Core Data

Master class v2

JV - Alana

O que vamos modelar?

Banco de Produtos

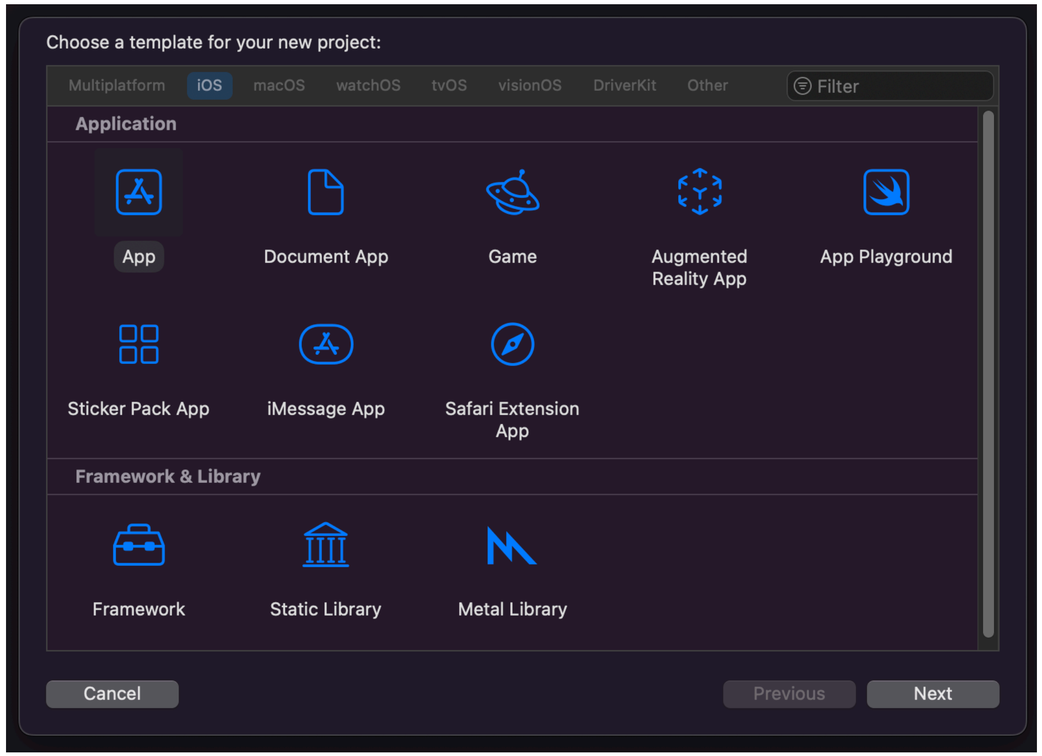
ID	Nome	Quantidade	Preço	Data de compra
1	Arroz	10	15	30/05/2025
2	Batata	20	5	30/05/2025
3	Cenoura	25	4	30/05/2025
4	Café	5	25	30/05/2025

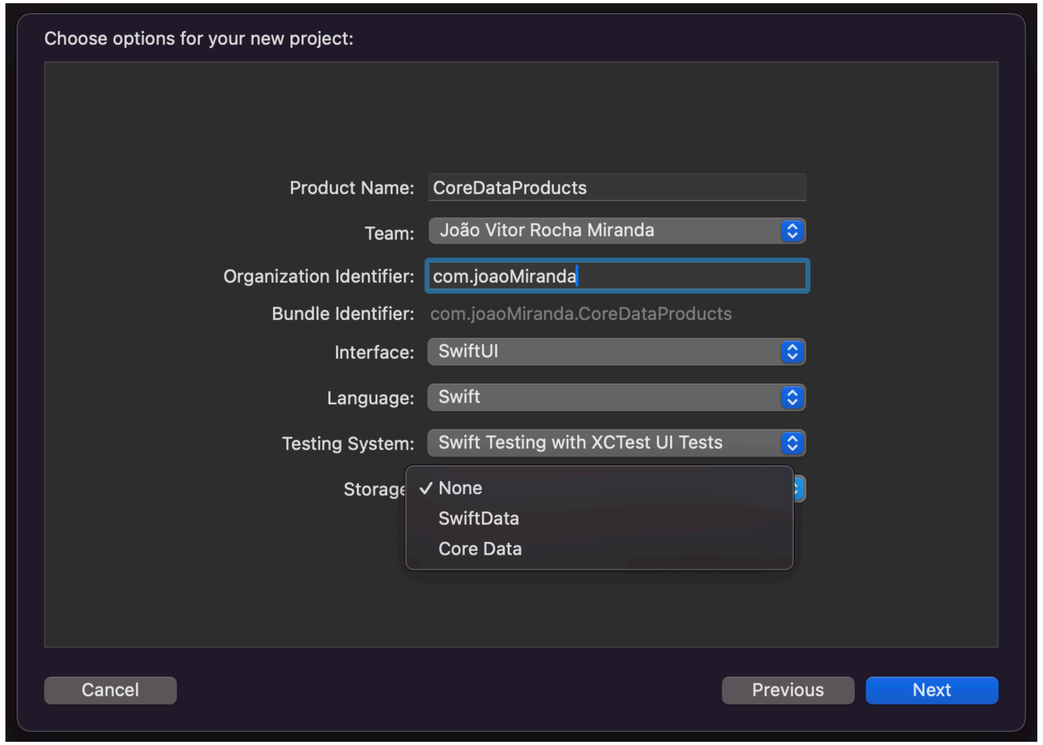
Banco de Vendas

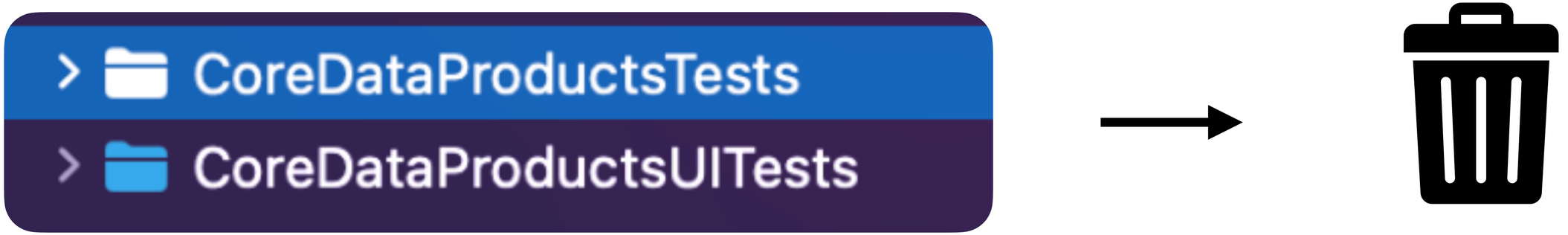
ID	ID_Produto	Data da Venda
12	2	2/6/25
13	4	2/6/25
14	1	2/6/25

Começando no Xcode

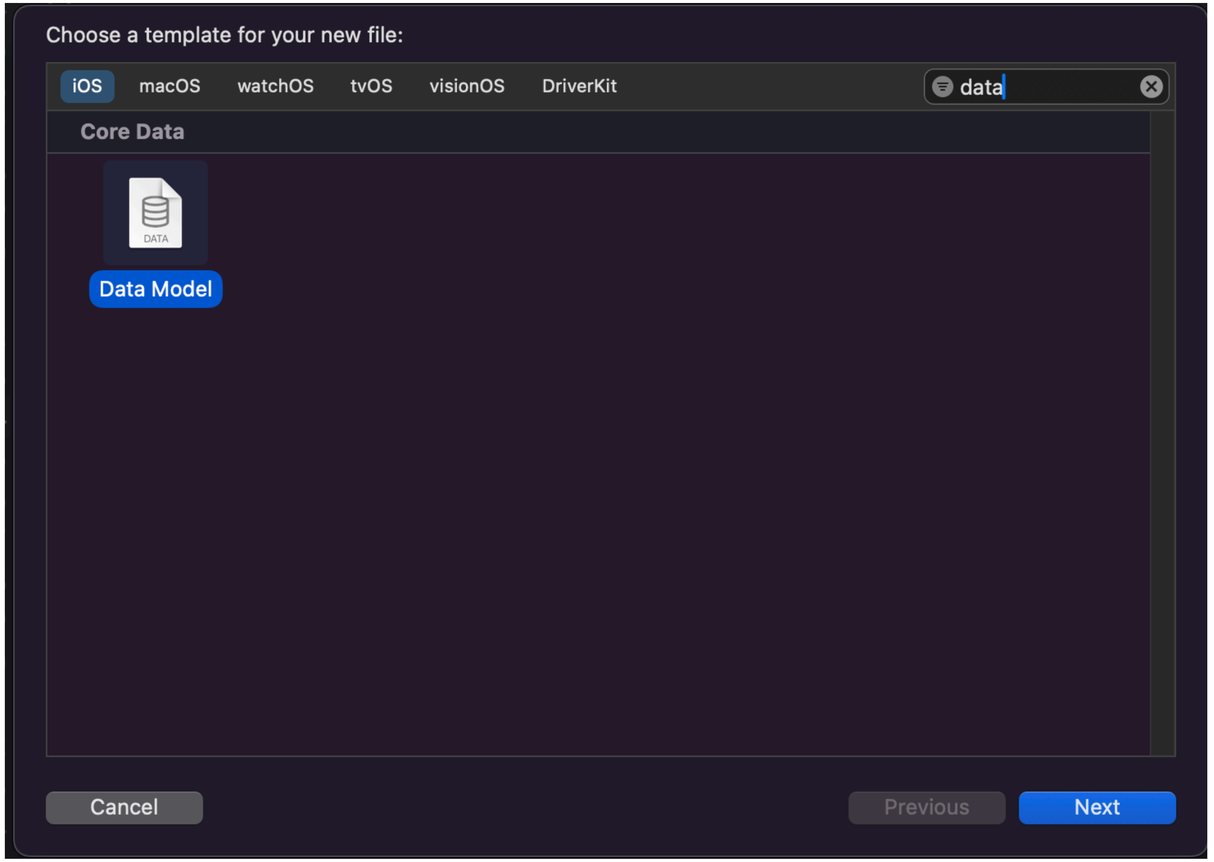
Iniciando o projeto:

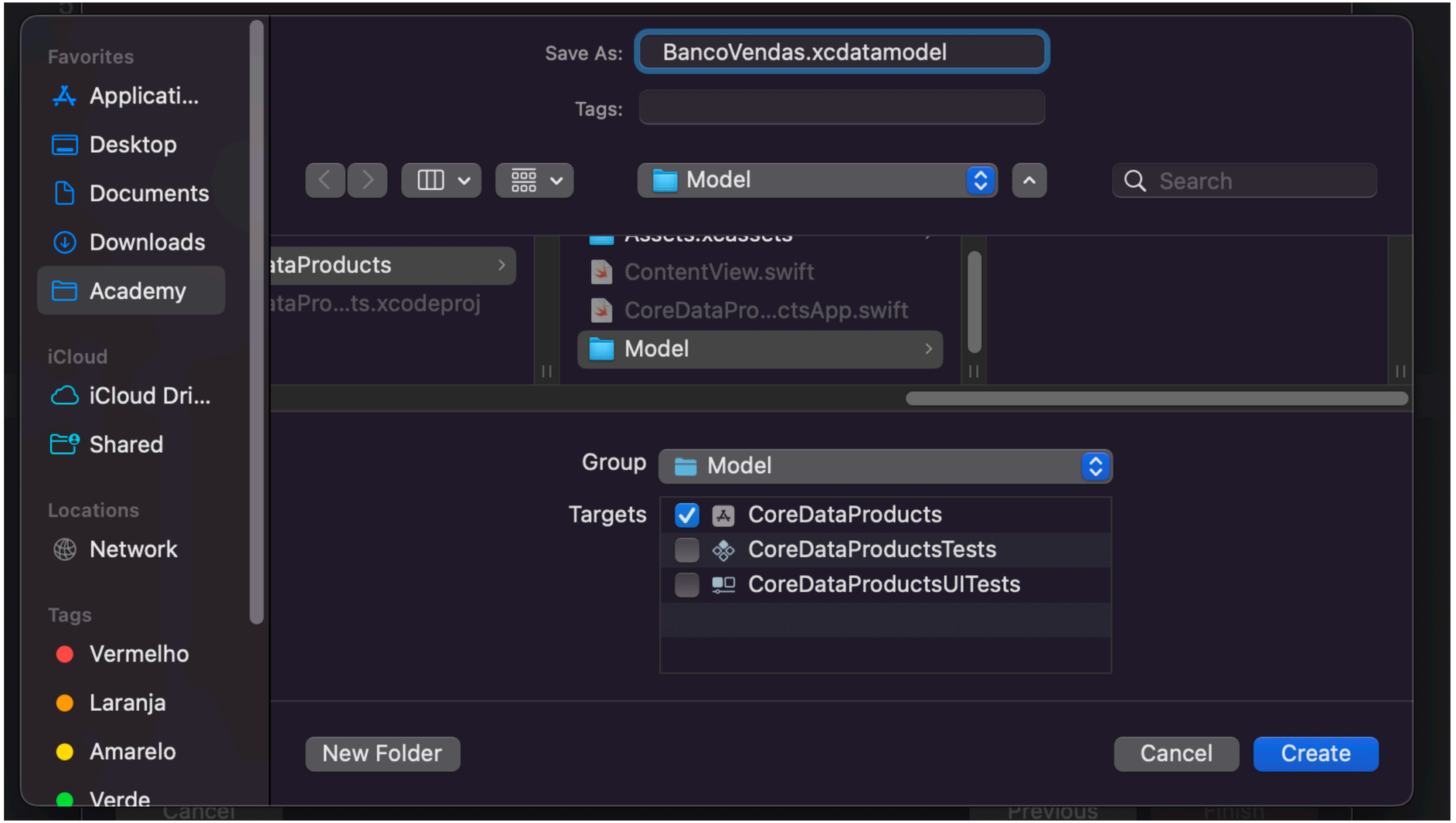


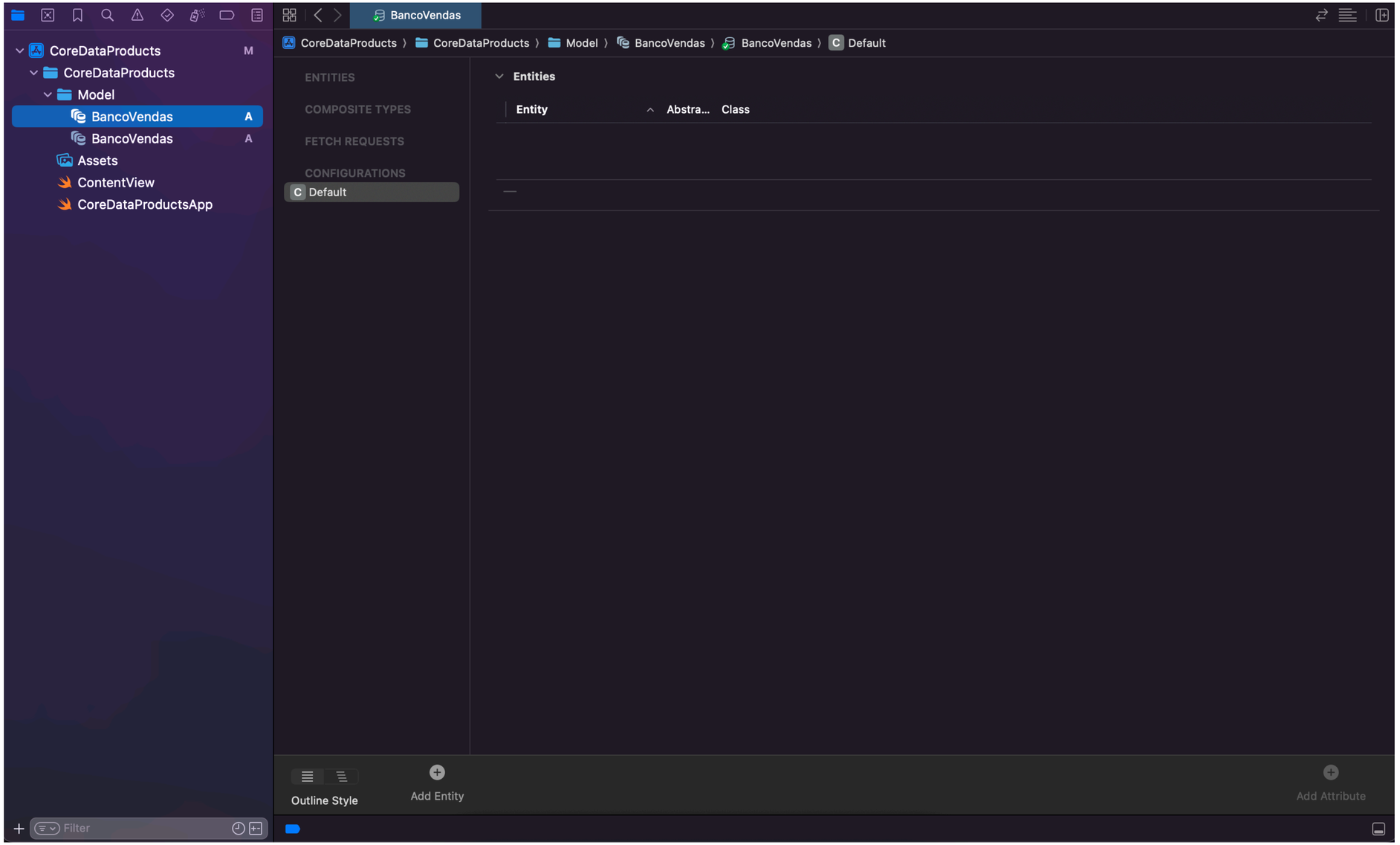



CoreDataProductsTests
CoreDataProductsUITests

Criar um novo Arquivo
do tipo **Data Model**




Favorites
Applicati...
Desktop
Documents
Downloads
Academy
iCloud
iCloud Dri...
Shared
Locations
Network
Tags
Vermelho
Laranja
Amarelo
Verde
Save As:
BancoVendas.xcdatamodel
Tags:
Model
Search
ataProducts
ataPro...ts.xcodeproj
ContentView.swift
CoreDataPro...ctsApp.swift
Model
Group
Model
Targets
CoreDataProducts
CoreDataProductsTests
CoreDataProductsUITests
New Folder
Cancel
Create

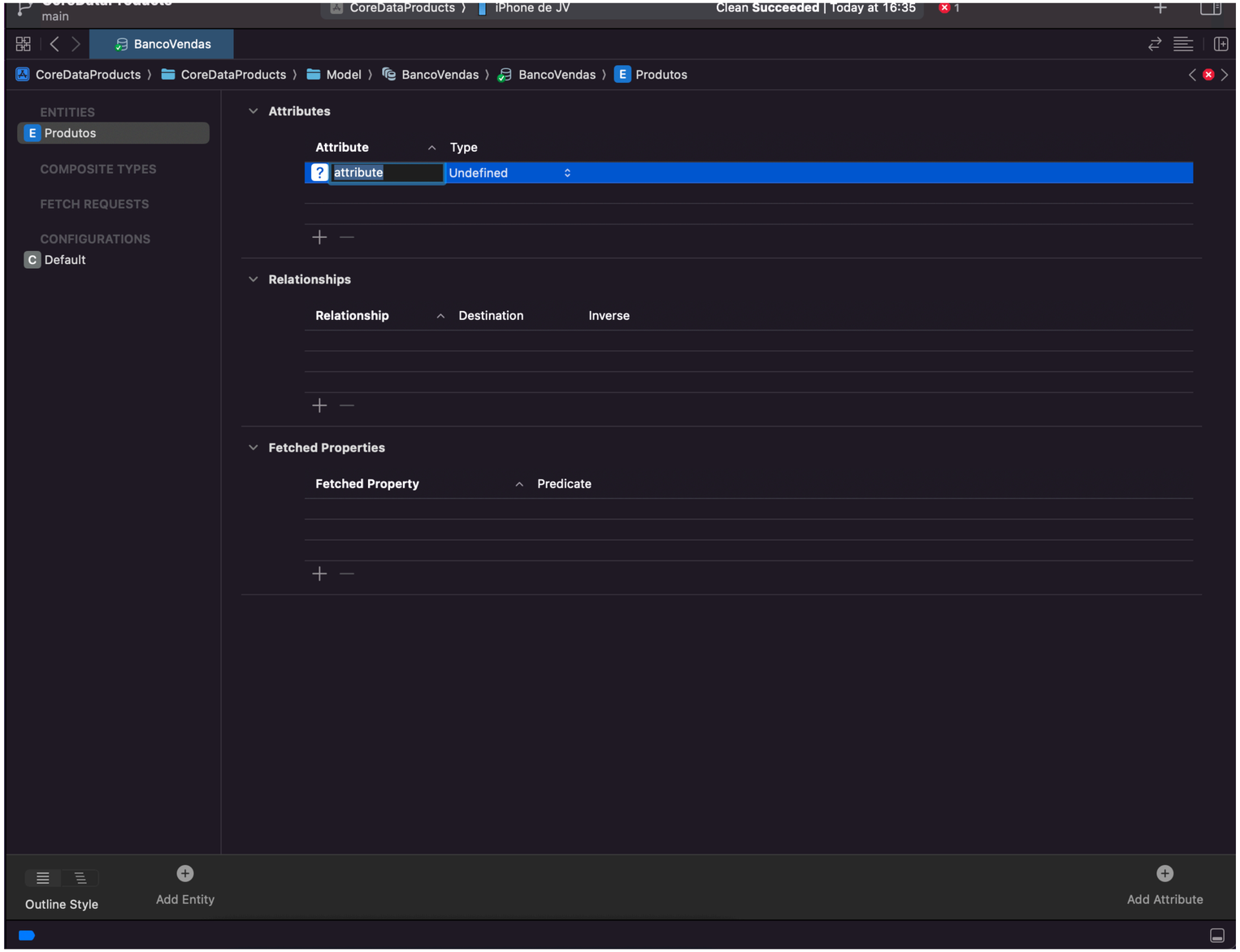

BancoVendas
CoreDataProducts
CoreDataProducts
Model
BancoVendas
BancoVendas
Assets
ContentView
CoreDataProductsApp
Filter
CoreDataProducts ⟩ CoreDataProducts ⟩ Model ⟩ BancoVendas ⟩ BancoVendas ⟩ Default
ENTITIES
COMPOSITE TYPES
FETCH REQUESTS
CONFIGURATIONS
Default
Entities
Entity
Abstra...
Class
Outline Style
Add Entity
Add Attribute

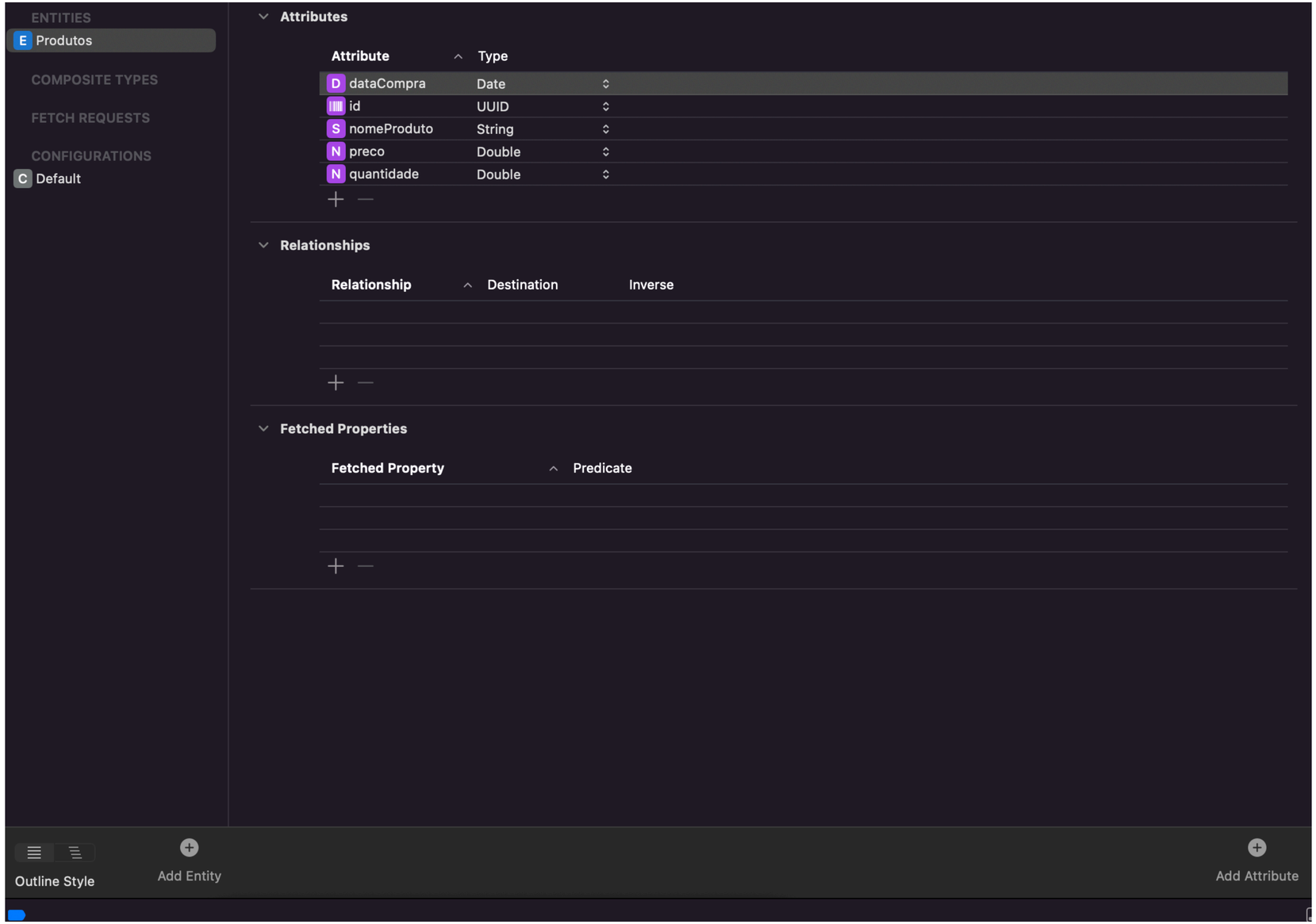
DICA:

Em caso de erro Com o coreData Limpe o app

CMD + Shift + K

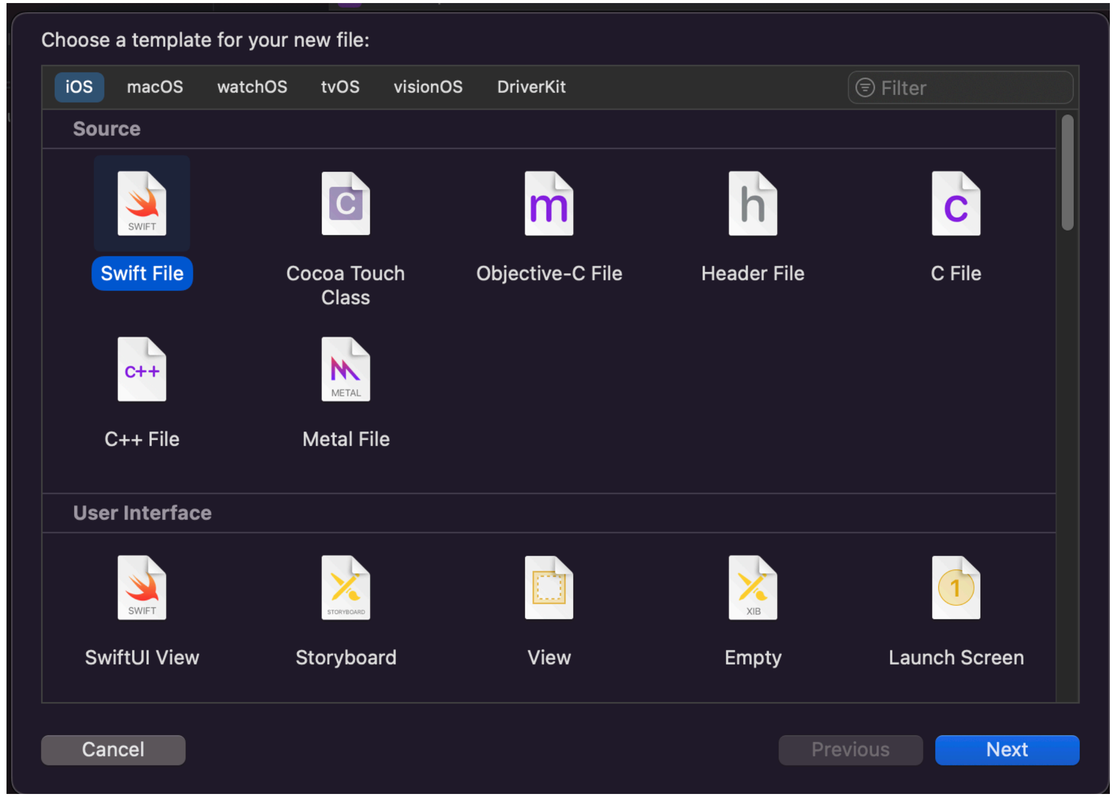
Apos clicar em ADD **ENTITY** no canto inferior direito, Vamos definir os **atributos** do nosso Banco de Produtos.




ENTITIES
Produtos
COMPOSITE TYPES
FETCH REQUESTS
CONFIGURATIONS
Default
Attributes
Attribute
Type
dataCompra
Date
id
UUID
nomeProduto
String
preco
Double
quantidade
Double
Relationships
Relationship
Destination
Inverse
Fetched Properties
Fetched Property
Predicate
Outline Style
Add Entity
Add Attribute

CoreDataControler

Criar um novo arquivo vazio chamado **CoreDataControler**



Após importar o CoreData, o esqueleto inicial da classe deve se parecer com algo assim:

```
8  import Foundation
9  import CoreData
10
11 class CoreDataController: ObservableObject{
12
13 }
14
```

ObservableObject

Seus atributos podem mudar e as interfaces podem atualizar seus dados

Caso for trabalhar com gráficos, os dados devem ser Identifiable, neste caso as classes ja é.

Init da padrão da Classe CoreDataControler. Este trecho garante a **comunicação** do Data Model com o código

```
import Foundation
import CoreData

class CoreDataController: ObservableObject{
    static let shared = CoreDataController()

    let persistentContainer: NSPersistentContainer
    var viewContext: NSManagedObjectContext {
        persistentContainer.viewContext
    }

    private init() {
        persistentContainer = NSPersistentContainer(name: "BancoVendas")

        persistentContainer.loadPersistentStores { _, error in
            if let error = error {
                fatalError("Could not load CoreData stack:
                    \(error.localizedDescription)")
            }

        }
    }
}
```

Funções

Caso haja alteração ele **salva** no banco

```
/*Save after changing the content*/
func saveContext() {
    if viewContext.hasChanges {
        do {
            try viewContext.save()
        } catch {
            print("Error saving context: \(error.localizedDescription)")
        }
    }
}
```

Cria um produto com base nos atributos passados

```
/*Create Product*/
func createProduct(nomeProduto: String, preco: Double, quantidade: Double,
    dataCompra: Date ) -> Produto {

    let produto = Produto(context: viewContext)

    produto.id = UUID() // Func nativa que cria um uuid
    produto.nomeProduto = nomeProduto
    produto.preco = preco
    produto.quantidade = quantidade

    saveContext()
    return produto
}
```

Busca os produtos no banco de produtos

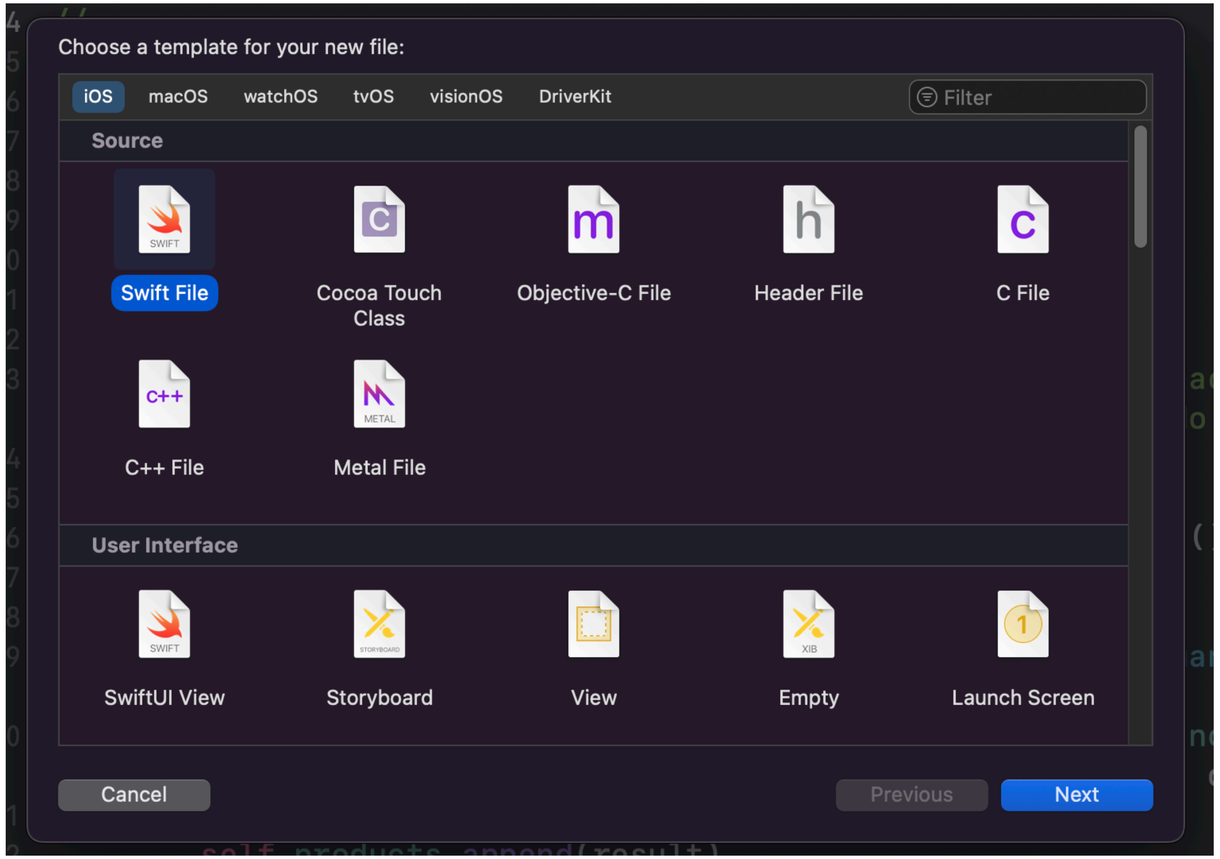
```
/*Search for products*/
func fetchAllProducts() -> [Produto] {
    let fetchRequest: NSFetchRequest<Produto> = Produto.fetchRequest()

    do {
        let result = try viewContext.fetch(fetchRequest)
        saveContext()
        return result
    } catch {
        print("Error fetching Products: \(error.localizedDescription)")
        return []
    }
}
```

Deleta o produto passado como parâmetro

```
/*Delete Produtct*/
func deleteProduct(_ product: Produto){
    viewContext.delete(product)
    saveContext()
}
```

ContentViewModel

Crie um novo arquivo chamado **ContentViewModel**, nele vamos interagir com o banco e as Views

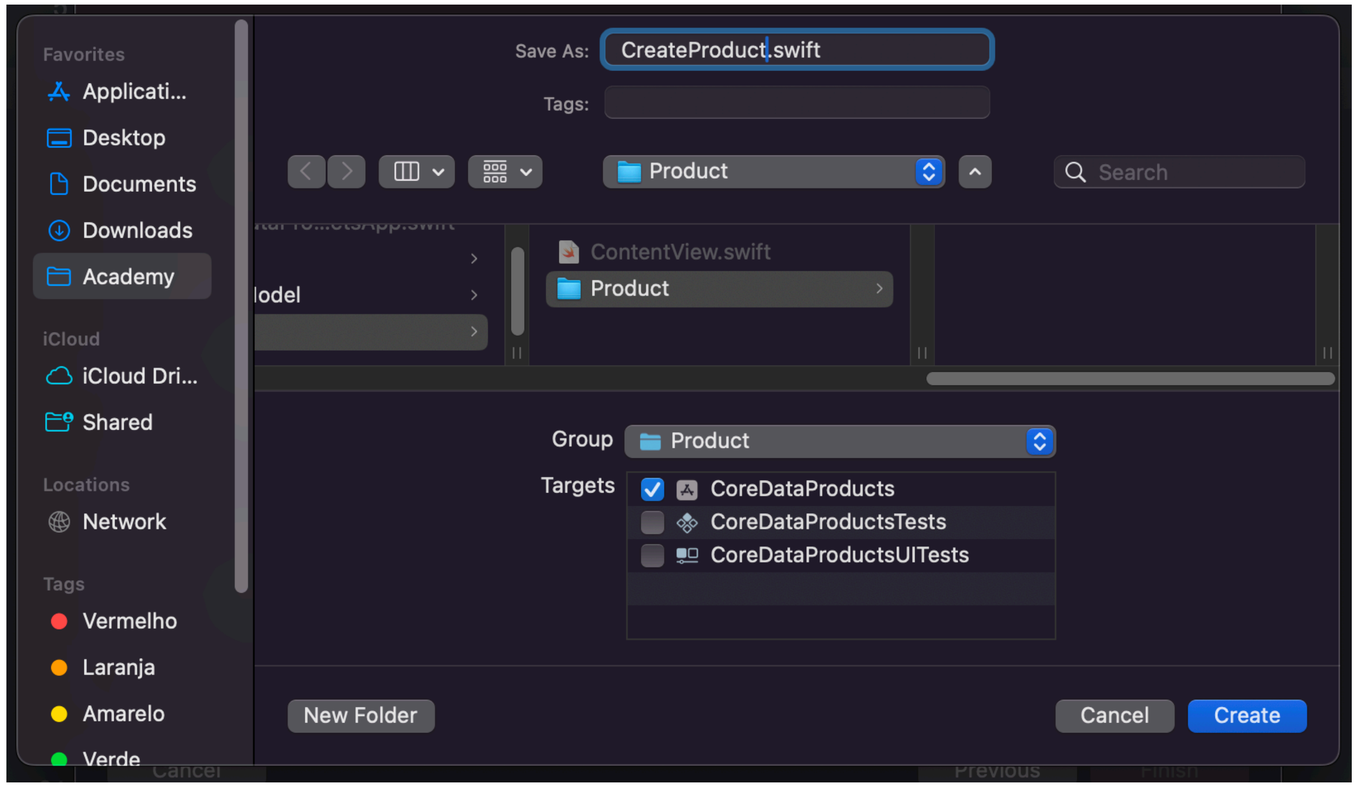


A classe deve conter estas **funções**, que se referenciam a tudo que criamos anteriormente na ContentViewModel

```swift
import Foundation
import CoreData
import SwiftUI

class ContentViewModel: ObservableObject {
    @Published var products: [Produto] = [] // Quando formos acessar os produtos do
        banco, sera atravez dessa variavel, nao diretamente do banco.

    func getProduct() {
        products = CoreDataController.shared.fetchAllProducts()
    }

    func createProduct(nomeProduto: String, preco: Double, quantidade: Double,
        dataCompra: Date ) {
        let result = CoreDataController.shared.createProduct(nomeProduto: nomeProduto,
            preco: preco, quantidade: quantidade, dataCompra: dataCompra)

        self.products.append(result)
    }

    func deleteProduct(_ product: Produto) {
        CoreDataController.shared.deleteProduct(product)
    }

}
```

CreateProductView

Vamos criar agora as **vizualizações** do produto, começando pela view de **CreateProduct**



Devemos pegar o **input** do usuário para servir como **parâmetro** da nossa função createProduct

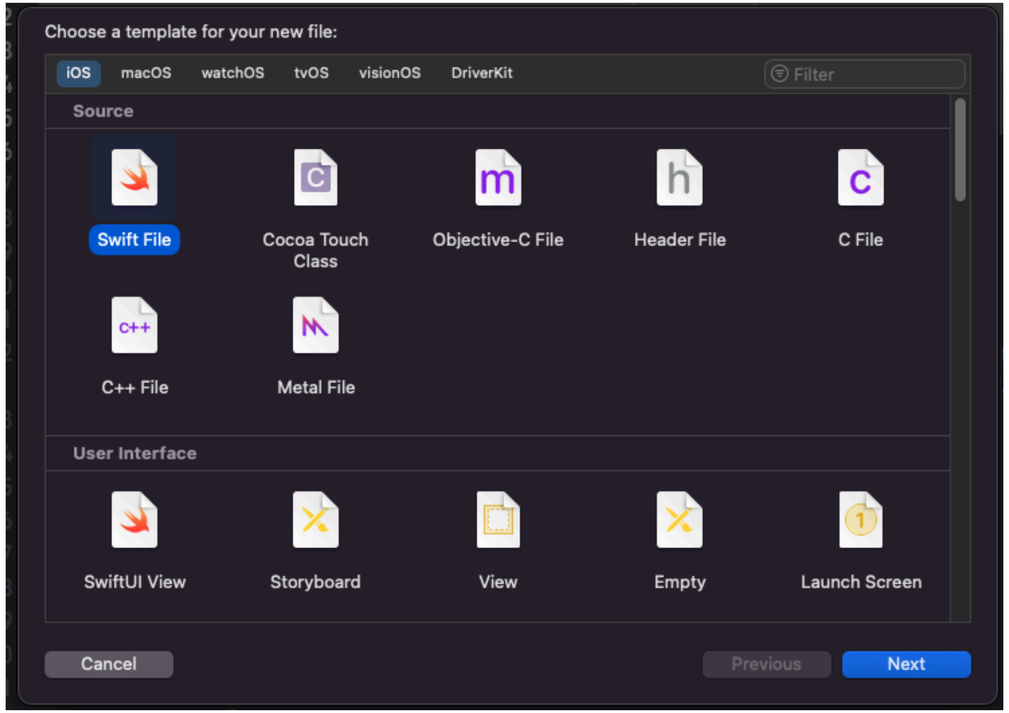
```
import SwiftUI

struct CreateProduct: View {
    @StateObject var viewModel: ContentViewModel //Acessaremos a
viewModel que sera passada como parametro quando chamarmos a view.

    @Environment(\.dismiss) var dismiss

    /*Products Atributes*/
    @State var nomeProduto: String = ""
    @State var preco: Double = 0
    @State var quantidade: Double = 0
    @State var dataCompra: Date = Date.now

    var body: some View {
        NavigationStack{

                Form {
                    /*Get user input*/
                    TextField("Nome", text: $nomeProduto)
                        .listRowSeparator(.visible)

                    HStack{
                        Text("Estoque:       ")
                        TextField("", value: $quantidade,
format: .number)
                    }
                    .listRowSeparator(.hidden)

                    HStack{
                        Text("Preço de Compra: ")
                        TextField("", value: $preco, format: .number)
                    }
                    .listRowSeparator(.hidden)


                }
                .scrollContentBackground(.hidden)
```

Date Component

Devemos formatar a data para extrair somente os atributos “Uteis"

Principalmente se formos comparar datas ou mostrar elas para o usuário

Vamos criar agora a extension da classe Date que irá realizar isso. Podemos chamar ela de **DateComponent**



Quando formos acessar qualquer tipo de data e for necessário que ela esteja **formatada**, apenas adicionamos o **.onlyDate**

```swift
import Foundation


extension Date {
    var onlyDate: Date {
        let calendar = Calendar.current
        let components = calendar.dateComponents([.year, .month, .day], from: self)
        return calendar.date(from: components)!
    }
}
```

Apenas formatando a dataCompra com nossa nova extension

```
/*Products Atributes*/
@State var nomeProduto: String = ""
@State var preco: Double = 0
@State var quantidade: Double = 0
@State var dataCompra: Date = Date.now.onlyDate
```

Após pegar o input do usuário, vamos enviar os dados para nossa viewModel

```
.toolbar {
                ToolbarItem(placement: .confirmationAction) {
                    /*Save Button*/
                    Button("Save") {
                        viewModel.createProduct(nomeProduto:
nomeProduto, preco: preco, quantidade: quantidade, dataCompra:
dataCompra)
                        dismiss()
                    }
                }
                /*Cancel Button*/
                ToolbarItem(placement: .cancellationAction) {
                    Button("Cancel") {
                        dismiss()
                    }
                }
            }
            .navigationTitle("Create")
            .navigationBarTitleDisplayMode(.inline)
        }
```

ProductView

Vamos criar agora a **vizualização** do produto

```
struct ProductView: View {
    @ObservedObject public var product: Produto
    @ObservedObject var viewModel = ContentViewModel()


    var body: some View {
        Grid (alignment: .leading) {
            HStack (){
                Text(product.nomeProduto ?? "No name found.")
                    .font(.title3)
                    .bold()
                    .padding(.top, 0.5)
                Spacer()
            }
            Divider()
            Text("Estoque: \(product.quantidade, specifier: "%.0f")")
                .font(.callout)
                .padding(.bottom, 0.5)
                .padding(.top)
            Text("Preço de Compra: R$\(product.preco, specifier: "%.2f")")
                .font(.callout)
                .padding(.bottom, 0.5)
            Divider()

            .padding(.top)
            .padding(.bottom, 0.5)
        }
        .onAppear(){
            viewModel.getProduct()
        }
    }
}
```

ProductListView

Filtro de Produtos

Busca de itens específicos dentro da lista de todos os produtos

Vamos pegar o input da String que o usuário deseja **filtrar** e mostrar uma lista dos produtos que correspondem ao que ele esta buscando

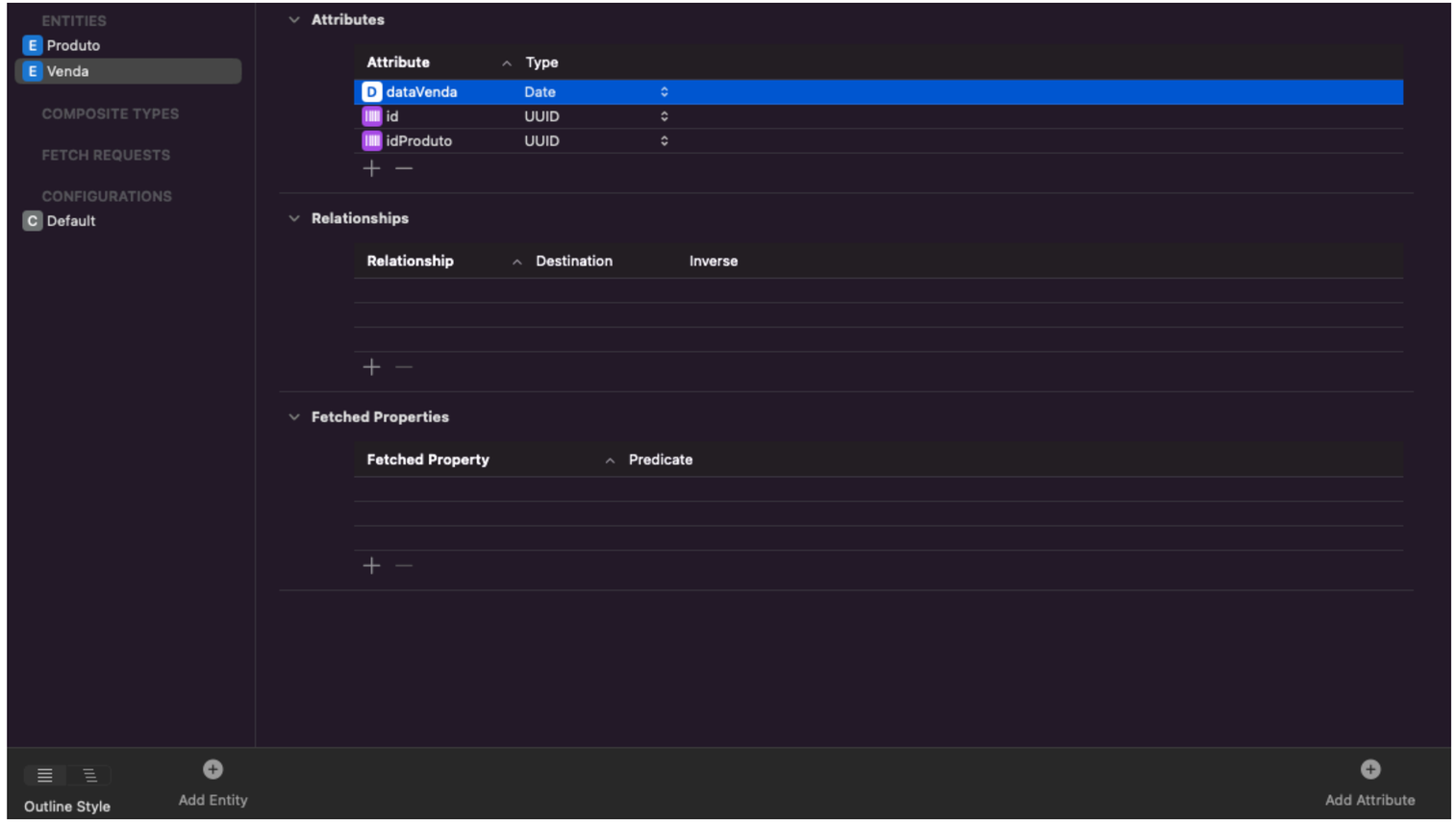
```
@State var searchText: String = "" // Used in Search bar

    var searchReults: [Produto] {
        guard !searchText.isEmpty else {
            return viewModel.products
        }

        return viewModel.products.filter {
            $0.nomeProduto!.localizedCaseInsensitiveContains(searchText)
        }
    }
```

Visão Completa da **ProductView**, já **integrada** com as visualizações criadas anteriormente.

```
struct ProductListView: View {
    @StateObject var viewModel = ContentViewModel()
    @State private var showingSheet: Bool = false
    @State var searchText: String = "" // Used in Search bar

    var body: some View {
        NavigationStack {
            VStack {
                List(searchReults) { product in
                        Section {
                            ProductView(product: product)
                        }
                        .swipeActions {
                            Button(role: .destructive) {
                                viewModel.deleteProduct(product)
                            } label: {
                                Image(systemName: "trash")
                            }
                        }
                }
                .scrollContentBackground(.hidden)
                .listStyle(.insetGrouped)
            }
            .navigationTitle("Produtos")
            .toolbar {
                ToolbarItem(placement: .confirmationAction) {
                    Button {
                        showingSheet.toggle()
                    } label: {
                        Image(systemName: "plus")
                    }

                    .sheet(isPresented: $showingSheet, onDismiss: {
                        viewModel.getProduct()
                    }) {
                        CreateProduct(viewModel: viewModel)
                    }
                }
            }
            .onAppear {
                viewModel.getProduct()
            }
            .searchable(text: $searchText)
        }
```

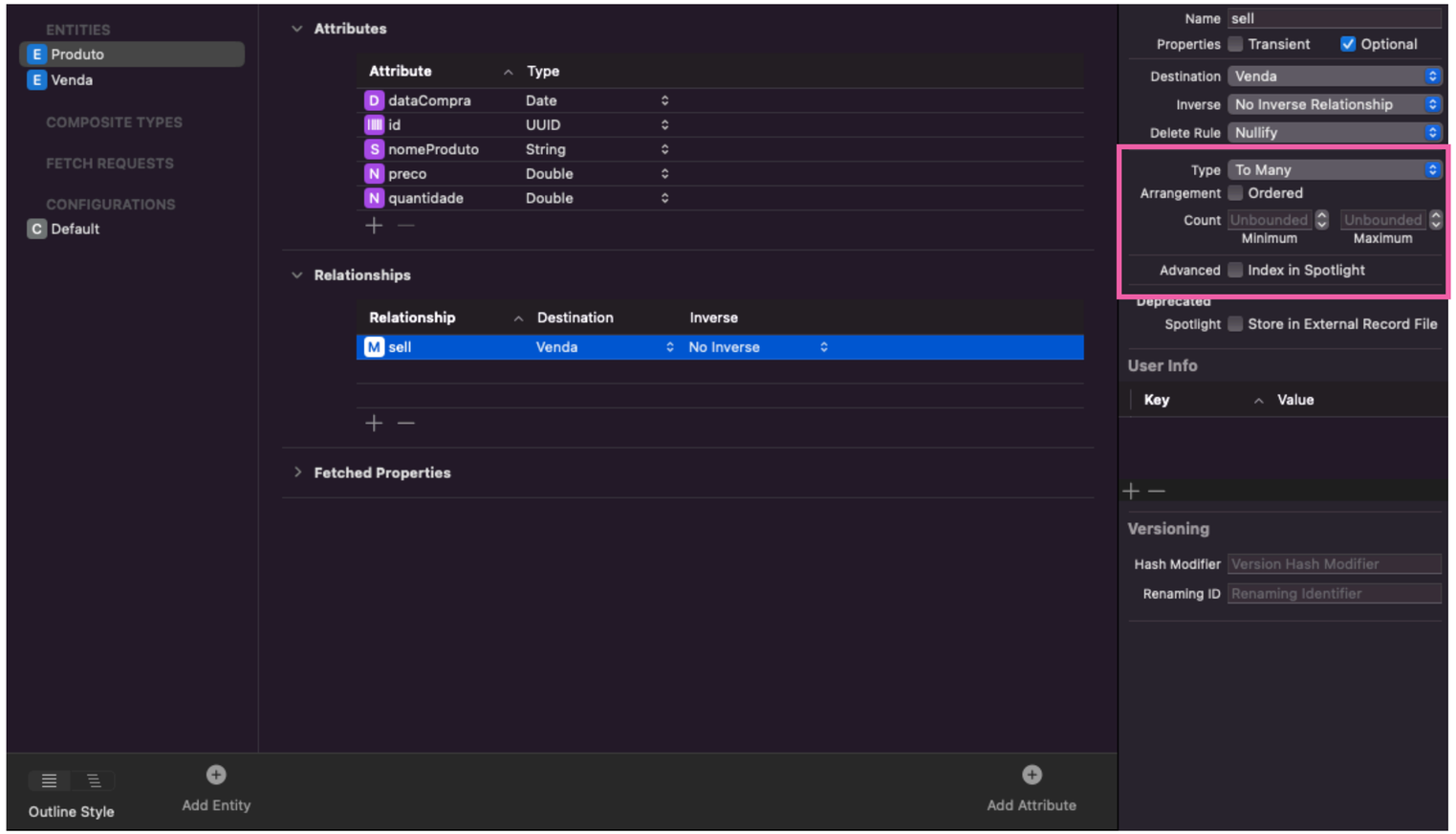
Nova Entidade de Vendas

Novamente no **BancoVendas** vamos clicar em **add Entity** e criar o banco de vendas com os atributos ao lado.



Relationships

Agora em Produto, Iremos clicar no ícone de + e criar uma **relação** com venda, onde um produto pode ter diversas vendas.



De Volta a CoredataControler

Vamos adicionar estas funções básicas somente para criar um protótipo de log de vendas.

```
func createSell(_ product: Produto) -> Venda {
        let sell = Venda(context: viewContext)

        sell.id = UUID()
        sell.idProduto = product.id
        sell.dataVenda = Date.now.onlyDate
         saveContext()
        return sell
    }

    func fetchAllSells() -> [Venda]{
        let fetchRequest: NSFetchRequest<Venda> = Venda.fetchRequest()

        do {
            let resultSell = try viewContext.fetch(fetchRequest)
            saveContext()
            return resultSell
        } catch {
            print("Error fetching Products: \(error.localizedDescription)")
            return []
        }
    }
```

De Volta a ContentViewModel

```swift
func getSell() {
    sells = CoreDataController.shared.fetchAllSells()
}

func createSell(_ product: Produto){
    let result = CoreDataController.shared.createSell(product)

    self.sells.append(result)
}
```

Novo Botão na Product View

```
Stepper("Nova Venda: ", onIncrement: {
    viewModel.createSell(product)
}, onDecrement: {
    print("TODO")

})
.padding(.top)
.padding(.bottom, 0.5)
```

Link do projeto completo:

https://github.com/J040VRM/CoreDataTutorial.git