
citi

Hilton **Pintor**

Desenvolvedor (iOS/tvOS/watchOS)

hiltonpintor@gmail.com

citi

// Aula 08

// Dúvidas da Aula 07

/*
Como fazer unwind pelo código?
*/

citi

// 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

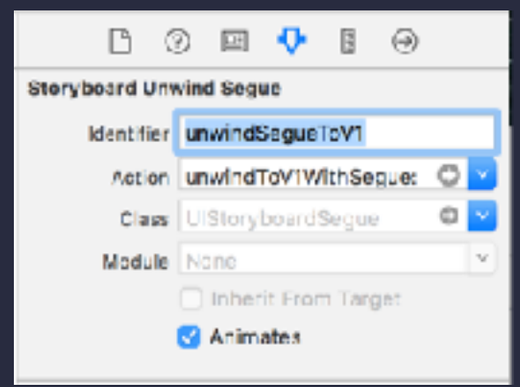

Navigation Controller
V1
IR PARA V2
V2
IR PARA V3
V3
POP
VOLTAR PV V1

```
// 1. no VC1

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

// 2. criando segue

// 2. criando segue

// 2. nomeando segue



```
// 3. chamar segue

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

// 3. chamar segue

// 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

// extra. pop

// Persistência

// Core Data

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

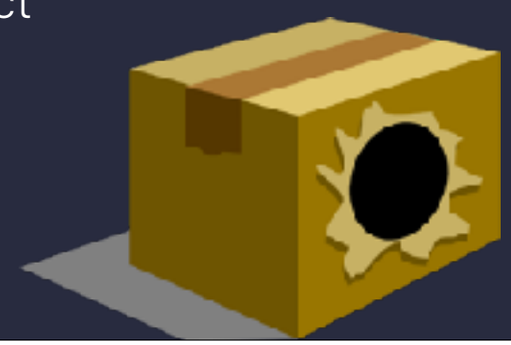
cit

// Core Data

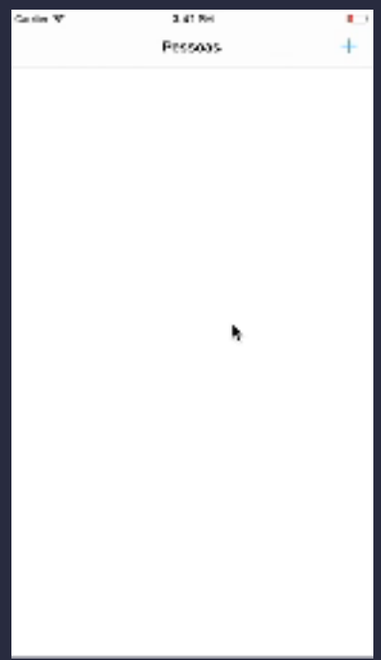
1. NSPersistentContainer
2. NSManagedObjectContext
3. NSManagedObject

// Core Data

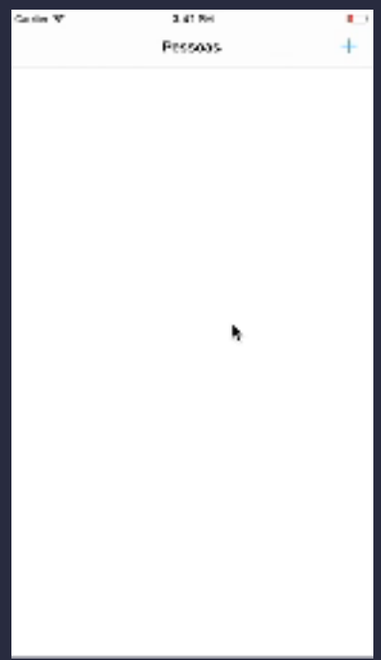
1. NSPersistentContainer
2. NSManagedObjectContext
3. NSManagedObject



// Core Data



// Core Data

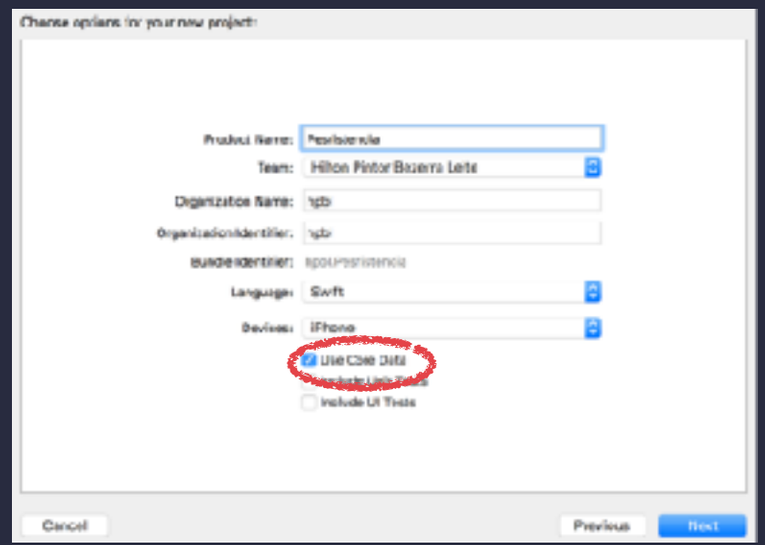


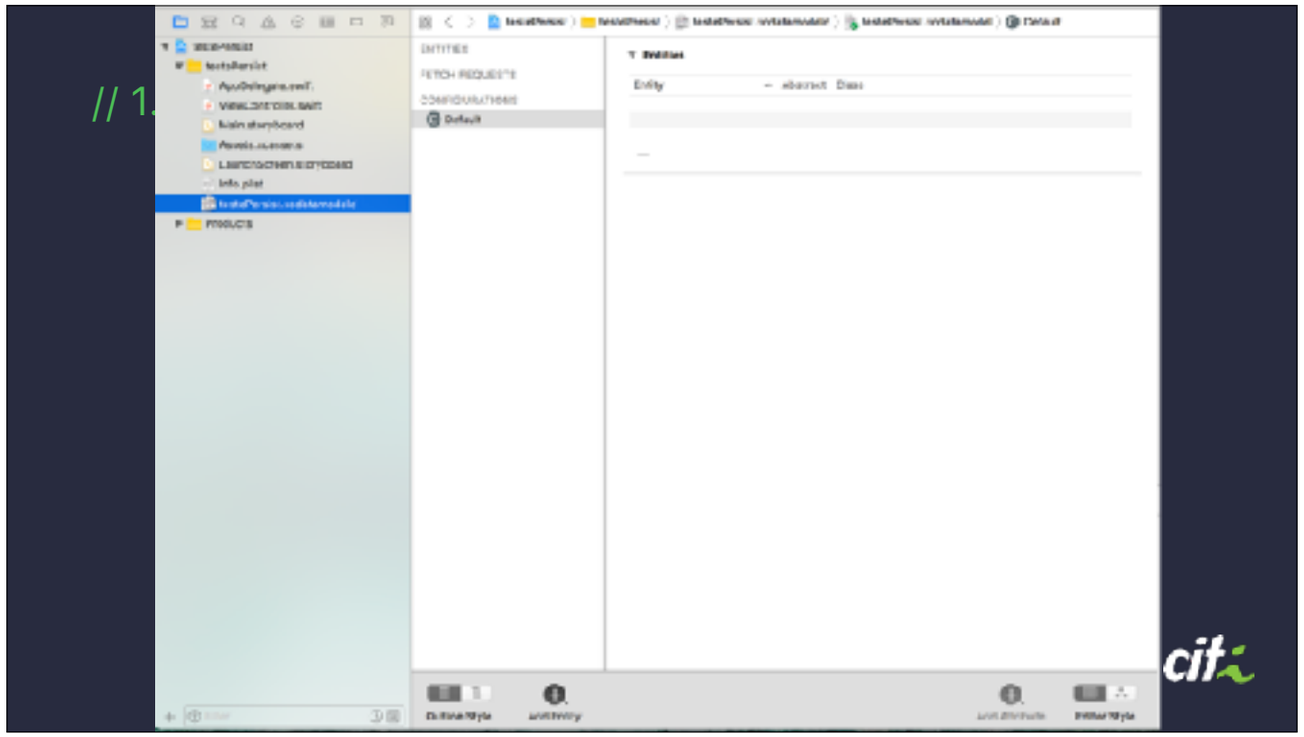
// algoritmo geral

1. Criar projeto com **Core Data**
2. Criar **Entity** e dar **Attributes**
3. Instanciar **App Delegate** para pegar o NSManagedContext
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



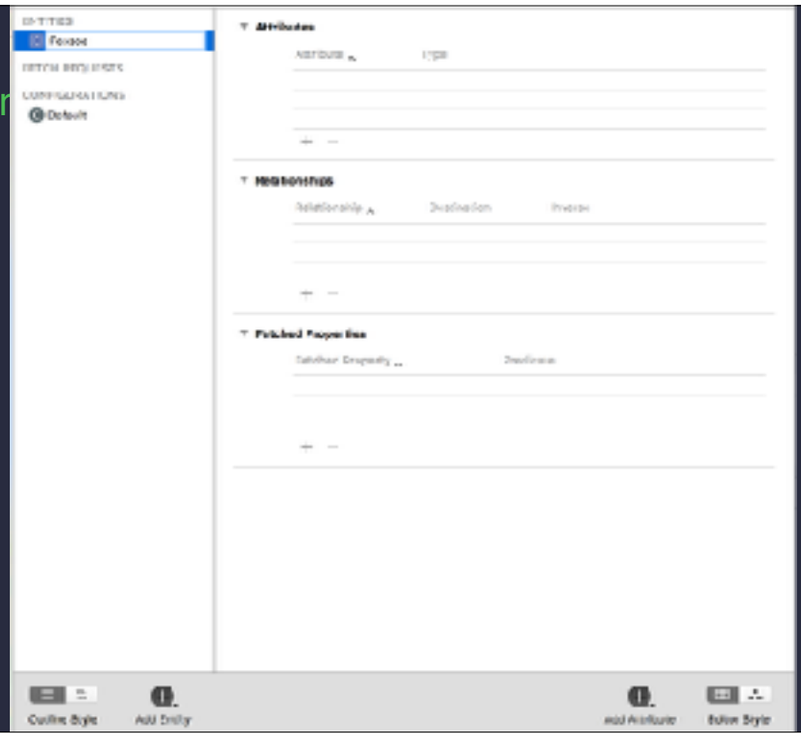

// 1.
ENTITIES
FETCH REQUESTS
CONFIGURATIONS
Default
Entity

//

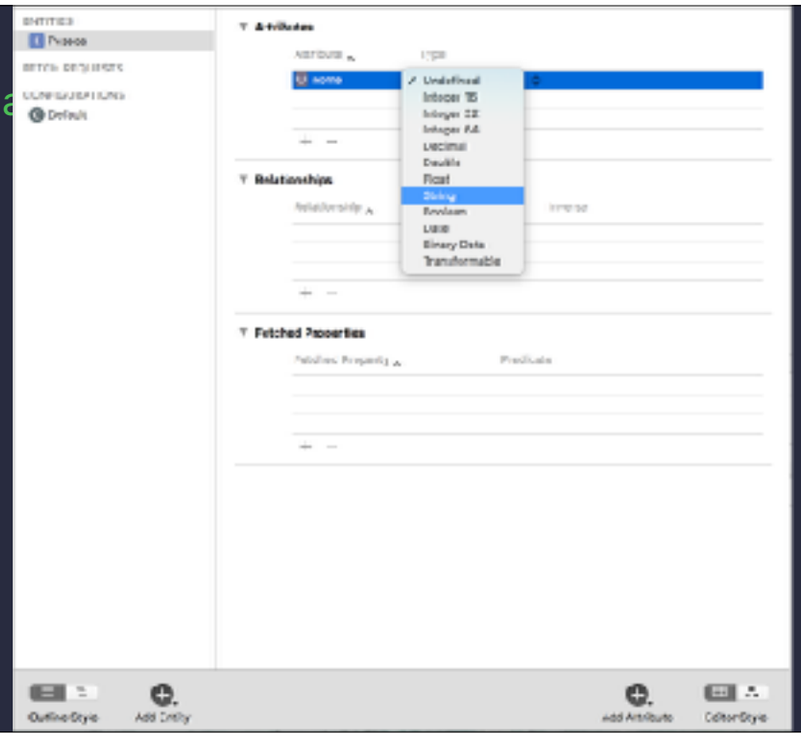
```
// MARK: - Core Data stack

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: "TestePersist")
    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
                and terminate. You should not use this function in a shipping
                application, although it may be useful during development.

            /*
             Typical reasons for an error here include:
             * The parent directory does not exist, cannot be created, or
                 disallows writing.
             * The persistent store is not accessible, due to permissions or
                 data protection when the device is locked.
             * The device is out of space.
             * The store could not be migrated to the current model version.
             Check the error message to determine what the actual problem
                 was.
             */
            fatalError("Unresolved error \(error), \(error.userInfo)")
        }
    })
    return container
}()

// MARK: - Core Data Saving support

func saveContext() {
    let context = persistentContainer.viewContext
    if context.hasChanges {
        do {
            try context.save()
```

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

// 2. criar



// 2. dar a


Person
Default
Attributes
Relationships
Fetched Properties
Undefined
Integer 16
Integer 32
Integer 64
Decimal
Double
Float
String
Boolean
Date
Binary Data
Transformable
Outline Style
Add Entity
Add Attribute
Editor Style

/*
3. Instanciar **App Delegate** para pegar o NSManagedContext
*/

citi

```
// 3. instanciar app delegate e context


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

// 3. instanciar 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 = 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)")
        }
    }
}
```

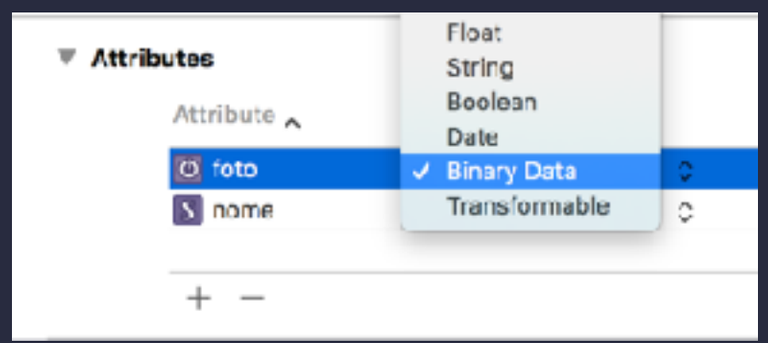
/*
5. Quando adicionar novos dados, **save**
*/

```swift
// 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 = #imageLiteral(resourceName: "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
}
```

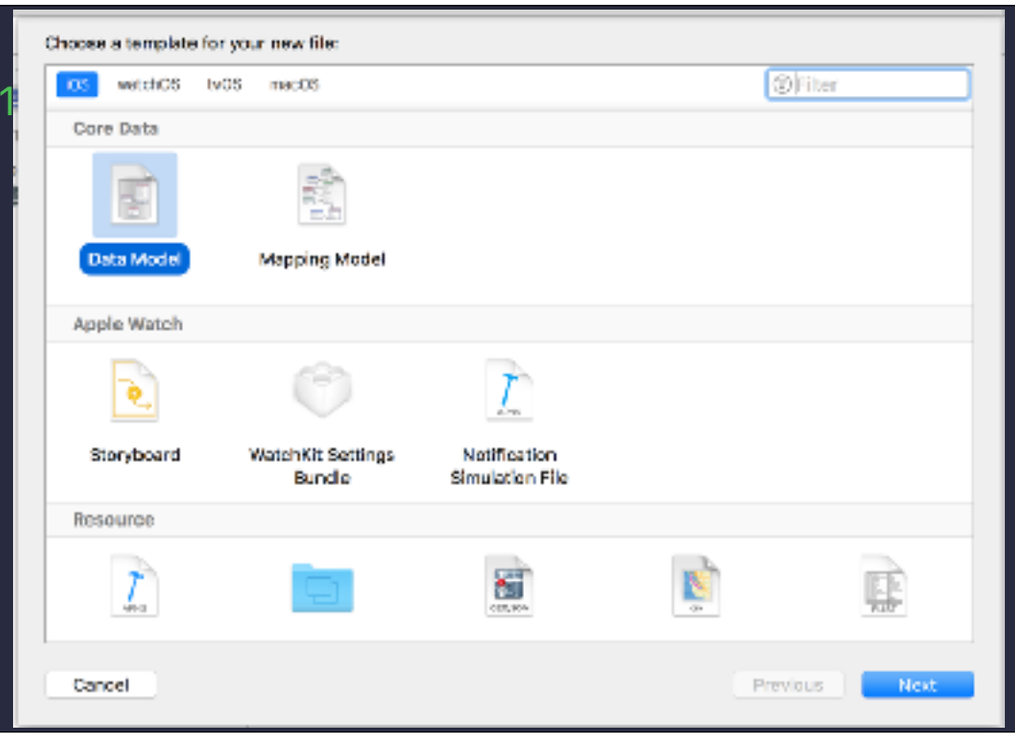
/*
Como adicionar Core Data a um projeto existente?
*/

citi

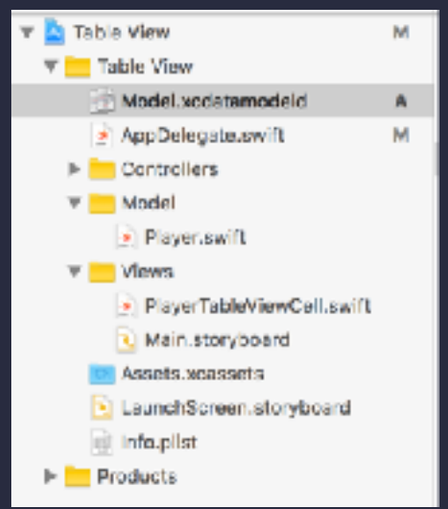
// algoritmo

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

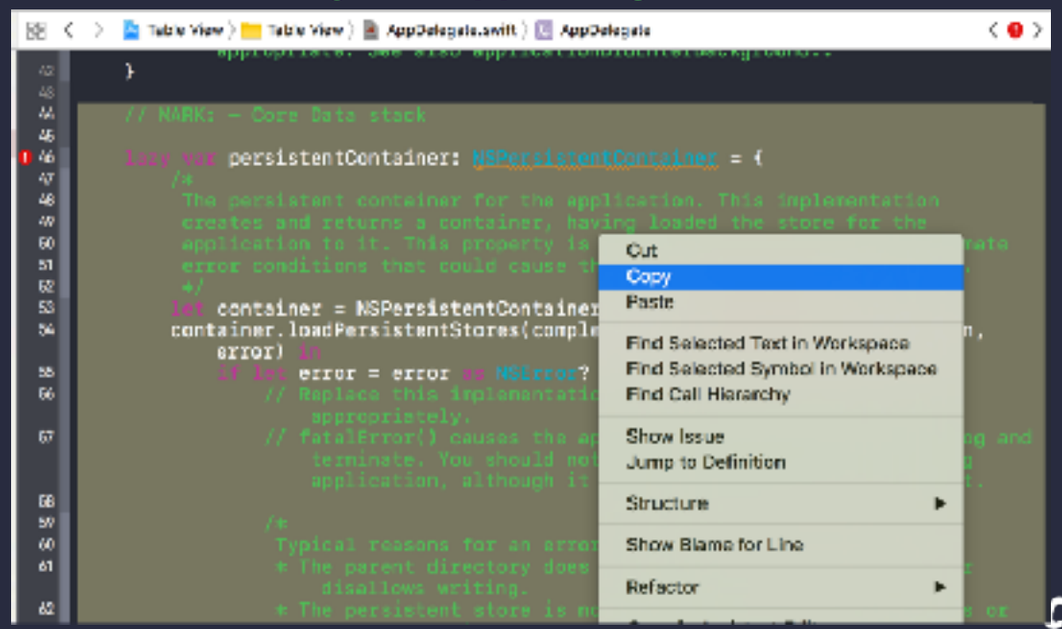
// 1


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**



// 2. Importar Core Data

Table View 〉 Table View 〉 AppDelegate.swift 〉 No Selection

```
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

```
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 legitim
     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 lo
                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

cit


DÚVIDAS
?
citi