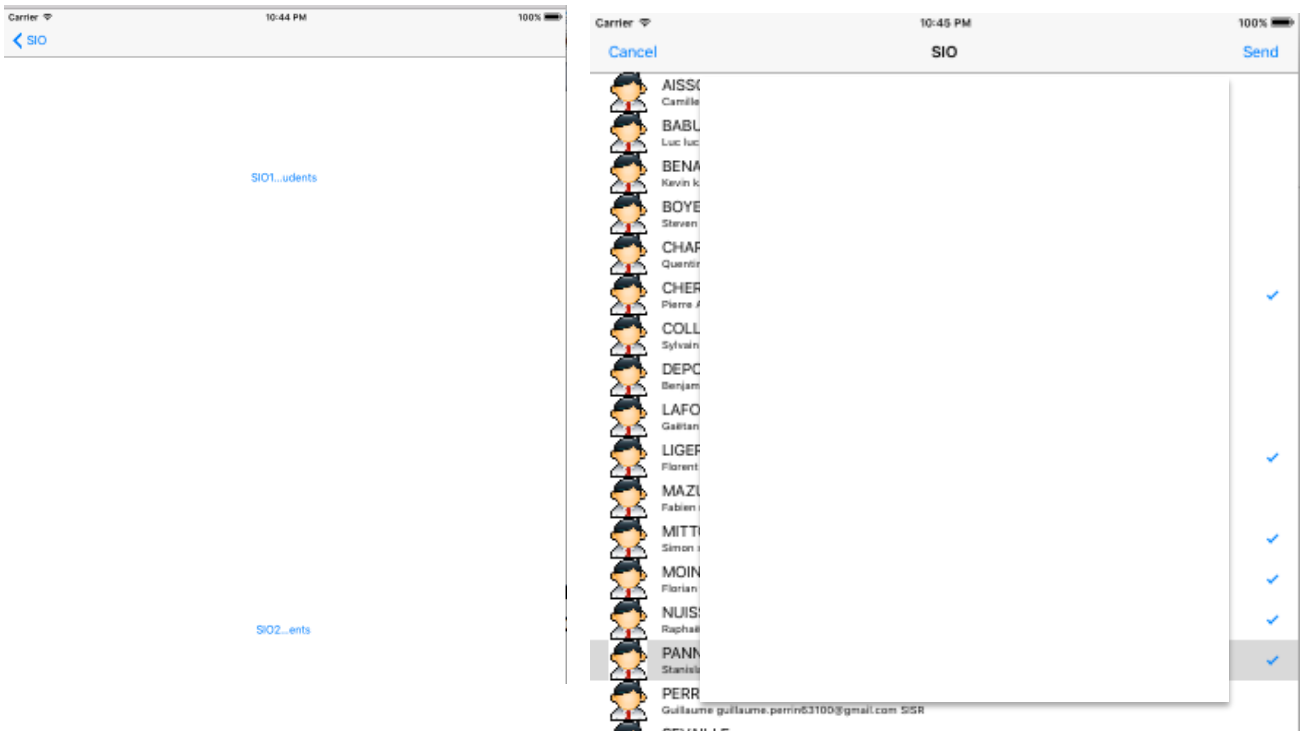


TP1 lesAbsents

Cette application permet de traiter les absences des étudiants, par l'envoi d'un mail avec la liste des absents du jour.

Il faut choisir la classe, et les absents dans la liste, chaque ligne sélectionnée est cochée (v)



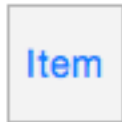
L'envoi de la liste des absents se fait par mail.



Partie 1 : le story board

Cette application **doit pouvoir supporter les différents formats d'écran.**

Elle permet d'utiliser tout un ensemble de composants utiles à une application de base :



Bar Button Item - Represents an item on a UIToolbar or UINavigationController object.

Button : bouton qui permet d'envoyer un message ou annuler



Navigation Item - Represents a state of the navigation bar, including a title.

Item : permet d'afficher un bandeau de navigation au dessus de l'écran

Button

Button - Intercepts touch events and sends an action message to a target object when it's tapped.

Button : bouton qui permet sélectionner la liste de classe

Le même story board peut être utilisé pour tous les devices qui utilisent iOS 9, sans écrire une seule ligne de code.

C'est l'intérêt de Universal Story Board.

Vous allez devoir commencer par vous familiariser avec le story board pour générer les vues portraits et paysages (landscape) des appareils : iphone, ipad.

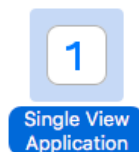
Tous les devices doivent pouvoir fonctionner en mode portrait et paysage.

iPhone 3.5-inch
iPhone 4-inch
iPhone 4.7-inch
iPhone 5.5-inch
iPad

iPhone 4S
iPhone 5, 5S
iPhone 6
iPhone 6 Plus
iPad 2, iPad Air (attention en Air = Retina x2) -> affichage point 7 du document

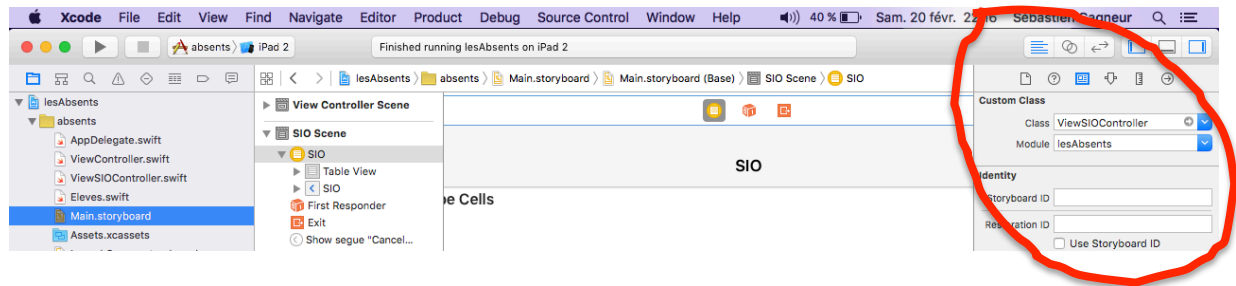
1. Setting the project

Pour créer le projet, la procédure habituelle est requise. Prendre un projet « single view application »

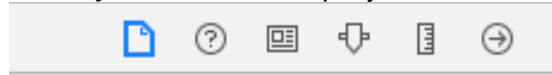


2. Setting the story board

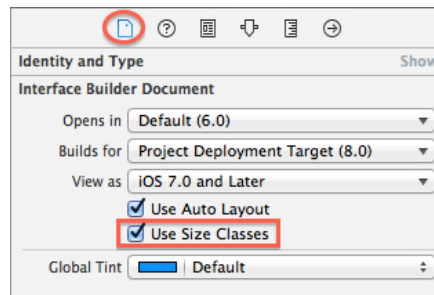
L'inspecteur est la fenêtre située à droite de Xcode :



Choisir le fichier du Main.storyboard, dans le projet, et aller dans le File Inspector :

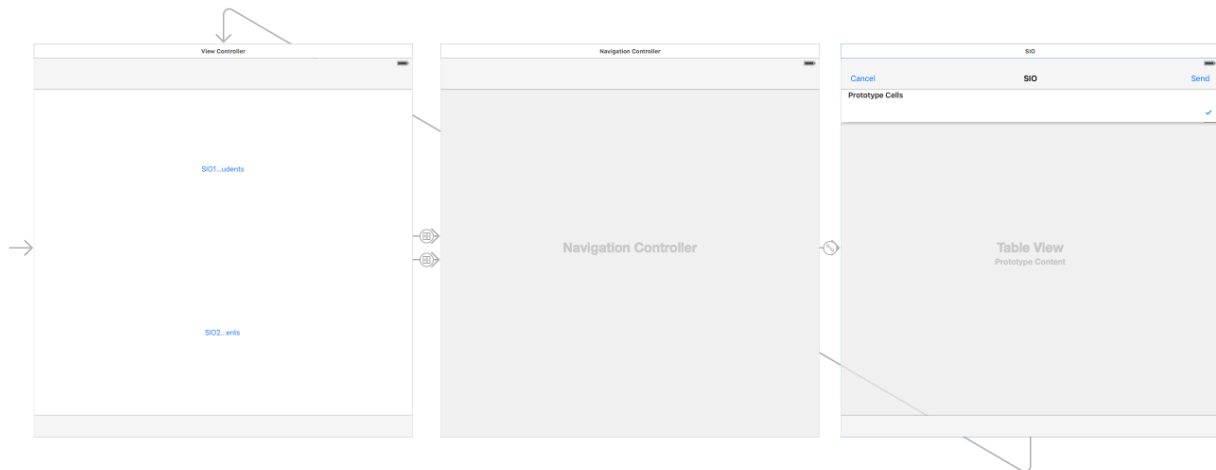


Il est nécessaire d'avoir activé, l'option suivante, qui en principe est activée par défaut.

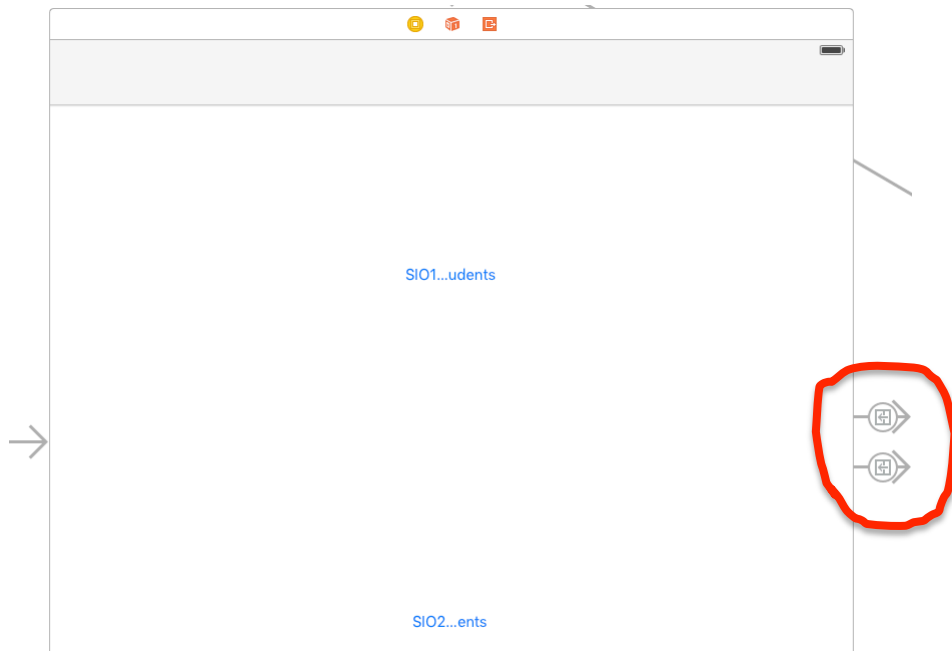


3. Création du story board

Créer un story board qui sera toujours le même.



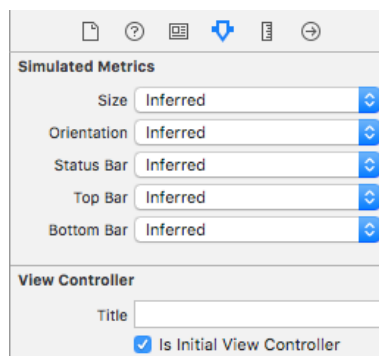
Le story board utilise dans une première fenêtre un ViewController, qui est la fenêtre d'accueil. La flèche à gauche constitue l'entry point, qui désigne la première fenêtre à afficher.



note : entourés en rouge, les « segues » !

Dans ce ViewController, deux boutons sont posés, chaque bouton est lié par un « segue » au UINavigationController

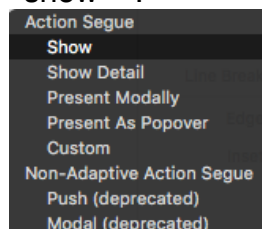
L'inspecteur doit permettre de vérifier si la fenêtre en question est bien : « is initial View Controller »



La création des « segues » se fait facilement.

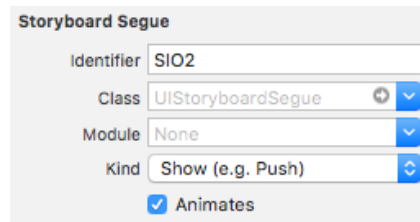
Il faut appuyer sur la touche « ctrl », pendant ce temps cliquer avec la souris sur le bouton SIO1 et glisser sur le navigationController. Vous pouvez lâcher la souris et le bouton « ctrl » à ce moment là.

Dans le menu contextuel choisir « show » :

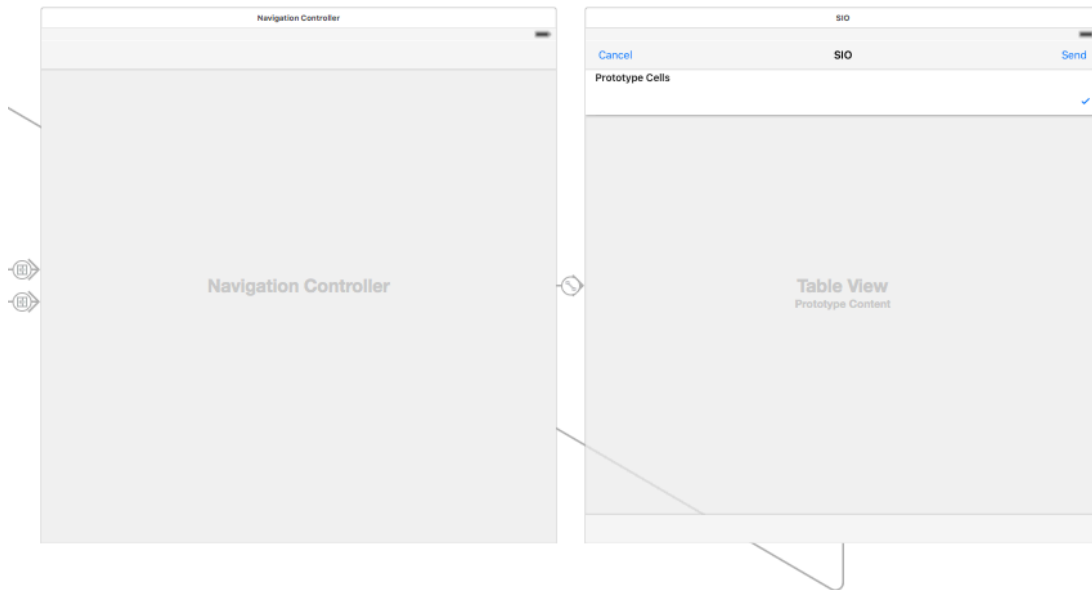


Vous aurez un autre « segue » à faire avec le bouton « cancel »

Le nommage des « segues » pour les repérer dans le code se fait dans l'inspecteur :

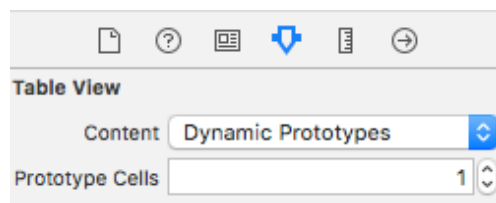


Nous avons ensuite un UINavigationController qui permet d'héberger une tableView.

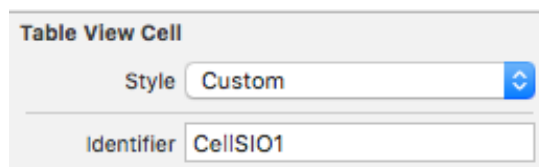


Le navigationItem est posé en premier, puis ensuite on pose les barButtonItem. Les barButtonItem sont renommés suivant l'utilisation que l'on souhaite en faire.

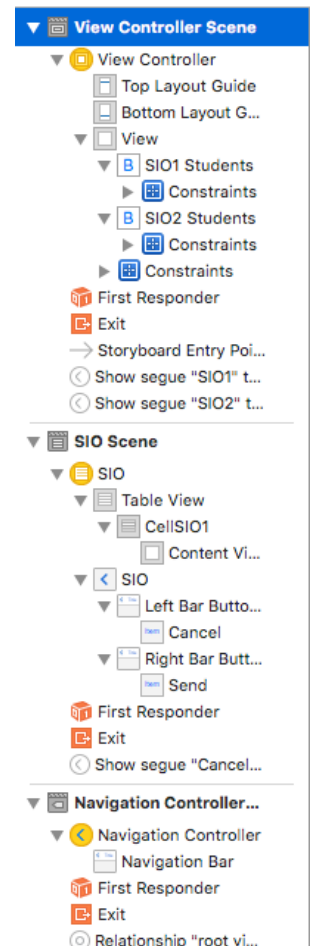
La tableView adopte les réglages suivants :








Les cellules ont également un identifiant :

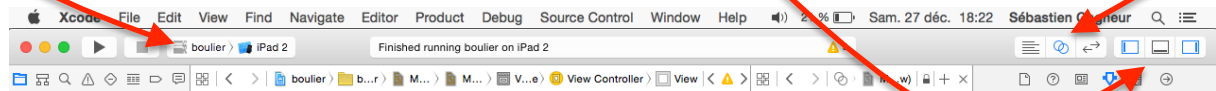


La hiérarchie du storyboard est la suivante :



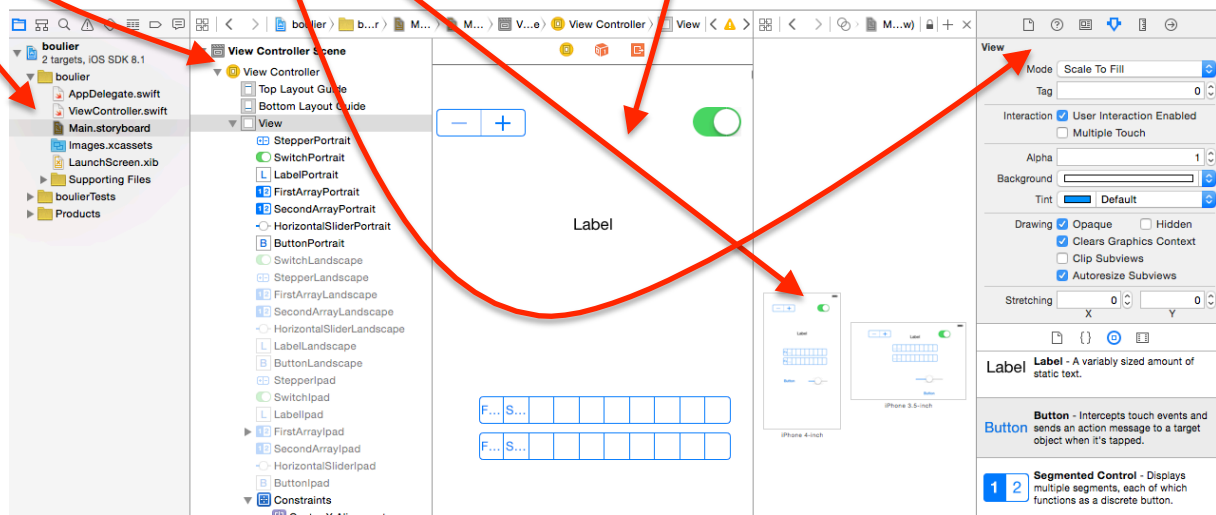
4. Xcode en mode story board

1. Choix du device  pour le lancement de l'application
2. Choix de la timeline  (passage à preview -> vue réelle au niveau story board)
3. Panneau gauche  (arborescence), panneau bas  (debug), panneau droit  (inspecteur)



L'écran de Xcode se compose de 5 zones au maximum.

1. L'arborescence du projet
2. L'arborescence du ViewController
3. La frame (l'écran que vous réalisez)
4. La vue réelle
5. L'inspecteur



Les boutons en bas sont assez importants :

1. Réglages Universal story board
2. + ajout de device en vue réelle

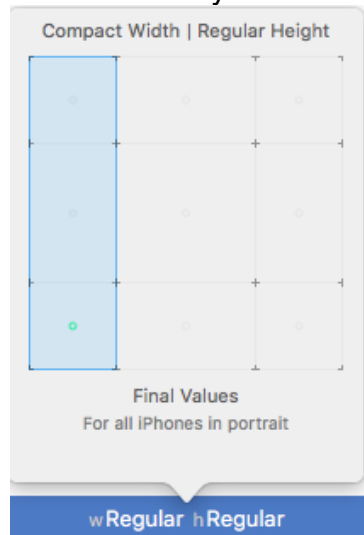


5. Universal story Board

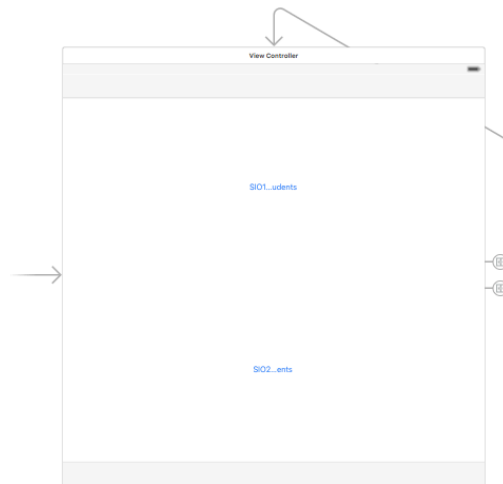
Le système une fois compris est assez simple, voir même intuitif, et déconcertant.

Pour sélectionner le mode portrait des iPhones, vous ajustez à la souris le nombre de carrés qui correspond. Cela ajuste la frame du viewController à la taille du device concerné. Au niveau de la vue réelle vous voyez ce qui se passe dans le mode sélectionné.

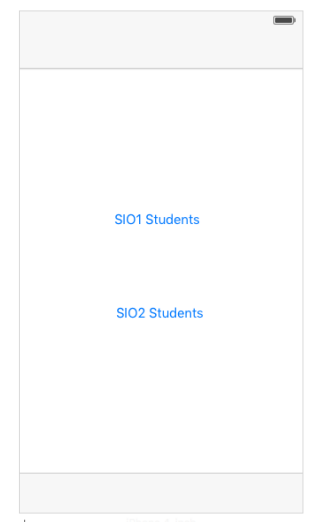
Universal Story Board



Frame



Vue réelle

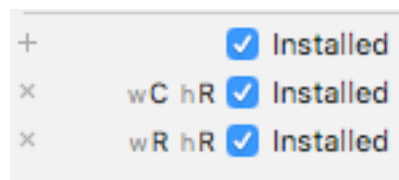


Le mode Regular/Regular correspond aux vues iPad dans tous les sens

Le mode Compact/Height correspond à la vue iPhone en mode portrait

Vous trouverez facilement la taille d'écran à utiliser pour votre travail.

Chaque composant graphiques doit valider les tailles d'écran ad hoc :

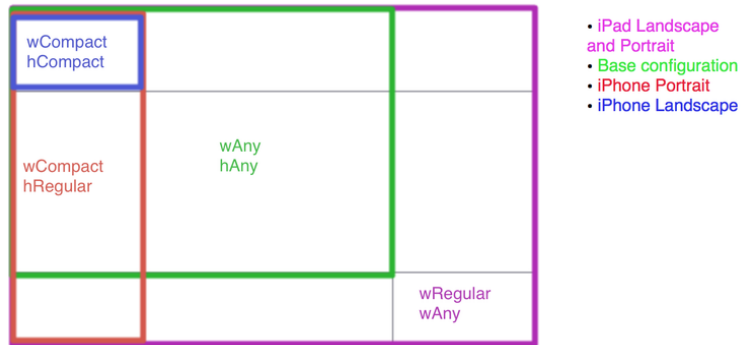


A partir de là, tous les composants posés sur la frame (viewController) sont visibles en réel dans le mode qui correspond. Si vous basculez dans le mode non sélectionné en vue réelle, vous ne voyez plus vos composants, il faut alors les installer comme sur l'image précédente.

Ce qui veut dire que le story board mémorise pour chaque mode, de chaque device, les composants utilisés et leur disposition.

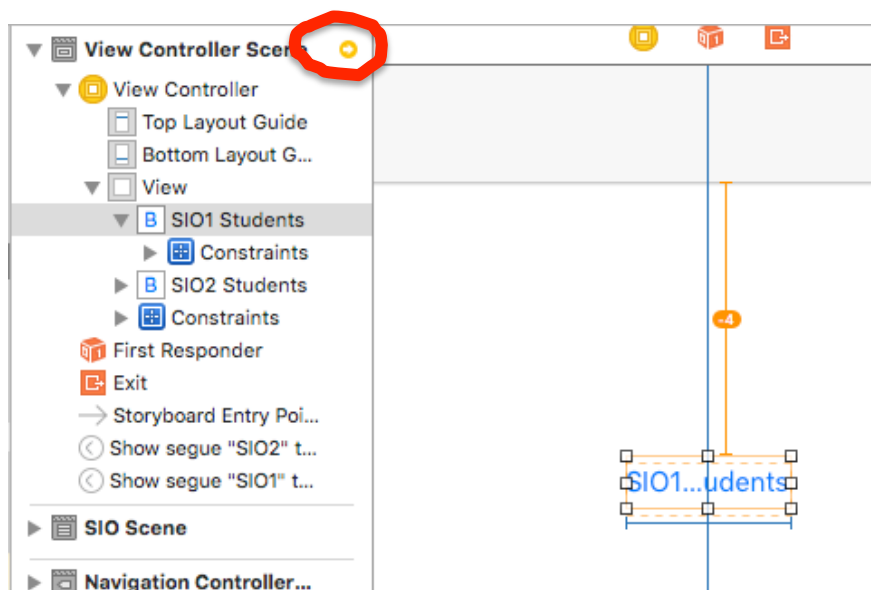
Il ne vous reste plus qu'à faire le travail, et prendre le temps d'ajuster les composants suivants les modes d'affichage et les devices.

Les différentes tailles :



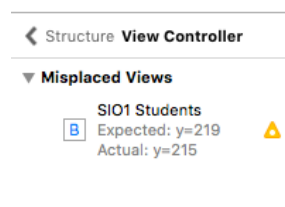
6. Disposition des composants

Pour la disposition des composants il existe un assistant assez utile.

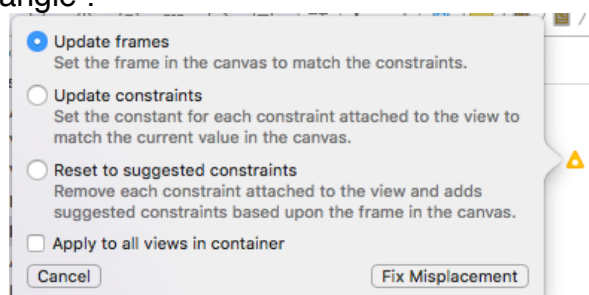


Pour chaque déplacement ou ajustement, le petit symbole (->) apparaît, il faut cliquer dessus.

Vous passez en mode assistance :



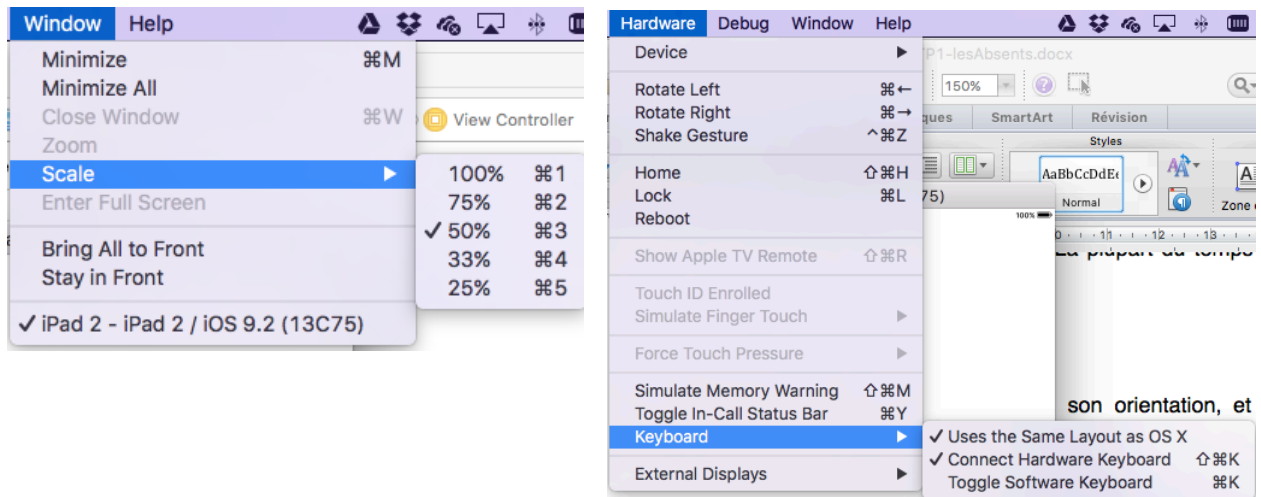
Vous cliquer sur le triangle :



Et vous choisissez la meilleure opération à réaliser. La plupart du temps « reset to suggested constraint » permet de régler le problème.
Vous finalisez avec « fix misplacement ».

7. iOS simulator

Le simulateur permet de régler la taille de l'écran, son orientation, et le clavier connecté :



Partie 2 : le codage de l'application

Cette version de l'application n'utilise ni base de données, ni API pour récupérer les listes de classes. Celles-ci sont codées en dur dans l'application.

2.1. Le ViewController

```
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
        if (segue.identifier == "SI01")
        {
            let navController = segue.destinationViewController as! UINavigationController
            let detailController = navController.topViewController as! ViewSIOController
            detailController.whichClass = "SI01"
        }
        if (segue.identifier == "SI02")
        {
            //let m : Motor = Motor(_ihm : self)
            let navController = segue.destinationViewController as! UINavigationController
            let detailController = navController.topViewController as! ViewSIOController
            detailController.whichClass = "SI02"
        }
    }
}
```

Le viewController s'occupe simplement de détecter quel « segue » est activé à partir d'un clic sur le bouton lié au « segue » en question.

Un « segue » prononcé « segway » est un lien qui permet de passer d'une fenêtre à une autre. Les « segues » sont créés sur le storyboard, ils sont nommés pour pouvoir les repérer.

L'appel d'un « segue » permet d'envoyer à la fenêtre suivante, l'information du type de classe à afficher. Ici, SIO1 ou SIO2.

2.2. La classe métier Eleve

La classe Eleve permet de décrire les instances associées. Cette classe possède des attributs et des méthodes. Les tableaux manipulés plus loin, « lesAbsents », et « lesEleves » sont composés d'instances « Eleve ».

```
import Foundation

class Eleve {

    var lastname : String = ""
    var firstname : String = ""
    var mail : String = ""
    var phone : String = ""
    var option : String = ""
    var selected : Bool = Bool()

    init() {
        self.lastname = ""
        self.firstname = ""
        self.mail = ""
        self.phone = ""
        self.option = ""
        self.selected = false
    }

    init(last : String, first : String, mail : String, phone : String, option : String, sel : Bool) {
        self.lastname = last
        self.firstname = first
        self.mail = mail
        self.phone = phone
        self.option = option
        self.selected = sel
    }

    func getLastName() -> String
    {
        return self.lastname
    }

    func getFirstName() -> String
    {
        return self.firstname
    }

    func getMail() -> String
    {
        return self.mail
    }

    func getPhone() -> String
    {
        return self.phone
    }
}
```

```

func getOption() -> String
{
    return self.option
}

func getSelected() -> Bool
{
    return self.selected
}

func setSelected(val : Bool)
{
    self.selected = val
}

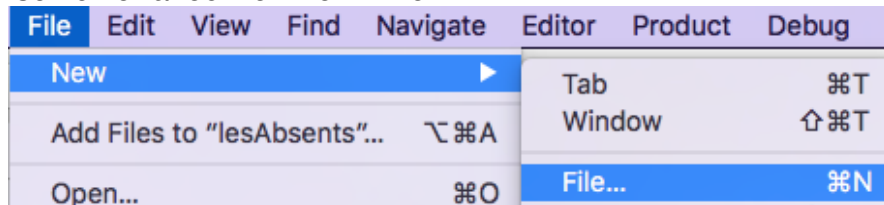
}

```

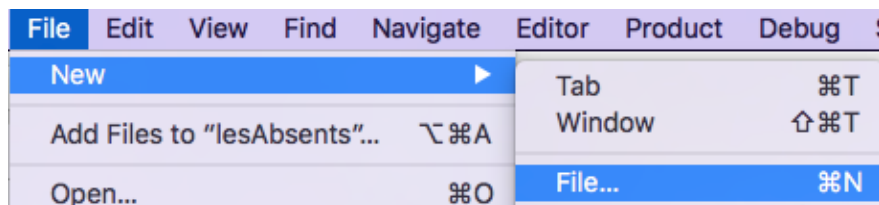
2.3 Le ViewSIOController

C'est le code du controller de la seconde fenêtre qui s'affiche lorsque vous arrivez sur la tableView.

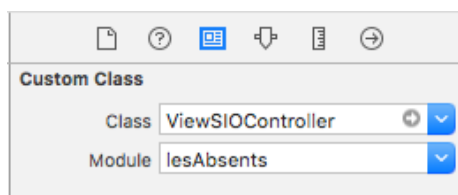
Il faut créer ce fichier avec file->new->file



Il faut aussi associer ce controller à la tableView du navigationController :



L'inspecteur permet là encore de régler la chose :



Au chargement de la vue :

```
//
// ViewSI01Controller.swift
// absents
//
// Created by Sébastien Gagneur on 09/02/2016.
// Copyright © 2016 Sébastien Gagneur. All rights reserved.
//

import UIKit
import MessageUI

class ViewSI0Controller : UITableViewController, MFMailComposeViewControllerDelegate
{
    var lesEleves : [Eleve] = [Eleve]()
    var lesAbsents : [Eleve] = [Eleve]()
    var whichClass : String = ""

    override func viewDidLoad() {
        super.viewDidLoad()
        if whichClass == "SI01"
        {
            lesEleves.append(Eleve(last: "ANTUNES", first: "Alexandre", mail: "antunes719@gmail.com",
            // ...
            })
        }
        if whichClass == "SI02"
        {
            lesEleves.append(Eleve(last: "AISSOU", first: "Camille", mail: "aiissoucamille@gmail.com",
            // ...
            })
        }
    }
}
```

Deux tableaux sont utilisés, lesEleves et lesAbsents. lesAbsents se complétera au fur et à mesure de la sélection des absents dans lesEleves.

Comme on utilise une tableView associé à un protocole (UITableViewController), il faut implémenter des méthodes pour gérer son fonctionnement :

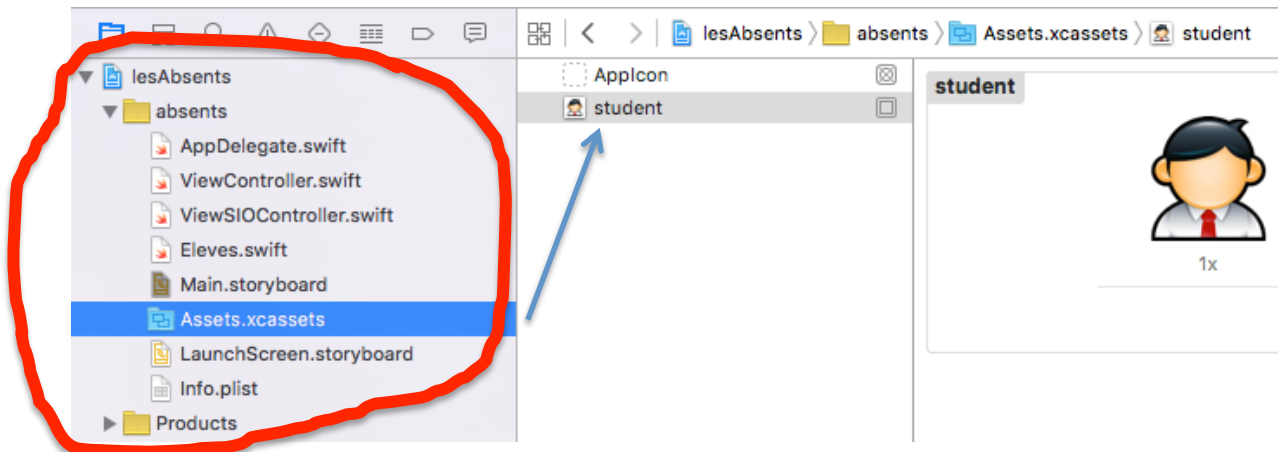
Il faut pouvoir compter combien d'élément il faut afficher dans la tableView à partir du tableau lesEleves

```
//COUNT
override func tableView(tableView: UITableView, numberOfRowsInSectionSection section: Int) -> Int {
    return lesEleves.count
}

//INSERT
override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) ->
UITableViewCell {
    print("scroll")
    let cell: UITableViewCell = UITableViewCell(style: UITableViewCellStyle.Subtitle, reuseIdentifier:
    "cellSI01")
    if lesEleves[indexPath.row].getSelected() == true
    {
        cell.accessoryType = UITableViewCellAccessoryType.Checkmark
    }
    let image : UIImage = UIImage(named: "student")!

    cell.imageView!.image = image
    cell.textLabel?.text = lesEleves[indexPath.row].getLastName()
    cell.detailTextLabel?.text = lesEleves[indexPath.row].getFirstName() + " " + lesEleves[indexPath.row]
    .getMail() + " " + lesEleves[indexPath.row].getOption()
    return cell
}
```

L'image utilisée est glissée dans Assets.Xcassets, vous pouvez aussi apercevoir à gauche (dans l'image de la page suivante) l'arborescence complète du projet. Il vous faut tous les fichiers présents, pour avoir une application fonctionnelle.



Il faut aussi pouvoir insérer les valeurs dans la liste.

Un clic sur une ligne de la tableView doit permettre de sélectionner l'élève en question dans les absents.

```
//SELECT
// gestion du Segue sur le disclosure qui ne marche pas par défaut
override func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
    let indexPath = tableView.indexPathForSelectedRow!
    let cell = tableView.cellForRowAtIndexPath(indexPath)
    if cell?.accessoryType == UITableViewCellAccessoryType.None
    {
        cell?.accessoryType = .Checkmark
        print("\(lesEleves[indexPath.row].getFirstName())")
        lesEleves[indexPath.row].setSelected(true)
        lesAbsents.append(lesEleves[indexPath.row])
        print("\(lesAbsents.count)")
    }
    else
    {
        cell?.accessoryType = .None
        print("\(lesEleves[indexPath.row].getFirstName())")
        lesEleves[indexPath.row].setSelected(false)
        var i : Int = 0
        var trouve : Int = 0
        for a in lesAbsents
        {
            if a.getLastName() == cell?.textLabel?.text
            {
                trouve = i
            }
            i++
        }
        lesAbsents.removeAtIndex(trouve)
        print("\(lesAbsents.count)")
    }
}
//verif de la collection
for a in lesAbsents
{
    print("coll = \(a.getLastName()) \(a.getFirstName()) \(a.getMail())")
}
}
```

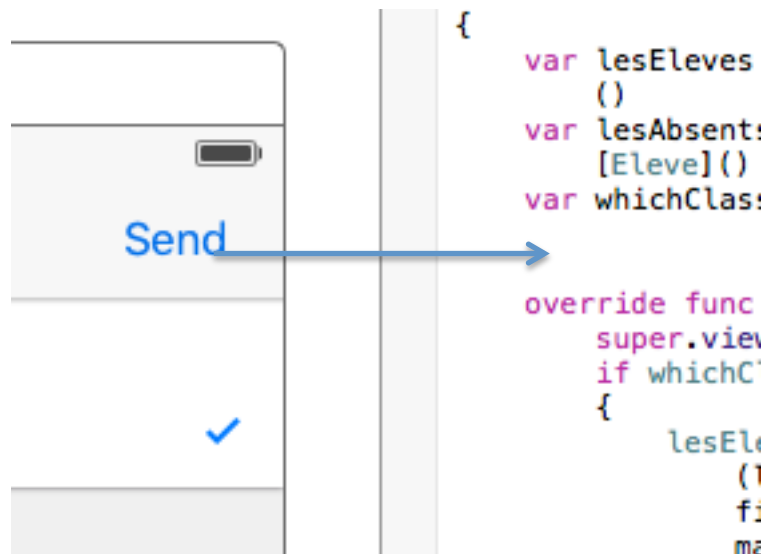
Bien évidemment un nouveau clic sur un élève déjà sélectionné, le supprime des absents.

Le clic sur le bouton envoyé (send) est lié à la méthode suivante :

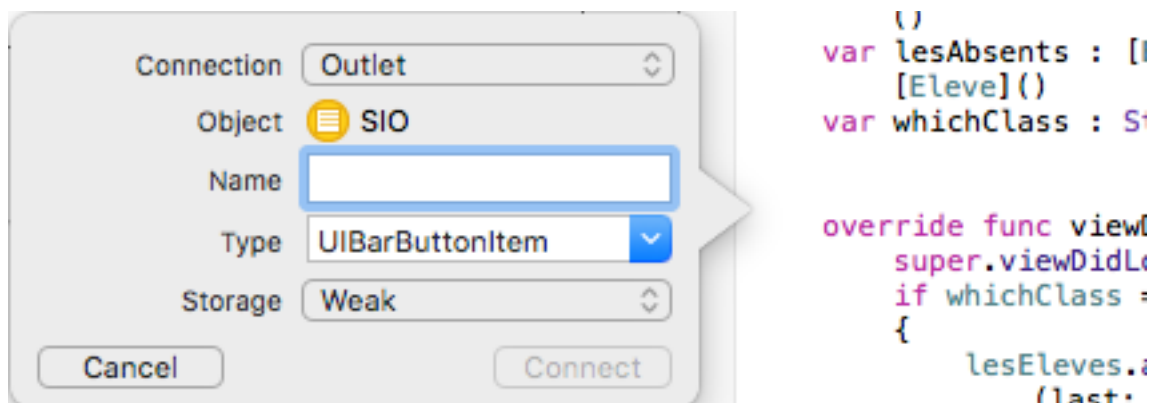
```
@IBAction func sendEmailButtonTapped(sender: AnyObject) {
    let mailComposeViewController = configuredMailComposeViewController()
    if MFMailComposeViewController.canSendMail() {
        self.presentViewController(mailComposeViewController, animated: true, completion: nil)
    } else {
        self.showSendMailErrorAlert()
    }
}
```

Pour lier le « clic » du bouton à une méthode :

Appuyer sur « ctrl », cliquer sur Send, glisser jusqu'au code du ViewController, relâchez.



Choisissez Action, afin de créer une méthode événementielle :



N'oubliez pas de donner un nom à votre méthode

Une méthode correctement liée, mentionne une petite croix (+) dans un rond noir.



Vient ensuite la composition du mail.

```
func configuredMailComposeViewController() -> MFMailComposeViewController {
    let mailComposerVC = MFMailComposeViewController()
    mailComposerVC.mailComposeDelegate = self // Extremely important to set the --mailComposeDelegate--
    property, NOT the --delegate-- property
}
```

Je choisis à qui j'envois le mail :

```
mailComposerVC.setToRecipients(["s.gagneur@univ-lyon1.fr", "s.gagneur@univ-lyon1.fr"],  
                                ["s.gagneur@univ-lyon1.fr", "s.gagneur@univ-lyon1.fr"])
```

Le compose l'entête du mail :

```
subject = "Liste de(s) " + String(lesAbsents.count) + " absent(s) du " + jour + " en SI01"  
  
mailComposerVC.setSubject(subject)
```

Je rédige le corps du message :

```
var text : String = ""  
text = "Bonjour Geneviève, \n\n Voici la liste de(s) " + String(lesAbsents.count) + " absent(s) de  
      SI01 du " + jour + " dans mon cours : \n\n"  
  
mailComposerVC.setMessageBody(text, isHTML: false)  
  
return mailComposerVC  
}
```

Le reste est une formalité :

```
func showSendMailErrorAlert() {  
    let sendMailErrorAlert = UIAlertController(title: "Could Not Send Email", message: "Your device could  
    not send e-mail. Please check e-mail configuration and try again.", preferredStyle:  
    UIAlertControllerStyle.Alert)  
  
    showViewController(sendMailErrorAlert, sender: self)  
}  
  
// MARK: MFMailComposeViewControllerDelegate  
  
func mailComposeController(controller: MFMailComposeViewController, didFinishWithResult result:  
    MFMailComposeResult, error: NSError?) {  
    controller.dismissViewControllerAnimated(true, completion: nil)  
}  
}
```

L'application est terminée :

