OC P10 BONUS

Concept

Reciplease est une application qui permet à l'utilisateur de rechercher des recettes suivant les ingrédients présent dans son frigo. L'appilcation repose sur l'API REST de Yummli

Bonus

L'application présente 2 bonus :

- Une option de partage de la recette
- Un système de gestion d'utilisateurs (creation, login, Facebook Login)

Bonus 1: Partage de recette

Dans le Vue Detail, nous ajoutons un bouton à la bar de navigation.

```
let shareImage = UIImage(named: "export.png")
    let shareItem = UIBarButtonItem(image: shareImage, style: .plain, target: self
, action: #selector(self.share(sender:)))
    //instantiate right barButton in navBar
    self.navigationItem.rightBarButtonItems = [item,shareItem]
```

Ce dernier a une action share item. Cette dernière fait appel à un UIActivityViewController qui permet de partager le lien de la recette.

```
  @objc func share(sender:UIBarButtonItem){
    if recipe?.url != nil {
      let recipeUrl = recipe?.url
      let activityController = UIActivityViewController(activityItems: [recipeUrl!
], applicationActivities: nil)
    present( activityController, animated: true, completion: nil )}
    else {
      showAlert(message: "Sorry you cannot share this recipe")
    }
  }
```

L'API renvoie parfois un objet recette qui ne contient pas d'url, c'est pour cela que nous validons l'existence de cette dernière par la condition `if recipe?.url != nil`.

Bonus 2: Gestion d'utilisateurs

La gestion d'utilisateurs se fait en plusieurs étapes :

1. Un Login Manager controller en charge de l'affichage et de la validation des paramètres pour créer un compte et se logguer.
2. L'implémentation du FacebookLoginSDK et facebookCoreSDK pour récupérer les informations
3. Le stockage et la transmission de de l'utilisateur "loggué"
4. La création et la gestion des comptes utilisateurs et de leurs recettes

Bonus 2: Login Manager

Installation du framework ILLoginKit

Pour implémenter le login manager nous avons pris l'option d'implémenter un framework exterieur ILLoginKit [lien vers le repo github](#).

Afin d'installer ce dernier, nous ajoutons utilisons CocoaPod et la ligne de commande suivante dans notre fichier pod `pod'ILLoginKit'` . ensuite nous entrons la commande `pod install` .

Ce framework install aussi 3 framework de dépdendance:

- **Validator** en charge de la validation des schéma alphanumériques
- **FBSDKLoginKit** en charge de l'authentification via Facebook
- **FBSDKCoreKit** en charger du rapatriement es informations du compte Facebook

Implémentation du LoginController

Suivant la documentation, nous créons un controller et une vue. Dans ce dernier nous instations un login coordinator de la class LoginCoordinator() et nou appelons la fonction start() dans la méthod viewDidLoad().

Ce login coordinator prend en charge l'affichage des différentes vues :

- login
- Inscription
- login via FB

Il hérite de la classe ILLoginKit.LoginCoordinator. Dans cette dernière, nous parametrons l'apparence et les actions à effectuer en cas de succès du login ou de l'insription.

L'affichage du LoginController se fait dans le fichier AppDelegate.swift.

Afin de le faire apparaitre avant l'application, nous definissons le login Controller comme

rootViewController.

Afin d'afficher, le MainTabBarController en cas de succès, nous définissons ce dernier comme root view Controller et nous le présentons.

```
func presentTabBarController(){
  let tabVc = MainTabBarController()
  let appDelegate = UIApplication.shared.delegate
  appDelegate?.window!?.rootViewController = tabVc
}
```

Nous évoquerons le passage de données plus tard dans un autre chapitre.

La gestion des données utilisateurs

Maintenant que nous avons un login controller qui permet à l'utilisateur d'entrer ses informations, nous devons gérer ces informations et les sauvegarder. Pour cela, nous créons un jeu de méthode dans le CoreDataManager.

- Création d'un utilisateur pour l'inscription et le login fb (première utilisation)
- Récupération des données utilisateurs
- Vérification si l'utilisateur est déjà inscrit

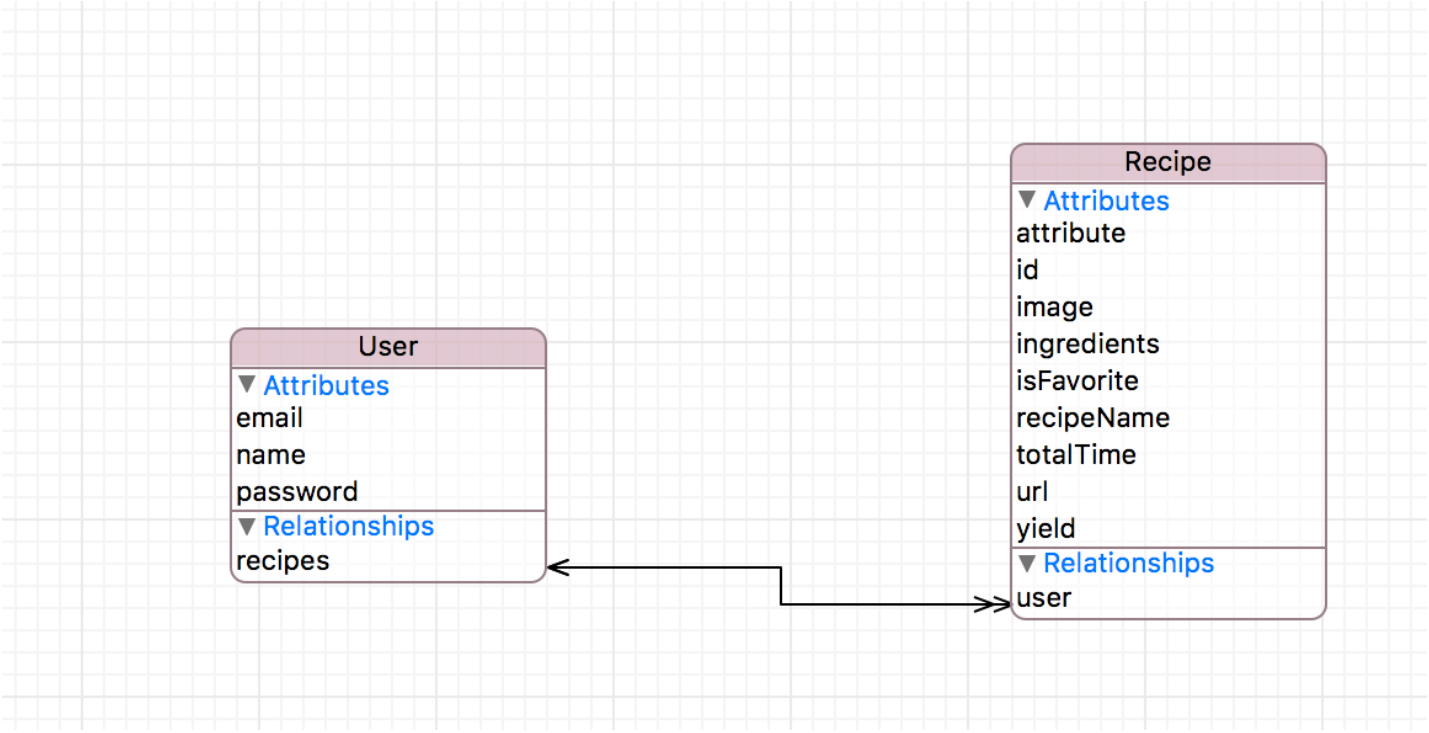
L'ensemble de ses action nécessite une modification du model lui-même.

Modèle de données

Nous créons donc une entité User avec les propriétés

- nom
- email
- password

Chacun de ses utilisateurs aura accès à ses recettes favorites. Pour cela nous créons une relation 1 à plusieurs.



Création d'un utilisateur

Nous créons une nouvelle instance de l'entité pour chaque utilisateur inscrit.

```
  func createUser(name: String, email: String, password: String) {
      // summons entity
      let entity = NSEntityDescription.entity(forEntityName: "User", in: managedOb
jectContext())
      // create new entity instance
      let newUser = NSManagedObject(entity: entity!, insertInto: managedObjectCont
ext())
      // sets value
      newUser.setValue(name, forKeyPath: "name")
      newUser.setValue(email, forKeyPath: "email")
      newUser.setValue(password, forKeyPath: "password")
      // save Context
      do {
        try managedObjectContext().save()
      } catch {
        print("saving error")
      }
    setLoggedUser(email: email)
  }
```

Dans le cas d'un facebook Login, nous récupérons, le nom, l'email et assignons le facebookID comme mot de passe.

Verification d'un utilisateur existant

Afin de se logguer l'utilisateur doit être inscrit, nous créons donc la fonction suivante qui renvoie un booléen.

```
func checkExistingUsers(email:String) -> Bool{
    let users : [User] = loadUsers()
    var existingUsers : [String] = []
    for user in users {
      existingUsers.append(user.email!)
    }
    if existingUsers.contains(email) {
      return true
    } else {
      return false
    }
  }
```

Nous récupérons tous les utilisateur enregistrés et comparons les emails avec l'email de login. Dans le cas ou celui correspond à un utilisateur de la base de données, la fonction retourne true.

Sauvegarde de l'identifiant de l'utilisateur logué

L'utilisateur logué ne faisant pas parti des informations à sauvegarder dans la base de donnée, nous utiliserons le userDefaults pour le sauvegardé.

Nous ne pouvons utiliser le delegate pattern car le loginController et les autres controller ne sont pas dans la même pile.

```
  func setLoggedUser(email:String){
    defaults.set(email, forKey: "currentUser")
  }
```

Implémentation dans les méthodes de callback du loginCoordinator

Dans le cas de l'option **Inscription**, nous implémentons, la méthode user, puis nous sauvegardons l'utilisateur dans le userDefaults et présentons le tabBarController.

Dans le cas des options **SignIn** et **Facebook SignIn**, nous implémentons les fonctions loginUser(), checkIfUserExists(), sauvegardons dans le userDefaults l'utilisateur et présentons le tabBarController.

Recupération de l'utilisateur dans le mainTabBarController

Afin de personnalisé l'expérience, une fois l'utilisateur enregistré il doit avoir accès à ses recettes, ses favoris. Pour cela nous devons récupéré les informations de l'utilisateur qui validé les conditions d'enregistrement dans le loginController.

Nous devons donc faire face à 2 problématiques :

- Récupération de l'utilisateur
- passage de ses données utilisateur à travers le différents controllers.

Récupération de l'utilisateur

La récupération se fait dans le MainTabBArController qui est le controller parent de l'application.

Nous définissons une variable user de type User Optionnel. Nous récupérons l'utilisateur en récupérant la String email stockée dans le userDefaults puis nous obtenons l'instance de l'entité user avec cet email.

```
func getCurrentUser() -> User {
    var userLogged: User?
    // grab user's email
    let user = defaults.string(forKey: "currentUser")
    if user != nil {
      print("LOGGED :(user!)")
      // fetch user from Core Data
      userLogged = fetchUser(email: user!)
    }
    return userLogged!
  }
```

transfert de l'information à travers les controllers

Nous utilisons le delegate pattern pour passer les informations entre les controllers.

Pour cela nous créons un protocol

```
protocol userLoggedDelegate {
  ///
  /// Current User
  ///
  /// - Returns: User
  func CurrentUser() -> User
```

Le detail Controller récupère donc l'utilisateur, du RecipeDisplayController ou du favoriteDisplayController, qui'lauront récupérer du HomeViewController.

Nous aurions ou le charger directement dans chaque controller, mais nous souhaitions utiliser le delegate pattern

Conséquences sur la sauvegarde et le display des favoris

Sauver une recette

La première conséquence viens de la nécessité d'attribuer les favoris à un utilisateur pur qu'il puisse les

retrouver.

Nous ajoutons donc un paramètres à la fonction saveRecipe(). Nous ajoutons donc le paramètre user de type User.

dans la méthode, nous assignons cette utilisateur au paramètre de relation : user

```
recipeItem.user = user
```

afficher les recettes

L'affichage des recettes dans le tableViewController Favorites est geré par un fetchResultController.

Afin de n'afficher que les recettes de l'utilisateur loggué, nous ajoutons a la requete du FRC un predicate.

Le predicate va filtrer les recettes favorites stockées dans le Core DAta et n'afficher que celle de l'utilisateur choisi.

```
request.predicate = NSPredicate(format: "user == %@", user!)
```

Chaque utilisateur peut donc avoir uniquement ses recettes.