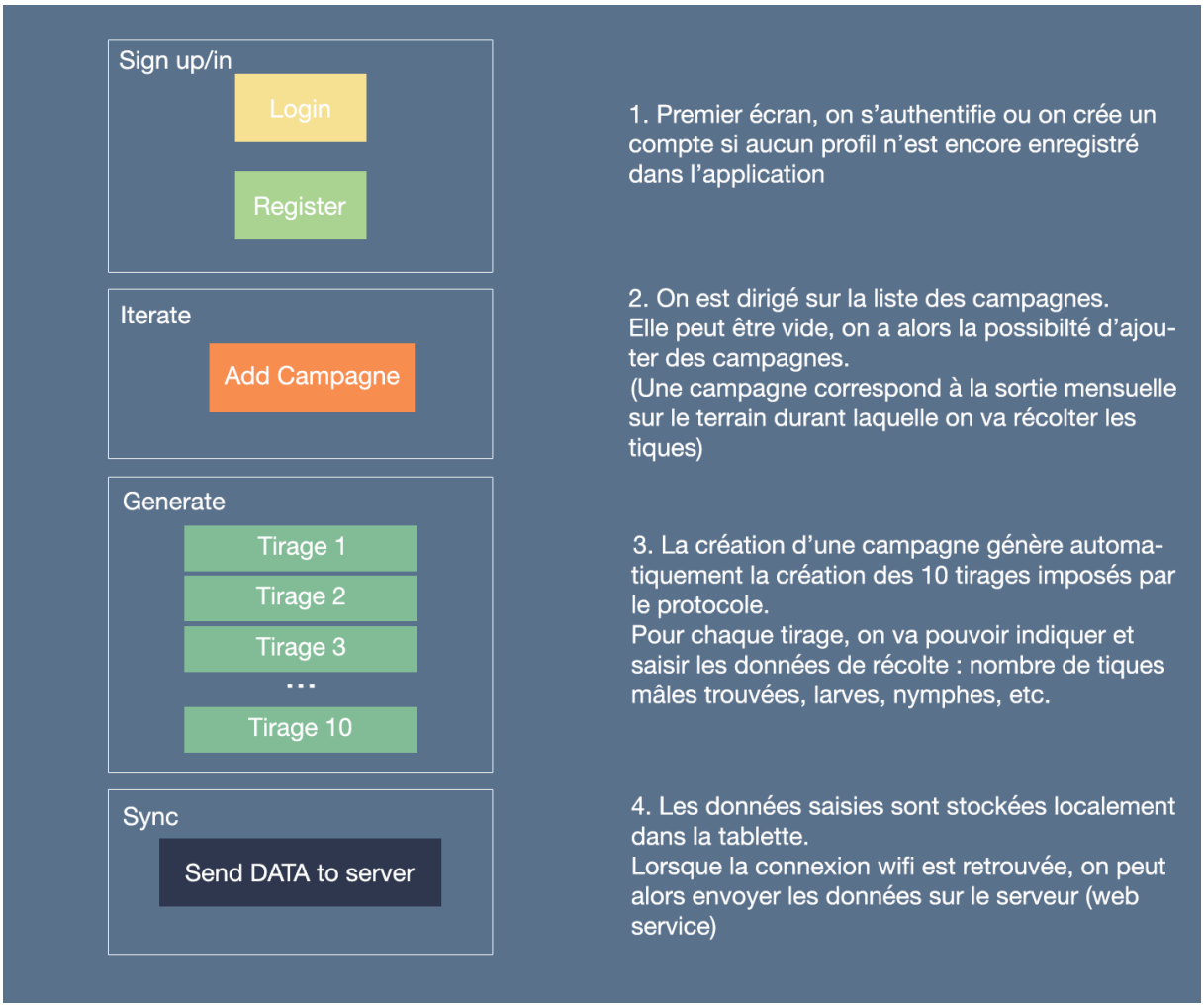


Figure 7. Work flow

Interfaces TypeScript

En profitant du TypeScript et des interfaces, on va pouvoir générer les modèles de notre application : on obtient un code solide, débarrassé des problèmes coutumiers et autres effets de bord du JavaScript liés aux erreurs de type : les erreurs sont affichées dès l'exécution du code (annexe 1).

Services

Les modèles de l'App étant définis, il nous faut un moyen de les gérer. Pour cela, on utilisera les services pour chacune de nos entités (campagne, user, photo, etc.). C'est une excellente pratique que de concentrer la logique applicative dans des services que l'on pourra utiliser où bon nous semble dans l'application chaque fois que nécessaire (annexe 2).

Résultats

On peut voir l'App sur les store Apple et Google ici :

App Store: https://apps.apple.com/us/app/climatick/id1470038941
Google Play Store: https://play.google.com/store/apps/details?id=fr.inra.climatick&hl=fr_CA

:

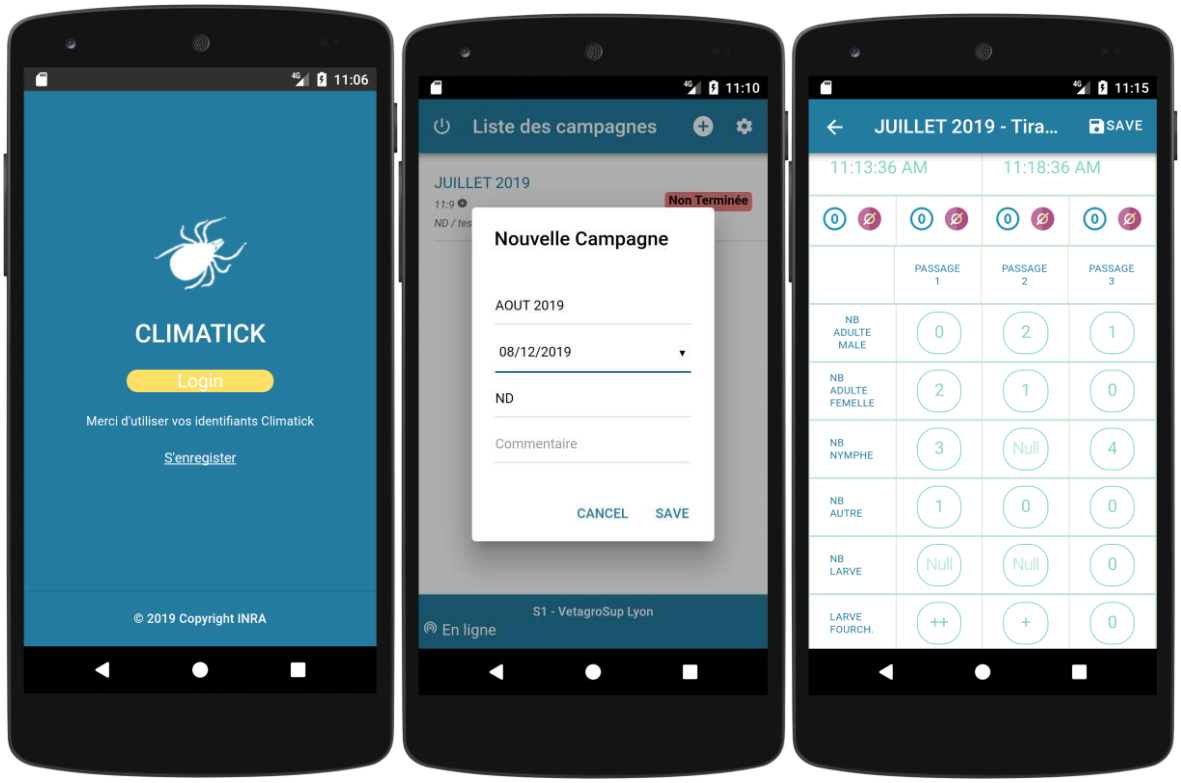
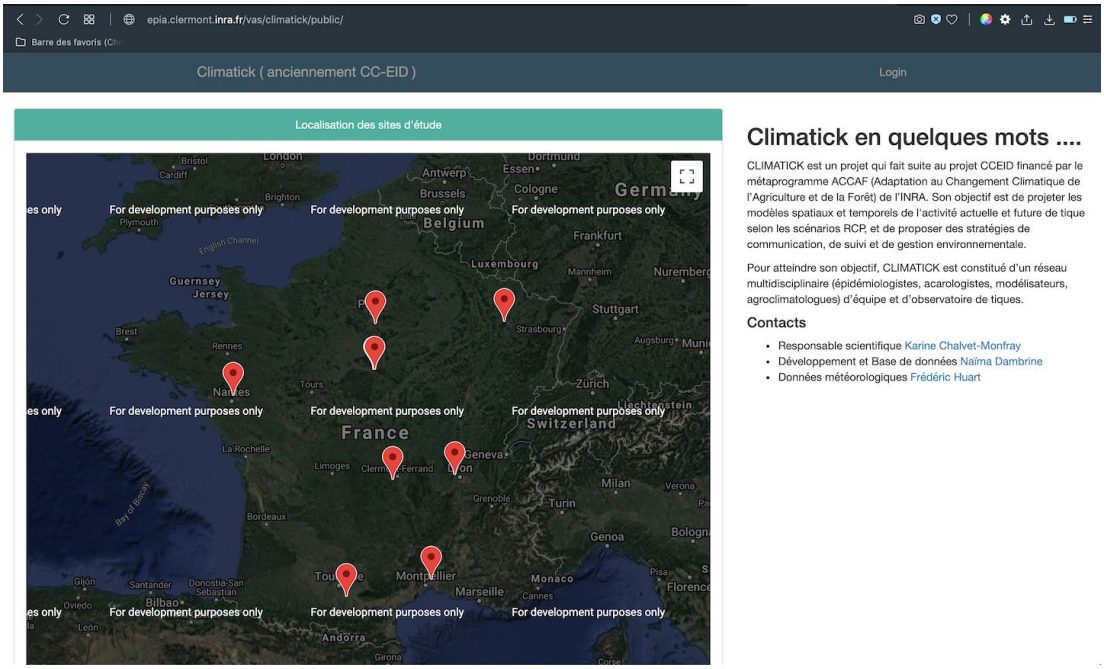


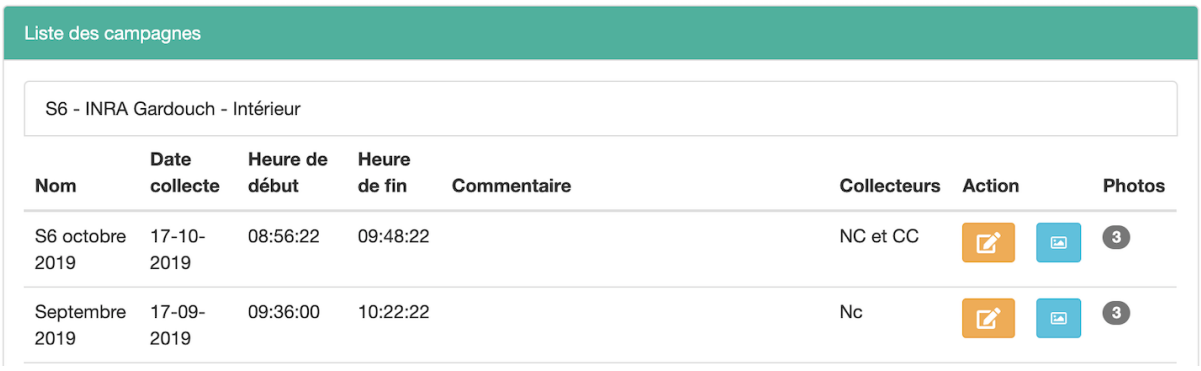
Photo 1. Exemple de saisie des données tiques via mobile

Les données saisies peuvent être consultées, a postériori, sur le serveur via l'application Climatick en ligne à l'url suivante : http://epia.clermont.inra.fr/vas/climatick/public/

Et voici quelques illustrations tirées de l'application en ligne, ici, les sites participants au projet :



Liste des campagnes de terrain envoyées via la tablette :

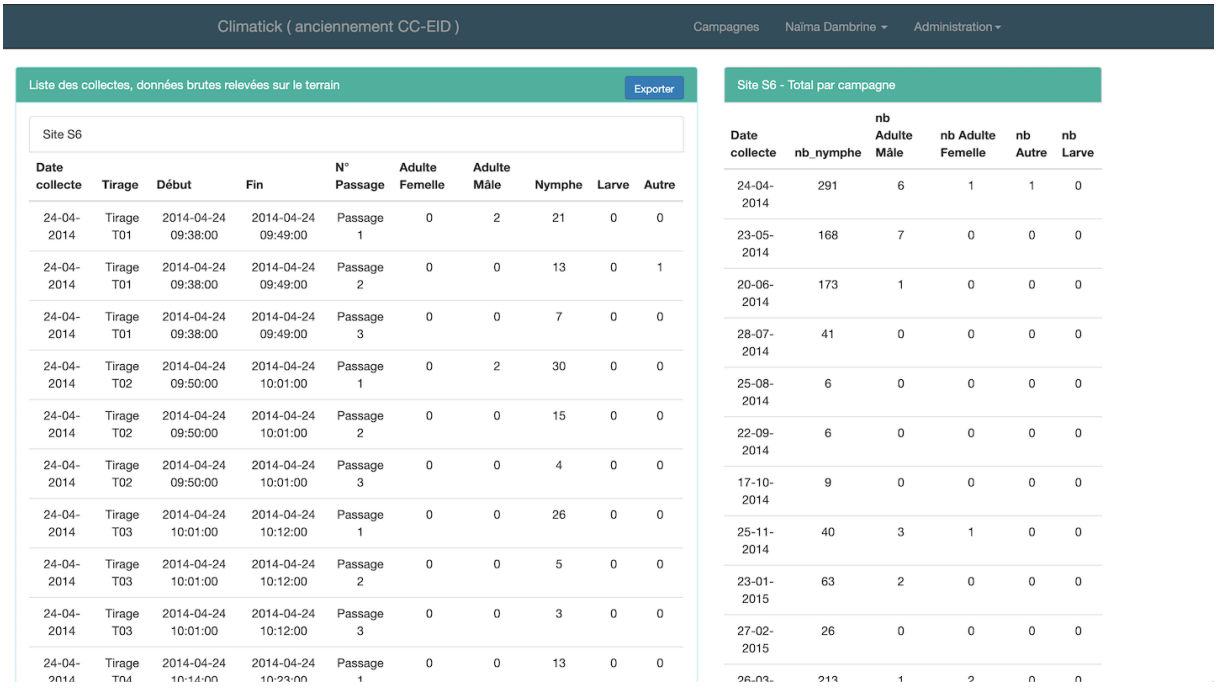


Photos prises sur le terrain avec la caméra de l'appareil (tablette ou mobile) :



© 2018 Copyright INRA Naïma Dambrine

Données brutes de terrain, exportable au format csv/Excel :



Climatick (anciennement CC-EID) — Campagnes — Naïma Dambrine — Administration

Liste des collectes, données brutes relevées sur le terrain — Exporter

Site S6

Date collecte	Tirage	Début	Fin	N° Passage	Adulte Femelle	Adulte Mâle	Nymphe	Larve	Autre
24-04-2014	Tirage T01	2014-04-24 09:38:00	2014-04-24 09:49:00	Passage 1	0	2	21	0	0
24-04-2014	Tirage T01	2014-04-24 09:38:00	2014-04-24 09:49:00	Passage 2	0	0	13	0	1
24-04-2014	Tirage T01	2014-04-24 09:38:00	2014-04-24 09:49:00	Passage 3	0	0	7	0	0
24-04-2014	Tirage T02	2014-04-24 09:50:00	2014-04-24 10:01:00	Passage 1	0	2	30	0	0
24-04-2014	Tirage T02	2014-04-24 09:50:00	2014-04-24 10:01:00	Passage 2	0	0	15	0	0
24-04-2014	Tirage T02	2014-04-24 09:50:00	2014-04-24 10:01:00	Passage 3	0	0	4	0	0
24-04-2014	Tirage T03	2014-04-24 10:01:00	2014-04-24 10:12:00	Passage 1	0	0	26	0	0
24-04-2014	Tirage T03	2014-04-24 10:01:00	2014-04-24 10:12:00	Passage 2	0	0	5	0	0
24-04-2014	Tirage T03	2014-04-24 10:01:00	2014-04-24 10:12:00	Passage 3	0	0	3	0	0
24-04-2014	Tirage T04	2014-04-24 10:14:00	2014-04-24 10:23:00	Passage 1	0	0	13	0	0

Site S6 - Total par campagne

Date collecte	nb_nymphe	nb Adulte Mâle	nb Adulte Femelle	nb Autre	nb Larve
24-04-2014	291	6	1	1	0
23-05-2014	168	7	0	0	0
20-06-2014	173	1	0	0	0
28-07-2014	41	0	0	0	0
25-08-2014	6	0	0	0	0
22-09-2014	6	0	0	0	0
17-10-2014	9	0	0	0	0
25-11-2014	40	3	1	0	0
23-01-2015	63	2	0	0	0
27-02-2015	26	0	0	0	0
26-03-	213	1	2	0	0

Conclusion

Supprimer les intermédiaires en faisant usage d'App mobile permet de faire l'économie d'erreur et de temps.

Un système d'information correctement bâti permet l'usage de ce type d'application, en bout de chaîne. Le développement final peut être généralisé sur tous types de projet.

Ce projet est autonome pour les applications suivantes :

- pour des projets orientés mono-utilisateur et ne nécessitant pas une base de données tiers, on peut imaginer un simple export csv pour une reprise des données sous R.
- d'autres solutions ont vu le jour, notamment à base de Raspberry Pi, où cette fois on embarque tout l'écosystème nécessaire : base de données + saisie en direct et/ou script de récupération des données issues de capteurs.
- enfin, nous pourrions imaginer rendre générique ce type d'application en générant à la volée les champs à saisir. Il faudrait préciser les champs et leur type et générer le masque de saisie idoine.

Ce projet peut être multi-utilisateur :

- dans ce cas de figure, il faudrait une API Rest générique également. Cela conviendrait particulièrement à des sauvegardes sur le Cloud, sous forme de fichiers. Dans le cas d'une base de données mutualisée sur un serveur, cela s'avère plus délicat en raison des contraintes d'intégrité des données inhérentes à tout système de gestion de base de données relationnel (SGBDR).
- on pourrait dès lors s'orienter vers des Web Services sémantiques, ce qui permettrait de réduire le couplage client-serveur et d'introduire de l'intelligence artificielle au sein des interfaces utilisateurs. Pour aller plus loin, une introduction au concept du modèle de maturité de Richardson : https://guide-api-rest.marmicode.fr/api-rest/le-modele-de-maturite-de-richardson et une mise en application du modèle avec le framework Symfony https://openclassrooms.com/fr/courses/4087036-construisez-une-api-rest-avec-symfony/4343816-rendez-votre-api-auto-decouvrable-dernier-niveau-du-modele-de-maturite-de-richardson

Cet article est publié sous la licence Creative Commons (CC BY-SA).
https://creativecommons.org/licenses/by-sa/4.0/
Pour la citation et la reproduction de cet article, mentionner obligatoirement le titre de l'article, le nom de tous les auteurs, la mention de sa publication dans la revue « Le Cahier des Techniques de l'Inra», la date de sa publication et son URL).

Annexe

1. TypeScript

Le TypeScript permet de modéliser nos données ; ainsi, pour une campagne de terrain, on utilisera les interfaces suivantes :

```
import { User } from '../interfaces/user';
export interface Campagne {
    id:string;
    nom: string;
    date: Date;
    heure_debut:Date;
    heure_fin:Date;
    comment:string;
    collecteurs: string;
    tirages: Tirage[];
    user_id:number;
    site_id:number;
    station_id:number;
    saved:boolean;
    photos : Photo[];
}

export interface Photo {
    name: string,
    path: string,
    data: string,
    dateTaken: Date
}

export interface Tirage{
    libelle: string;
    heure_debut: Date;
    heure_fin: Date;
    comment:string;
    passages: Passage[];
    saved:boolean;
}

export interface Passage{
    libelle:string;
    adulte_m:number;
    adulte_f:number;
    nymphe:number;
    autre:number;
    larve:number;
    larve_fourchette:string;
    comment:string;
}
```

On peut ainsi profiter de ce typage pour, par exemple, mettre à jour une campagne :

```
  update(campagne : Campagne): void {
    let index = this.campagnes.findIndex(c => c.id === campagne.id);
    this.campagnes[index] = campagne;
    this.save();
  }
```

2. Les services

Voici un exemple d'utilisation des services : liste des méthodes proposées dans le service campagne.service.ts

```
  ...
  getCampagne(id): Campagne {
    return this.campagnes.find(campagne => campagne.id === id);
  }

  createCampagne(data): void {...}

  renameCampagne(campagne, data): void {
    let index = this.campagnes.indexOf(campagne);

    if (index > -1) {
      this.campagnes[index].nom = data.nom;
      this.campagnes[index].date = data.date;
      this.campagnes[index].comment = data.comment;
      this.campagnes[index].collecteurs = data.collecteurs;
      this.save();
    }
  }

  removeCampagne(campagne): void {
    let index = this.campagnes.indexOf(campagne);

    if (index > -1) {
      this.campagnes.splice(index, 1);
      this.save();
    }
  }
  ...
}
```

Les méthodes offertes par le service pourront être utilisées depuis la page des campagnes : bouton faisant appel à la fonction addCampagne() depuis campagne.page.html

```
  ...
  <ion-button (click)="addCampagne()">
    <ion-icon slot="icon-only" name="add-circle"></ion-icon>
  </ion-button>
  ...
```

Via le composant : campagne.page.ts, en important le service :

```
...
import { CampagneService } from "../../services/campagne.service";
...
constructor(
    ...
    public campagneService: CampagneService,
    public authService: AuthService,
    public http : HttpClient,
    public toastController: ToastController,
    public popOverCtrl: PopoverController,
    public loadingController: LoadingController,
    public photoService : PhotoService,
    private plt: Platform  ... ) {}
...
```

Puis en l'utilisant lors de notre appel addCampagne() : campagne.page.ts

```
  addCampagne(): void {
    ...
    this.campagneService.createCampagne(data);
    ...
  }
```

On va pouvoir également gérer la prise de photo et déléguer toute la gestion de création, sauvegarde, etc. au service photo.service.ts

```
takePhoto(idCampagne:string): void{
    this.photoService.load(idCampagne);

    this.loadingController.create({
      message: 'Enregistrement de la photo ...',
    }).then((overlay) => {

      overlay.present();

      this.photoService.takePhoto().then( (photo) =>{

        overlay.dismiss();

      }, (err) => {
        overlay.dismiss();
        this.alertCtrl.create({
          header: 'Oops!',
          message: err
        }).then( (alert) =>{
          alert.present();
        });
      });
    });
  }
```