Application Maraîchage

RAPPORT DE PROJET

BTS Systèmes Numériques

Option Informatique & Réseaux

Année 2021/2022

Le Festin d'Élian

TOURNACHE Nicolas

BERCHEL Rudy

MAUREL Robin

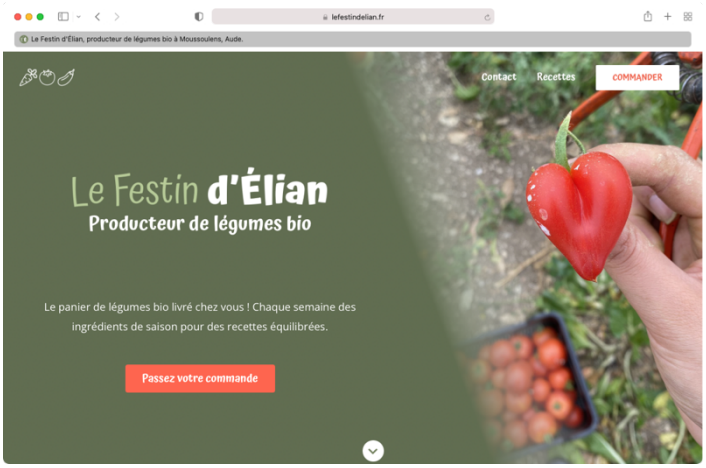
LORIN Dorian

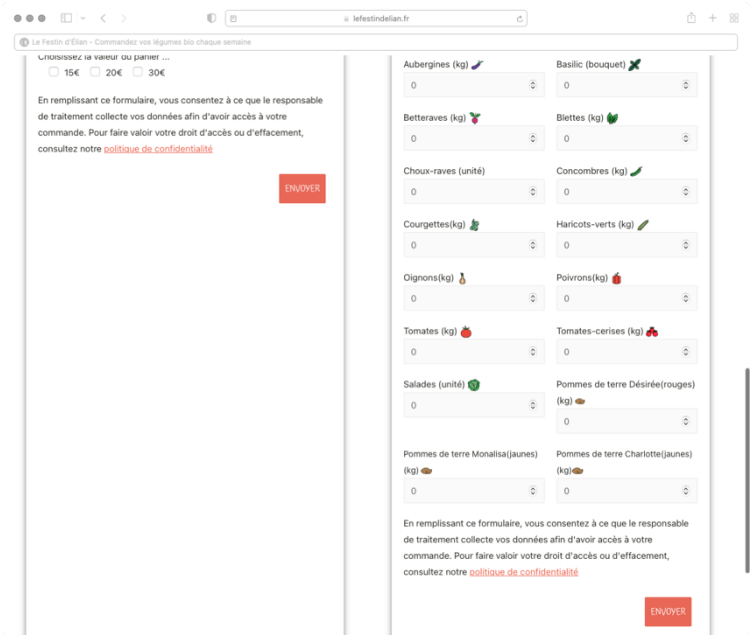
Table des matières

I – Présentation

1. Présentation de l'entreprise

Le partenaire pour lequel nous avons réalisés ce projet est l'entreprise Le Festin d'Elian. Le Festin d'Elian est un producteur de légumes Bio qui livre des paniers de légumes Bio chaque semaine. Un site web est à disposition afin de pouvoir commander ses paniers légumes par internet :





2. Contexte du projet

L'entreprise permet aussi de prendre des paniers en utilisant un système de distribution de légumes qui se compose de plusieurs casiers en libre-service. Il suffit d'utiliser un monnayeur ou une carte bleue afin d'ouvrir un casier qui correspond à un panier de légumes.

II – Objectif du projet

1. Objectif principal

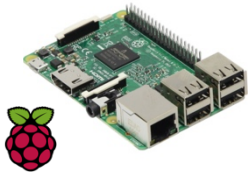
L'entreprise souhaite mettre à jour son site Web et souhaite également avoir une application mobile sur iPhone/iPad. Elle souhaiterait pouvoir suivre l'évolution des prises de paniers de légumes dans les casiers. Il faut donc développer une solution permettant de visualiser l'état de stocks des casiers en temps réel et d'envoyer des notifications lorsque les clients prennent des paniers de légumes.

Objectifs Principaux :

- **Utilisation de capteurs :** Permettre de détecter l'ouverture d'un casier lors de la prise d'un panier de légumes et de savoir lorsqu'un casier est vide.
- **Serveur WEB :** Mise en œuvre d'un site WEB amélioré sous Wordpress permettant d'afficher l'état des casiers.
- **Base de données :** La base de données permettra de stocker les informations des casiers comme leurs état ainsi que l'horodatage de la dernière prise de paniers légumes.
- **Application iOS :** L'application permettra de visualiser l'état des casiers en direct (plein ou vide), l'horodatage de la dernière ouverture et une notification sera envoyée lorsqu'un casier sera ouvert.

2. Matériel

Pour réaliser ce projet, nous avons eu besoin du matériel suivant :

- **1 Raspberry PI** 
- **28 Capteurs d'ouverture Sonoff ZIGBEE SNZB-04** 

- **1 Module de communication ZIGBEE** 

3. Logiciels & Langages utilisés

Afin de créer le site WEB, la base de données pour le stockage des informations des casiers, l'API pour communiquer avec la base de données, la mise en place des capteurs et la création de l'application, on a utilisé les logiciels et langages suivants :

- **HTML & CSS pour le site WEB**

- **MySQL & PhpMyAdmin pour la base de données**

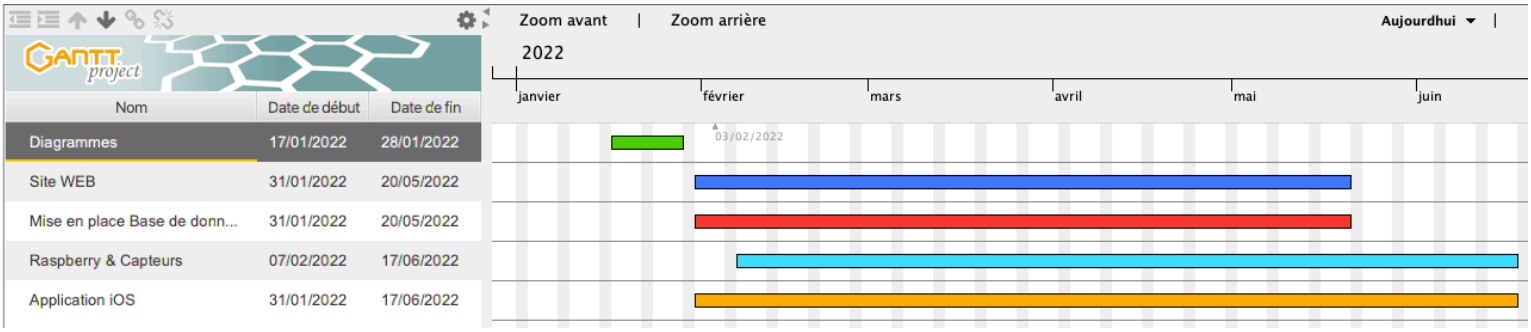
- **Le langage Python pour les capteurs**

- **Le langage SwiftUI pour l'application iOS**

- **Le langage PHP pour l'API afin de communiquer avec la base de données**

III – Spécifications générales

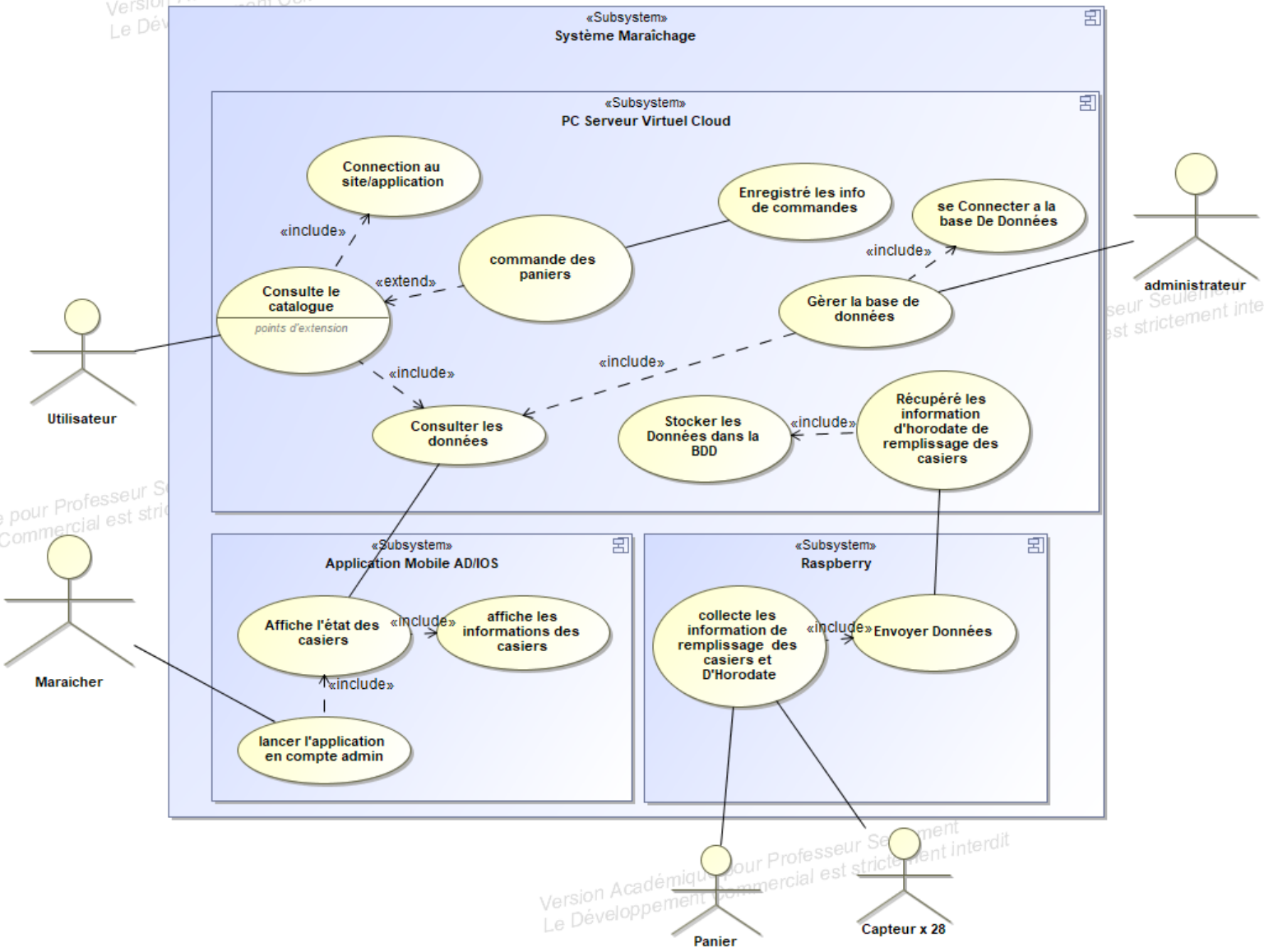
1. Diagramme de Gantt (planification des tâches)



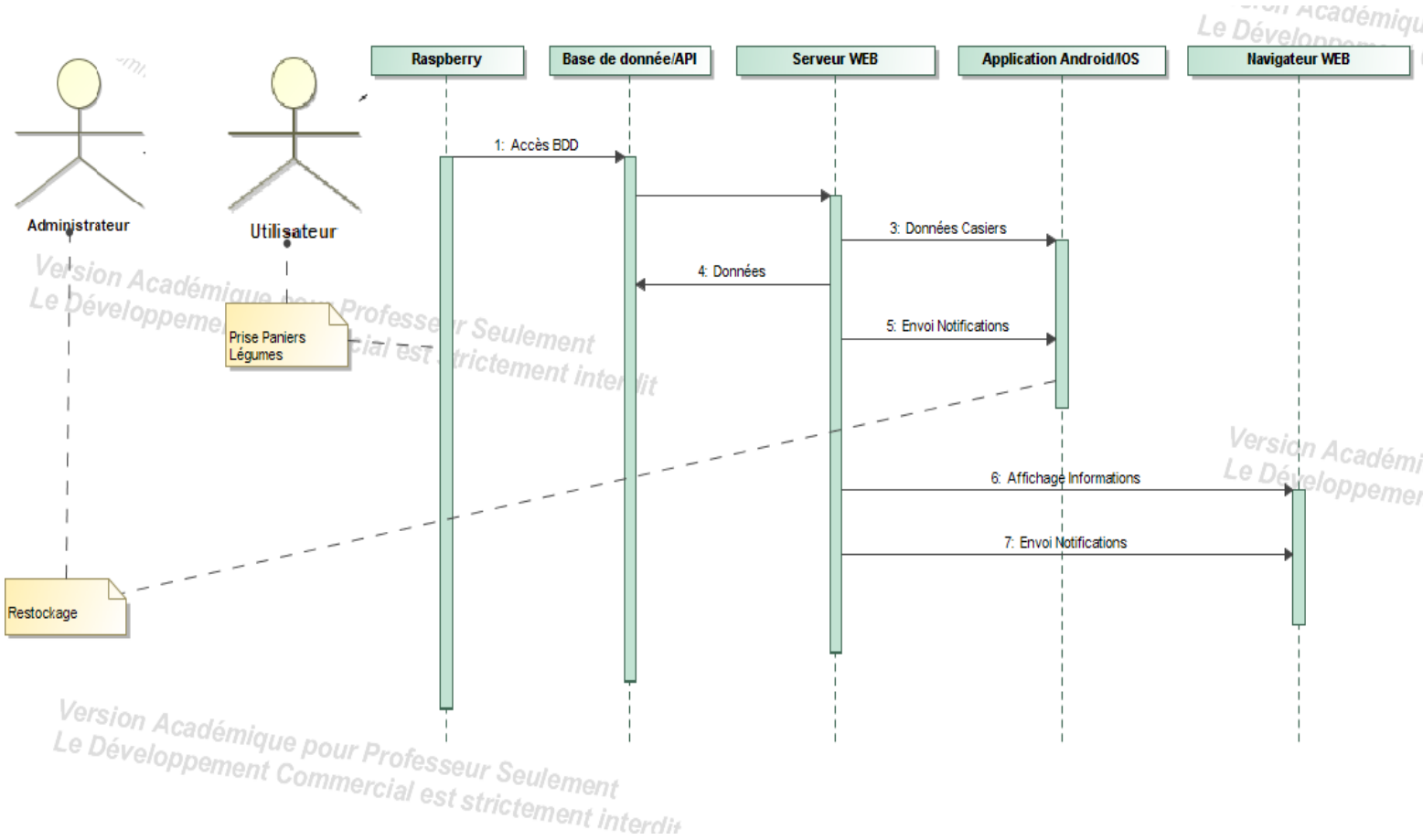
Nom	Date de début	Date de fin
Diagrammes	17/01/2022	28/01/2022
Site WEB	31/01/2022	20/05/2022
Mise en place Base de donn...	31/01/2022	20/05/2022
Raspberry & Capteurs	07/02/2022	17/06/2022
Application iOS	31/01/2022	17/06/2022

- LORIN Dorian
- BERCHEL Rudy
- MAUREL Robin
- TOURNACHE Nicolas

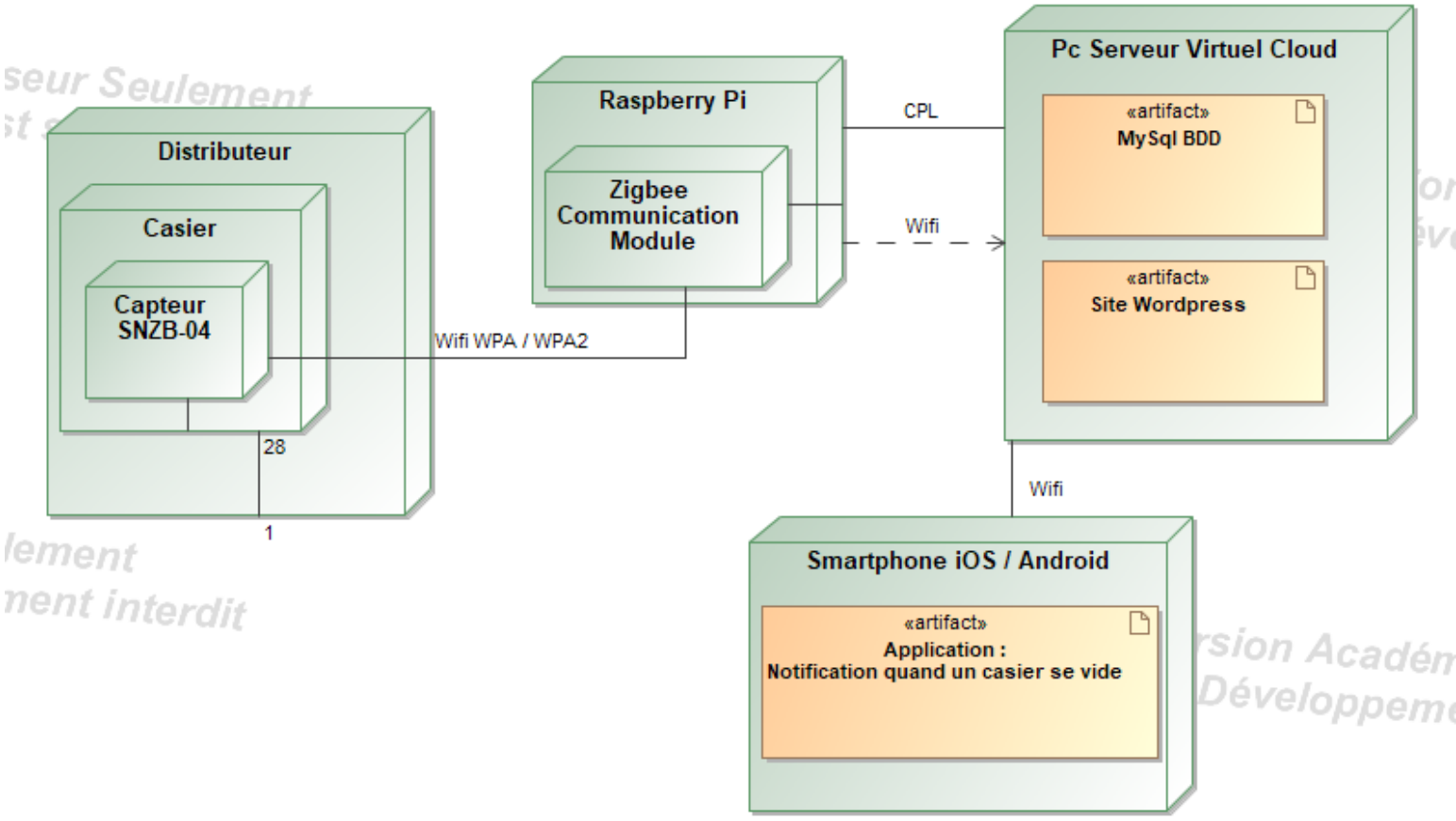
2. Diagramme de cas d'utilisation



3. Diagramme de séquence



4. Diagramme de déploiement



5. Structure de la base de données

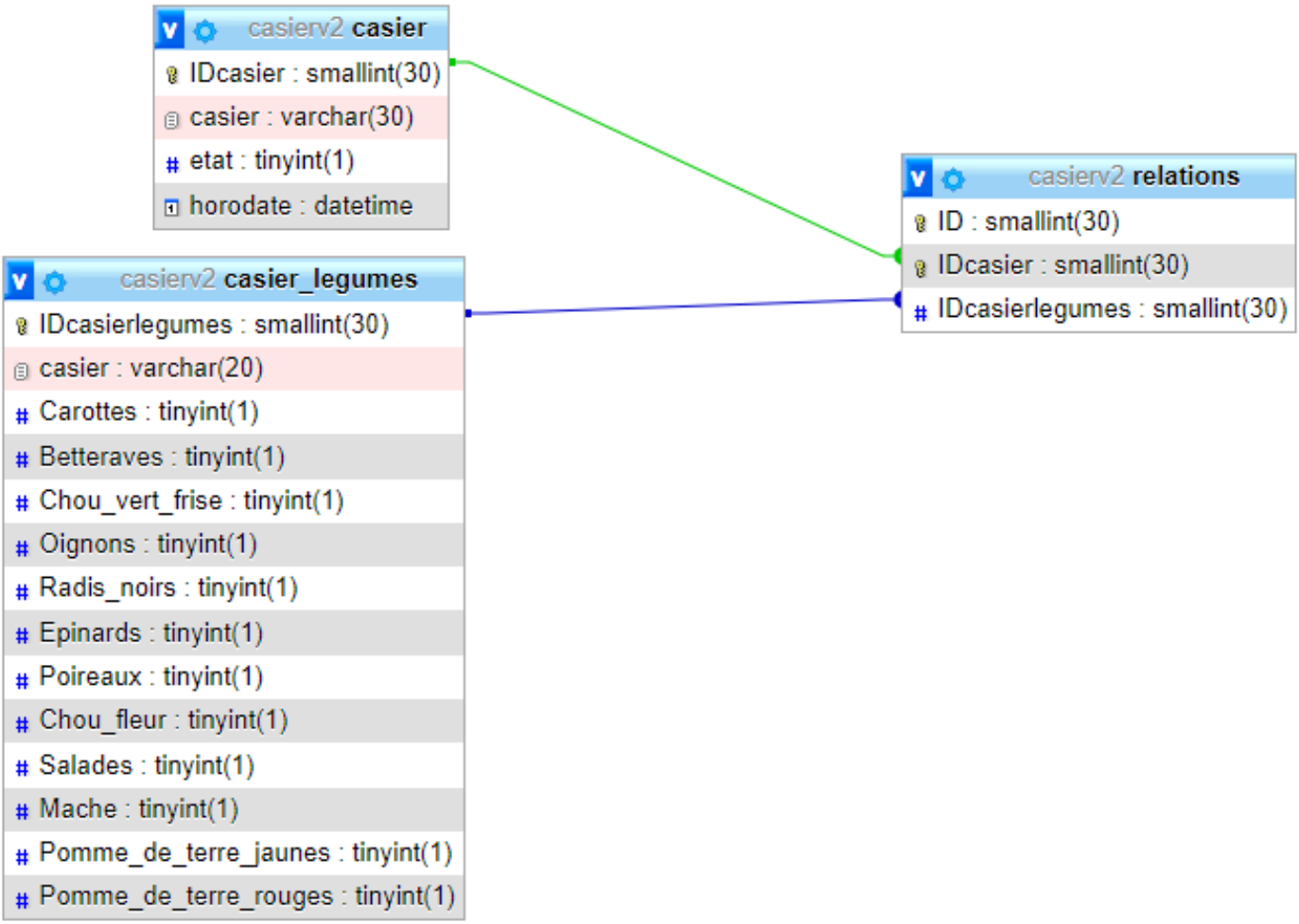


Table casier :

- IDcasier : Identifiants des casiers
- Casier : Noms des casiers (Casier 1, Casier 2, …)
- État : Permet de connaître l'état des casiers (vides ou plein)
- Horodate : Permet de stocker la date et l'heure de l'ouverture d'un casier

Table casier_légumes :

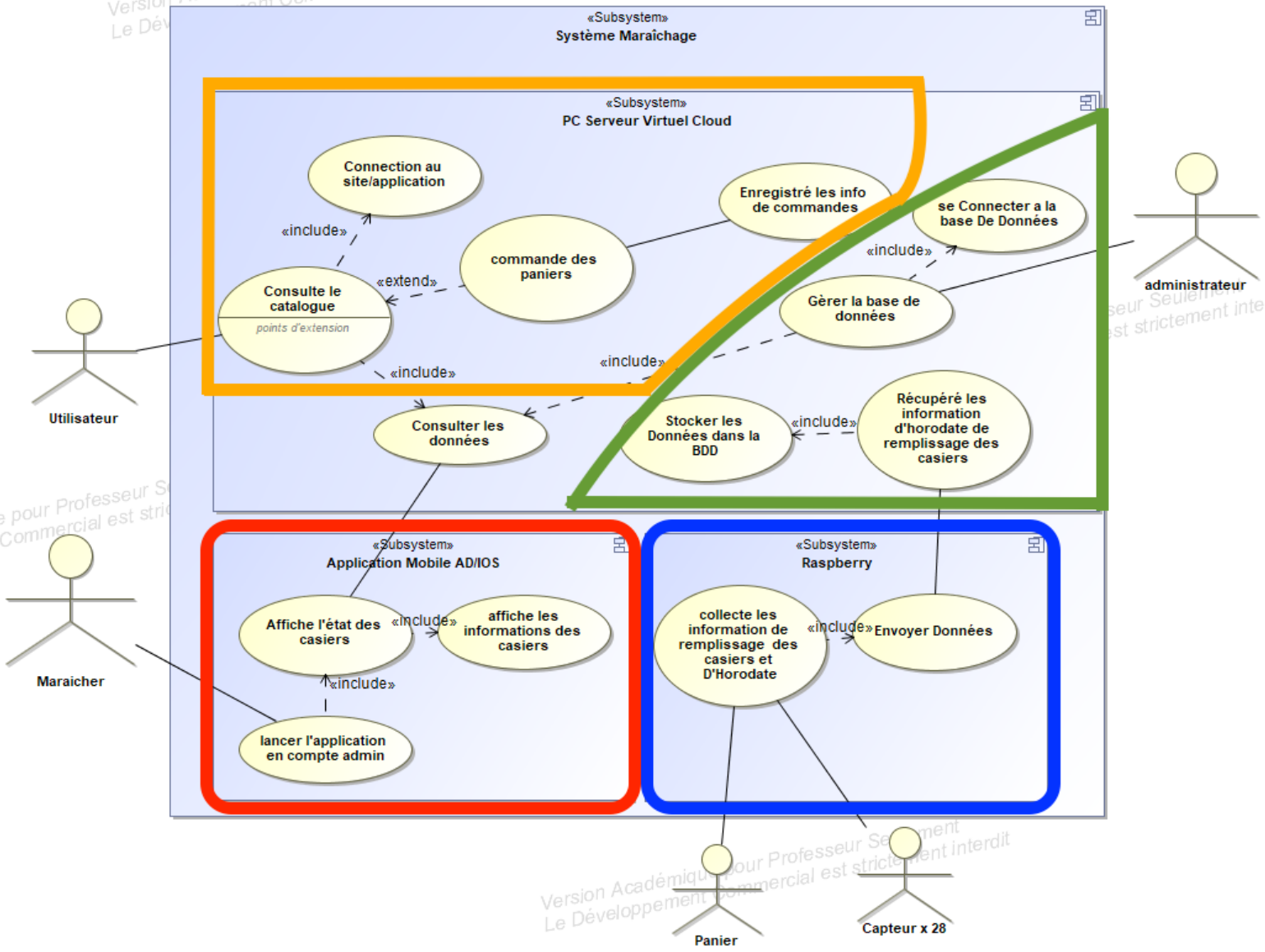
- IDcasierlegumes : Identifiant du casier
- Casier : Nom du casier
- Carottes […] Pomme_de_terre_rouges : Présence ou non du légume dans le casier

IV – Répartition des tâches

1. Tableau de répartition des tâches

NOM Étudiant	Tâche principale	Matériel/Solution
BERCHEL Rudy	Installer un serveur de base de données afin de stocker les informations des casiers et des informations pour l'administrateur. Élaborer une API pour communiquer avec le serveur WEB et l'application mobile. Accéder avec une interface graphique à la base de données	**Serveur sur le cloud comprenant la base de données ainsi que l'API permettant de communiquer avec le site web et l'application.**
LORIN Dorian	Installer un serveur WEB sous Wordpress et y rajouter du contenu supplémentaire. Élaborer un service permettant de faire l'interaction entre la base de données et le serveur et ainsi permettre l'envoi de notifications sur le navigateur WEB et les applications mobiles.	**Serveur sur le cloud avec le site web connecté à la base de données.**
TOURNACHE Nicolas	Élaborer un logiciel sous iOS/Android permettant de visualiser l'état des casiers en temps réel et y intégrer un micro-service permettant de connaître l'état du stock de paniers de légumes afin de connaître les légumes à amener au système. Intégrer l'envoi de notifications à l'utilisateur ainsi qu'un onglet paramètres afin de configurer la distribution des notifications.	**Téléphone/Tablette sous iOS/Android** **Xcode et le langage SWIFT pour le développement de l'application.**
MAUREL Robin	Choisir et dimensionner une technologie de connexion de données afin d'intégrer une solution matérielle sur le système afin de détecter la prise de paniers de légumes. Réaliser l'intégration logicielle dans le matériel sélectionné afin de détecter la prise de paniers de légumes. Élaborer un système de logs sur le matériel choisi en local pour horodater la prise de paniers de légumes.	**Raspberry PI** **Capteurs d'ouverture**

2. Tâches sur le diagramme de cas d'utilisation



- MAUREL Robin
- TOURNACHE Nicolas
- BERCHEL Rudy
- LORIN Dorian

V – Développement de ma partie

1. Description de ma partie

Pour ce projet, ma tâche est de réaliser une application mobile qui affichera en temps réel l'état des casiers s'ils sont vides ou pleins. L'application devra également envoyer une notification lorsqu'un panier de légumes est récupéré. Ainsi le maraîcher pourra alors savoir quand il faut remplir un casier.
Les tâches que j'ai réalisées sont :

- Réaliser une application iOS
- Récupérer les données de l'API
- Afficher les données de manière structurée dans l'application
- Intégrer un onglet permettant de gérer les notifications

2. Matériel & technologies employé(e)s

Pour réaliser mon application j'ai utilisé les technologies suivantes :

- **Xcode** afin de coder l'application 
- Le langage **SwiftUI** 

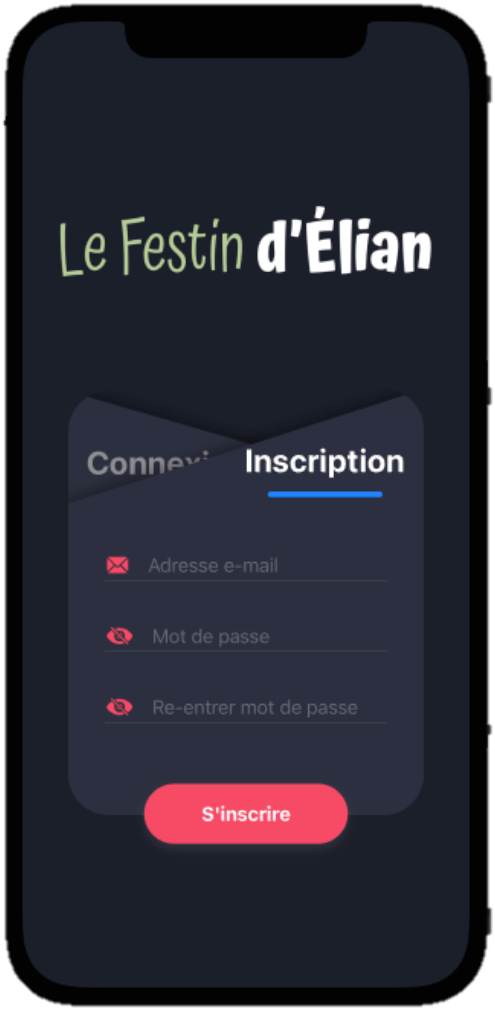
Comme matériel j'utilisais mon téléphone afin de tester tout au long du projet mon application mobile.

3. Écran d'accueil de l'application

A. Écran d'accueil

Au lancement de l'application l'utilisateur doit se connecter à l'aide d'un compte créé précédemment via l'onglet inscription.

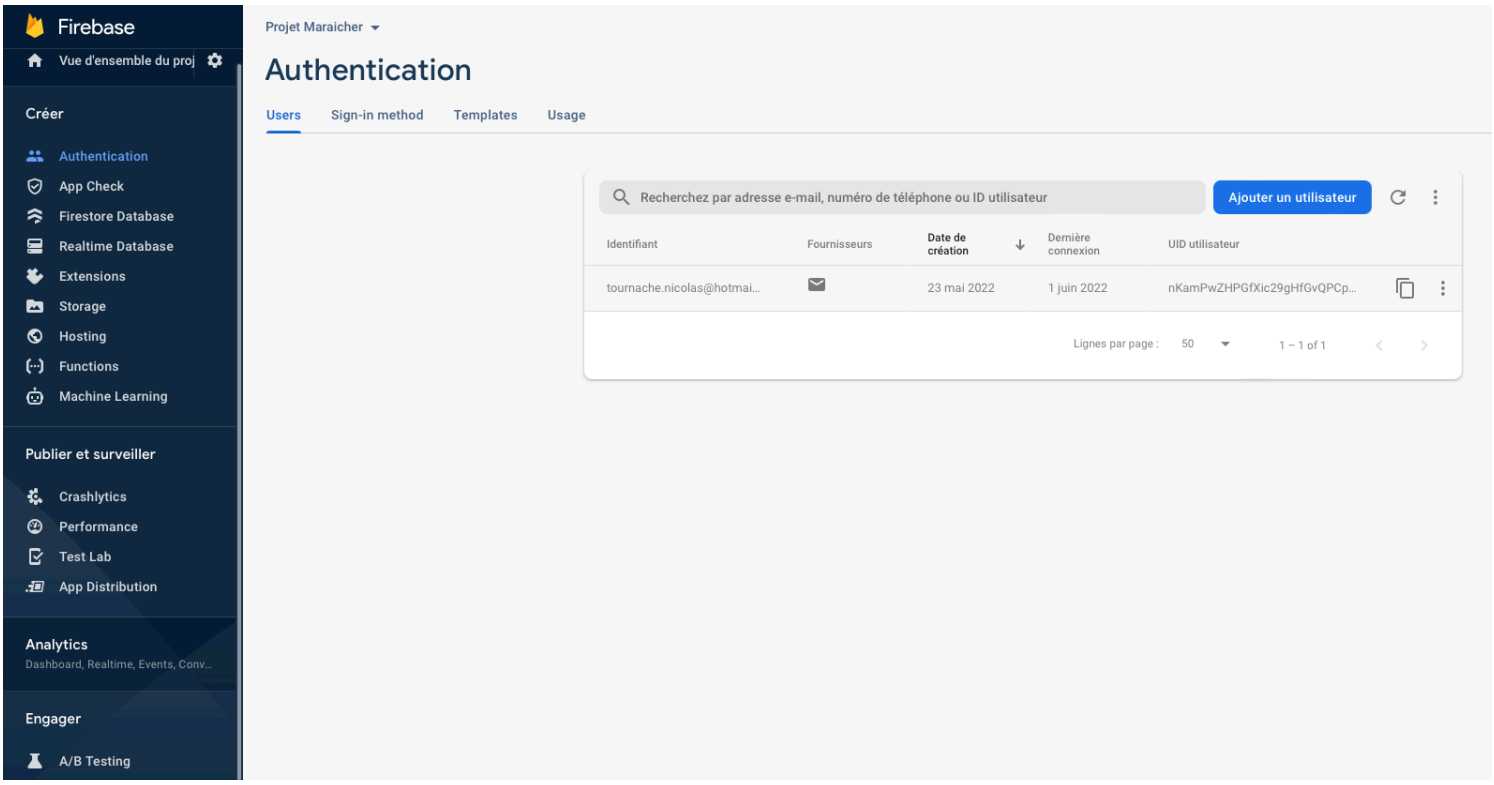
Connexion



Inscription

B. Fonctions d'authentification

Pour les fonctionnalités d'authentification, j'utilise Firebase, un service en ligne fournissant plusieurs fonctionnalités comme des bases de données, l'envoi de notification ou bien l'authentification.

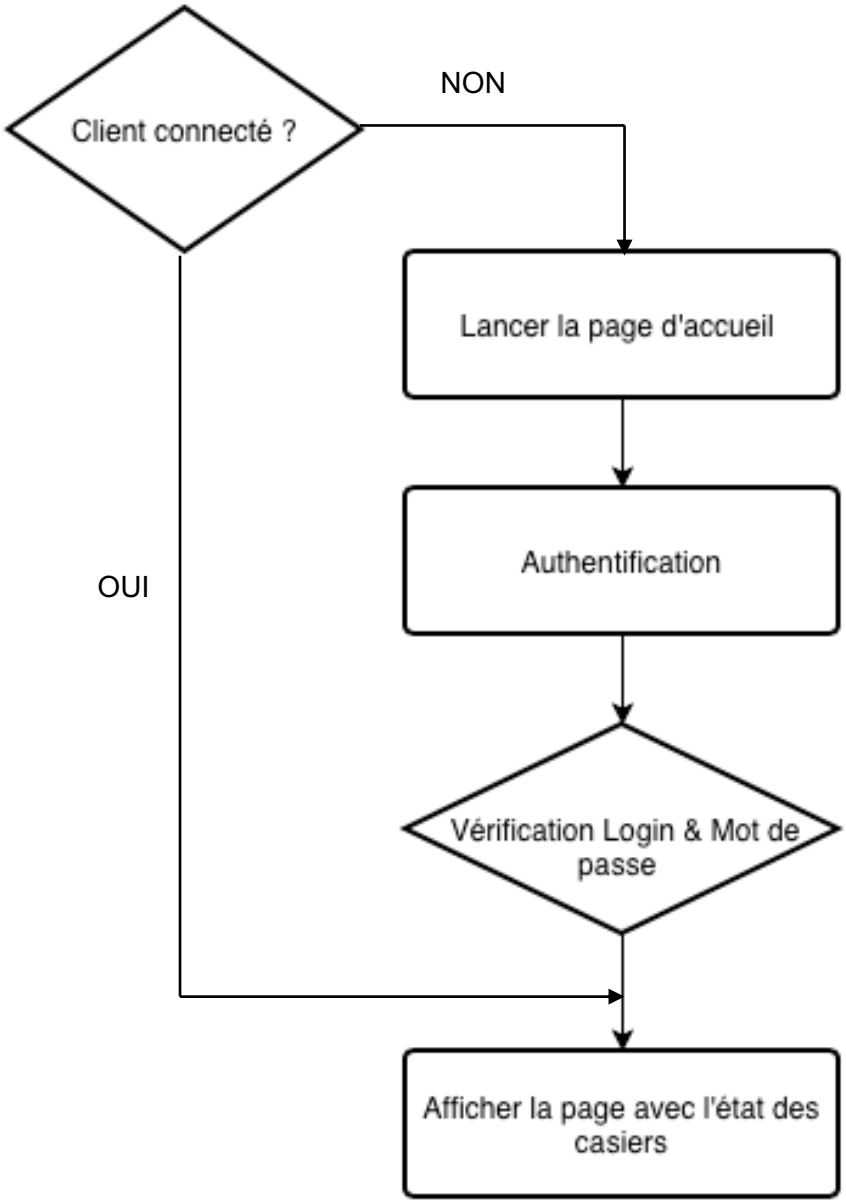


Capture d'écran de l'onglet Authentification sur Firebase.

Pour connecter Firebase a mon projet il a fallu rajouter quelques lignes de codes dans le fichier AppDelegate.

```swift
import SwiftUI
import Firebase

@main
struct AppliMaraichageApp: App {

    @UIApplicationDelegateAdaptor(AppDelegate.self) var appDelegate

    var body: some Scene {
        WindowGroup {
            let viewModel = AppViewModel()
            ContentView()
                .environmentObject(viewModel)
        }
    }
}

class AppDelegate: NSObject, UIApplicationDelegate {
    func application(_ application: UIApplication, didFinishLaunchingWithOptions lauchOptions:
                     [UIApplication.LaunchOptionsKey : Any]? = nil) -> Bool {
        FirebaseApp.configure()

        return true
    }
}
```

C. Algorigramme de l'écran d'accueil :



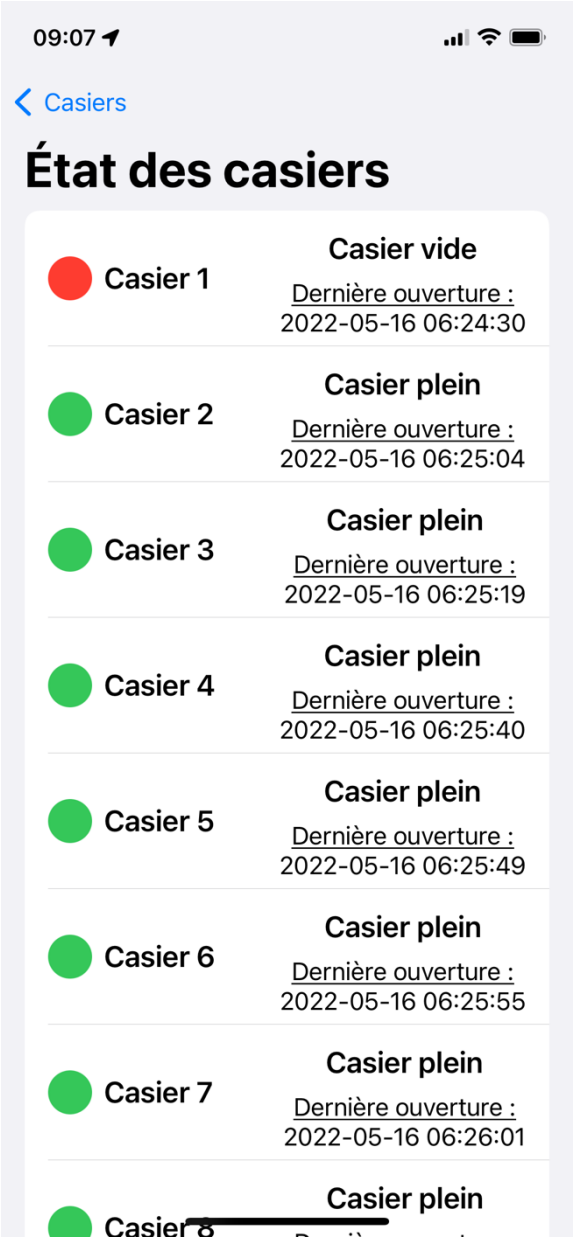
4. Affichage des casiers

A. Liste des casiers

Les parties Connexion et Inscription sont indépendantes et possèdent leurs propre vues. Une fois l'utilisateur connecté, la vue des casiers est affichée.

Pour l'interface j'ai décidé de dresser une liste affichant la totalité des casiers. Une ligne correspond à un casier et sur chaque ligne on retrouve trois informations correspondant aux casiers :

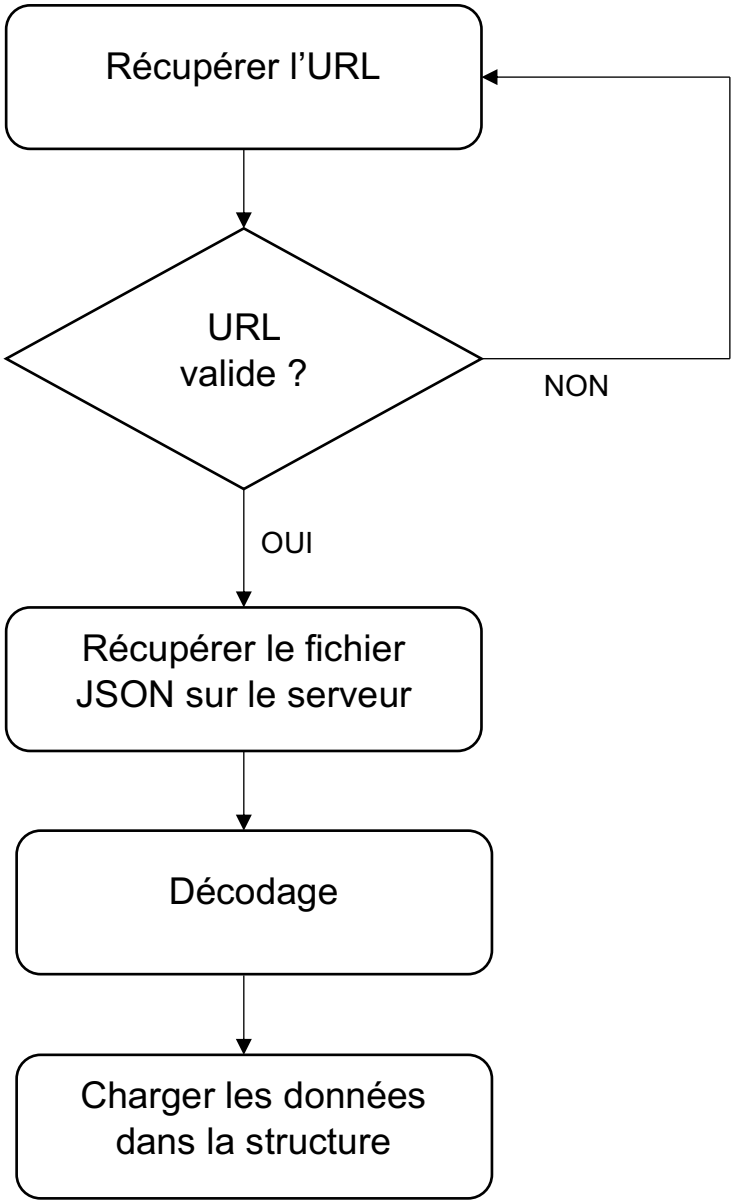
- Une forme ronde désignant l'état des casiers : VERT -> PLEIN
  ROUGE -> VIDE
- Le nom du casier avec son numéro
- L'heure et la date de la dernière ouverture du casier



Les noms des casiers sont récupérés comme tels depuis l'api.

La couleur indiquant l'état et le texte indiquant casier vide ou plein sont obtenu grâce à une fonction simple ; en fonction de l'état d'un casier représenté avec 0 ou 1 sur la base de données, le programme va afficher un point vert et afficher « casier plein », sinon, un point rouge et « casier vide »

B. Algorigramme partie casiers :



VI – Annexes

1. Code Écran d'accueil

A. Fichier AppViewModel

Le fichier AppViewModel défini les fonctions de connexion, inscription, déconnexion et mot de passe oublié.

```swift
import Foundation
import Firebase

class AppViewModel: ObservableObject {
    let auth = Auth.auth()
    @Published var signedIn = false
    var isSignedIn: Bool {
        return auth.currentUser != nil
    }

    //Fonction CONNEXION
    func signIn(email: String, password: String) {
        auth.signIn(withEmail: email,
                    password: password) { [weak self] result, error in
            guard result != nil, error == nil else {
                return
            }
            DispatchQueue.main.async {
                //Succès
                self?.signedIn = true
            }
        }
    }

    // Fonction INSCRIPTION
    func signUp(email: String, password: String) {
        auth.createUser(withEmail: email,
                        password: password) { [weak self] result, error in
            guard result != nil, error == nil else {
                return
            }
            DispatchQueue.main.async {
                //Succès
                self?.signedIn = true
            }
        }
    }

    // Fonction DÉCONNEXION
    func signOut() {
        try? auth.signOut()
        self.signedIn = false
    }
}
```

B. Fichier Login

Le fichier login correspond à la vue login avec les champs de connexion.

```swift
//
//  Login.swift
//  AppliMaraichage
//
//  Created by Nicolas on 22/04/2022.
//

import SwiftUI

struct Login: View {
    @EnvironmentObject var viewModel: AppViewModel
    @State var email = ""
    @State var password = ""
    @Binding var index : Int

    var body: some View{

        ZStack(alignment: .bottom) {
            VStack{
                HStack{
                    VStack(spacing: 10){
                        Text("Connexion")
                            .foregroundColor(self.index == 0 ? .white : .gray)
                            .font(.title)
                            .fontWeight(.bold)
                        Capsule()
                            .fill(self.index == 0 ? Color.blue : Color.clear)
                            .frame(width: 100, height: 5)
                     }
                    Spacer(minLength: 0)
                }
                .padding(.top, 30)// for top curve...

                VStack{
                    HStack(spacing: 15){
                        Image(systemName: "envelope.fill")
                        .foregroundColor(Color("Color1"))

                        TextField("Adresse e-mail", text: self.$email)
                    }
                    Divider().background(Color.white.opacity(0.5))
                }
                .padding(.horizontal)
                .padding(.top, 40)

                VStack{
                    HStack(spacing: 15){

                        Image(systemName: "eye.slash.fill")
                        .foregroundColor(Color("Color1"))

                        SecureField("Mot de passe", text: self.$password)
                    }
                    Divider().background(Color.white.opacity(0.5))
```

```
                    Divider().background(Color.white.opacity(0.5))
                }
                .padding(.horizontal)
                .padding(.top, 30)

                HStack{
                    Spacer(minLength: 0)
                    Button(action: {
                    }) { Text("Mot de passe oublié ?")
                            .foregroundColor(Color.white.opacity(0.6))
                        }
                }
                .padding(.horizontal)
                .padding(.top, 30)
            }
            .padding()
            .padding(.bottom, 65)
            .background(Color("Color2"))
            .clipShape(CShape())
            .contentShape(CShape())
            .shadow(color: Color.black.opacity(0.3), radius: 5, x: 0, y: -5)
            .onTapGesture {
                self.index = 0
            }
            .cornerRadius(35)
            .padding(.horizontal,20)

            // Button...

            Button(action: {
                guard !email.isEmpty, !password.isEmpty else {
                    return
                }
                viewModel.signIn(email: email, password: password)
            }) {

                Text("Connexion")
                    .foregroundColor(.white)
                    .fontWeight(.bold)
                    .padding(.vertical)
                    .padding(.horizontal, 50)
                    .background(Color("Color1"))
                    .clipShape(Capsule())
                    // shadow...
                    .shadow(color: Color.white.opacity(0.1), radius: 5, x: 0, y: 5)
            }
            // moving view down..
            .offset(y: 25)
            .opacity(self.index == 0 ? 1 : 0)
        }
    }
}

struct CShape : Shape {
    func path(in rect: CGRect) -> Path {
        return Path { path in

            path.move(to: CGPoint(x: rect.width, y: 90))
            path.addLine(to: CGPoint(x: rect.width, y: rect.height))
            path.addLine(to: CGPoint(x: 0, y: rect.height))
            path.addLine(to: CGPoint(x: 0, y: 0))
        }
    }
}
```

C. Fichier Inscription

```swift
//
//  Inscription.swift
//  AppliMaraichage
//
//  Created by Nicolas on 22/04/2022.
//

import SwiftUI

struct Inscription: View {
    @EnvironmentObject var viewModel: AppViewModel
    @State var email = ""
    @State var password = ""
    @State var Repass = ""
    @Binding var index : Int

    var body: some View{

        ZStack(alignment: .bottom) {

            VStack{

                HStack{

                    Spacer(minLength: 0)

                    VStack(spacing: 10){
                        Text("Inscription")
                            .foregroundColor(self.index == 1 ? .white : .gray)
                            .font(.title)
                            .fontWeight(.bold)

                        Capsule()
                            .fill(self.index == 1 ? Color.blue : Color.clear)
                            .frame(width: 100, height: 5)
                    }
                }
                .padding(.top, 30)// for top curve...

                VStack{
                    HStack(spacing: 15){
                        Image(systemName: "envelope.fill")
                            .foregroundColor(Color("Color1"))
                        TextField("Adresse e-mail", text: self.$email)
                    }
                    Divider().background(Color.white.opacity(0.5))
                }
                .padding(.horizontal)
                .padding(.top, 40)

                VStack{
                    HStack(spacing: 15){
                        Image(systemName: "eye.slash.fill")
                            .foregroundColor(Color("Color1"))
```

```
                    HStack(spacing: 15){
                        Image(systemName: "eye.slash.fill")
                            .foregroundColor(Color("Color1"))
                        SecureField("Mot de passe", text: self.$password)
                    }
                    Divider().background(Color.white.opacity(0.5))
                }
                .padding(.horizontal)
                .padding(.top, 30)

                // On remplace mot de passe oublié par ré-entrer le mot de passe
                VStack{
                    HStack(spacing: 15){
                        Image(systemName: "eye.slash.fill")
                            .foregroundColor(Color("Color1"))
                        SecureField("Re-entrer mot de passe", text: self.$Repass)
                    }
                    Divider().background(Color.white.opacity(0.5))
                }
                .padding(.horizontal)
                .padding(.top, 30)
            }
            .padding()
            // bottom padding...
            .padding(.bottom, 65)
            .background(Color("Color2"))
            .clipShape(CShape1())
            // clipping the content shape also for tap gesture...
            .contentShape(CShape1())
            // shadow...
            .shadow(color: Color.black.opacity(0.3), radius: 5, x: 0, y: -5)
            .onTapGesture {

                self.index = 1

            }
            .cornerRadius(35)
            .padding(.horizontal,20)

            // Button...

            Button(action: {
                guard !email.isEmpty, !password.isEmpty else {
                    return
                }
                viewModel.signUp(email: email, password: password)
            }) {
                Text("S'inscrire")
                    .foregroundColor(.white)
                    .fontWeight(.bold)
                    .padding(.vertical)
                    .padding(.horizontal, 50)
                    .background(Color("Color1"))
                    .clipShape(Capsule())
```

```
                    .background(Color("Color1"))
                    .clipShape(Capsule())
                    // shadow...
                    .shadow(color: Color.white.opacity(0.1), radius: 5, x: 0, y: 5)
            }
            // moving view down..
            .offset(y: 25)
            // hiding view when its in background...
            // only button...
            .opacity(self.index == 1 ? 1 : 0)
        }
    }
}


struct CShape1 : Shape {
    func path(in rect: CGRect) -> Path {
        return Path { path in

            path.move(to: CGPoint(x: 0, y: 100))
            path.addLine(to: CGPoint(x: 0, y: rect.height))
            path.addLine(to: CGPoint(x: rect.width, y: rect.height))
            path.addLine(to: CGPoint(x: rect.width, y: 0))
        }
    }
}
```

D. Fichier ContentView

Le fichier ContentView contient les instructions au lancement de l'application.

```swift
//
//  ContentView.swift
//  AppliMaraichage
//
//  Created by Nicolas on 20/04/2022.
//

import SwiftUI
import FirebaseAuth

struct ContentView: View {
    @EnvironmentObject var viewModel: AppViewModel

    var body: some View {
        NavigationView {
            //Si utilisateur est deja connecté
            if viewModel.signedIn {
                //On afficher CasierView
                CasierList()
            }
            //Sinon on affiche l'écran d'acceuil
            else {
                Home()
                    .preferredColorScheme(.dark)
            }
        }
        .onAppear {
            viewModel.signedIn = viewModel.isSignedIn
        }
    }
}
```

Dans le fichier ContentView, une deuxième structure nommée « Home » permet de lier les vues Connexion et Inscription.

```
34  struct Home : View {
35      @State var email = ""
36      @State var password = ""
37      @State var index = 0
38
39      var body: some View {
40          GeometryReader { _ in
41              VStack {
42                  Image("logoFestinElianLight")
43                      .resizable()
44                      .frame(width: 350, height: 100)
45                      //.padding(.top, 60)
46                  ScrollView {
47                      ZStack {
48                          Inscription(index: self.$index)
49                          //Changer ordre des vues ...
50                              .zIndex(Double(self.index))
51                          Login(index: self.$index)
52                      }
53                      .padding(.horizontal, 20)
54                      .padding(.top, 70)
55                      }
56                  }
57          }
58          .background(Color.black.edgesIgnoringSafeArea(.all))
59      }
60  }
```

2. Code affichage des casiers

A. Fichier CasierModel

Le fichier CasierModel défini la structure des données récupérées de l'API soit :

- Le nom des casiers avec leurs numéros
- L'état des casiers
- L'horodatage des casiers

```swift
//
//  CasierModel.swift
//  ProjetMaraicher
//
//  Created by Nicolas on 14/03/2022.
//


import Foundation

struct Casier: Codable, Identifiable {
    let id = UUID()
    var casier: String
    var etat: String
    var horodate: String
}

class Api {
    func decode(completion: @escaping ([Casier]) -> ()) {
        guard let url = URL(string:
            "http://172.16.99.2/projetmaraicher/API/") else { return }

        URLSession.shared.dataTask(with: url) { (data, _, _) in
            let infocasier = try! JSONDecoder().decode([Casier].self, from: data!)

            DispatchQueue.main.async {
                completion(infocasier)
            }
        }
        .resume()
    }
}
```

Une fonction decode() permet de se connecter à l'API et de charger les données dans la structure.

B. Fichier CasierView

Le fichier CasierView est la vue qui s'affiche lorsque l'utilisateur est connecté. Cette vue affiche une liste des casiers.

```swift
//
//  CasierView.swift
//  ProjetMaraicher
//
//  Created by Nicolas on 14/03/2022.
//

import SwiftUI

// Liste des casiers
struct CasierList: View {
    @State var Casiers: [Casier] = []
    @EnvironmentObject var viewModel : AppViewModel

    var body: some View {
        //Liste
        List(Casiers) { item in
            HStack {
                if (item.etat == "0") {
                    Circle()
                        .fill(.green)
                        .frame(width: 30, height: 30)
                } else {
                    Circle()
                        .fill(.red)
                        .frame(width: 30, height: 30)
                }

                //Spacer()

                //Afficher les noms des casiers
                Text(item.casier)
                    .font(Font.system(.title3))
                    .bold()

                Spacer()

                VStack {
                    //Espace entre cette partie et le nom des casiers
                    Spacer()

                    //Si l'état d'un casier est égal à zéro alors on affiche Casier plein
                    if (item.etat == "0") {
                        Text("Casier plein")
                            .font(Font.system(size: 20))
                            .bold()
                    //Sinon on affiche Casier vide
                    } else {
                        Text("Casier vide")
                            .font(Font.system(size: 20))
                            .bold()
                    }
                    Spacer()

```

```
53                      Spacer()
54
55                      Text("Dernière ouverture :")
56                          .underline()
57                      //Afficher l'horodatage des casiers
58                      Text(item.horodate)
59                  }
60              }
61          }.navigationTitle("État des casiers")
62              .toolbar {
63                  Button(action: { viewModel.signOut() },
64                         label: { Text("Déconnexion")})
65              }
66
67          .onAppear {
68              Api().decode { (Casiers) in
69                  self.Casiers = Casiers
70              }
71          }
72      }
73  }
```