



# Reciplease

Présentation du projet 10  
Openclassrooms

-  
Mentor : Bertrand Bloc'h  
Élève: Lauriane Haydari



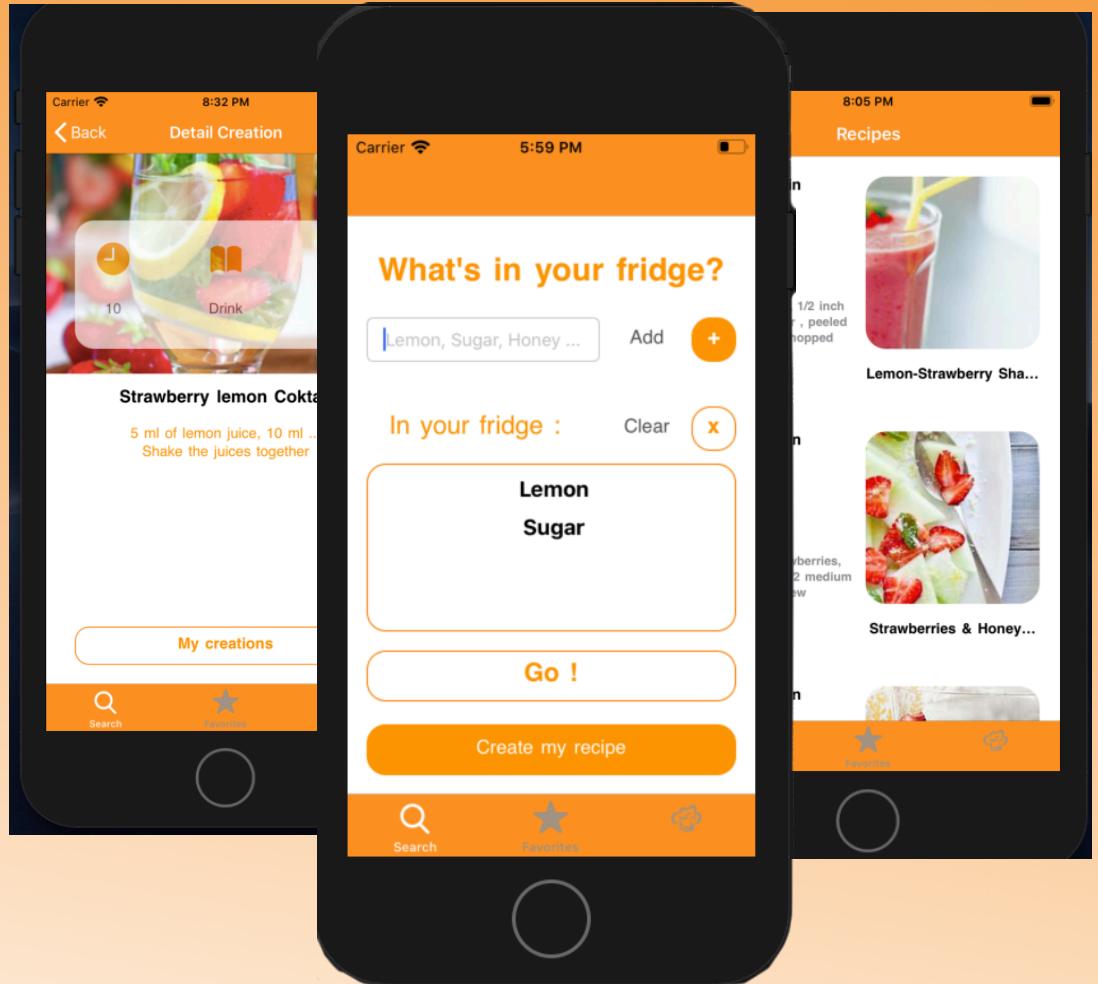
# Reciplease

Reciplease est une application de trois pages :

1. Entrez des ingrédients
2. Recherchez des recettes
3. Sélectionnez la recette qui vous intéresse pour voir le détail
4. Ajoutez la en favoris
5. Découvrez la recette en ligne sur safari
6. Consultez vos recettes favorites

Bonus :

1. Possibilité de créer sa propre recette avec photo et descriptif
2. Ajout et suppressions de la recette dans ma liste de créations





# SIMULATION DE L'APPLICATION

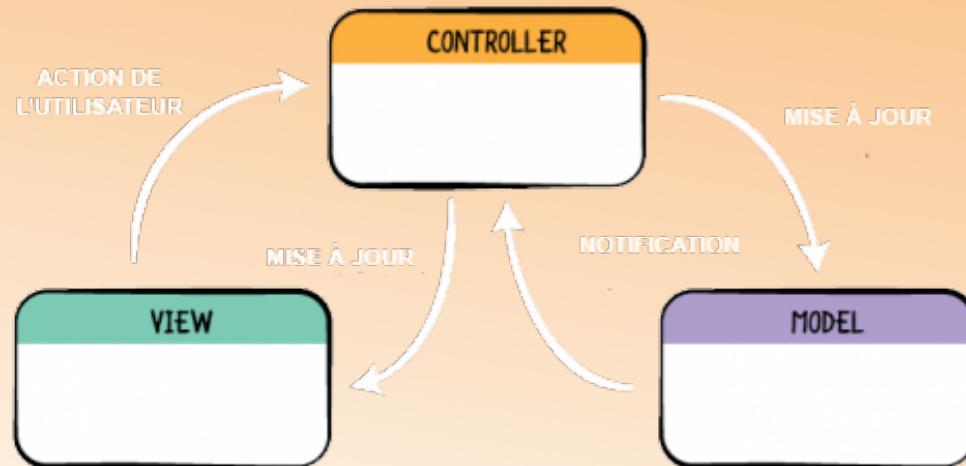




# QU'EST-CE QUE LE MODÈLE MVC ?

Le principe MVC est un patron de conception célèbre et très utilisé qui signifie : Model View Controller.

- **Le Modèle** : gère la logique du programme
- **La Vue** : se concentre sur l'affichage
- **Le Contrôleur** : récupère les informations du modèle et les affiche dans la vue.



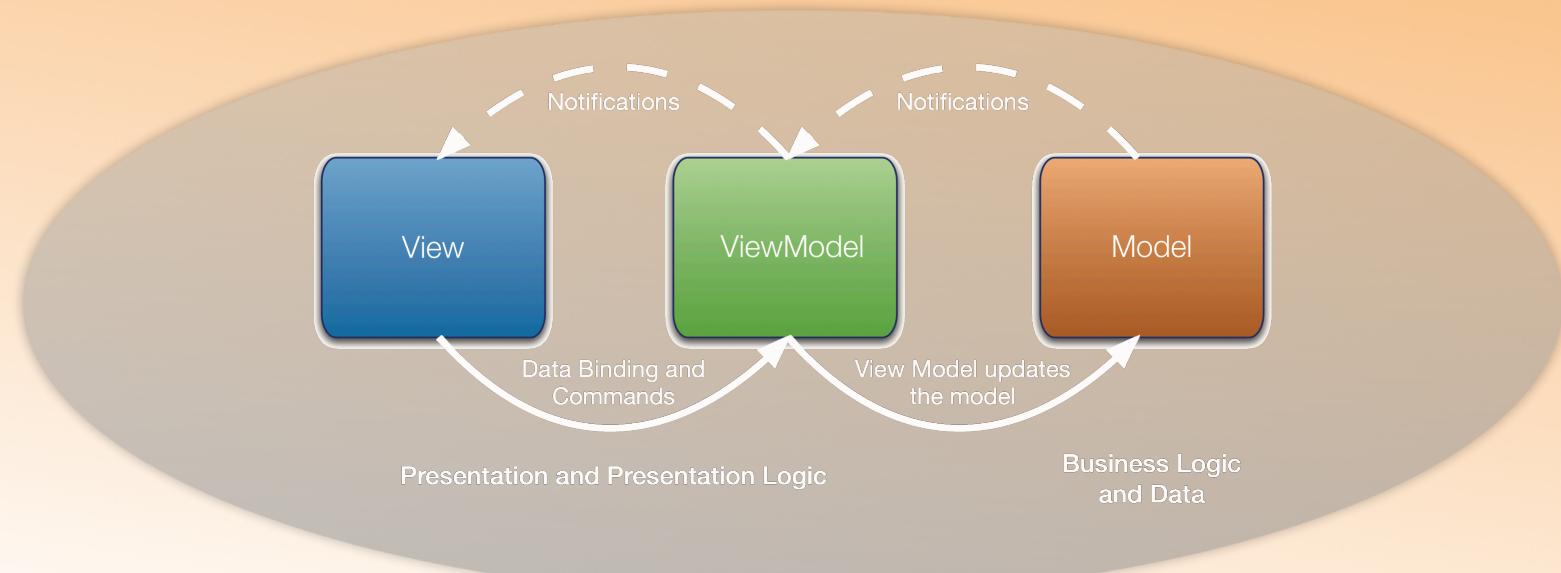


# QU'EST-CE QUE LE MODÈLE MVVM ?

Le Model-View-ViewModel, est un modèle d'architecture.

**Pour ce projet il a été préférable de choisir le MVVM Réactif natif.**

- Les inputs : les événements
- Les outputs : receptacles des événements des inputs correspondants.
- Le ViewController : contrôle la vue, il est claire, simple, il ne connaît pas les données.



# LA DÉMARCHE UTILISÉE



- 1 - Crédation du projet en implémentant Coredata**
- 2 - Crédation de 6 modules (HomeView / Recipes / RecipeDetail / SavingCreation / CreationDetail / CreationsList) en MVVM utilisant le Coordinator et le storyboard de chaque vue**
- 3 - Implémentatation de UI TabBar avec 3 items :**
  - .search**
  - .favorite**
  - .creation**
- 4 - Crédation du network avec installation d'Alamofire par Cocoapod**
- 5 - Écriture des repository par module => requêtes réseaux**
- 6 - Crédation du modèle**
- 7- Injection de Core data dans le context et le context dans l'appDelegate et création des Entity**
- 8- Unit tests**
- 9- UI tests**



# LA LOGIQUE DES MODULES

## a) HomeSearchViewModel

```
// MARK: - Output

var titleLabel: ((String) -> Void)?
var placeHolderTextField: ((String) -> Void)?
var addButton: ((String) -> Void)?
var listTitleLabel: ((String) -> Void)?
var clearButton: ((String) -> Void)?
var ingredients: (([String]) -> Void)?
var ingredientTextField: ((String) -> Void)?
var searchButton: ((String) -> Void)?
var searchButtonIsHidden: ((Bool) -> Void)?

// MARK: - Input

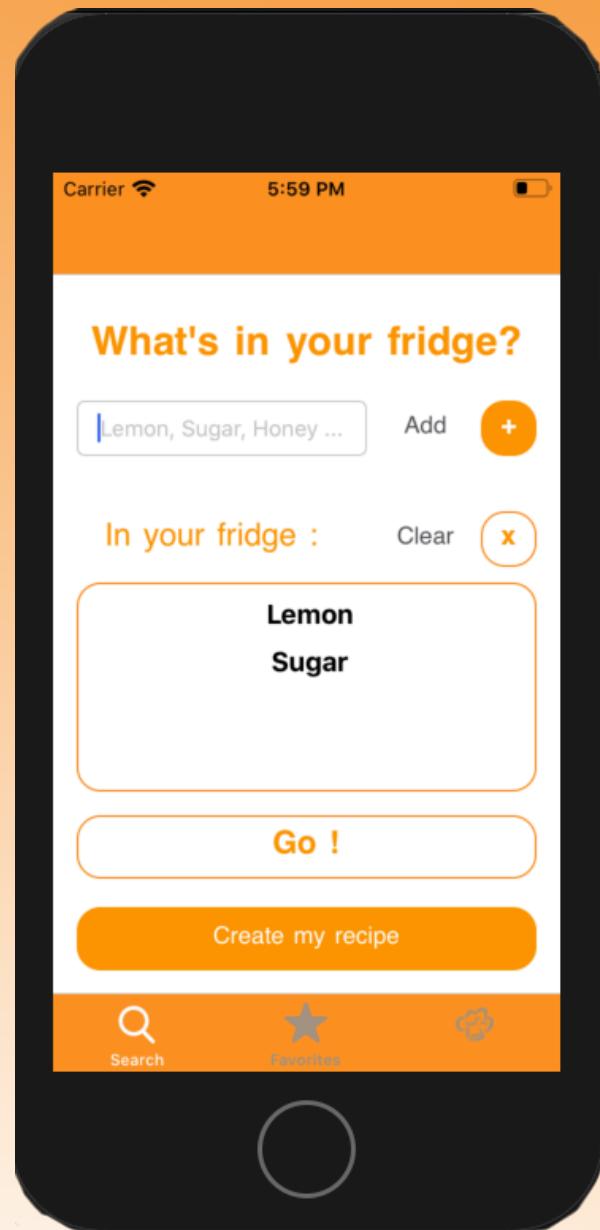
func viewDidLoad() {
}

func didPressAdd(ingredientSelected: String) {
}

func didPressClear() {
}

func didPressSearchForRecipes() {
}

func didPressCreateRecipe() {
```





# LA LOGIQUE DES MODULES

## b) RecipesViewModel

```
// MARK: - Output

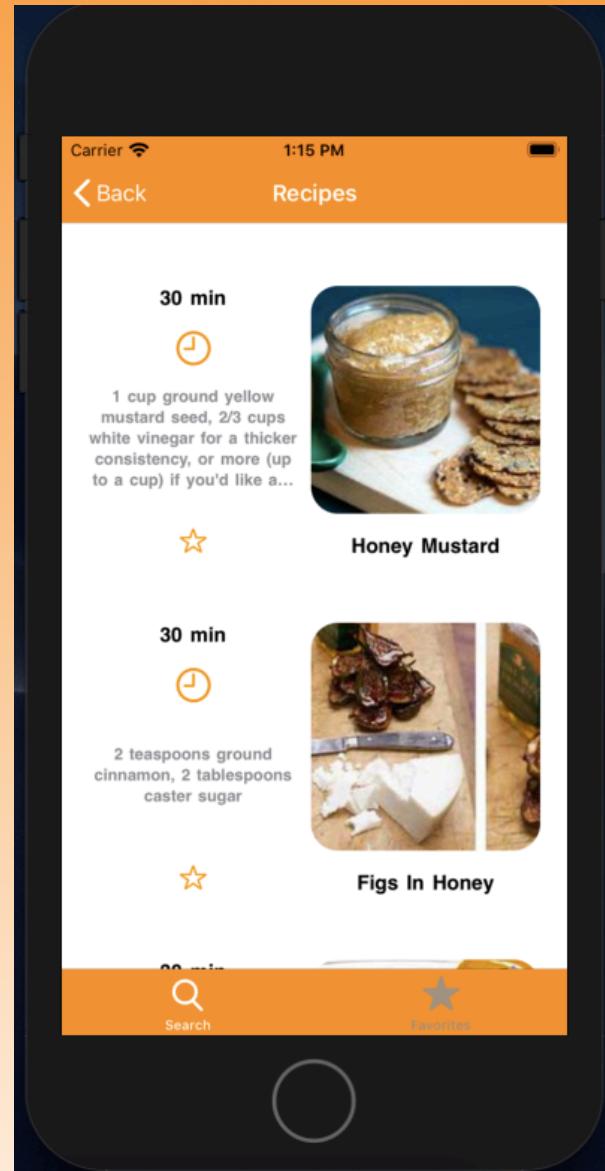
var recipesDisplayed: (([RecipeItem]) -> Void)?

var activityIndicatorIsLoading: ((Bool) -> Void)

// MARK: - Input

func viewWillAppear() {
}

func didSelectRecipe(recipe: RecipeItem) {
```





# LA LOGIQUE DES MODULES

## c) RecipeDetailViewModel

```
// MARK: - Output

var recipeDisplayed: ((RecipeItem) -> Void)?
var image: ((String) -> Void)?
var timeLabel: ((String) -> Void)?
var categoryLabel: ((String) -> Void)?
var yieldLabel: ((String) -> Void)?
var nameRecipeButton: ((String) -> Void)?
var favoriteState: ((Bool) -> Void)?
var favoriteImageState: ((String) -> Void)?

var navBarTitle: ((String) -> Void)?


// MARK: - Input

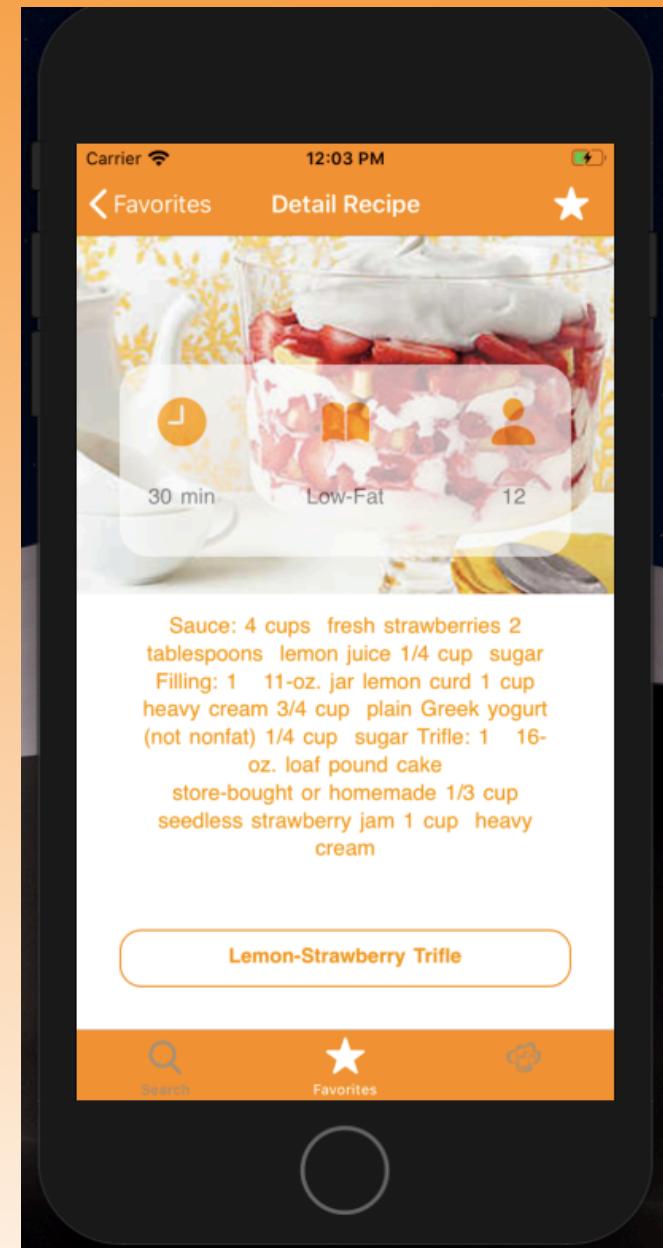
func viewDidLoad() {

}

func didPressSelectFavoriteRecipe() {

}

func didPressSafariButton() {
```





# LA LOGIQUE DES MODULES

## c) SavingCreationViewModel

```
// MARK: - Output

var creationDisplayed: (([CreationItem]) -> Void)?

var label: ((String) -> Void)?

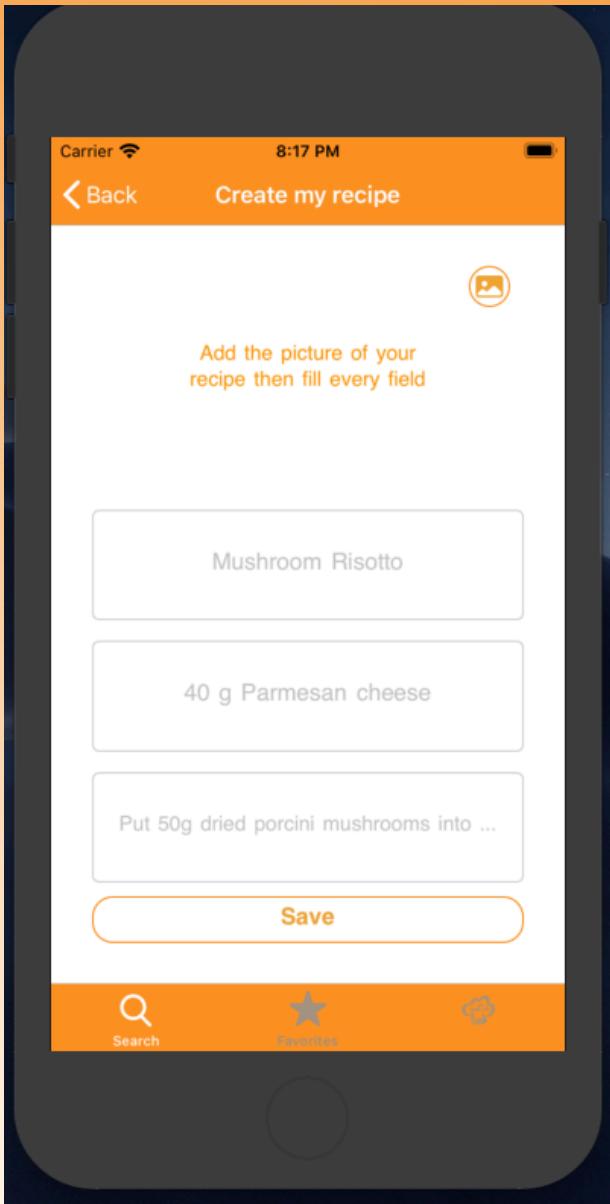
var timePlaceholder: ((String) -> Void)?
var categoryPlaceholder: ((String) -> Void)?
var yieldPlaceholder: ((String) -> Void)?
var titlePlaceholder: ((String) -> Void)?
var ingredientsPlaceholder: ((String) -> Void)?
var methodPlaceholder: ((String) -> Void)?
var navBarTitle: ((String) -> Void)?
var saveButton: ((String) -> Void)?


// MARK: - Input

func viewDidLoad() {
}

func didPressSaveButton(titleTextField: String,
                        ingredientTextField: String,
                        methodTextField: String,
                        timeTextField: String,
                        dietCategoryTextField: String,
                        yieldTextField: String) {
}

func didPressAddPhoto(imageAdded: Data?) {
}
// Alert
func restrictedCase() {
}
func deniedCase() {
}
```





# LA LOGIQUE DES MODULES

## c) CreationDetailViewModel

```
// MARK: - Output

var titleLabel: ((String) -> Void)?
var ingredientsAndMethodLabel: ((String) -> Void)?
var timeLabel: ((String) -> Void)?
var categoryLabel: ((String) -> Void)?
var yieldLabel: ((String) -> Void)?
var navBarTitle: ((String) -> Void)?
var imageData: ((Data?) -> Void)?
var creationButton: ((String) -> Void)?

// MARK: - Input

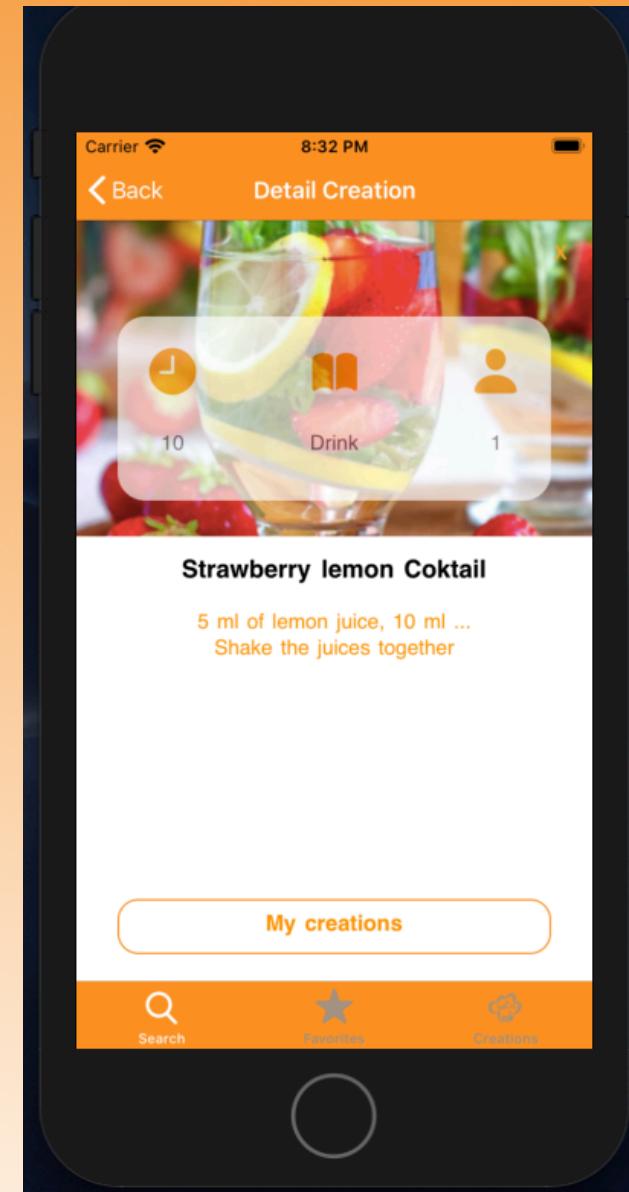
func viewDidLoad() {

}

func didPressMyCreationsButton() {

}

func didPressDeleteCreationButton() {
```





# LA LOGIQUE DES MODULES

## c) CreationsListViewModel

```
// MARK: - Output

var creationItem: (([CreationItem]) -> Void)?

// MARK: - Input

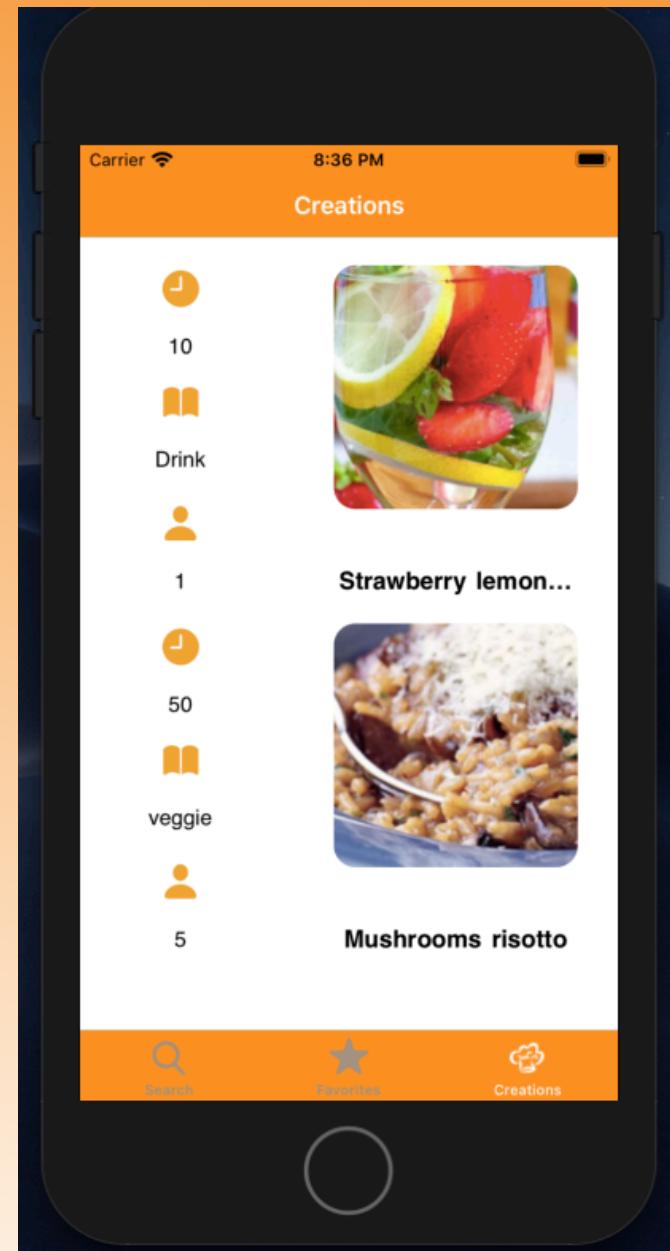
func viewDidLoad() {

}

func didSelectCreation(creation: CreationItem) {

}

func didPressDeleteCreation(name: String) {
}
```





# LE DIAGRAMME MVVM RÉACTIF NATIF DE L'APPLICATION



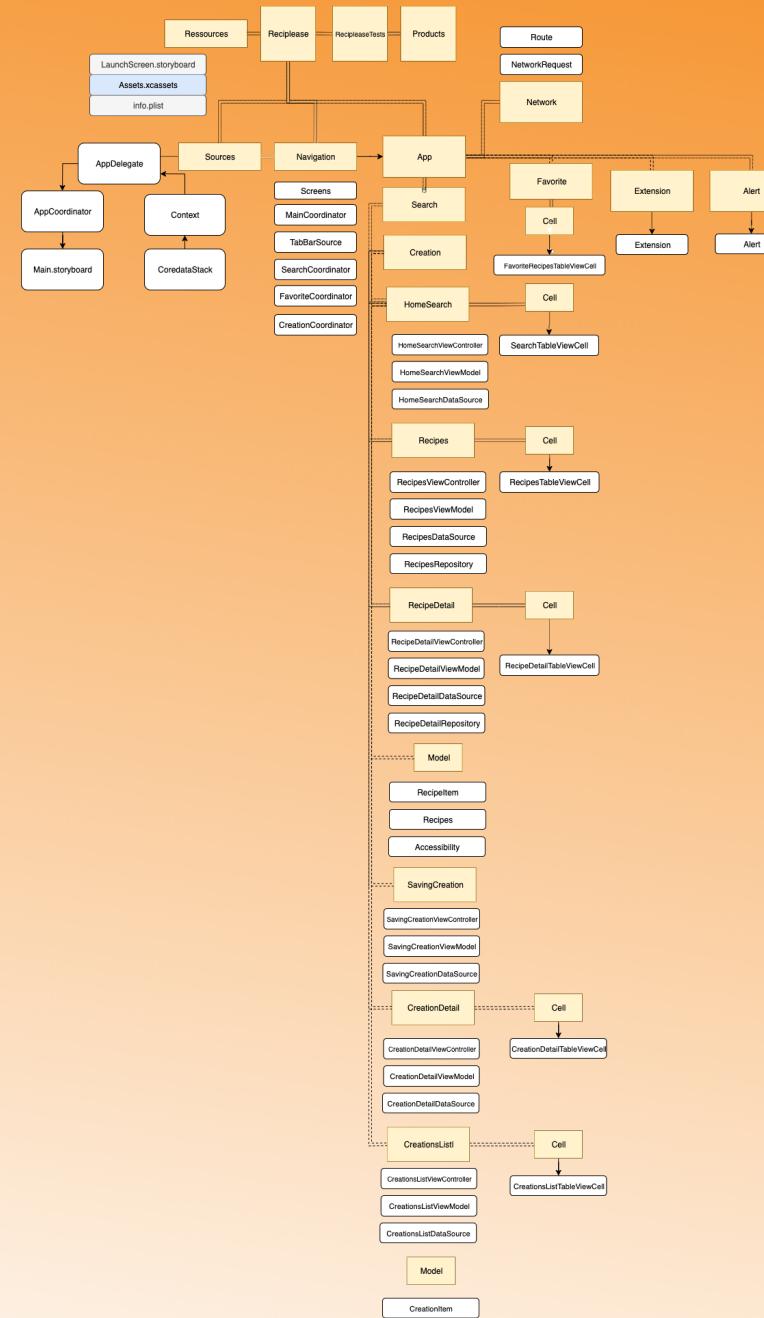
SOURCES



NETWORK



NAVIGATION





# PRÉSENTATION DU CODE





# COMPÉTENCES MAÎTRISÉES

**Perfectionnement de :**

- MVVM avec Coordinator
- Requêtes réseaux avec une couche network et repository
- Affichage de tabBar Item sans storyboard
- TableView et MVVM

**Nouveauté :**

- Persistence de données avec Core data
- Requêtes réseaux avec Alamofire
- Unit Tests avec utilisation de Mock
- UI Tests avec Protocol





# QUESTIONS ?

?

**Je vous remercie pour votre attention**