Localizing an app in iOS has a standard process. We will walk through steps involved in it. For that we will create a sample project and follow the process to fully localize it.

1. Create a project

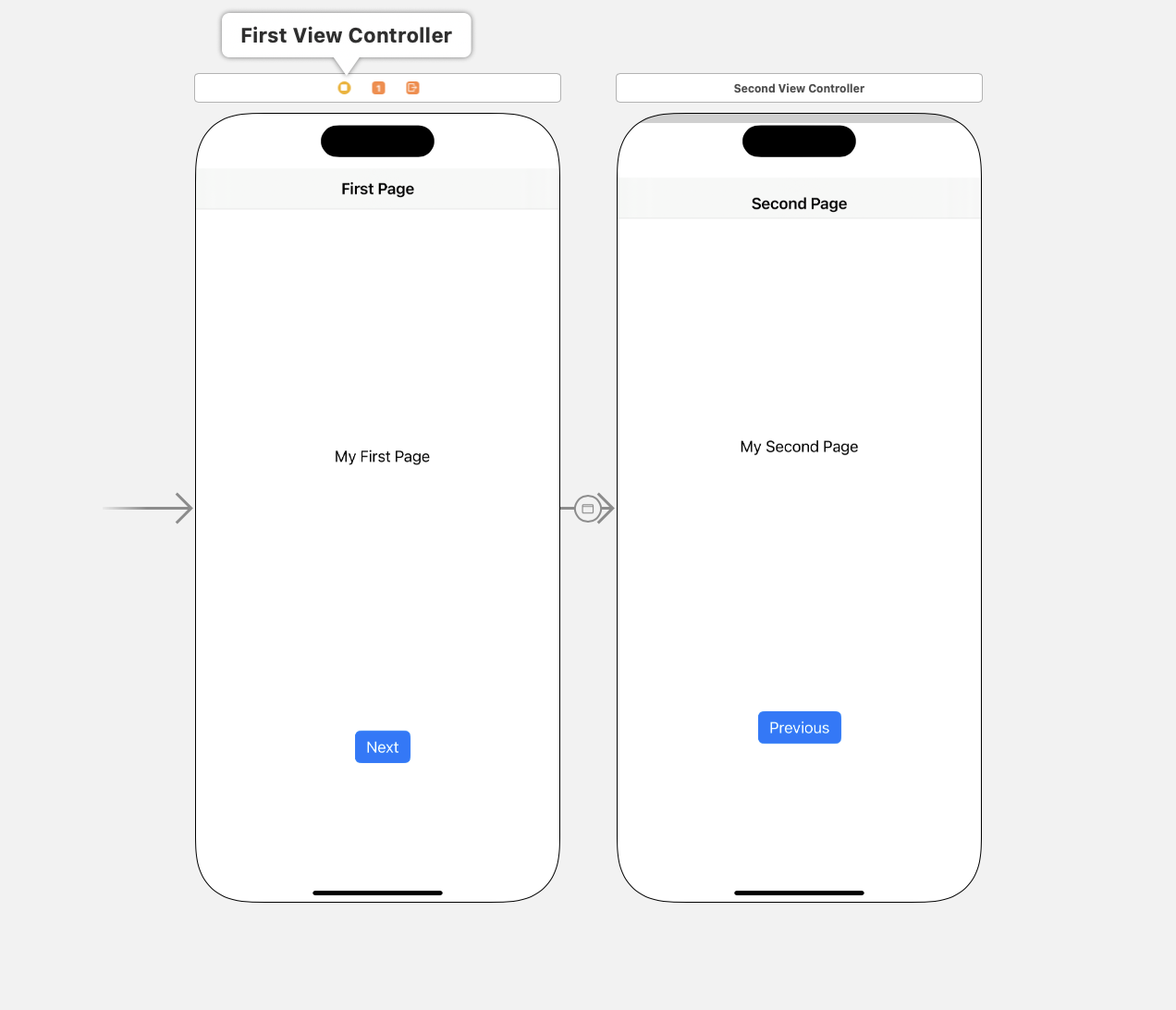
Creat a new iOS project in xcode. You can use either SwiftUI or UIKit interface. In this example we have used UIKit interface. You can checkout the project from [here](https://github.com/reverieinc/localization_demo_iOS).

In this project, we have created 2 screens.

First screen has a navigation bar with title ‘First Page’, one UILabel to display text content and a Button with label ‘Next’

The second screen has a navigation bar with title ‘Second Page’, one UILable to display text content and a Button with label ‘previous’.

When user presses the ‘Next’ button on the first screen app goes to the second screen. When user presses the ‘previous’ button on the second screen app dismisses the second screen and come back to the first screen.

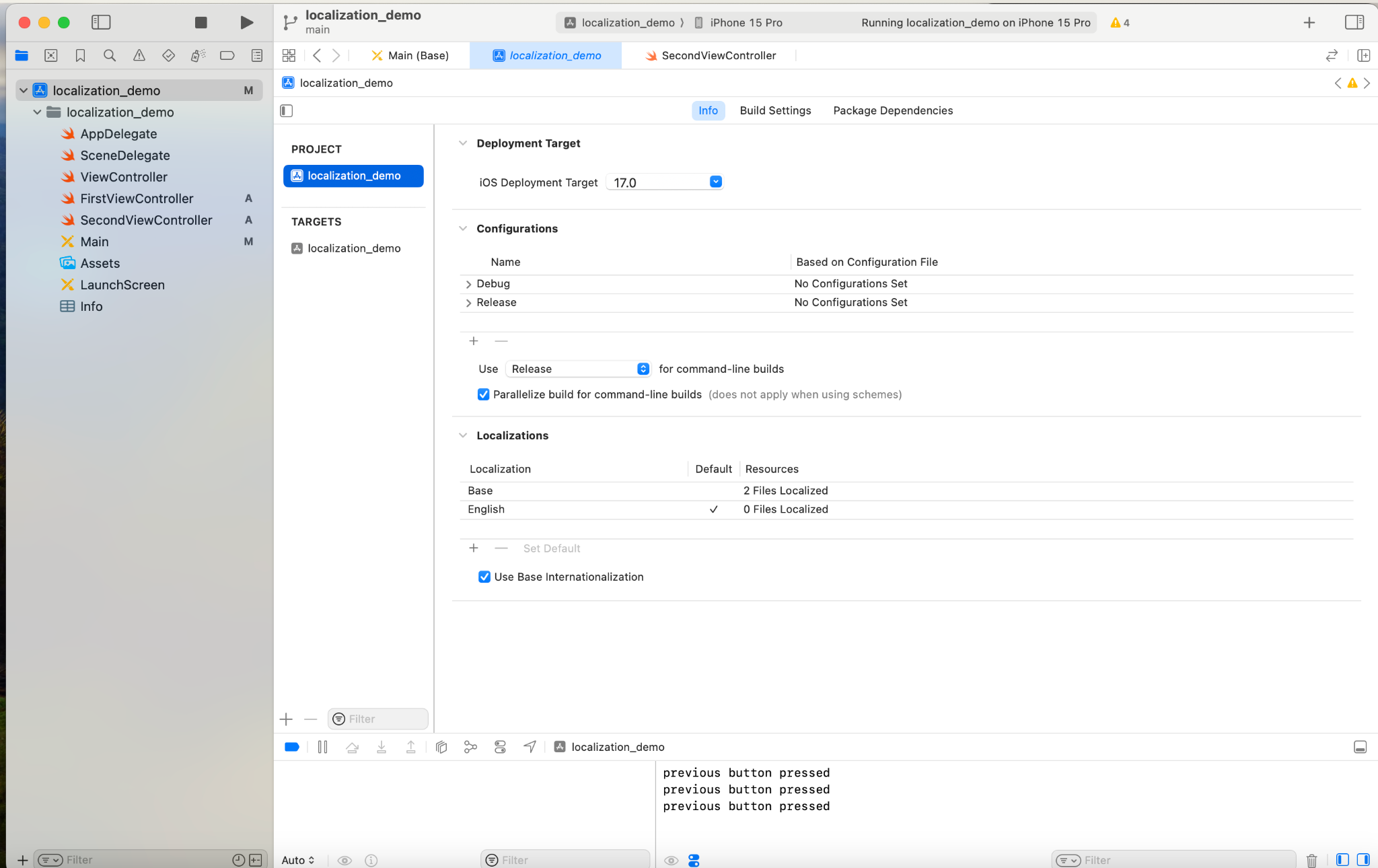


1. Add supported languages to the app

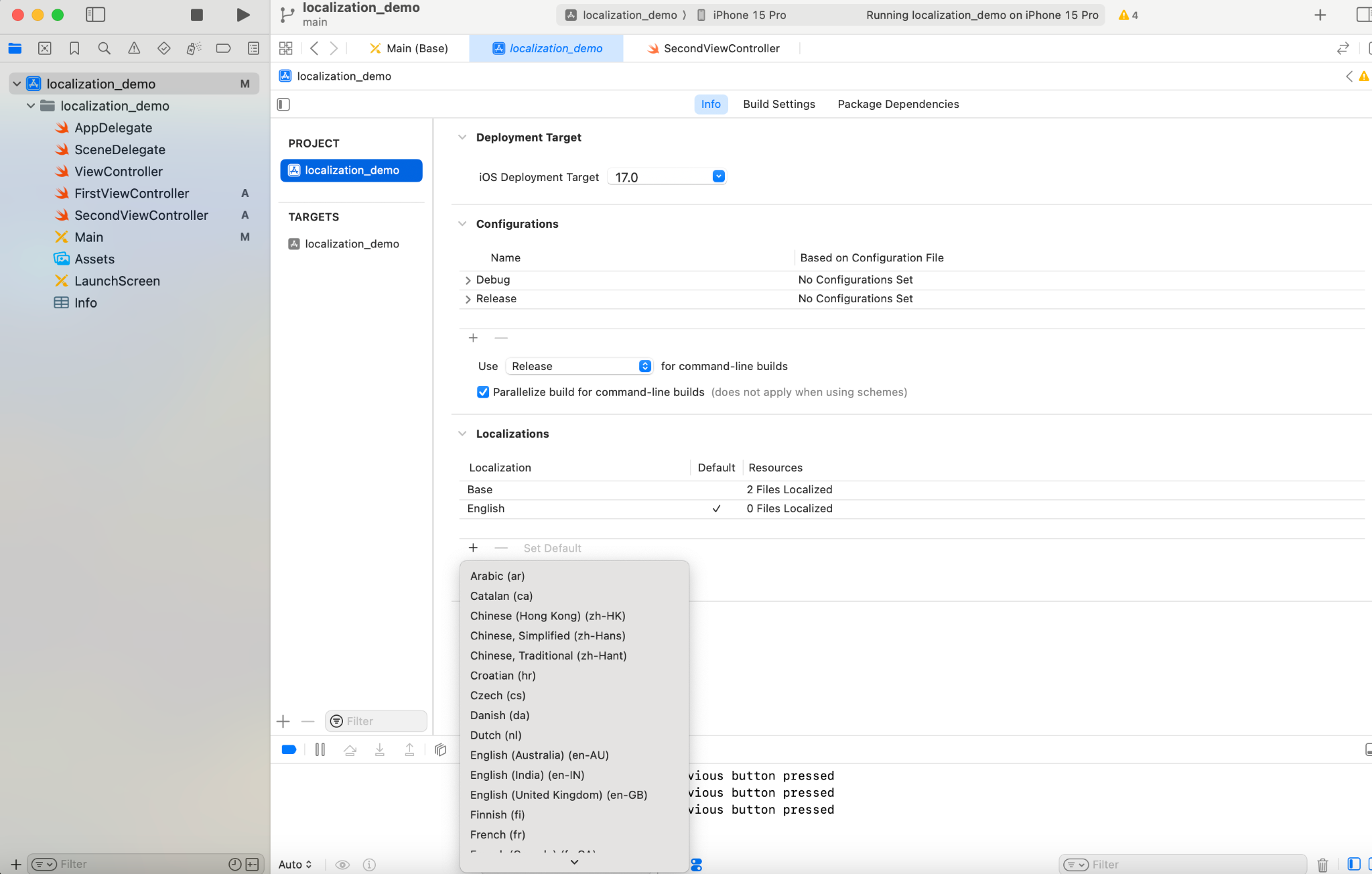
Since our app is ready, we will localize the app in Indian languages.

Lets go to Project navigator -> Select the project name

Under Localization, make sure ‘Use base internationalization’ is checked.



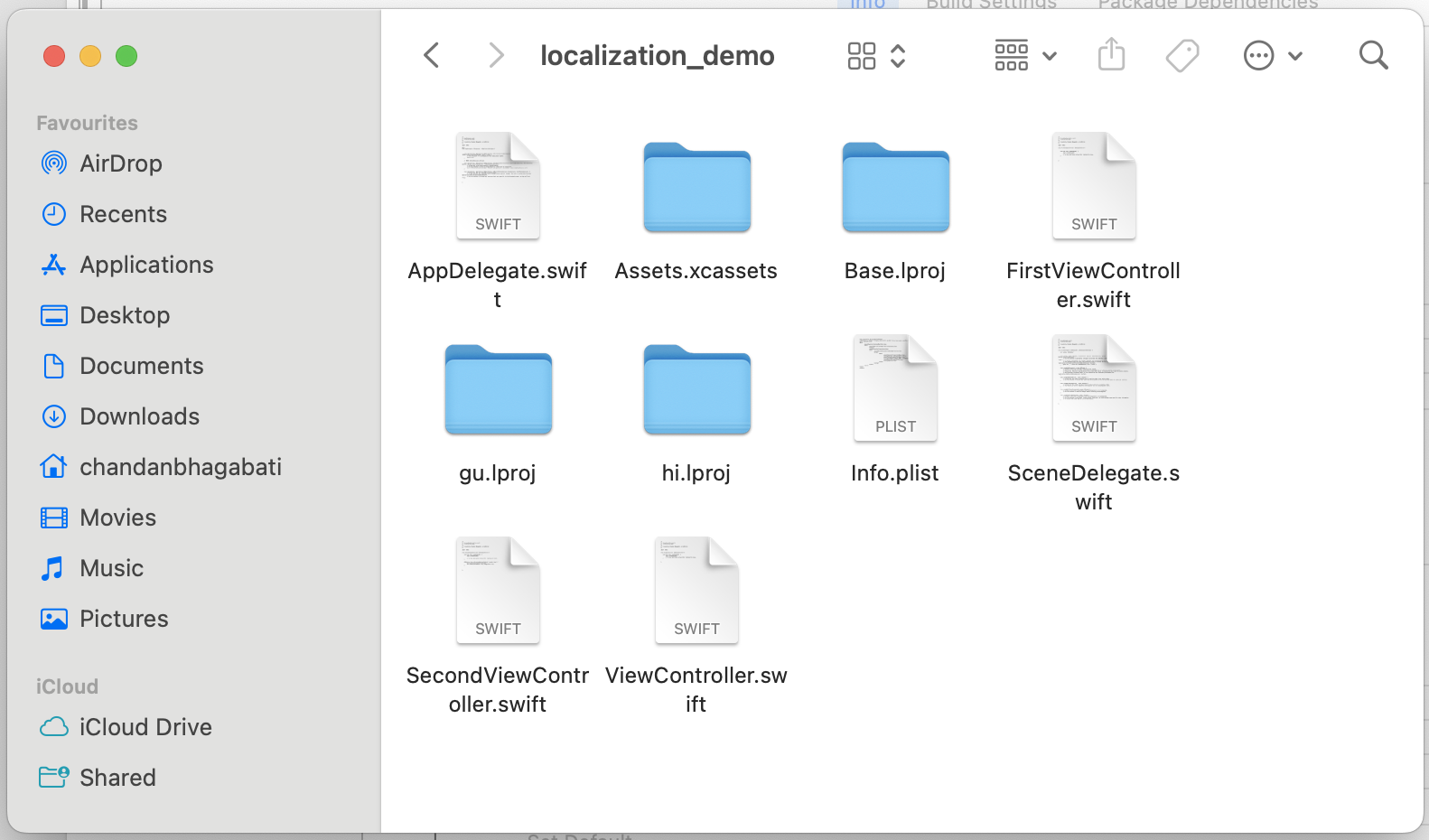
Under Localization click the plus(+) symbol to ad languages. We will add Hindi and Gujarati from the list of languages.



1. Changes in project folder

After adding languages there will be some changes in the project folder. Language folder like hi.lproj, gu.lproj and en.lproj will be created. Each folder will contain main.strings file which contains all the main.storyboard strings.

And main.storyboard will be moved to base.lproj.



1. Test

Let us test if localization works with our application after these steps. We will change the Title of the first page in Hindi and Gujarati.

Open **Main(Hindi)** file, find the text ‘First page’ and replace it with ‘पहला पन्ना’

Similarly open Main(Gujarati) file , find the text ‘First Page’ and replace it with ‘પ્રથમ પૃષ્ઠ’

We can check it in simulator by changing the system language. To do that, open

Product -> Scheme -> Edit Scheme.

In the Edit Scheme window select ‘Options’ tab and change the app language to ‘Hindi’.

Now run the application on simulator. The Title of the first screen should change to Hindi.

Repeat the steps for checking Gujarati.

1. Handling localization fallback

What happens if a user is using a language that our app does not support or our app supports the language but all strings are not localized.

The system should fallback to default language.

To prevent any unexpected behavior, we can create an extension function that encapsulates NSLocalizedString with the desired defaultLanguage to fall back on when the preferred language isn’t found.

Creat a new class Utils.swift and add below String extension.

/\*

Utils.swift

\*/

**import** Foundation

**extension** String {

**func** localize(comment: String = "") -> String {

**let** defaultLanguage = "en"

**let** value = NSLocalizedString(**self**, comment: comment)

**if** value != **self** || NSLocale.preferredLanguages.first == defaultLanguage {

**return** value // String localization was found

}

// Load resource for default language to be used as

// the fallback language

**guard** **let** path = Bundle.main.path(forResource: defaultLanguage, ofType: "lproj"), **let** bundle = Bundle(path: path) **else** {

**return** value

}

**return** NSLocalizedString(**self**, bundle: bundle, comment: "")

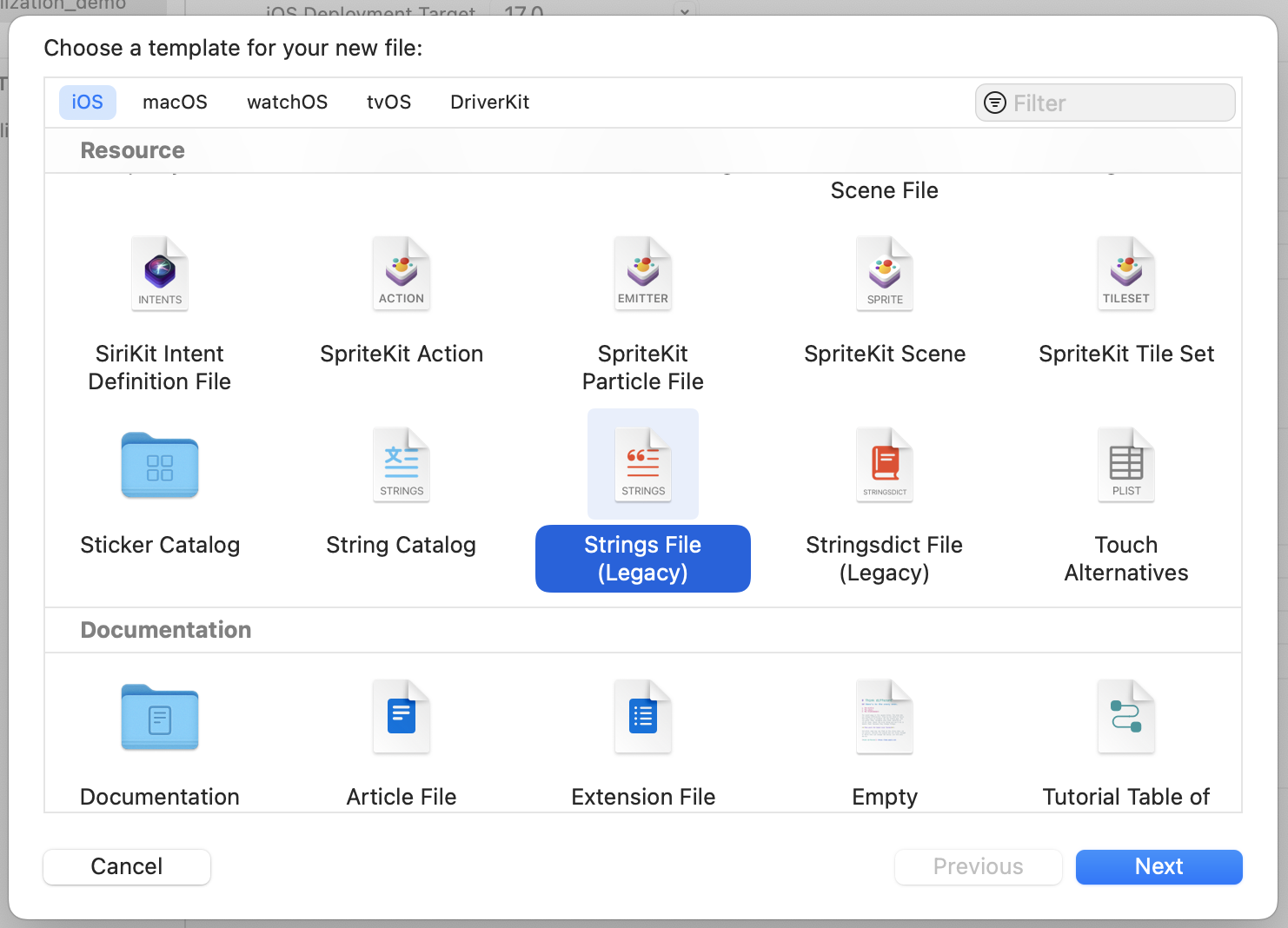
}

}

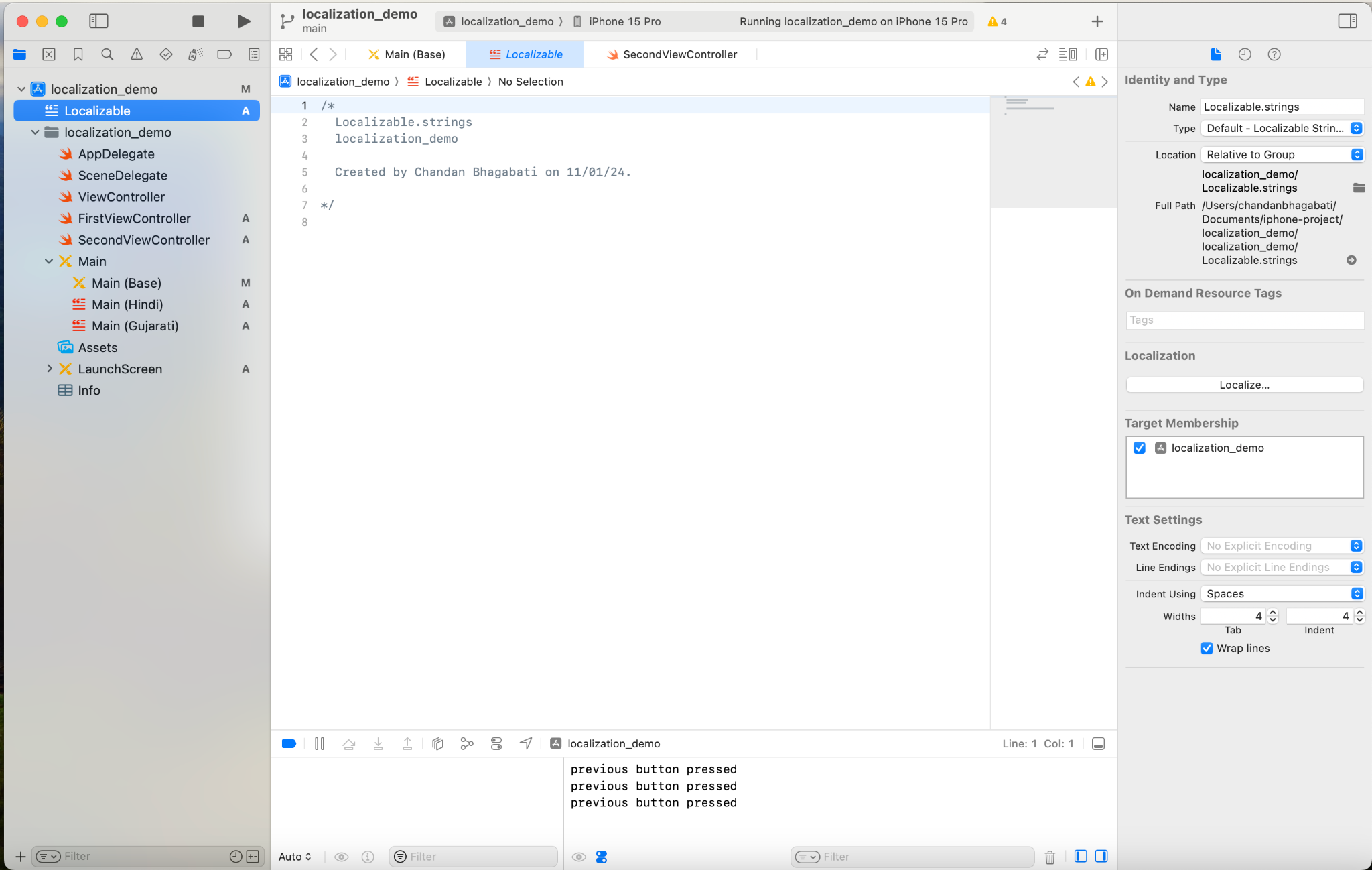
1. In code String localization

So far we are able to localize the Storyboard content. Now we will localize in-code strings. We will use String.localize() extension created above.

Let us define localizable.strings file by going to **File -> New File** and select Strings file.

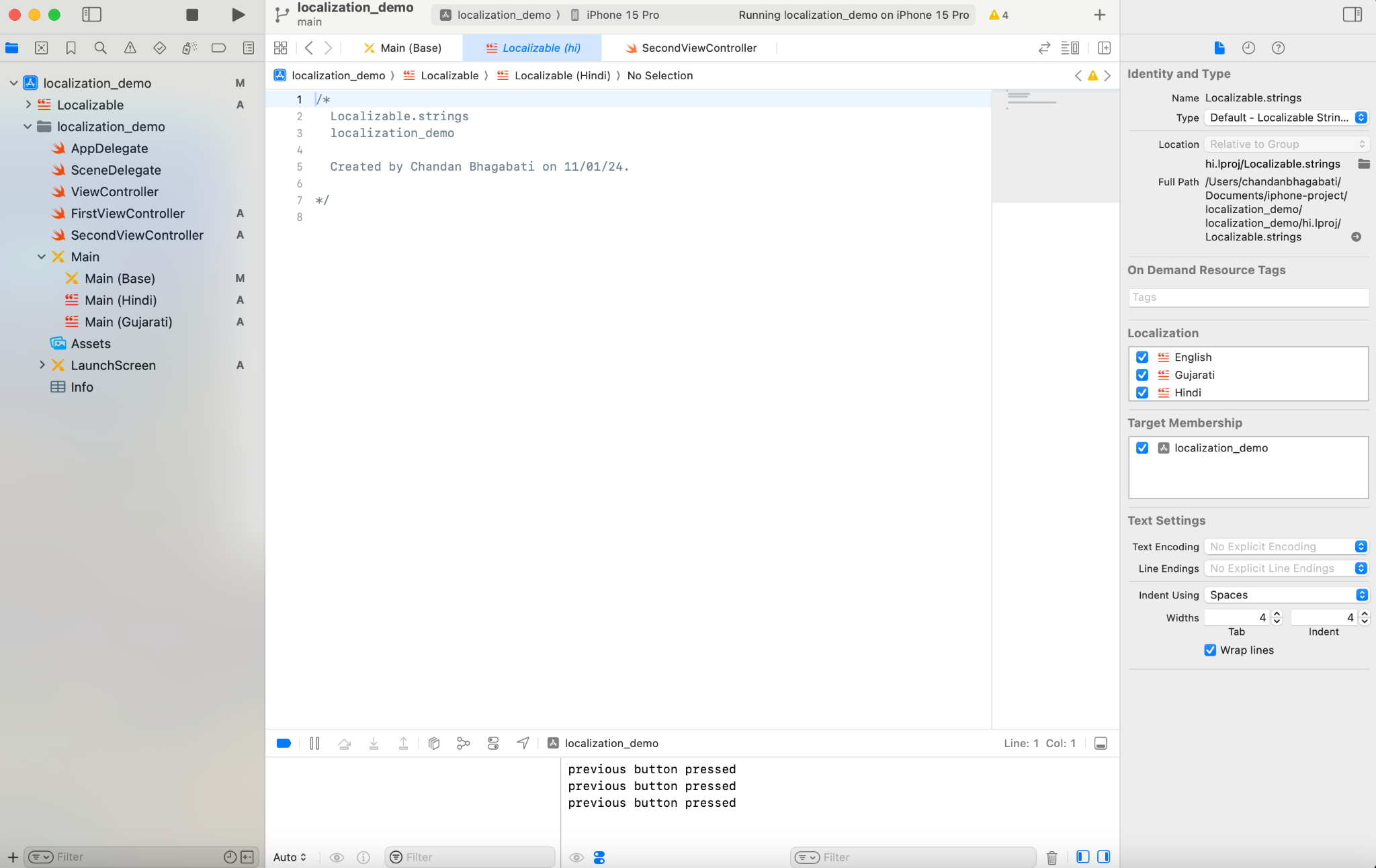


Open **Localizable.strings** file and click on **localize** button in Inspector window.



You will see all the available language selected in step 2. Check all the languages.

Eventually it will create individual Localizable.strings for each language.



We will define all the strings to be localized in localizable file. Let us start with ‘My First Page’ label.

Open Localizable(English) file and define following key value.

"page.content.first" = "My first Page";

Then open Localizable(Hindi) and Localizable(Gujarati) and add following key value pairs respectively.

"page.content.first" = "मेरा पहला पेज";

"page.content.first" = "મારું પ્રથમ પૃષ્ઠ";

Create an outlet for the UILabel in FirstViewController.swift

**@IBOutlet** **weak** **var** first\_page\_content: UILabel!

Update the UILabel content in viewDidLoad method.

**override** **func** viewDidLoad() {

**super**.viewDidLoad()

**self**.first\_page\_content.text = "page.content.first".localize()

}

You will notice we have used the key.localize() extension here to update the content.

1. Test implementation

Now change the app language by going to Edit Scheme as we did in step 4.

You should see the ‘My First Page’ text in respective language when you change the app language.

1. Working with localized file (Export/Import)

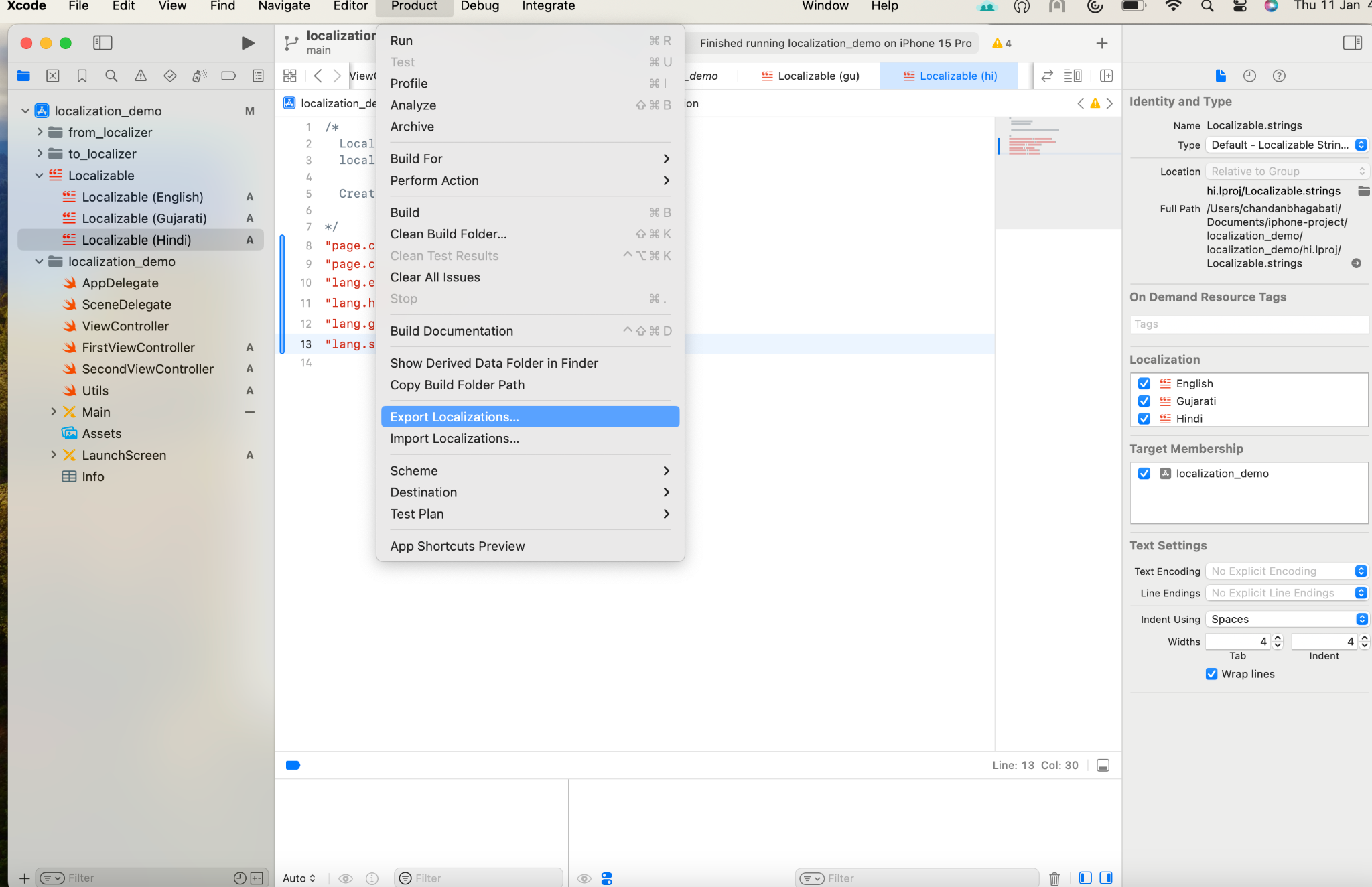
So far we have translated the localizable file ourselve. If your project has 100s of such strings it is a tedious process to update these strings to everyfile in their respective languages. You need language knowledge and time for that.

Instead of doing that, you can define only the English strings and rely on Reverie localization service for rest of the languages.

For that define all strings to be localized in Lcalizable(English) file.

Create 2 folders in your project. a) to\_localizer b)from\_localizer. These 2 folders will make your life easy.

Go to Product -> Export localizations and select ‘to\_localizer’ as destination folder. All your app strings will be exported to this folder. Share this folder with Reverie.



Reverie will update all the strings to be localized in their respective languages. Once done Rverie will give back the folder to you.

Copy the folder to ‘from\_localizer’ folder. Go to Product -> Import localizations and select ‘from\_localizer’ as your source.

All Localizable files will be updated instantly.

1. Enable local change in application

We have tested the app in different languages by changing the language in Edit Scheme. Now we have to give an option to the app user to change the language.

Ideally user should be redirected to system settings for changing system language.

The second option is to change the system language programmatically. Apple does not recommend this approach as it may create some inconsistency in the app.

We will implement both the approach and see if we see any issue.

For redirecting user to system settings we will use below code.

**func** invokeLangSettings(){

**if** **let** settingsURL = URL(string: UIApplication.openSettingsURLString) {

UIApplication.shared.open(settingsURL, options: [:], completionHandler: **nil**)

}

}

For setting language programmatically we will use below code. When we change system language programmatically, the application needs to restart to get the change effect. For that we will give a notification alert to the user to reopen the app.

**func** setLangauge(languageCode: String, language: String) {

**let** alert = UIAlertController(title: **nil**, message: "lang.change.alert".localize(), preferredStyle: .alert)

alert.addAction(UIAlertAction(title: "exit.app".localize() , style: .default, handler: { [**weak** **self**](\_) **in**

// Update app's language with the language code

UserDefaults.standard.set([languageCode], forKey: "AppleLanguages")

UserDefaults.standard.synchronize()

**self**?.closeApplicationWithNotification(language: language)

}))

// Show change language alert to user

**self**.present(alert, animated: **true**, completion: **nil**)

}

**func** closeApplicationWithNotification(language: String) {

**let** content = UNMutableNotificationContent()

content.title = "Language changed to" + " \(language)"

content.body = "Tap to reopen the application"

content.sound = UNNotificationSound.default

**let** trigger = UNTimeIntervalNotificationTrigger(timeInterval: 0.5, repeats: **false**)

**let** identifier = "revlocal"

**let** request = UNNotificationRequest.init(identifier: identifier, content: content, trigger: trigger)

**let** center = UNUserNotificationCenter.current()

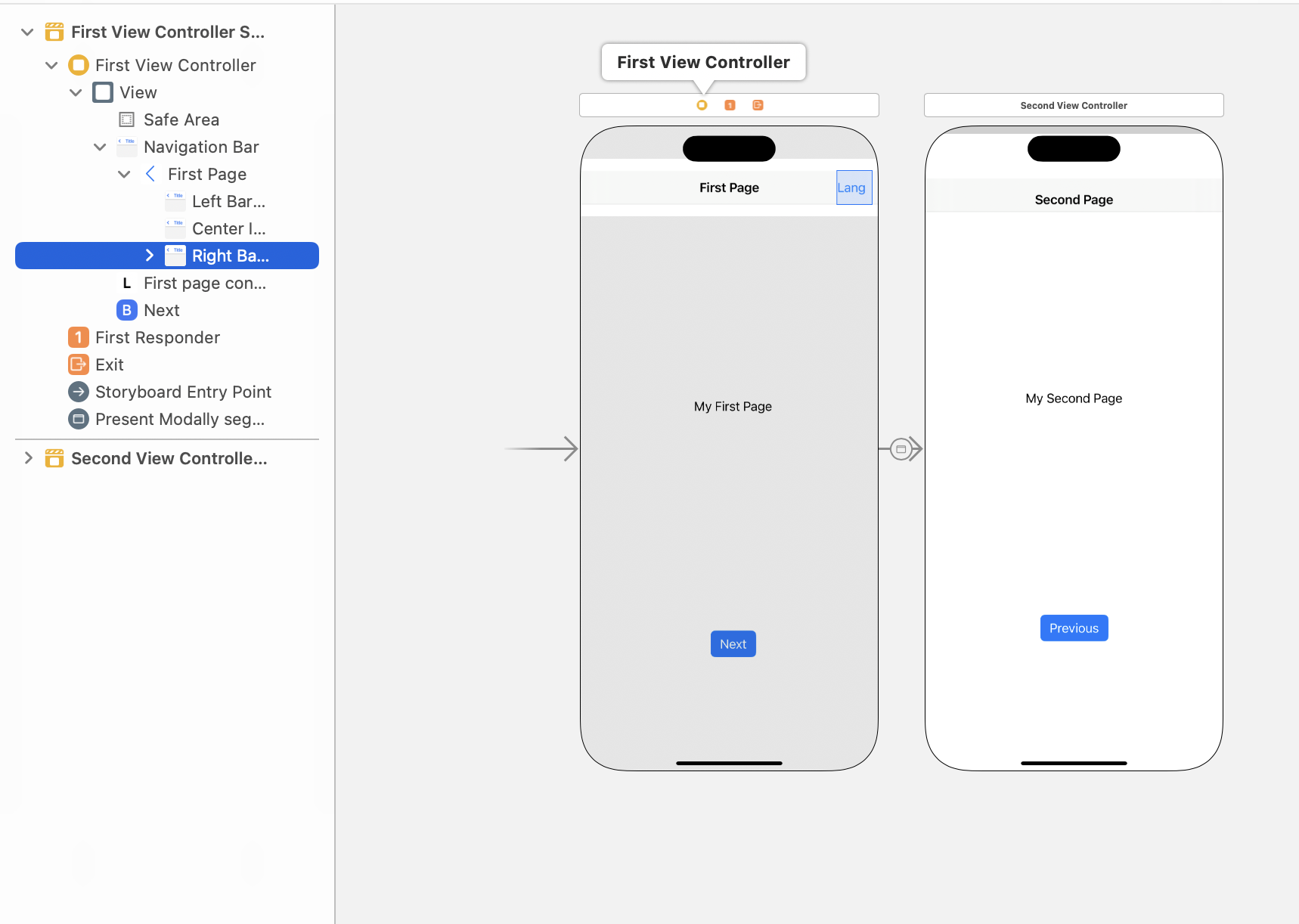
center.add(request)

exit(EXIT\_SUCCESS)

}

Now we will stitch together both the options in a single menu in the first screen.

Add a UIBarButtonItem to the Navigation Bar and rename it to Lang.



Create an outlet of the BarButton Item in FirstViewController.swift

**@IBOutlet** **weak** **var** lang\_option\_button: UIBarButtonItem!

Define UIMenu with specific actions.

**func** addBarMenu(){

**let** barButtonMenu = UIMenu(title: "", children: [

UIAction(title: "lang.english".localize(), handler: { [**weak** **self**] (\_) **in**

**self**?.setLangauge(languageCode: "en", language: "English")

}),

UIAction(title: "lang.hindi".localize(), handler: { [**weak** **self**] (\_) **in**

**self**?.setLangauge(languageCode: "hi", language: "Hindi")

}),

UIAction(title: "lang.gujarati".localize(), handler: { [**weak** **self**] (\_) **in**

**self**?.setLangauge(languageCode: "gu", language: "Gujarati")

}),

UIAction(title: "lang.settings".localize(), handler: { [**weak** **self**] (\_) **in**

**self**?.invokeLangSettings()

})])

lang\_option\_button?.menu = barButtonMenu

}

Add a bar UIMenu to the button.

**override** **func** viewDidLoad() {

**super**.viewDidLoad()

**self**.first\_page\_content.text = "page.content.first".localize()

addBarMenu()

}

1. Test complete application

Run the application on simulator and play with the menu.

We are good to go with the localized application. The only thing to take care is adding the in-code strings to localizable files and share it with localization provider i.e. Reverie. Update with the received file.

And always use the string extension method key.localize().