

cit 

Hilton Pintor

Desenvolvedor (iOS/tvOS/watchOS)



hiltonpintor@gmail.com

// Aula 08

/*

Como fazer unwind pelo
código?

*/

// algoritmo

1. Criar action no **VC de destino**
 1. @IBAction func unwindToX(segue:)
2. Criar segue do VC de **origem** para o **Exit**
 - 2.1. dar um **identificador** para a segue
3. **Chamar** a segue
 1. performSegue(withIdentifier: sender:)

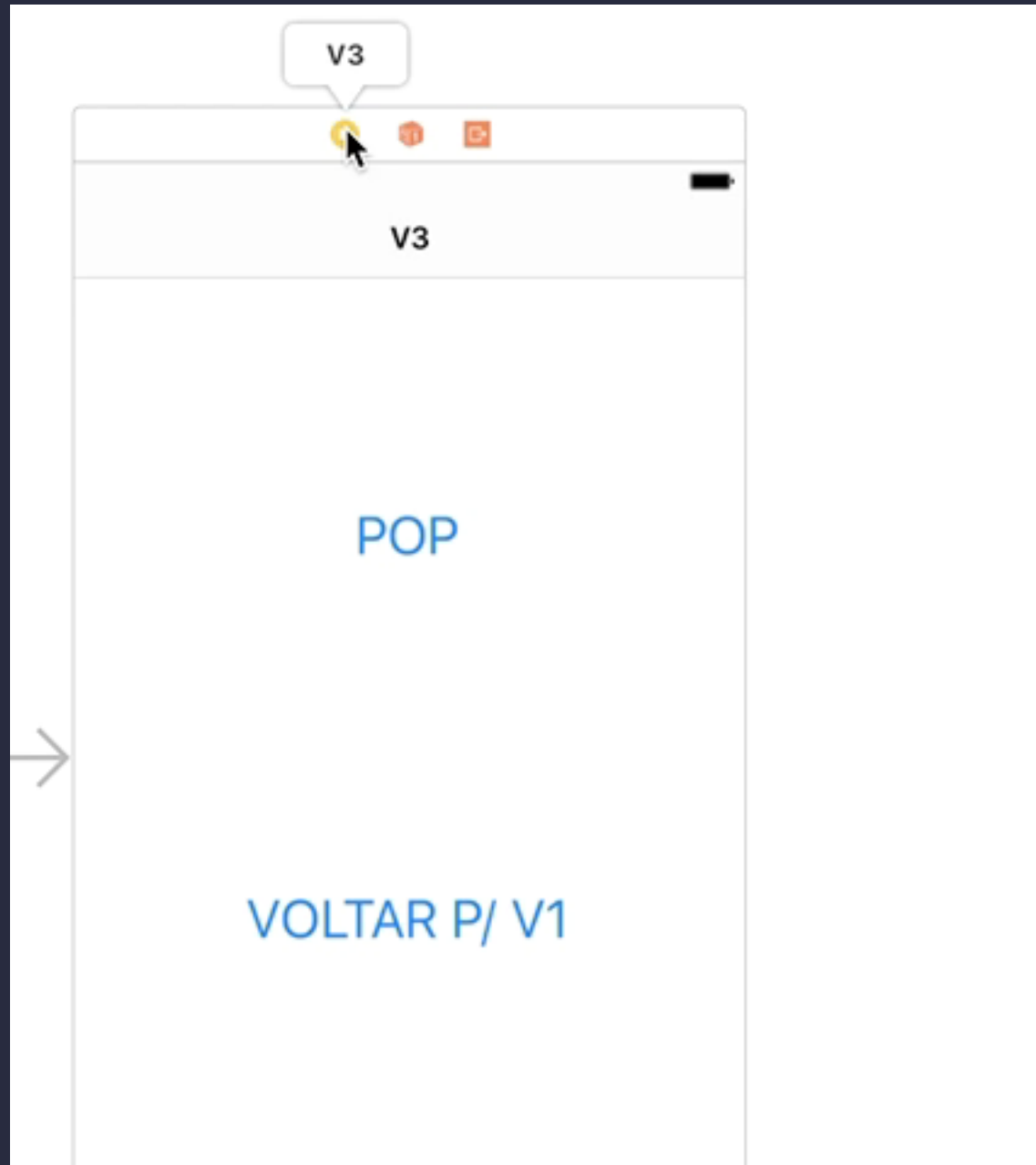
// algoritmo



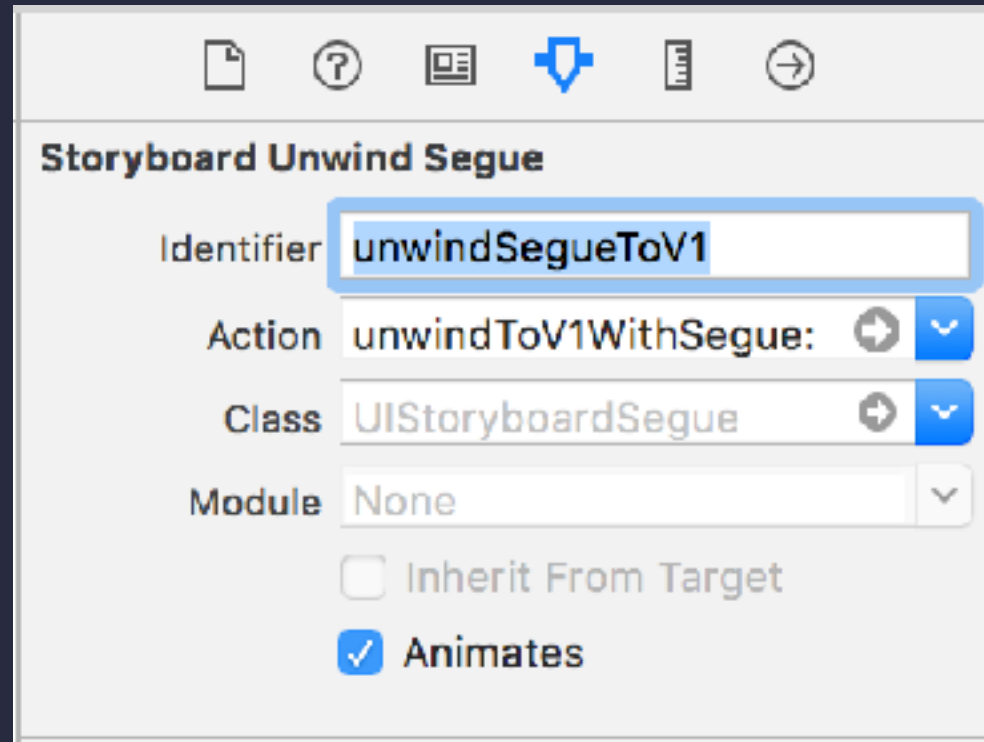
```
// 1. no VC1
```

```
@IBAction func unwindToVC1(sender: UIStoryboardSegue)  
{ ... }
```

// 2. criando segue



// 2. nomeando segue



The image shows the 'Storyboard Unwind Segue' configuration panel in Xcode. The panel has a title bar with icons for file operations. Below the title, the 'Identifier' field is highlighted with a blue selection box and contains the text 'unwindSegueToV1'. The 'Action' field is set to 'unwindToV1WithSegue:' with a dropdown arrow. The 'Class' field is set to 'UIStoryboardSegue' with a dropdown arrow. The 'Module' field is set to 'None' with a dropdown arrow. At the bottom, there are two checkboxes: 'Inherit From Target' (unchecked) and 'Animates' (checked).

Storyboard Unwind Segue

Identifier

Action

Class

Module

☐ Inherit From Target

☒ Animates

// 3. chamar segue

```
@IBAction func voltarV1(_ sender: Any) {  
    self.performSegue(withIdentifier: "unwindSegueToV1", sender: self)  
}
```

// 3. chamar segue



```
// extra. pop
```

```
@IBAction func popBtn(_ sender: Any) {  
    /* se fosse modal:  
    dismiss(animated: true, completion: nil)*/  
    self.navigationController?.popViewController(animated: true)  
}
```

// extra. pop



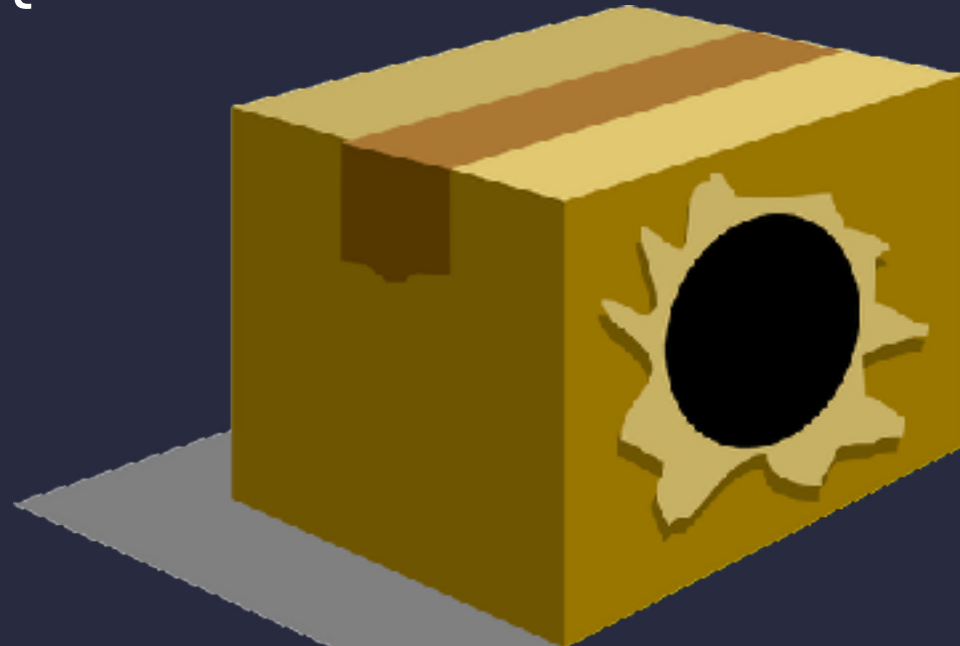
// Persistência

// Core Data

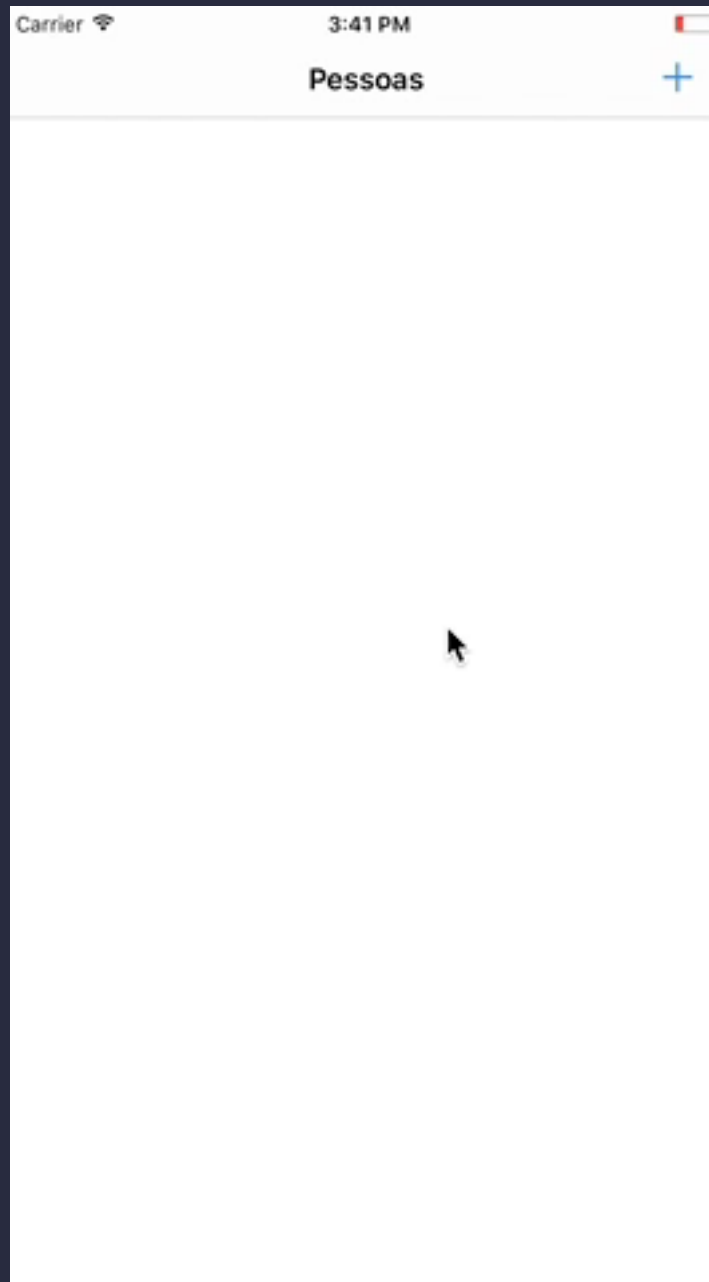
1. Framework de Persistência local
2. Usa a memória do iPhone
3. Usa SQLite
4. Modelar os dados

// Core Data

- 1.NSPersistentContainer
- 2.NSManagedObjectContext
- 3.NSManagedObject



// Core Data



// algoritmo geral

1. Criar projeto com **Core Data**
2. Criar **Entity** e dar **Attributes**
3. Acessar **App Delegate** para pegar o **NSManagedObjectContext**
4. Realizar **fetch** dos dados salvos
5. Quando adicionar novos dados, **save**
6. Quando remover dados, **delete**

/*

1. Criar projeto com **Core Data**

*/

// 1. Criar Projeto

Choose options for your new project:

Product Name:

Team:

Organization Name:

Organization Identifier:

Bundle Identifier:

Language:

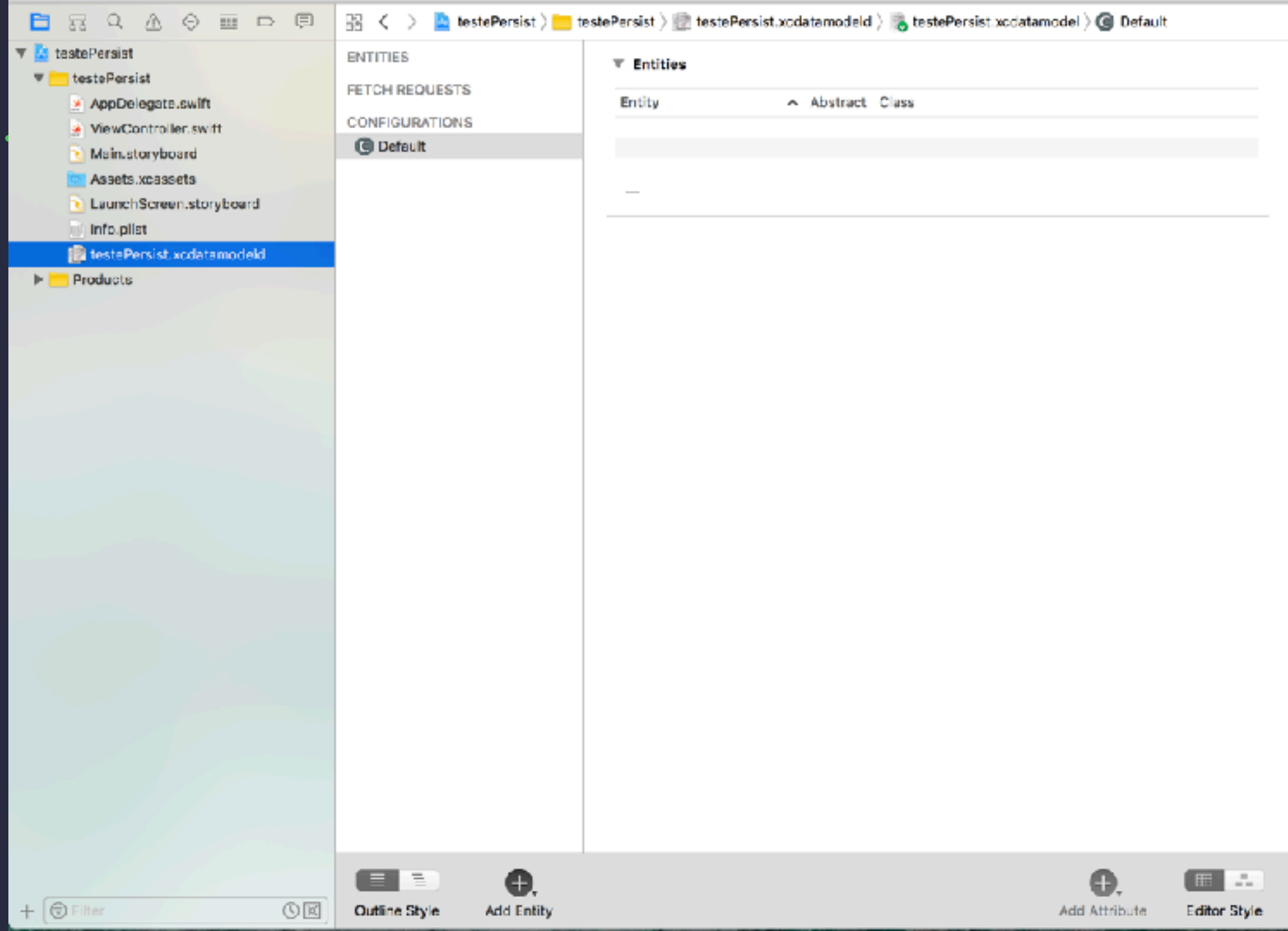
Devices:

☒ Use Core Data

☐ Include Unit Tests

☐ Include UI Tests

// 1.



//



```
45 // MARK: - Core Data stack
46
47 lazy var persistentContainer: NSPersistentContainer = {
48     /*
49     The persistent container for the application. This implementation
50     creates and returns a container, having loaded the store for the
51     application to it. This property is optional since there are legitimate
52     error conditions that could cause the creation of the store to fail.
53     */
54     let container = NSPersistentContainer(name: "testePersist")
55     container.loadPersistentStores(completionHandler: { (storeDescription,
56         error) in
57         if let error = error as NSError? {
58             // Replace this implementation with code to handle the error
59             // appropriately.
60             // fatalError() causes the application to generate a crash log
61             // and terminate. You should not use this function in a shipping
62             // application, although it may be useful during development.
63
64             /*
65             Typical reasons for an error here include:
66             * The parent directory does not exist, cannot be created, or
67             disallows writing.
68             * The persistent store is not accessible, due to permissions or
69             data protection when the device is locked.
70             * The device is out of space.
71             * The store could not be migrated to the current model version.
72             Check the error message to determine what the actual problem
73             was.
74             */
75             fatalError("Unresolved error \(error), \(error.userInfo)")
76         }
77     })
78     return container
79 }()
80
81 // MARK: - Core Data Saving support
82
83 func saveContext () {
84     let context = persistentContainer.viewContext
85     if context.hasChanges {
86         do {
87             try context.save()
88         } catch {
89             // Replace this implementation with code to handle the error
90             // appropriately.
91             // fatalError() causes the application to generate a crash log
92             // and terminate. You should not use this function in a shipping
93             // application, although it may be useful during development.
94         }
95     }
96 }
```

/*

2. Criar **Entity** e dar **Attributes**

*/

// 2. criar

ENTITIES

E

Pessoa

FETCH REQUESTS

CONFIGURATIONS

C

Default

▼ Attributes

Attribute ^	Type
+ -	

▼ Relationships

Relationship ^	Destination	Inverse
+ -		

▼ Fetched Properties

Fetches Property ^	Predicate
+ -	

≡ ≡

Outline Style

+

Add Entity

+

Add Attribute

≡ ≡

Editor Style

// 2. dar a

ENTITIES

Passoa

FETCH REQUESTS

CONFIGURATIONS

Default

Attributes

Attribute ^	Type
U nome	✓ Undefined
	Integer 16
	Integer 32
	Integer 64
	Decimal
	Double
	Float
	String
	Boolean
	Date
	Binary Data
	Transformable

+ -

Relationships

Relationship ^	
	Inverse

+ -

Fetches Properties

Fetches Property ^	Predicate

+ -

Outline Style

Add Entity

Add Attribute

Editor Style

```
/*
```

3. Acessar **App Delegate**
para pegar o
NSManagedObjectContext

```
*/
```

// 3. Acessar app delegate e context

```
class ViewController: UIViewController {  
    var appDelegate: AppDelegate?  
    var managedContext: NSManagedObjectContext?  
}
```

// 3. Acessar app delegate e context

```
class ViewController: UIViewController {  
    var appDelegate: AppDelegate?  
    var managedContext: NSManagedObjectContext?  
  
    override func viewDidLoad() {  
        super.viewDidLoad()  
        // ...  
  
        self.appDelegate = UIApplication.shared.delegate as? AppDelegate  
  
        self.managedContext = self.appDelegate?.persistentContainer.viewContext  
    }  
}
```

/*

4. Realizar **fetch** dos dados salvos

*/

// 4. fetch

```
class ViewController: UIViewController {  
    var appDelegate: AppDelegate?  
    var managedContext: NSManagedObjectContext?  
    var pessoas: [NSManagedObject] = []  
  
    override func viewWillAppear(_ animated: Bool) {  
        super.viewWillAppear(animated)  
  
        let fetchRequest = NSFetchRequest<NSManagedObject>(entityName: "Pessoa")  
  
        do {  
            try self.pessoas = (self.managedContext?.fetch(fetchRequest))!  
        } catch let error as NSError {  
            print("erro na hora de pedir. \(error), \(error.userInfo)")  
        }  
    }  
}
```

/*

5. Quando adicionar
novos dados, **save**

*/

// 5. save

```
class ViewController: UIViewController {  
    var appDelegate: AppDelegate?  
    var managedContext: NSManagedObjectContext?  
    var pessoas: [NSManagedObject] = []  
  
    func save(novoNome: String) {  
        let entity = NSEntityDescription.entity(forEntityName: "Pessoa", in: managedContext!)  
  
        let pessoa = NSManagedObject(entity: entity!, insertInto: managedContext)  
  
        pessoa.setValue(novoNome, forKey: "nome")  
  
        do {  
            try managedContext?.save()  
            self.pessoas.append(pessoa)  
        } catch let error as NSError {  
            print("erro na hora de salvar. \(error), \(error.userInfo)")  
        }  
    }  
}
```


/*

5. Quando remover
dados, **delete**

*/

```

extension ViewController: UITableViewDataSource {
// 5. save
    func tableView(_ tableView: UITableView,
                   commit editingStyle: UITableViewCellEditingStyle,
                   forRowAt indexPath: IndexPath) {

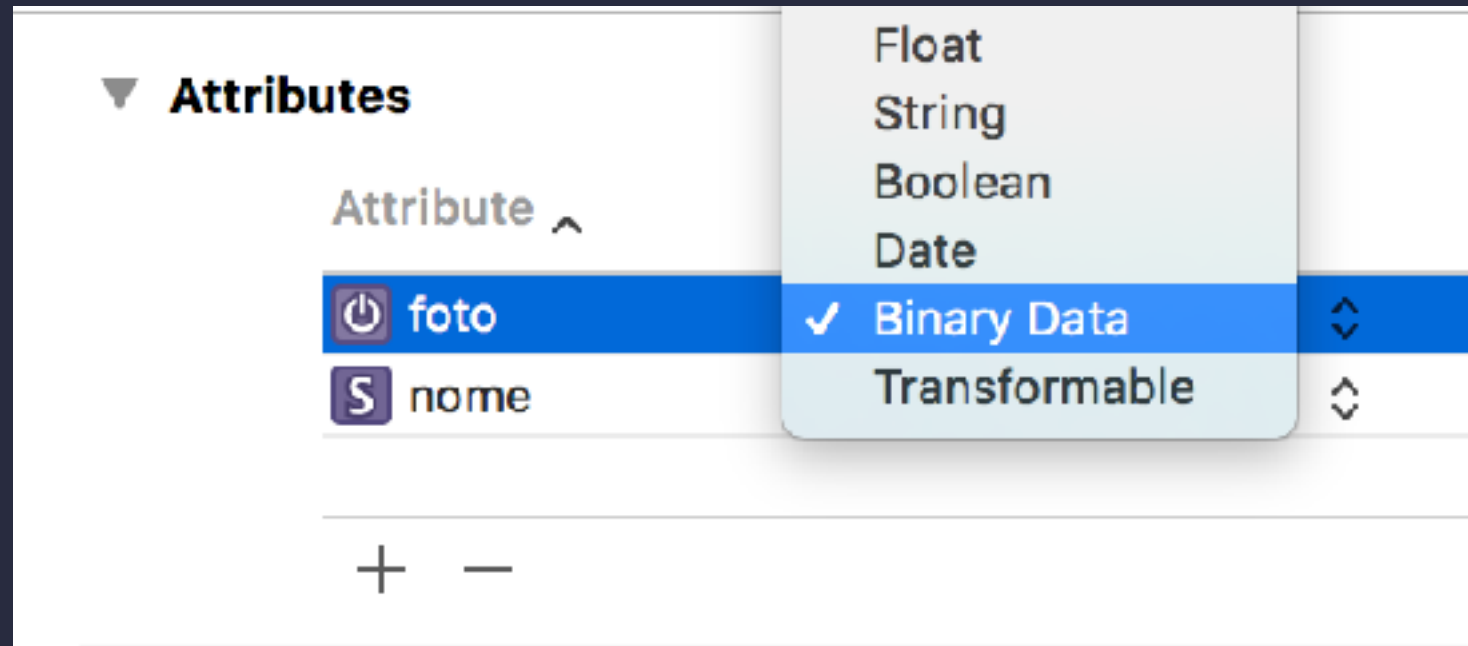
        if editingStyle == .delete {
            let pessoa = self.pessoas[indexPath.row]
            self.managedContext?.delete(pessoa)
            self.appDelegate?.saveContext()

            let fetchRequest = NSFetchRequest<NSManagedObject>(entityName: "Pessoa")

            do {
                try self.pessoas = (self.managedContext?.fetch(fetchRequest))!
                tableView.reloadData()
            } catch {
                print("Fetching Failed")
            }
        }
    }
}

```

// persistindo Imagens



// save Imagens

```
func save(novoNome: String) {  
    let entity = NSEntityDescription.entity(forEntityName: "Pessoa", in: managedContext!)  
  
    let pessoa = NSManagedObject(entity: entity!, insertInto: managedContext)  
  
    let img = UIImage(named: "diego")!  
    let imgData = UIImageJPEGRepresentation(img, 1)  
  
    pessoa.setValue(imgData, forKey: "foto")  
  
    pessoa.setValue(novoNome, forKey: "nome")  
  
    do {  
        try managedContext?.save()  
        self.pessoas.append(pessoa)  
    } catch let error as NSError {  
        print("erro na hora de salvar. \(error), \(error.userInfo)")  
    }  
}
```

// fetch Imagens

```
func tableView(_ tableView: UITableView,
               cellForRowAt indexPath: IndexPath) -> UITableViewCell {

    let cell = tableView.dequeueReusableCell(withIdentifier: "Cell",
                                           for: indexPath)

    let pessoa = pessoas[indexPath.row]

    cell.textLabel?.text = pessoa.value(forKey: "nome") as? String

    guard let imgData = pessoa.value(forKey: "foto") as? Data,
          let image = UIImage(data: imgData) else {
        return cell
    }

    cell.imageView?.image = image

    return cell
}
```

/*

Como adicionar Core
Data a um projeto
existente?

*/

// algoritmo

1. Adicionar arquivo **Data Model** (.xcdatamodeld)
2. Adicionar código ao **App Delegate**
 - 2.1. Mudar `let container = NSPersistentContainer(name: "nomeDoArquivo")`
3. Adaptar o código para usar Core Data

Choose a template for your new file:

iOS

watchOS

tvOS

macOS

Filter

Core Data



Data Model



Mapping Model

Apple Watch



Storyboard

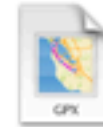


WatchKit Settings
Bundle



Notification
Simulation File

Resource

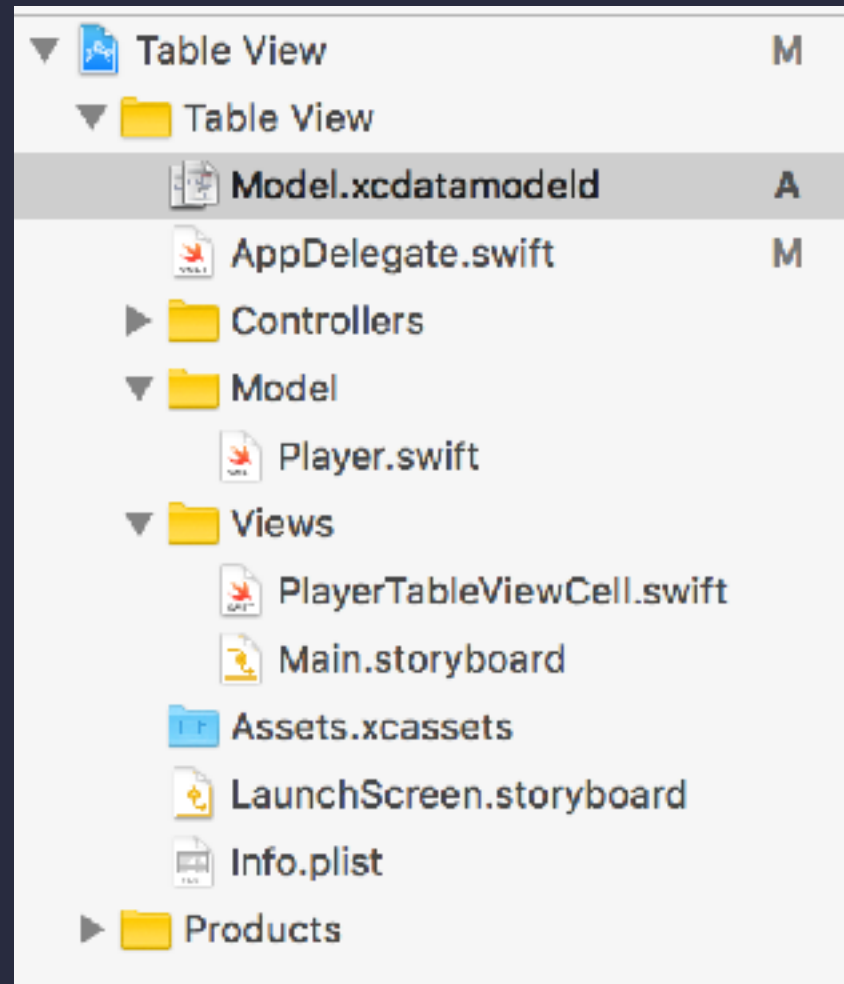


Cancel

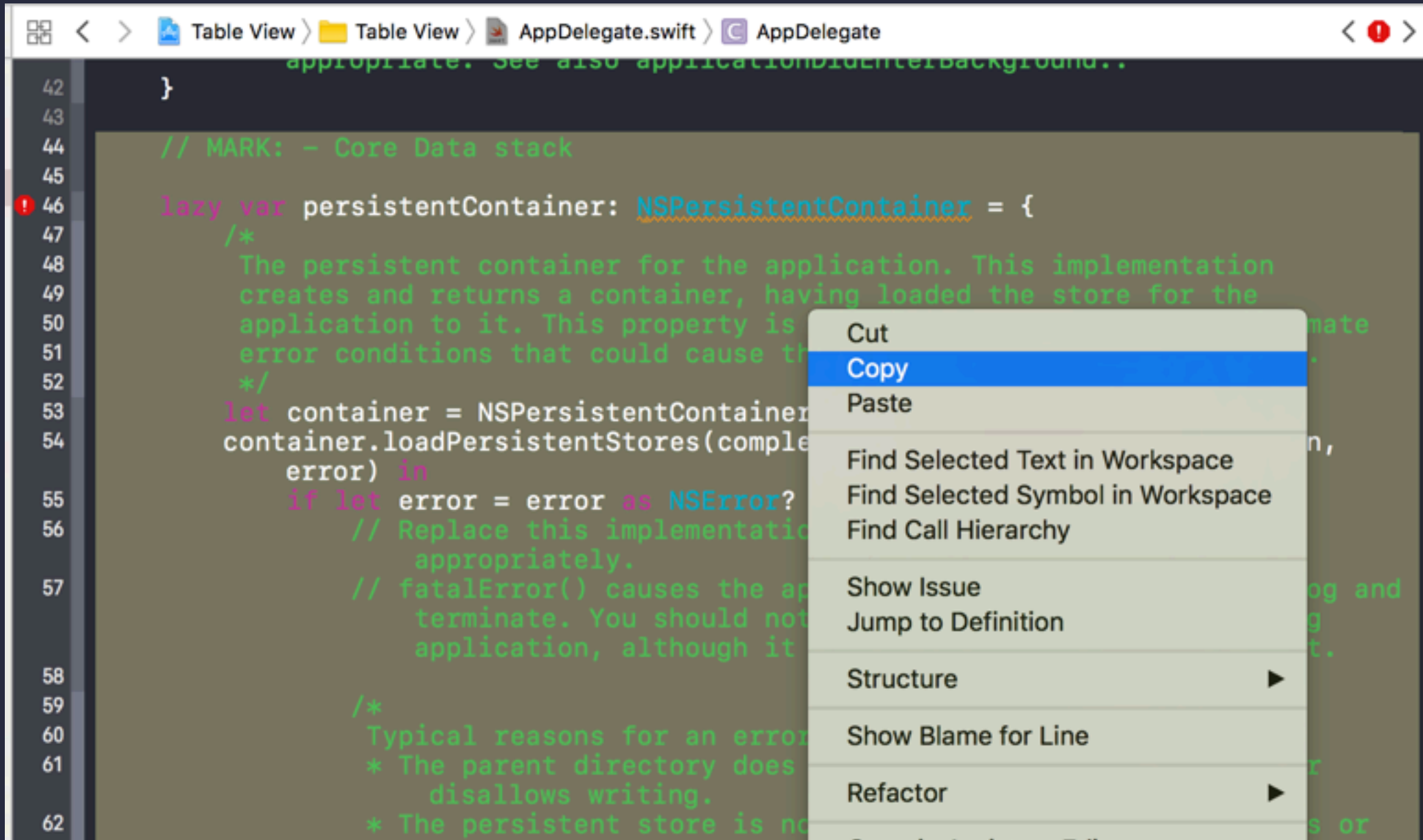
Previous

Next

// 1. Adicionar arquivo .xcdatamodeld



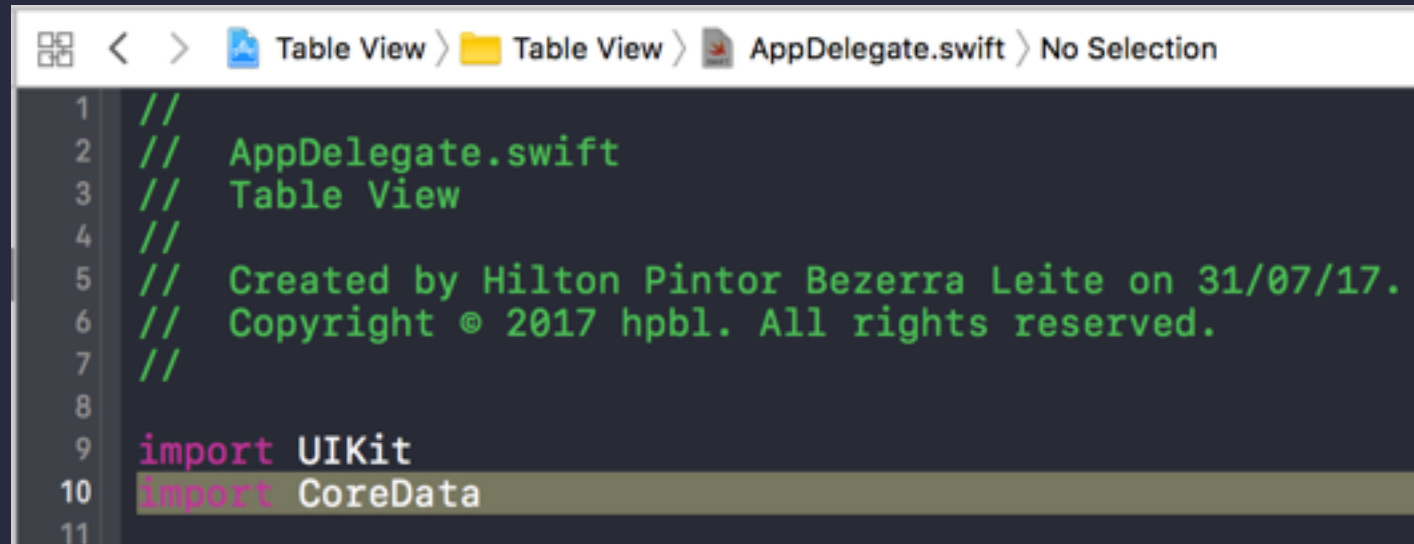
// 2. Adicionar código ao App Delegate



The screenshot shows the Xcode IDE with the AppDelegate.swift file open. The file path in the top bar is 'Table View > Table View > AppDelegate.swift'. The code is in Swift and includes a comment '// MARK: - Core Data stack'. On line 46, there is a declaration of a lazy variable 'persistentContainer' of type 'NSPersistentContainer'. A context menu is open over this line, with 'Copy' selected. The menu options include Cut, Copy, Paste, Find Selected Text in Workspace, Find Selected Symbol in Workspace, Find Call Hierarchy, Show Issue, Jump to Definition, Structure, Show Blame for Line, and Refactor. The code on line 46 is: `lazy var persistentContainer: NSPersistentContainer = {`

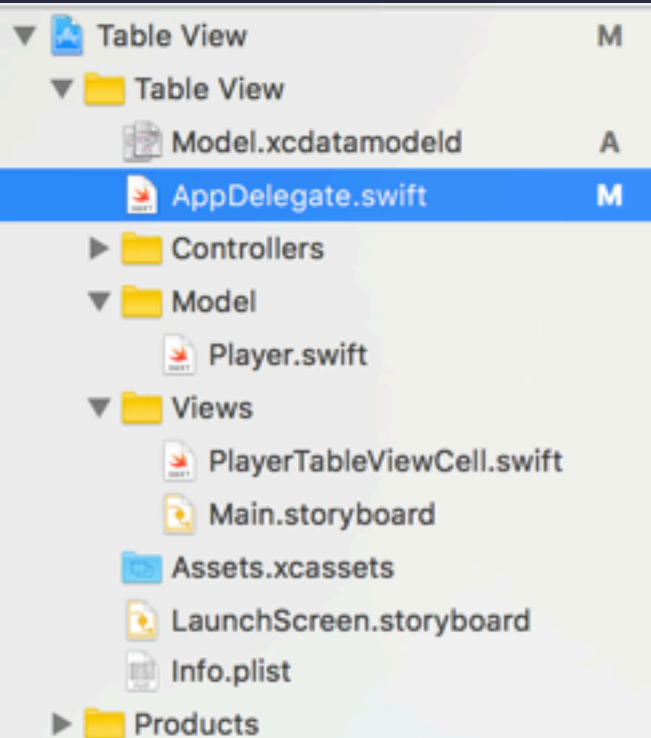
```
42     }
43
44     // MARK: - Core Data stack
45
46     lazy var persistentContainer: NSPersistentContainer = {
47         /*
48          The persistent container for the application. This implementation
49          creates and returns a container, having loaded the store for the
50          application to it. This property is lazy so it only loads the store
51          the first time it is used. It also lazily creates the store and logs
52          any error conditions that could cause the store to fail to load.
53          */
54         let container = NSPersistentContainer(name: "Table View")
55         container.loadPersistentStores(completionHandler: { (storeDescription, error) in
56             if let error = error as NSError? {
57                 // Replace this implementation with code to produce the error message
58                 // that was produced. This example shows how to format the error
59                 // to include the store's name and the error's details, but it
60                 // is not required.
61                 // Example: "The store could not be loaded. (error=1) The parent directory does
62                 // not allow writing."
63                 // * The parent directory does not allow writing.
64                 // * The persistent store is not a valid directory.
```

// 2. Importar Core Data



```
1 //  
2 // AppDelegate.swift  
3 // Table View  
4 //  
5 // Created by Hilton Pintor Bezerra Leite on 31/07/17.  
6 // Copyright © 2017 hpbl. All rights reserved.  
7 //  
8  
9 import UIKit  
10 import CoreData  
11
```

// 3. Mudar nome do container



46
47
48
49
50
51
52
53
54
55
56
57
58
59
60

```
lazy var persistentContainer: NSPersistentContainer = {  
    /*  
    The persistent container for the application. This implementation  
    creates and returns a container, having loaded the store for the  
    application to it. This property is optional since there are legitimate  
    error conditions that could cause the creation of the store to fail.  
    */  
    let container = NSPersistentContainer(name: "Model")  
    container.loadPersistentStores(completionHandler: { (storeDescription,  
        error) in  
        if let error = error as NSError? {  
            // Replace this implementation with code to handle the error  
            // appropriately.  
            // fatalError() causes the application to generate a crash log  
            // terminate. You should not use this function in a shipping  
            // application, although it may be useful during development  
        }  
    })  
}
```

// Exercício

// Exercício 14

Lista de coisas III

1. Adicione persistência local usando **Core Data** ao seu app
2. Dados **adicionados** devem ser mantidos
3. **Remoções** devem ser mantidas

// Extra

4. **Edições** devem ser mantidas

DÚVIDAS

