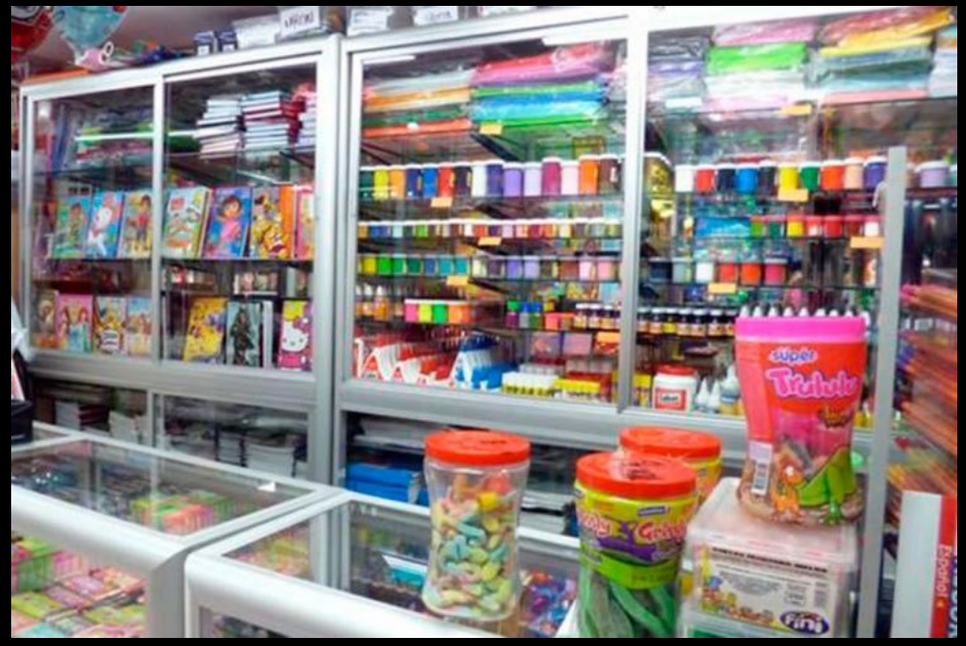
# Utilizando SQLite en Swift

iOS Dev Lab FCA
Cristian Tafolla Rosales

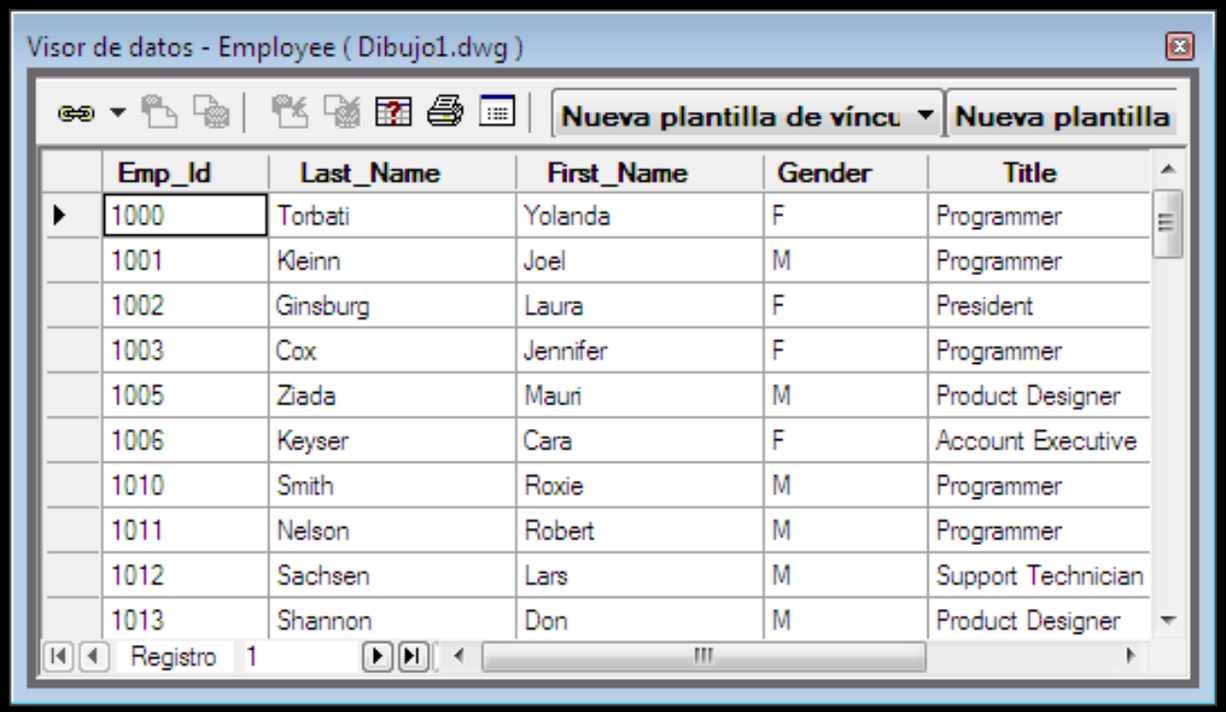
# Base de datos

- Conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su uso posterior.




# Base de datos relacional

- Colección de elementos de datos organizados en un conjunto de tablas.
- Nos permite acceder o administrar los datos sin tener que reorganizar las tablas de dicha base.

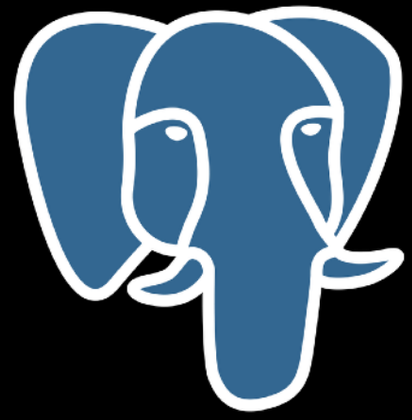


# Hay dos tipos de relaciones:

- Tablas
- De asociación o vinculación

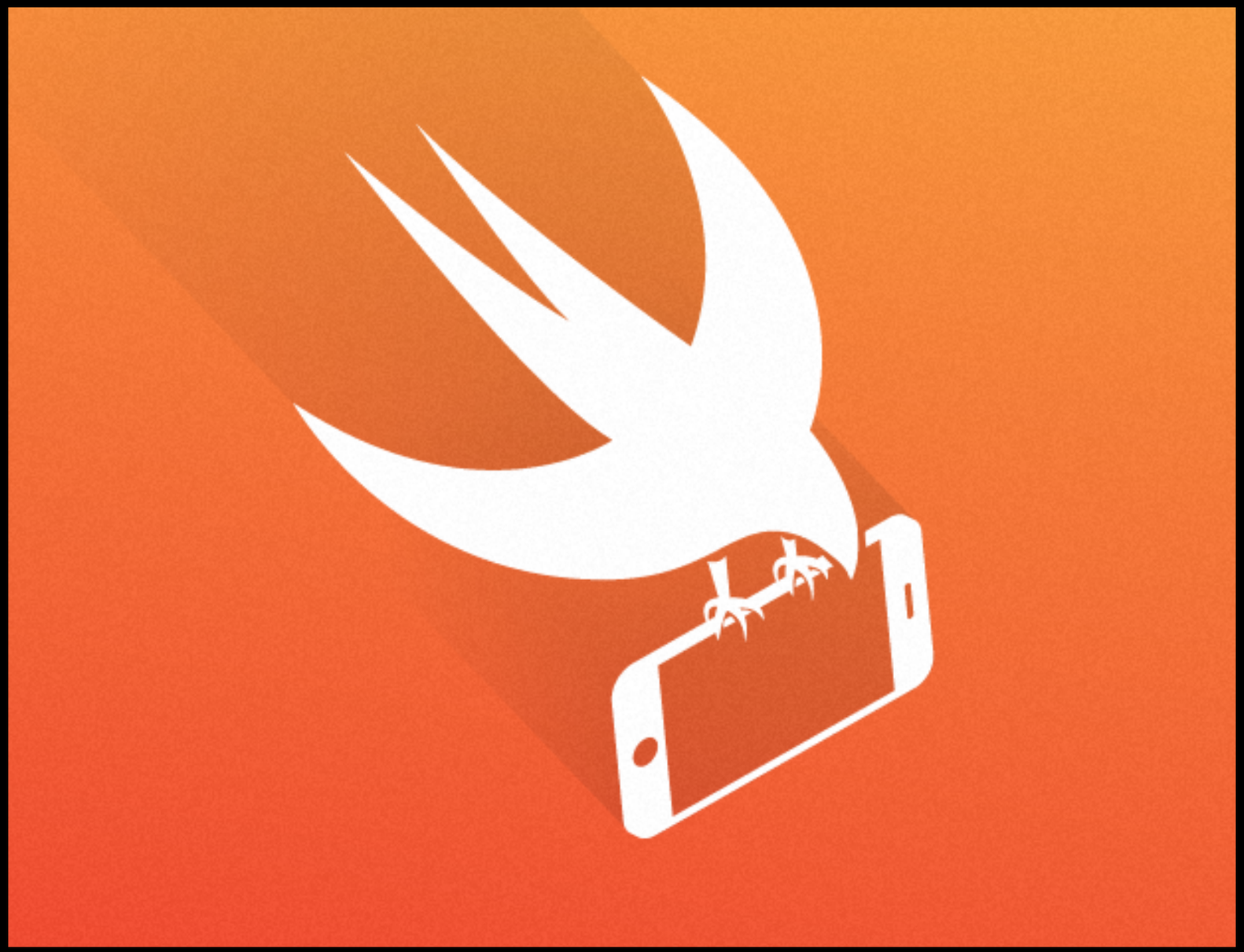
# RDBMS

- Software que permite gestionar una base de datos relacional. Por lo general utilizan el Lenguaje de Consultas Estructuradas (SQL).

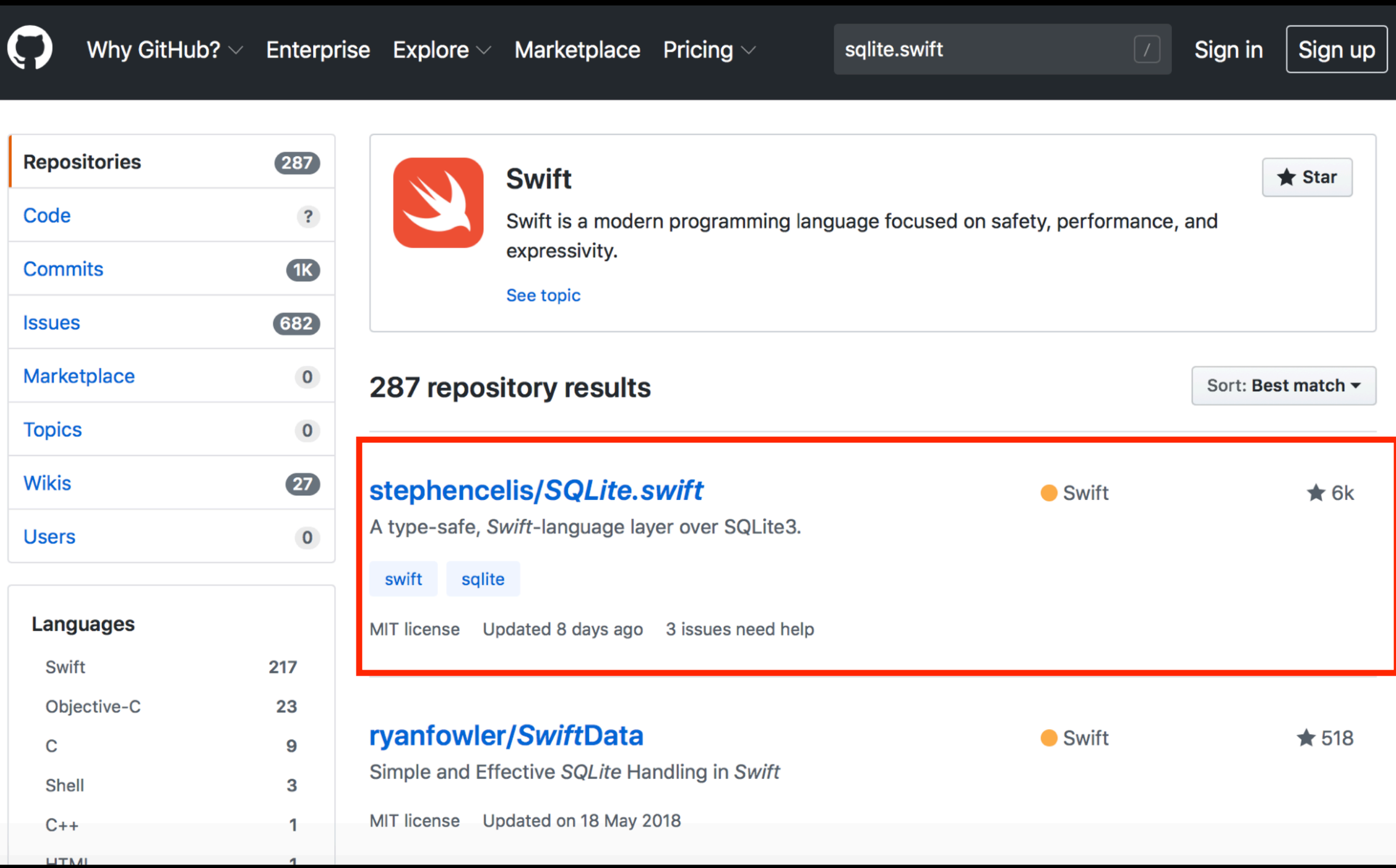




# SQLite

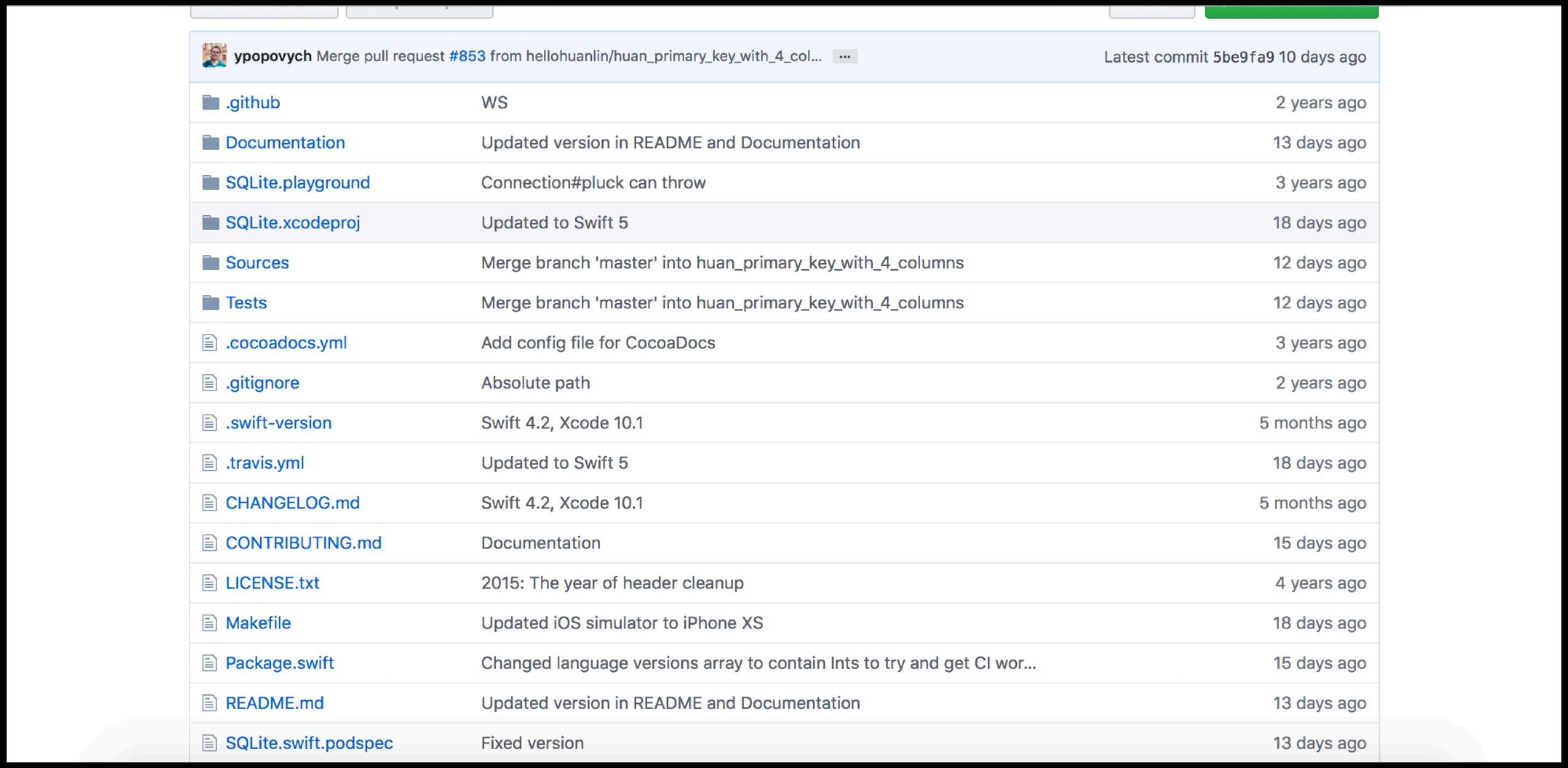
- Sistema de gestión de bases de datos relacionales (RDBMS), de dominio público.



# Manos a la obra

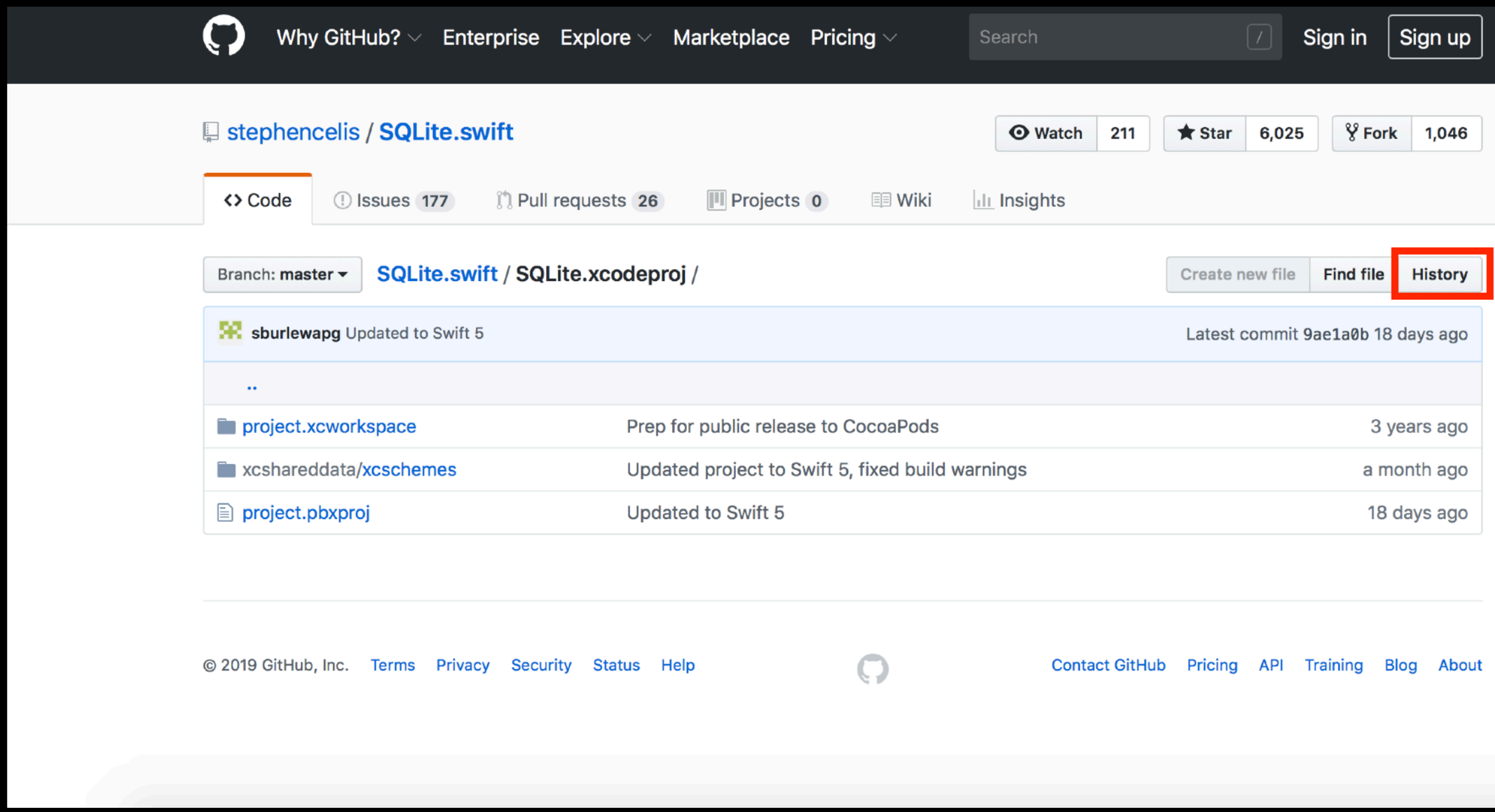



Acceder a Github y buscar “sqlite.swift”




Buscar el archivo “SQLite.xcodeproj” y verificar si concuerda con la versión de Swift que utilizamos

Nota: Nosotros estamos utilizando la versión 4.2 de Swift, así que tendremos que hacer un poco más de trabajo




Dentro del archivo “SQLite.xcodeproj” iremos a su historial

- Reverted to swift 4.2, will go to 5 in another version update ···
sburlewapg committed on 4 Apr ✗
7a98a95

Commits on Apr 3, 2019

Swift 5 ···
sburlewapg committed on 3 Apr ✗
e28f30c

Commits on Mar 30, 2019

Updated project to Swift 5, fixed build warnings
sburlewapg committed on 30 Mar
93f5568

Commits on Dec 19, 2018

Swift 4.2, Xcode 10.1 ···
jberkel committed on 19 Dec 2018 ✓
1 Verified
cf4183f

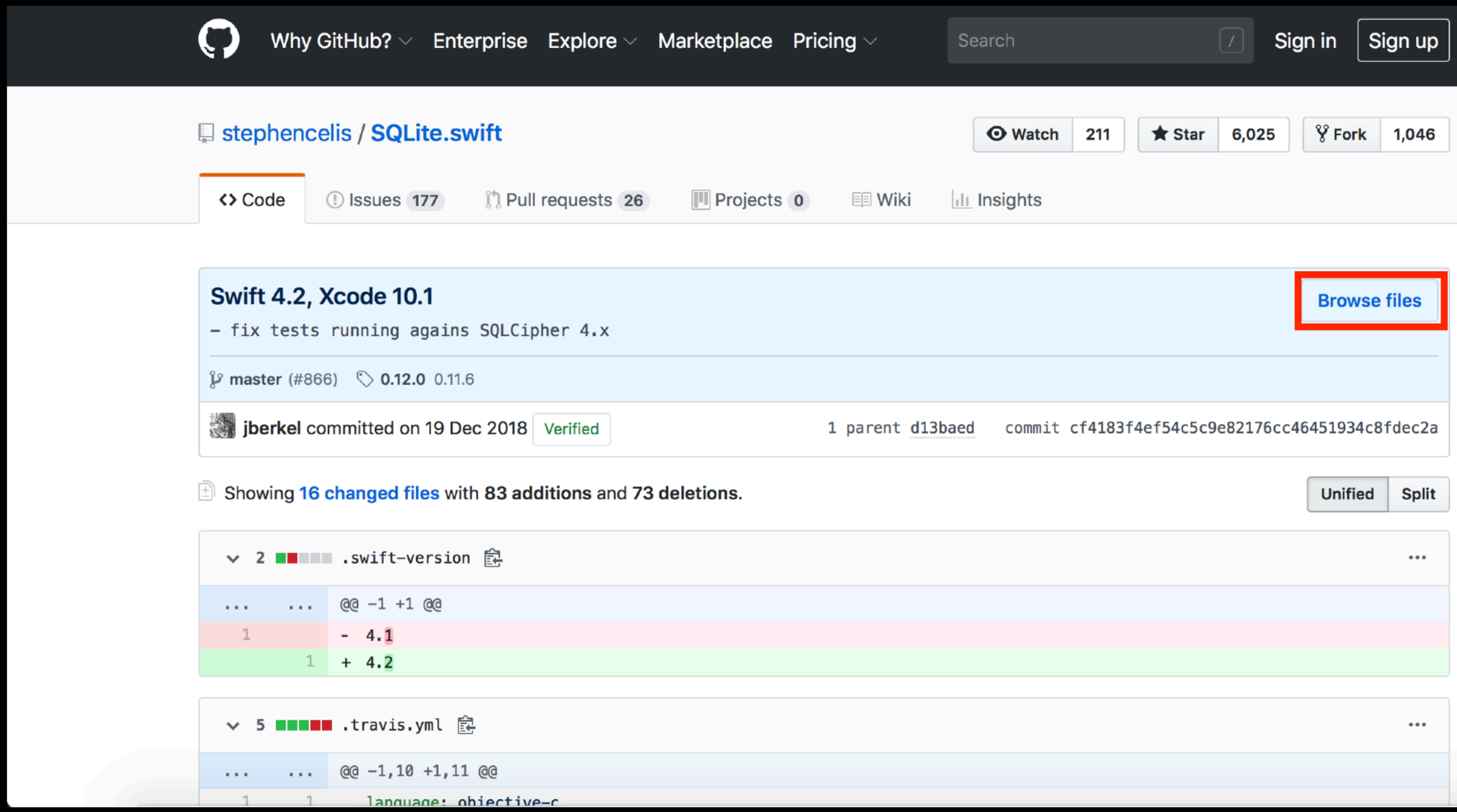
Commits on Apr 13, 2018

Exclude test on iOS/tvOS 9.x
jberkel committed on 13 Apr 2018 ✓
a1d478e

Commits on Apr 12, 2018

Fix Swift 4.1 warnings
jberkel committed on 12 Apr 2018 ✗
f40ca4d

Iremos al commit correspondiente a la versión 4.2 de Swift




Exploraremos los archivos correspondientes a este commit

A type-safe, Swift-language layer over SQLite3.

swift sqlite

807 commits | 3 branches | 13 releases | 54 contributors | MIT

Tree: cf4183f4ef | New pull request | Find File | Clone or download

jberkel Swift 4.2, Xcode 10.1 ...

| | | |
|---|---|---|
| .github | WS | |
| Documentation | fix markdown table formatting | |
| SQLite.playground | Connection#pluck can throw | |
| SQLite.xcodeproj | Swift 4.2, Xcode 10.1 | |
| Sources | Swift 4.2, Xcode 10.1 | 5 months ago |
| Tests | Swift 4.2, Xcode 10.1 | 5 months ago |
| .cocoadocs.yml | Add config file for CocoaDocs | 3 years ago |
| .gitignore | Absolute path | 2 years ago |

**Clone with HTTPS**

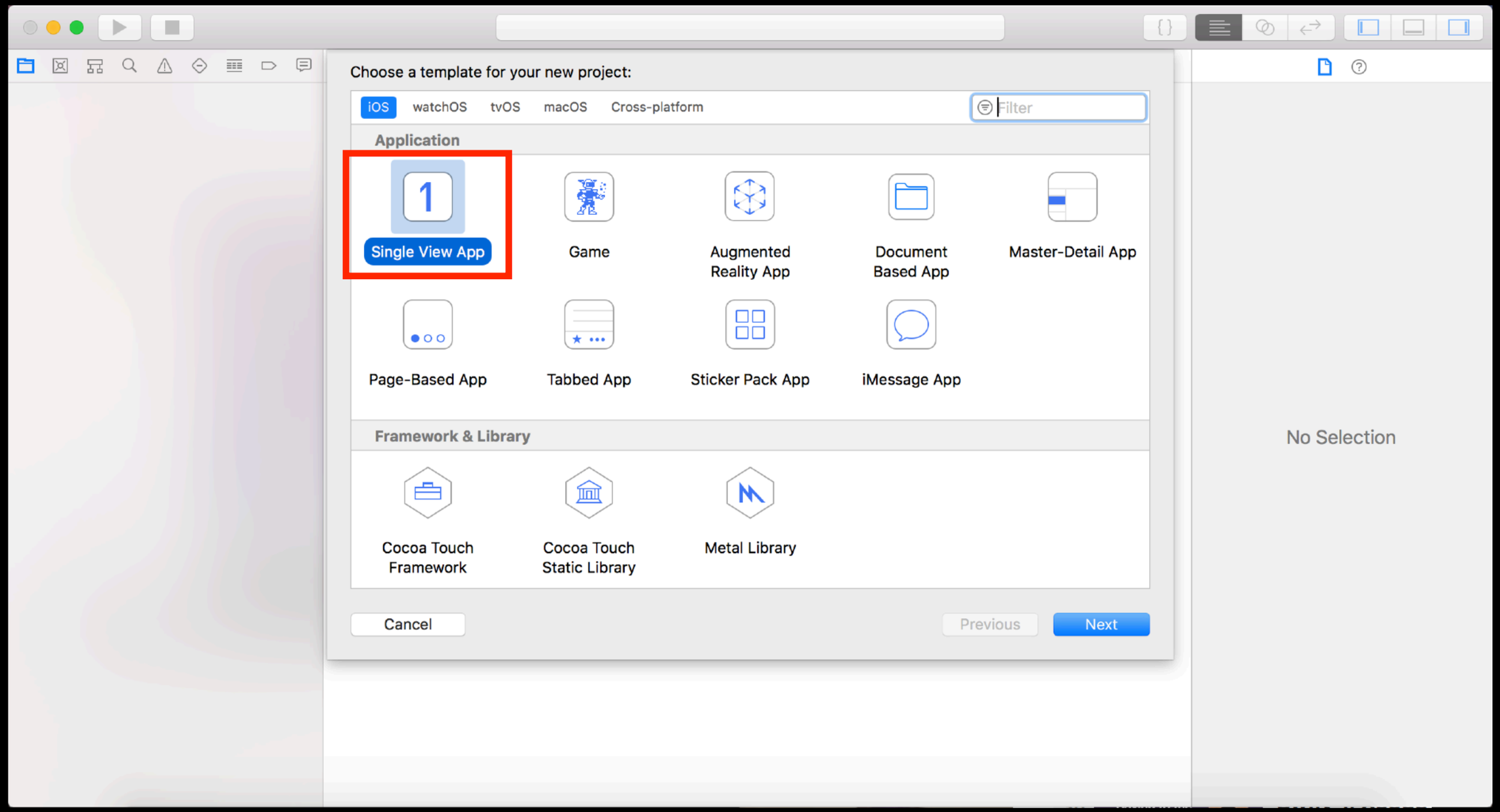
Use Git or checkout with SVN using the web URL.

https://github.com/stephencelis/SQLit

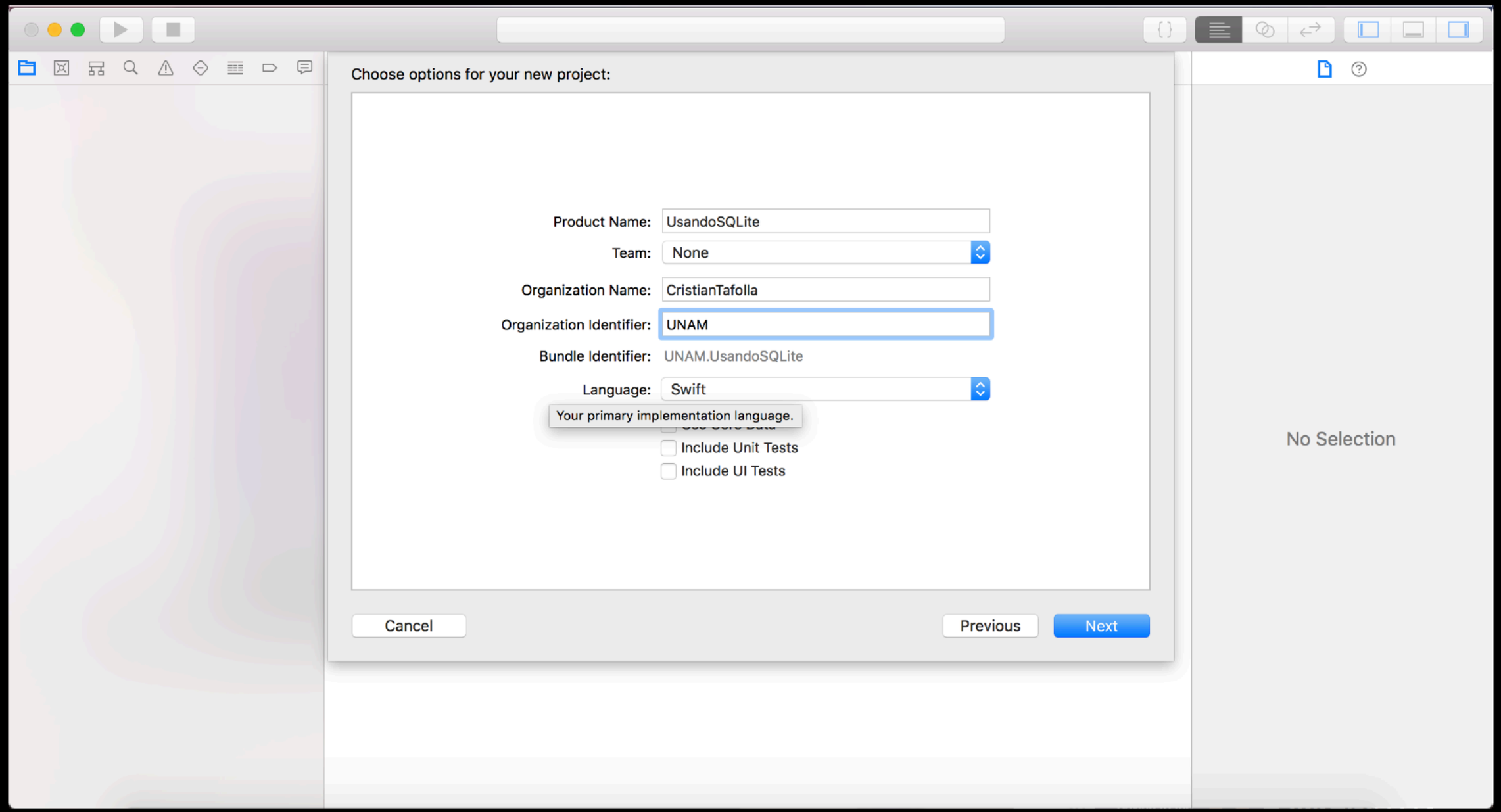
Open in Desktop | Download ZIP

Descargaremos el archivo ZIP

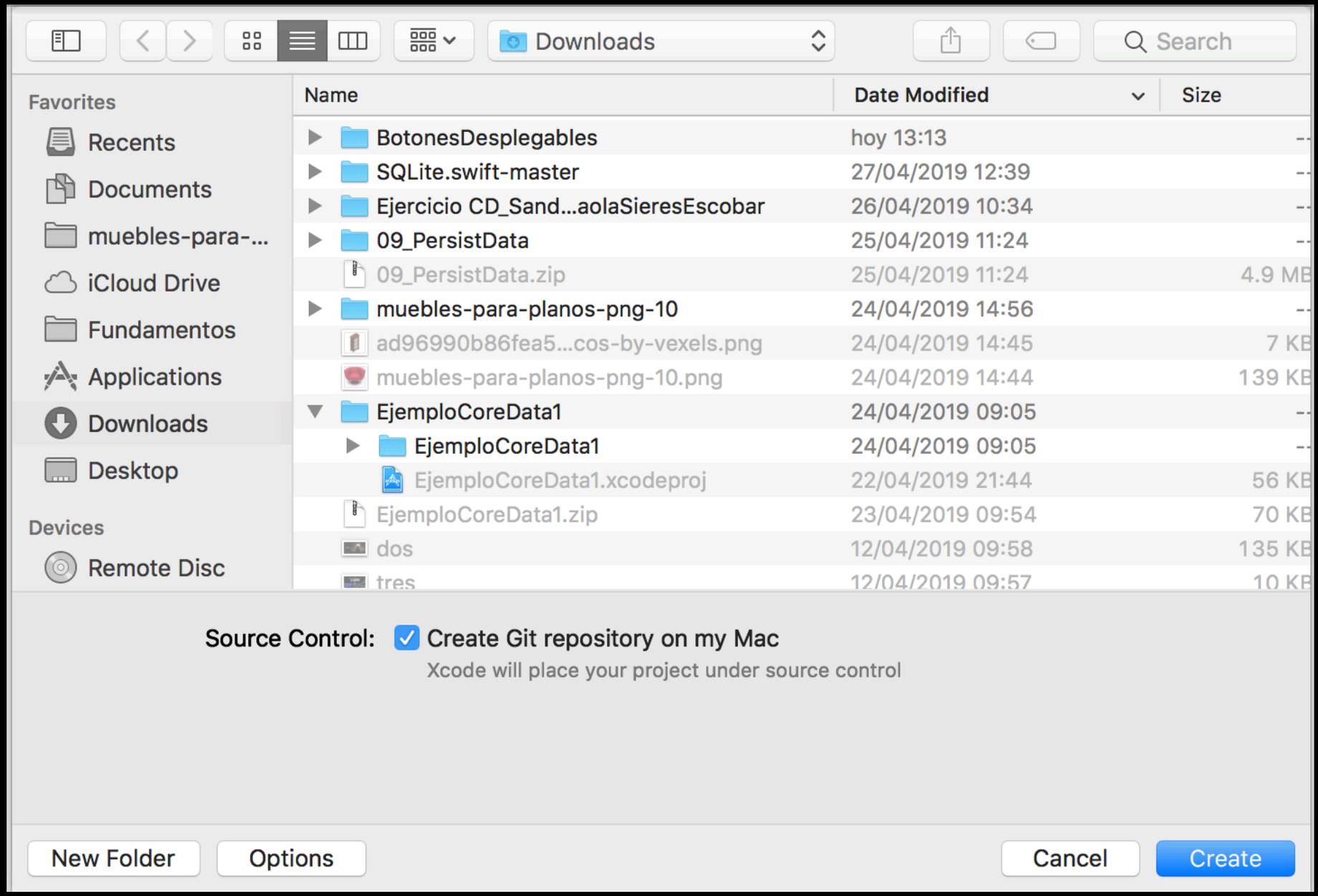

# Crearemos un nuevo proyecto en Xcode



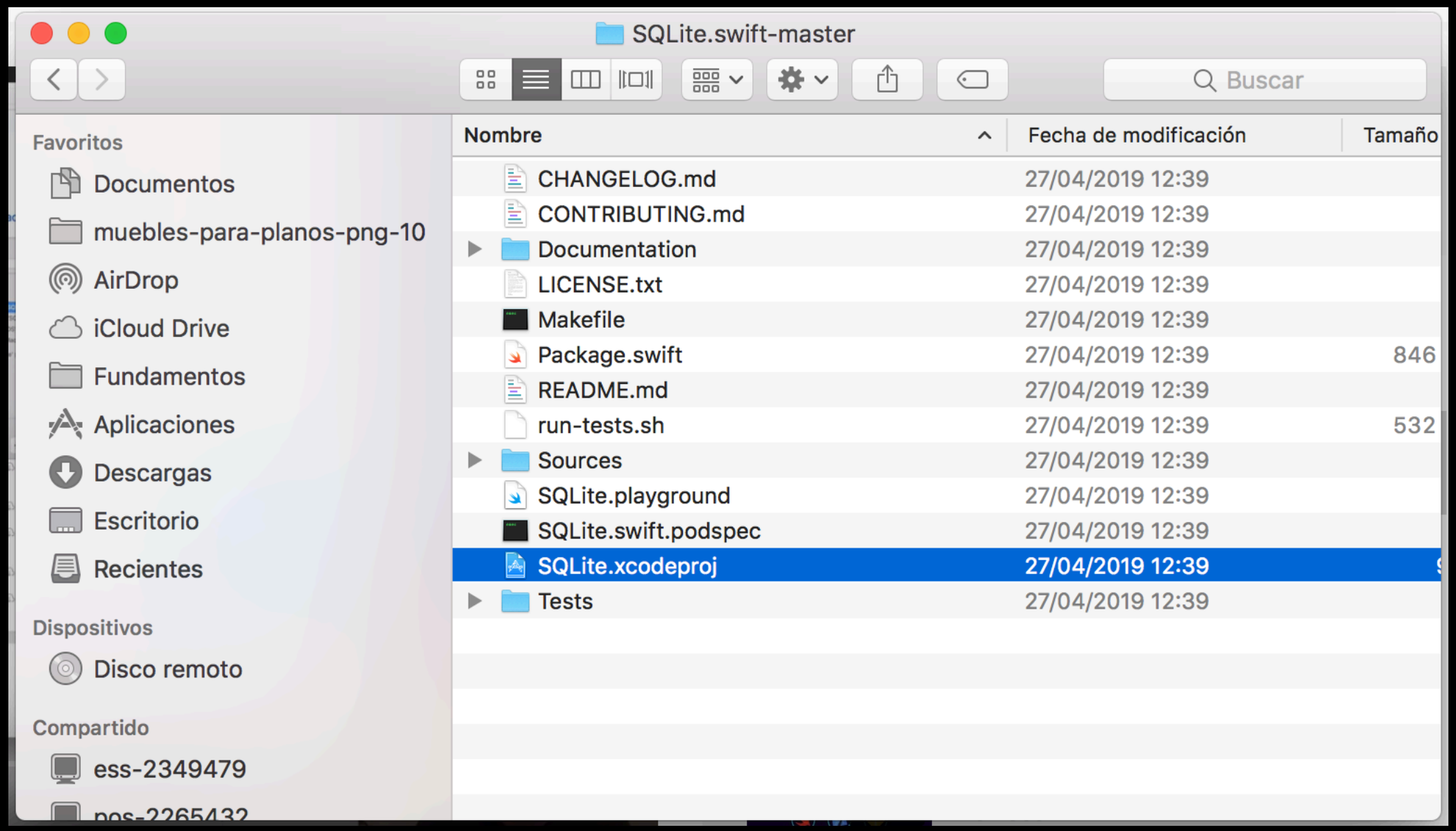

Seleccionamos la opción “Single View App”



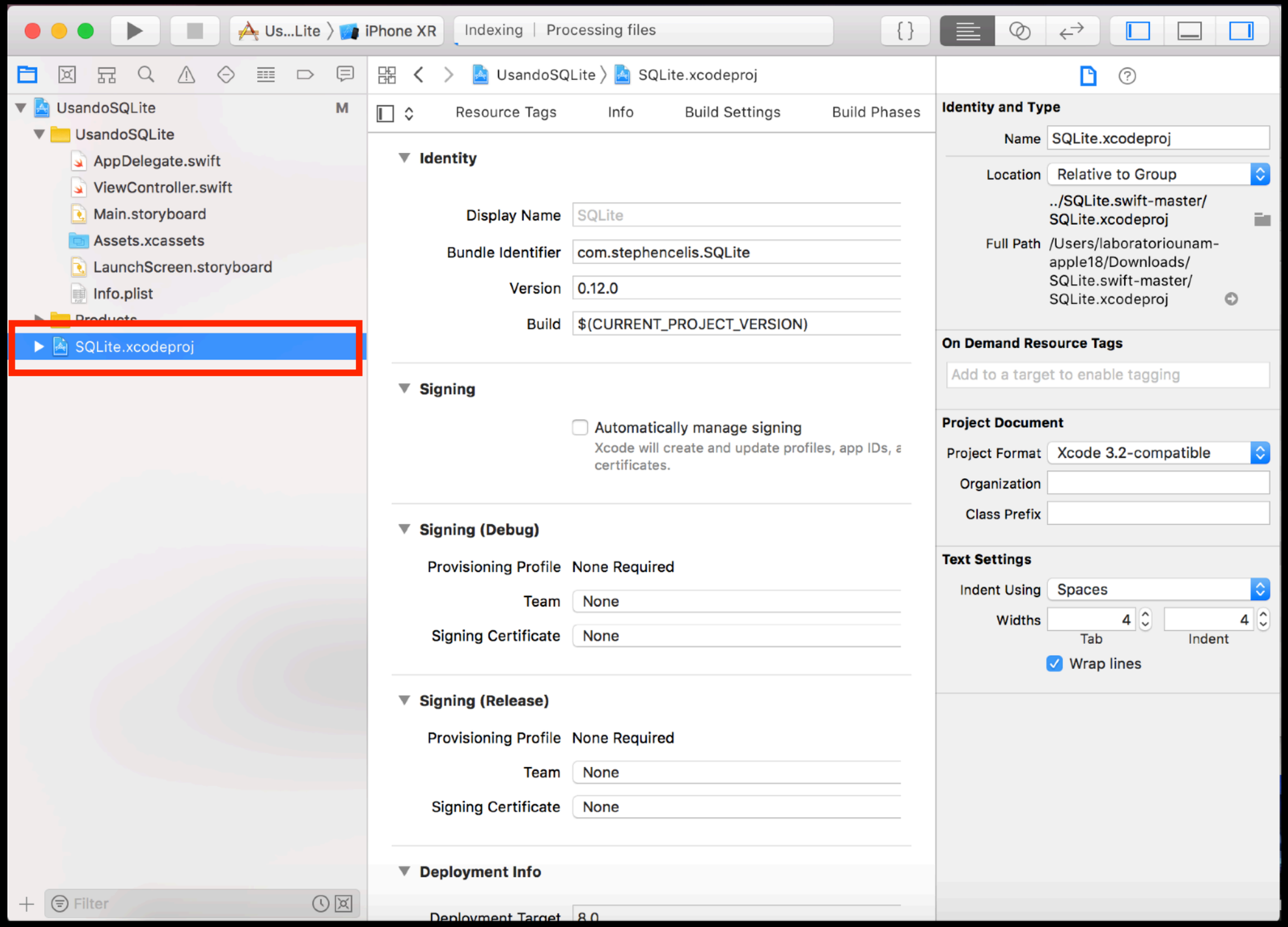

Agregamos los datos pertinentes



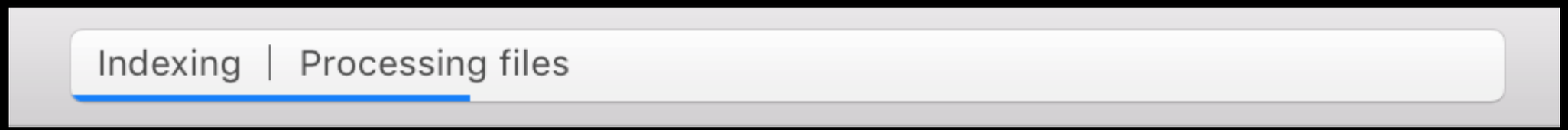

Lo guardamos en la ubicación deseada



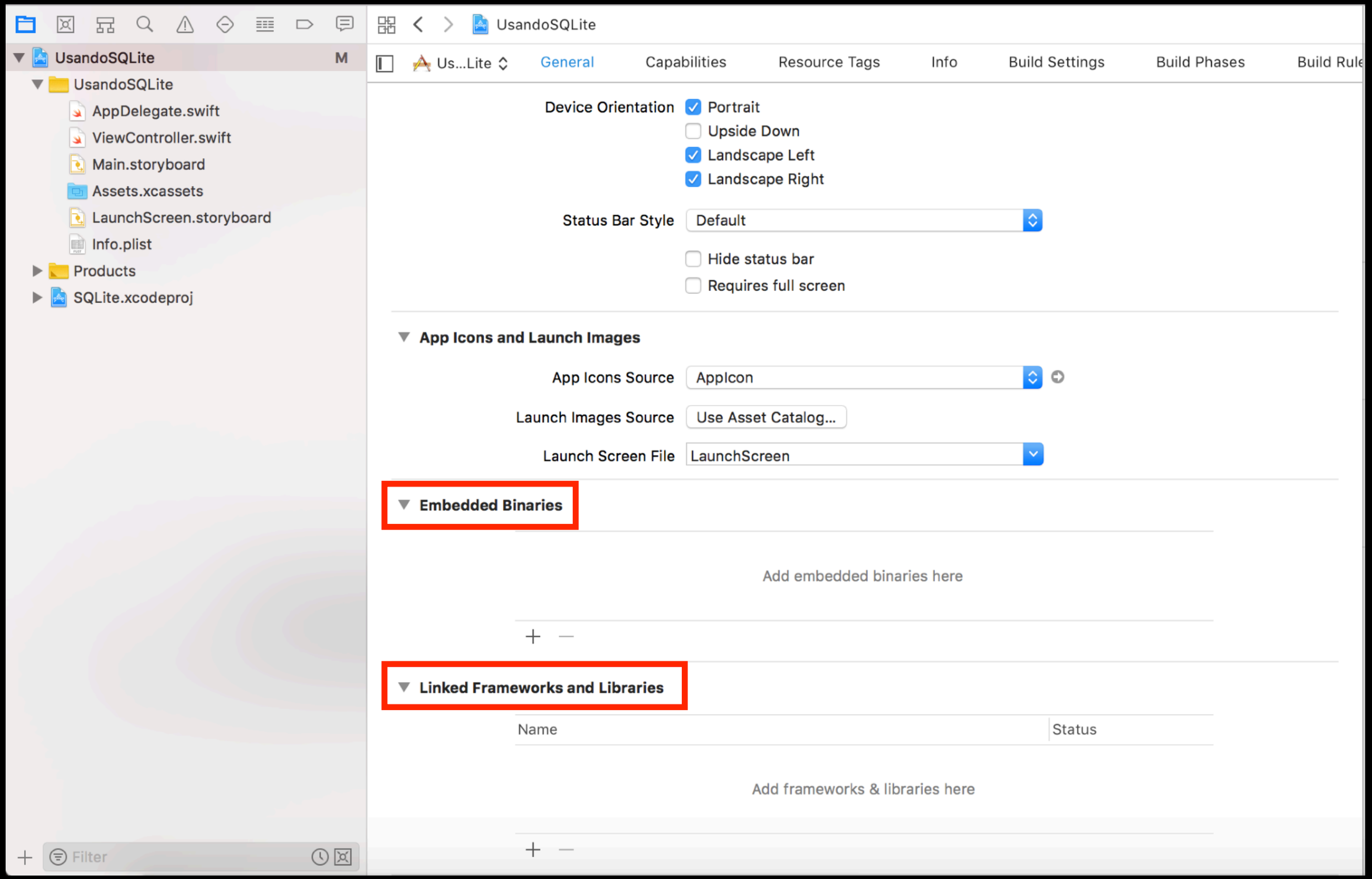

Ubicamos el archivo que descargamos desde Github, y dentro de este el correspondiente a “SQLite.xcodeproj”



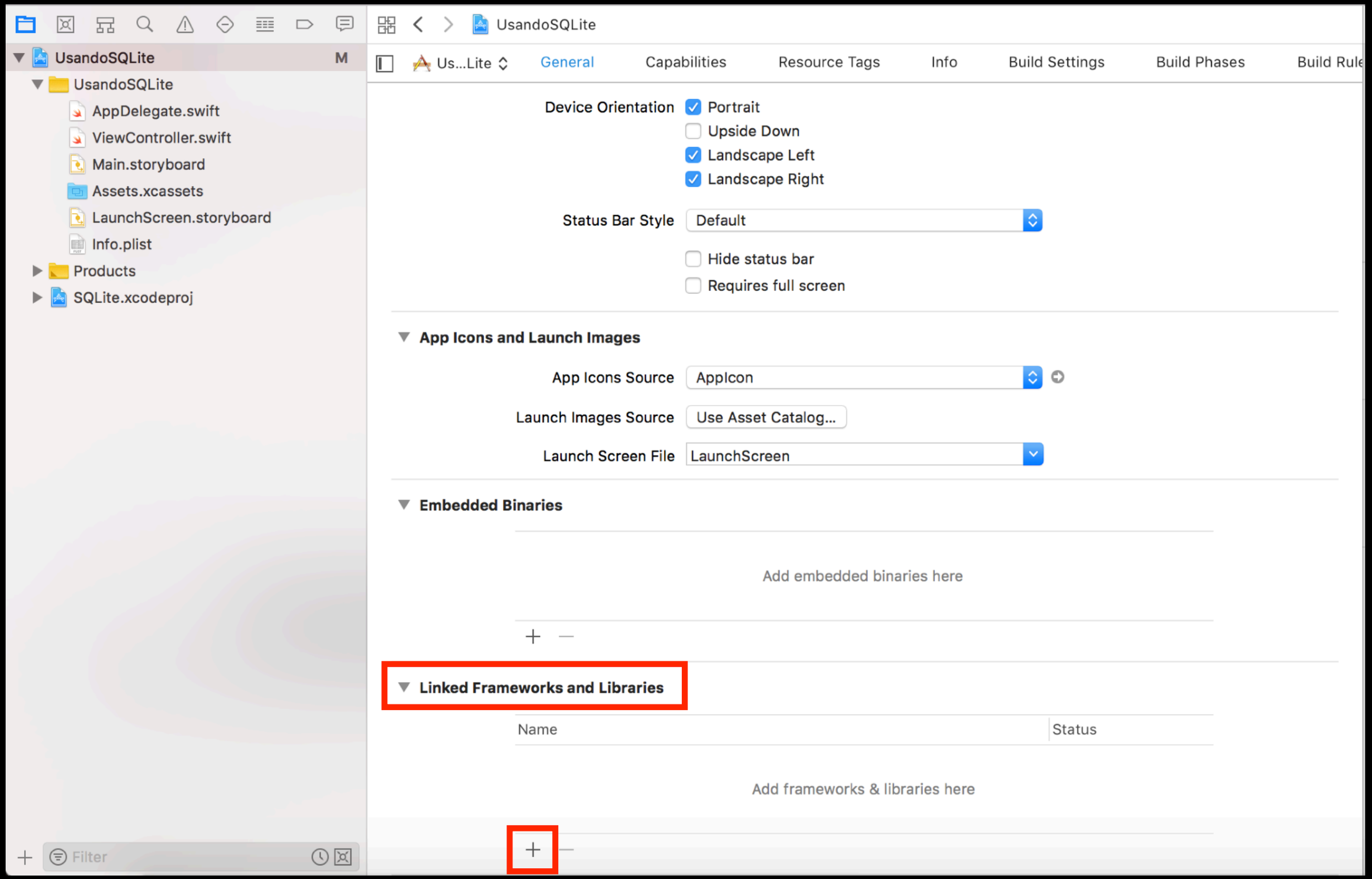

Lo arrastramos dentro del explorador de componentes de nuestro proyecto, en la parte inferior



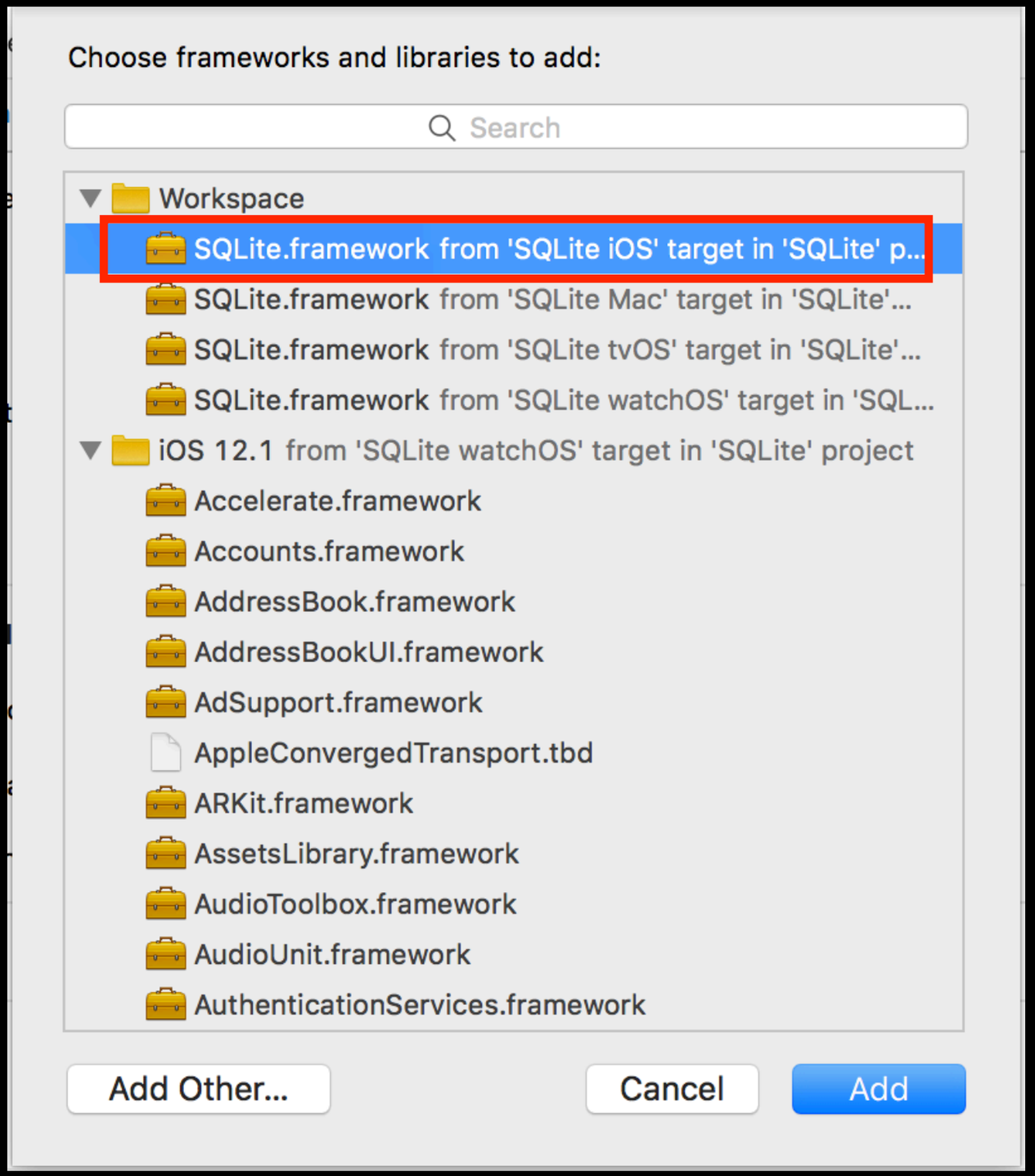

Esperamos a que termine de cargar la barra de estado correspondiente al proyecto (parte superior)




Ubiquemos dos secciones: “Embedded Binaries” y “Linked Frameworks and Libraries”

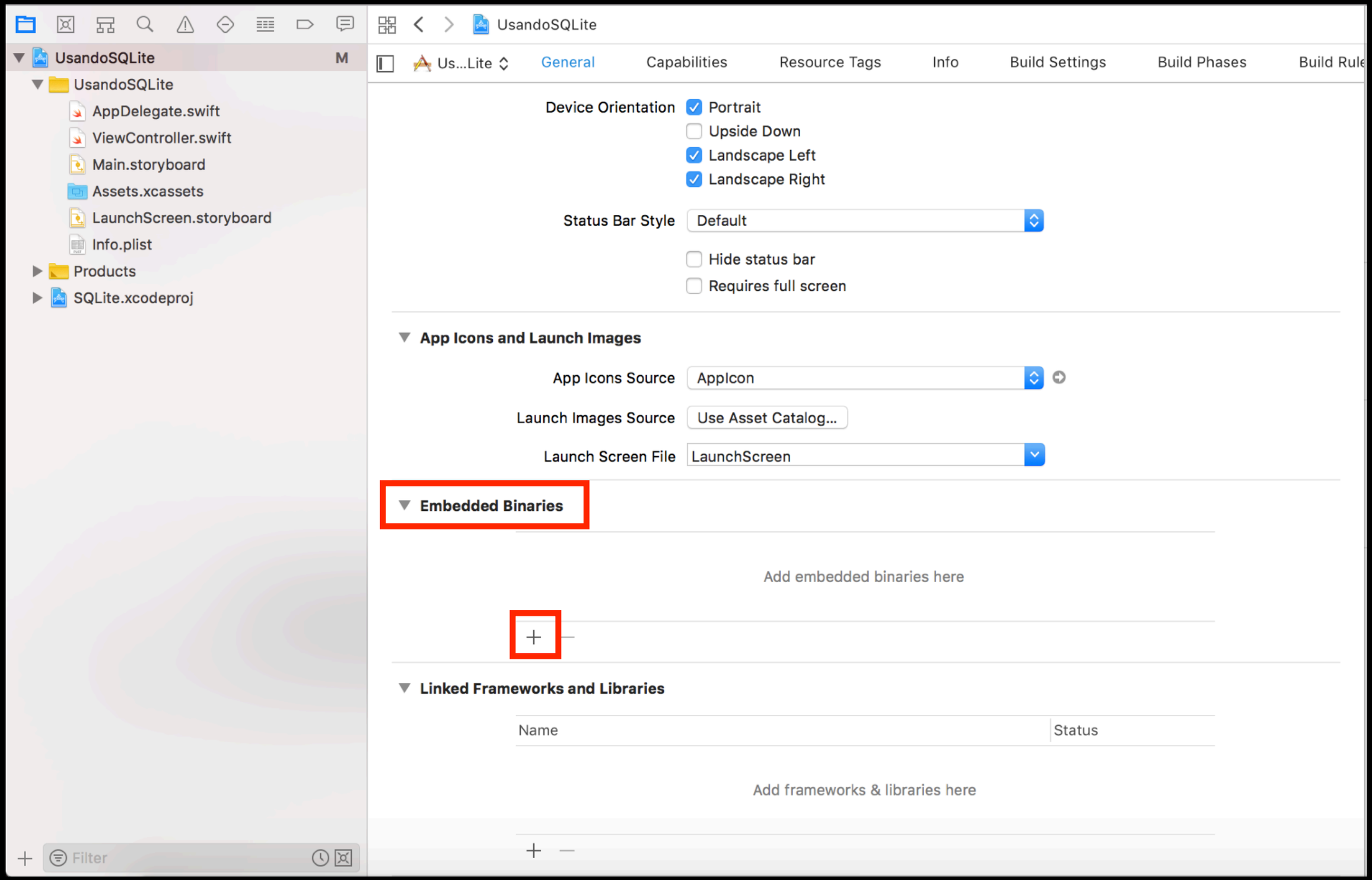



Ubiquemos el signo de más (+) dentro de “Linked Frameworks and Libraries”

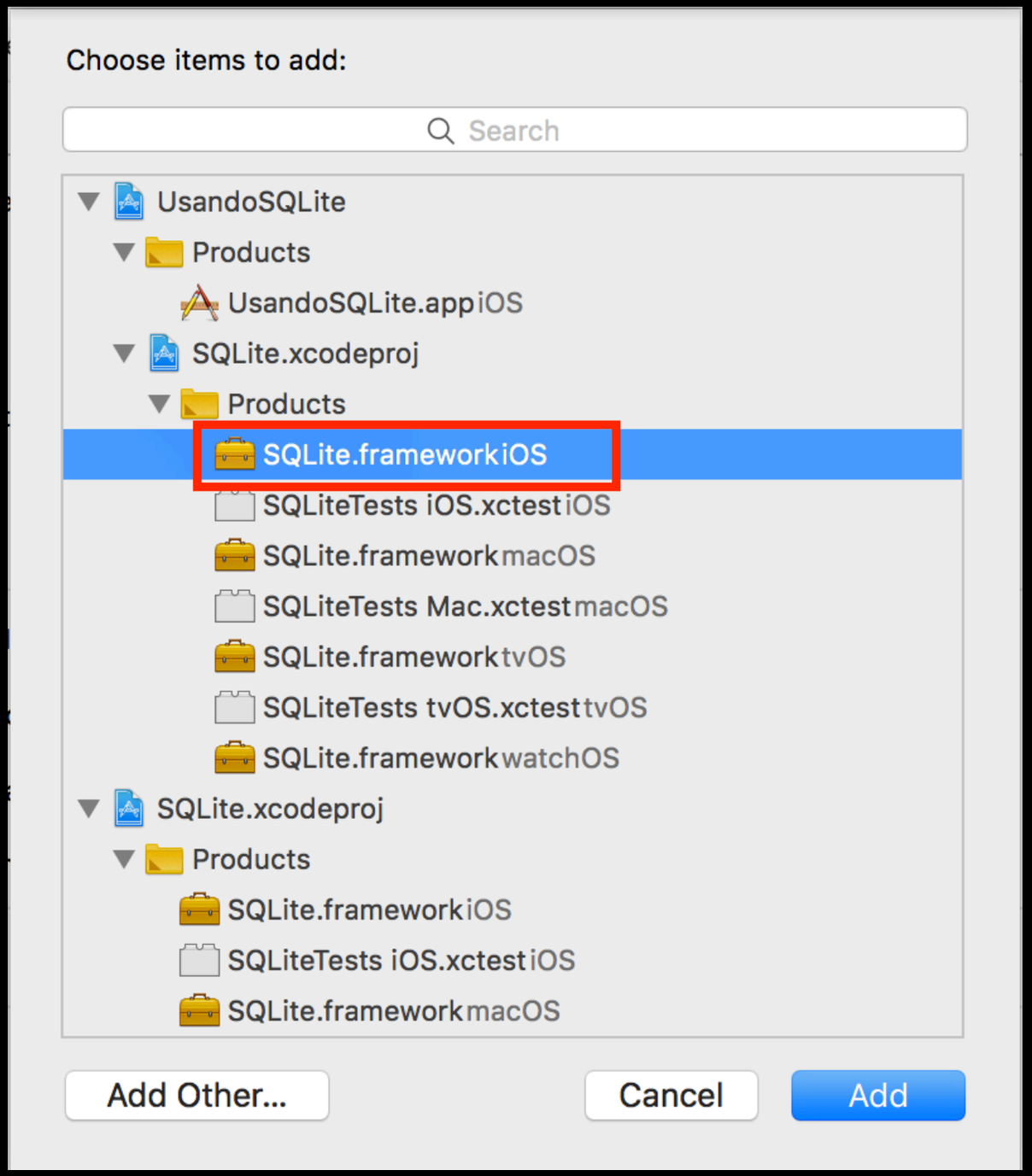



Seleccionemos SQLite.framework correspondiente a los dispositivos iOS

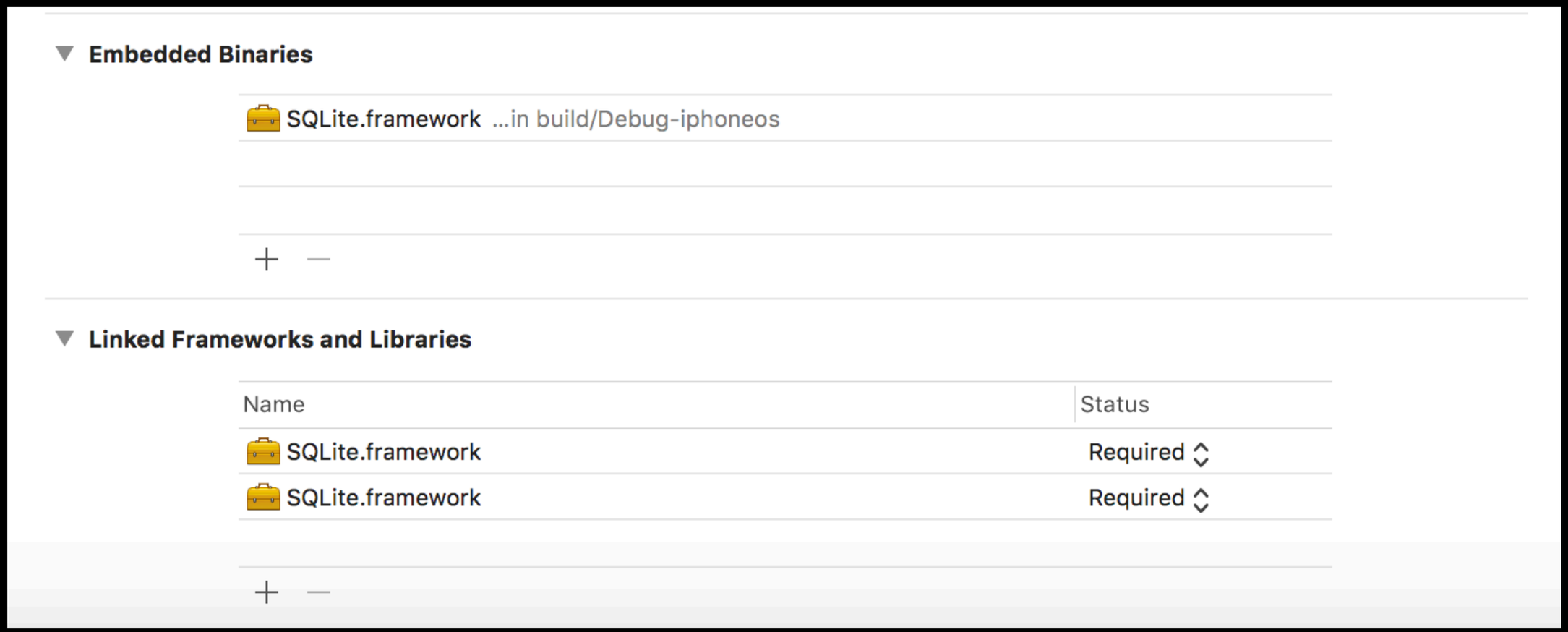

Veremos que fue agregado a nuestro proyecto dicho framework



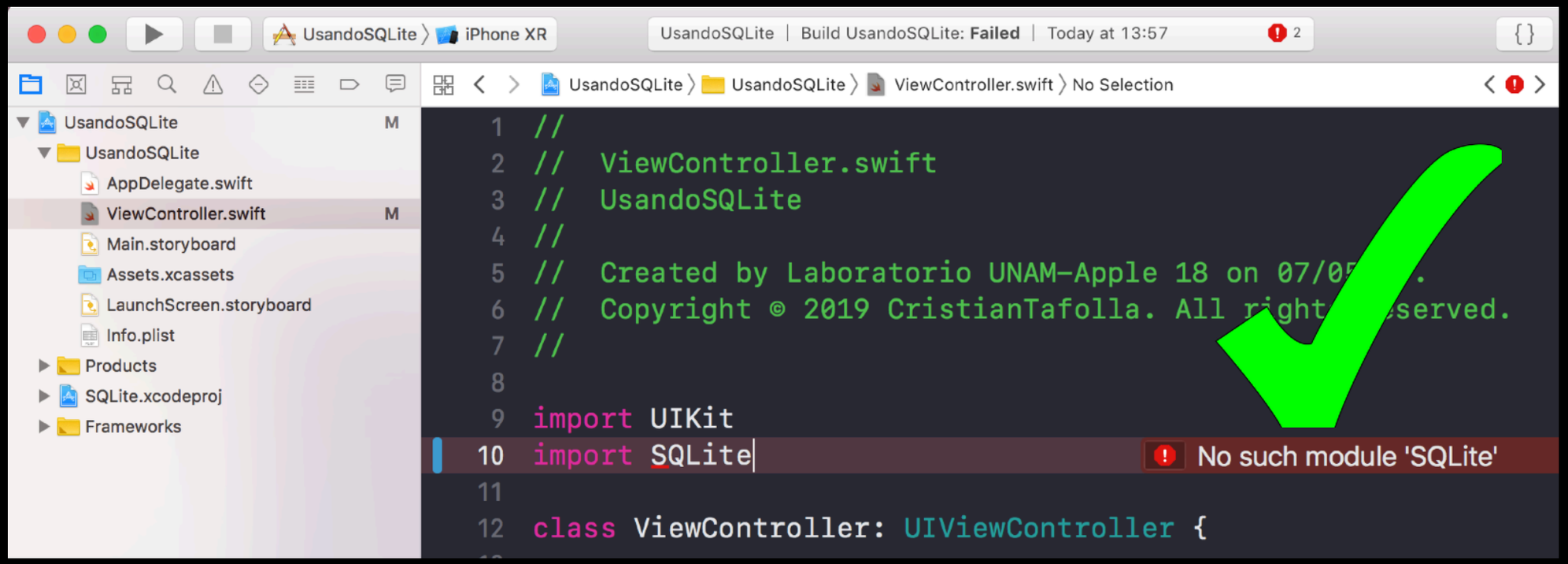

Ubiquemos el signo de más (+) dentro de “Embedded Binaries”



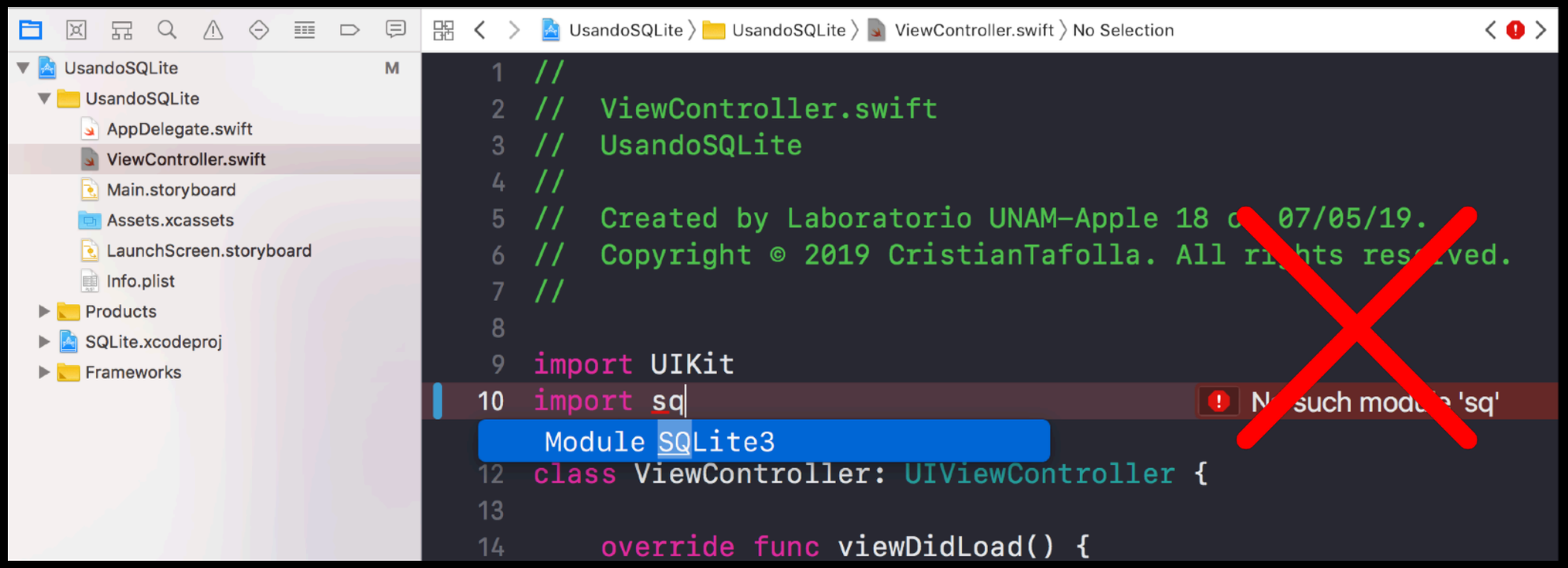

Seleccionemos el producto correspondiente

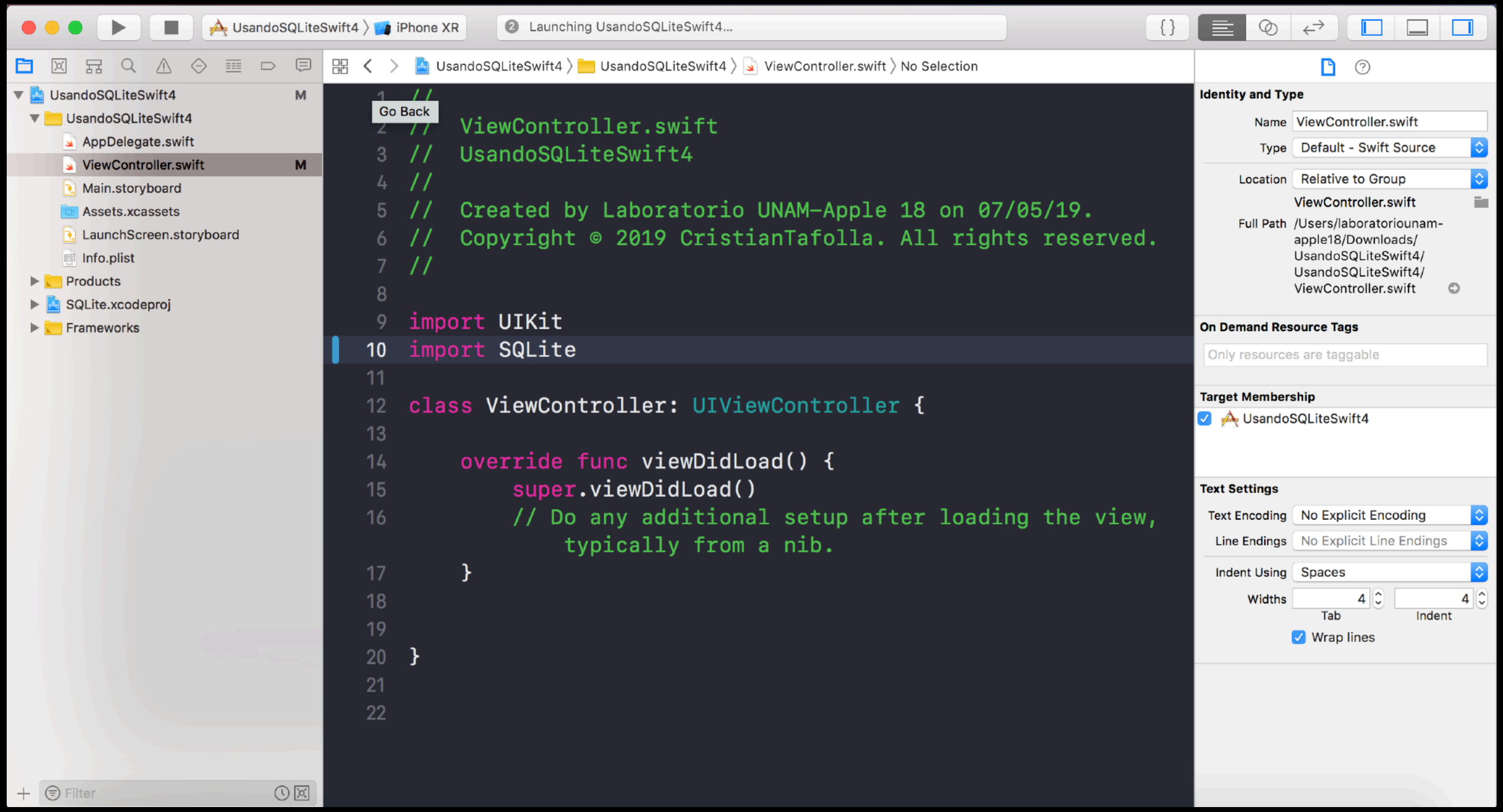



Veremos que el resultado es el siguiente: un componente en la parte superior y dos en la parte inferior




Dentro de ViewController.swift agregaremos el framework SQLite, evitando el autocompetado de SQLite3.






El error se quitará al compilar el proyecto, aunque no hayamos programado nada aún.

```
13
14      override func viewDidLoad() {
15          super.viewDidLoad()
16
17          do {
18              let documentDirectory = try
                    FileManager.default.url(for: .document
                    Directory, in: .userDomainMask,
                    appropriateFor: nil, create: true)
19          } catch {
20              print(error)
21          }
22      }
23
```

Vamos a crear un archivo URL para nuestra base de datos. Se utiliza la estructura try, do-catch. Esta se utiliza para administrar los errores. Un símil en Java es el do-catch. Cada función que represente un error potencial debe seguir esta estructura.

```
12  class ViewController: UIViewController {
13
14      override func viewDidLoad() {
15          super.viewDidLoad()
16
17          do {
18              let documentDirectory = try
                    FileManager.default.url(for: .documentDirect
                    ory, in: .userDomainMask, appropriateFor:
                    nil, create: true)
19              let fileUrl =
                    documentDirectory.appendingPathComponent("us
                    ers").appendingPathExtension("sqlite3")
20              let database = try
                    Connection(fileUrl.path)
21          } catch {
22              print(error)
23          }
24      }
```

Después hacemos una base de datos para los usuarios, utilizando la URL de la parte superior. Este tendrá de nombre “users” y de extensión “sqlite3", como lo sugiere la documentación de Github.

## Usage

```
import SQLite

let db = try Connection("path/to/db.sqlite3")
```

```
13
14      var database: Connection!
15
16      override func viewDidLoad() {
17          super.viewDidLoad()
18
19          do {
20              let documentDirectory = try
                    FileManager.default.url(for: .documentDirect
                    ory, in: .userDomainMask, appropriateFor:
                    nil, create: true)
21              let fileUrl =
                    documentDirectory.appendingPathComponent("us
                    ers").appendingPathExtension("sqlite3")
22              let database = try Connection(fileUrl.path)
23
24              self.database = database
25
26          } catch {
27              print(error)
28          }
```

Probamos la conexión con la base de datos.

```
13
14      var database: Connection!
15
16      override func viewDidLoad() {
17          super.viewDidLoad()
18
19          do {
20              let documentDirectory = try
                    FileManager.default.url(for: .documentDirect
                    ory, in: .userDomainMask, appropriateFor:
                    nil, create: true)
21              let fileUrl =
                    documentDirectory.appendingPathComponent("us
                    ers").appendingPathExtension("sqlite3")
22              let database = try Connection(fileUrl.path)
23
24              self.database = database
25
26          } catch {
27              print(error)
28          }
```

Crear una variable global, la cual se denomina database, esta será de tipo Connection!

```
13
14      var database: Connection!
15
16      override func viewDidLoad() {
17          super.viewDidLoad()
18
19          do {
20              let documentDirectory = try
                    FileManager.default.url(for: .documentDirect
                    ory, in: .userDomainMask, appropriateFor:
                    nil, create: true)
21              let fileUrl =
                    documentDirectory.appendingPathComponent("us
                    ers").appendingPathExtension("sqlite3")
22              let database = try Connection(fileUrl.path)
23
24              self.database = database
25
26          } catch {
27              print(error)
28          }
```

Posteriormente indicamos que será igual a la base de datos creada en la carga de nuestra aplicación.

```
16      let usersTable = Table("users")
17      let id = Expression<Int>("id")
18      let name = Expression<String>("name")
19      let email = Expression<String>("email")
20
```

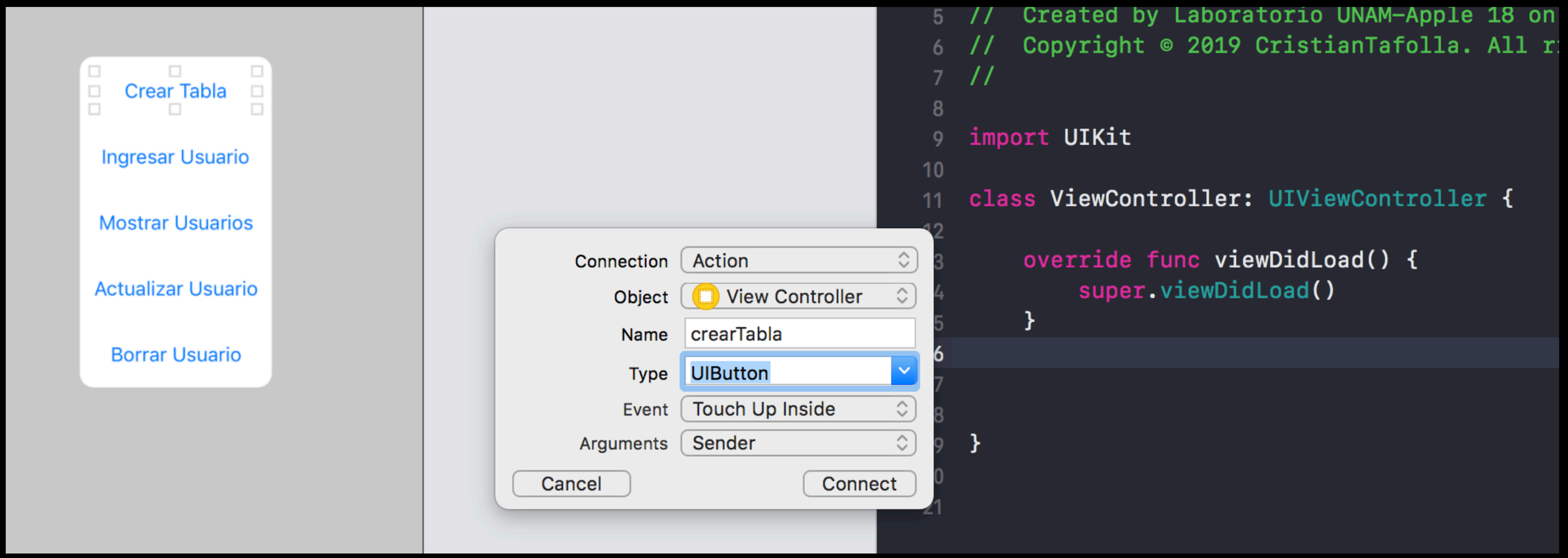
Declaro una constante correspondiente a la tabla de los usuarios.

```
16      let usersTable = Table("users")
17      let id = Expression<Int>("id")
18      let name = Expression<String>("name")
19      let email = Expression<String>("email")
20
```

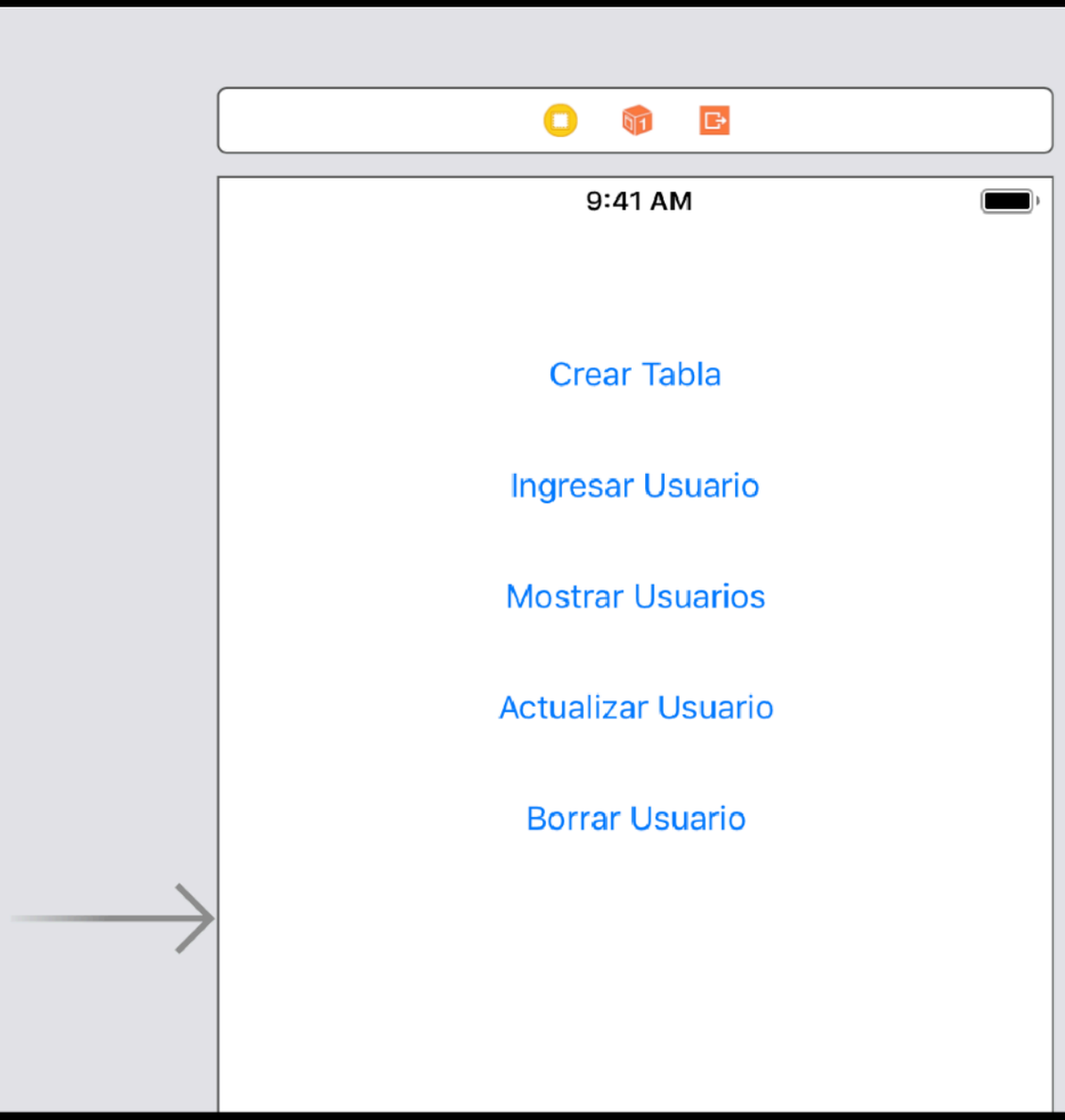
Después creamos los campos de nuestra BD, indicando que son expresiones, entre picoparéntesis ( < > ) el tipo de dicho campo y posteriormente el tipo de dicho campo.

# Tipos de dato

- Programación: Característica que define lo que un dato puede retener.
- Base de datos: Características que definen qué dato puede o no entrar en nuestro caso a una propiedad o atributo.




Dentro del Main.storyboard habrá que crear 5 botones como lo indica la diapositiva




```swift
class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    @IBAction func crearTabla(_ sender: UIButton) {
        print("Tabla creada")
    }

    @IBAction func ingresarUsuario(_ sender: UIButton) {
        print("Usuario dado de alta")
    }

    @IBAction func mostrarUsuarios(_ sender: UIButton) {
        print("Mostrando usuarios")
    }

    @IBAction func actualizarUsuario(_ sender: UIButton) {
        print("Actualizando usuario")
    }

    @IBAction func borrarUsuario(_ sender: UIButton) {
        print("Eliminando usuario")
    }

}
```

A cada uno de los botones hay que agregar un “Action”, con un breve mensaje

```swift
    @IBAction func ingresarUsuario(_ sender: UIButton) {
        print("Usuario registrado")
        let alert = UIAlertController(title: "Da de alta al usuario", message: nil,
            preferredStyle: .alert)
        alert.addTextField{ (tf) in tf.placeholder = "Name" }
        alert.addTextField{ (tf) in tf.placeholder = "Email" }
        let action = UIAlertAction(title: "Alta", style: .default){ (_) in
            guard let name = alert.textFields?.first?.text,
                let email = alert.textFields?.last?.text
                else { return }
            print(name)
            print(email)
        }
        alert.addAction(action)
        present(alert, animated: true, completion: nil)
    }
```

Dentro del “Action” ingresarUsuario, habrá que agregar este segmento de código…



…para que aparezca el siguiente recuadro emergente.

```
    @IBAction func crearTabla(_ sender: UIButton) {
        print("Tabla creada")

        let createTable = self.usersTable.create{ (table) in
            table.column(self.id, primaryKey: true)
            table.column(self.name)
            table.column(self.email, unique: true)
        }

        do{
            try self.database.run(createTable)
            print("La tabla se creó con éxito")
        } catch {
            print(error)
        }
    }
```

Dentro del “Action” crearTabla, habrá que agregar este segmento de código para crear la tabla con la estructura especificada al inicio.