

Bazy danych Dokumentacja

#learn/studies/2017

Nazwa projektu: Northwind iOS

Technologia: Aplikacja na iOS napisana w języku programowania Swift

Autorzy: Sebastian Korzeniak, Maciej Łobodziński, Michał Śmiałko

Data: 20/01/2018

Kierunek: Informatyka IV Niestacjonarne

Projekt aplikacji mobilnej na telefony iPhone możliwa do uruchomienia także na tabletach iPad.

Ustalenia początkowe i zakres projektu

Celem projektu jest zaimplementowanie systemu realizującego wybrane podstawowe operacje w przykładowej bazie Northwind w wybranej technologii.

- Operacje CRUD na wybranych tabelach.
- Operacje składania zamówień na produkty
- Operacje wyszukiwania informacji

Wizja aplikacji

Stworzyliśmy aplikację dla menadżerów magazynów i sklepów pomagającą w codziennych obowiązkach. Pozwala między innymi na:

- Sprawdzanie listy produktów
- Zmiany szczegółów produktów
- Sprawdzanie nowych zamówień od klientów
- Dodawanie nowych zamówień

Warstwa wizualna oraz nawigacja aplikacji

Staraliśmy się by warstwa wizualna trzymała się Apple Guidelines iOS 11.

Do nawigacji w aplikacji użyliśmy pattern Navigation Controller, który pozwala na pokazywanie zagnieżdżonych widoków w logiczny flow, dzięki któremu użytkownik wie, w którym miejscu aplikacji się znajduje.

Przejrzysty interfejs tabel umożliwia szybkie nawigowanie pomiędzy kategoriami, a przyciski dodawania i usuwania elementów umieszczone w Navigation Barze dają szybki dostęp do najczęściej używanych funkcji systemu.

Przykładowy scenariusz użycia

Przykładowy scenariusz użycia na przykładzie obsługi nowego zamówienia złożonego przez klienta.

Obsługę nowego zamówienia możemy podzielić na trzy etapy:

- Zakładamy, że klient dzwoni do pracownika przez telefon i podaje zamówienie
 - Pracownik dodaje do systemu nowe zamówieni
 - Pracownik wykonuje zamówienie
-
1. Aplikacja włącza się ekranem startowym, tak zwanym Launch screen, który jest pokazywany w czasie ładowania zasobów do pamięci podręcznej telefonu.
 2. Pierwszym ekranem który pokazujemy po faktycznym starcie aplikacji jest ekran "Select Flow of work View" (Products / Orders / Categories). Na tym ekranie użytkownik decyduje który obszar bazy w danym momencie go interesuje. W przypadku dodawania nowego zamówienia wybiera "Orders List"
 3. Kolejnym ekranem widzimy posortowaną Listę Zamówień. Po naciśnięciu przycisku znajdującego się w prawym górnym rogu użytkownik przechodzi do ekranu dodawania nowego zamówienia, gdzie uzupełnia dane zamówienia.
 4. Następnie użytkownik wraca do ekranu wyboru Procesów by otworzyć listę produktów i sprawdzić czy pozycje interesujące klienta są dostępne. Użytkownik używa "Search Bar" - wyszukiwarki by szybko znaleźć konkretny go produkt.

Technologia

Do zbudowania aplikacji użyliśmy frameworków zapewnianych przez firmę Apple: Core Data, Foundation oraz UIKit

Język programowania: Do stworzenia projektu wybraliśmy język programowania Swift stworzony przez firmę Apple jako następcę Objective-C.

Dependency manager: projekt nie wymagał posiada zewnętrznych zależności, więc nie

używamy menadżera zależności.

Baza danych: Najważniejszym elementem projektu jest warstwa zarządzania danymi do której używamy frameworku Core Data. Został on stworzony przez firmę Apple specjalnie na potrzeby produktów iOS oraz macOS. Służy do zarządzania warstwą modelu obiektów w aplikacjach. Zapewnia generalne i automatyczne rozwiązania do często występujących zadań związanych ze stanami obiektów, persystowaniem danych oraz graficznym zarządzaniem danymi.

Core Data może zapisywać obiekty do XML, Binarki lub SQLite.

Architektura Aplikacji

Aplikacja jest napisana w oparciu o wzorzec projektowy MVC Model View Controller polecaną przez firmę Apple jako typowy wzorzec projektowy do aplikacji.

W MVC staramy się odseparować Model czyli wszelkie obiekty bazy danych oraz logikę biznesową od Widoków, czyli os warstwy prezentowanej użytkownikowi. Widok z Modelem komunikuje się dzięki obiektowi Kontrolera, który niejako podaje dane z modelu w odpowiednie miejsca i zawiaduje widokiem. Widok na podstawie tych danych wie, jak powinien się konfigurować.

1. Widoki

Widoki zostały zaprogramowane w programie Interface Builder zapewnianym przez firmę Apple w swoim IDE Xcode. Jest to graficzna forma prezentowania XMLA, którą w czasie rzeczywistym można edytować i wizualnie wyświetlać w takiej formie, w jakiej user zobaczy ją finalnie w aplikacji.

Importer Danych

Ważnym komponentem projektu jest importer danych uruchamiany po kliknięciu przycisku "import" zlokalizowanego w głównym ekranie w Navigation Barze Navigation Controllera.

Uruchamia on Import klasy zlokalizowanej w pliku Import.swift

```
func importData() {  
    let names: [String] = [
```

```

    "products",
    "regions",
    "categories",
    "territories",
    "employees",
    "order-details",
    "customers",
    "orders",
    "suppliers",
    "shippers"]

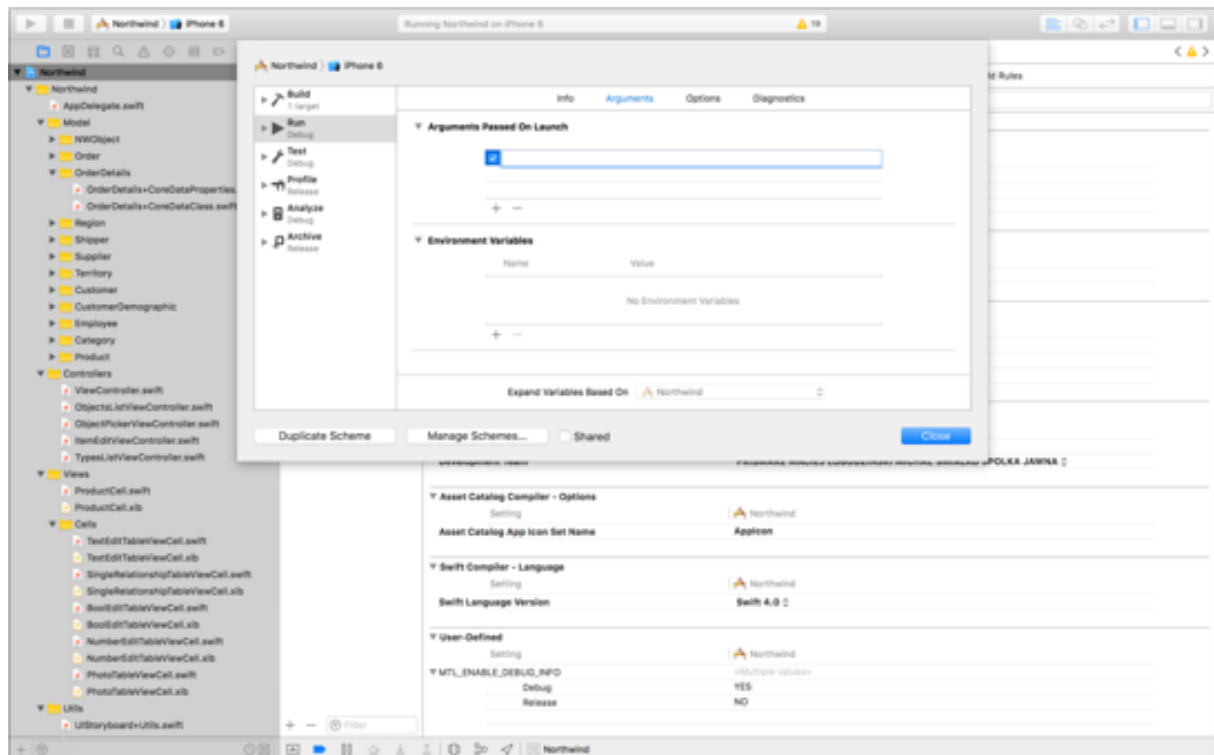
for idx in names.indices {
    let name = names[idx]
    let path = Bundle.main.path(forResource: name, ofType: "csv")!
    let url = URL(fileURLWithPath: path)
    let fileData = try! Data(contentsOf: url)
    let fileText = String(bytes: fileData, encoding: .utf8)!

    switch idx {
    case 0: importProducts(fileText: fileText)
    case 1: importRegions(fileText: fileText)
    case 2: importCategories(fileText: fileText)
    case 3: importTerritories(fileText: fileText)
    case 4: importEmployees(fileText: fileText)
    case 5: importOrderDetails(fileText: fileText)
    case 6: importCustomers(fileText: fileText)
    case 7: importOrders(fileText: fileText)
    case 8: importSuppliers(fileText: fileText)
    case 9: importShippers(fileText: fileText)
    default: break
    }
}
}

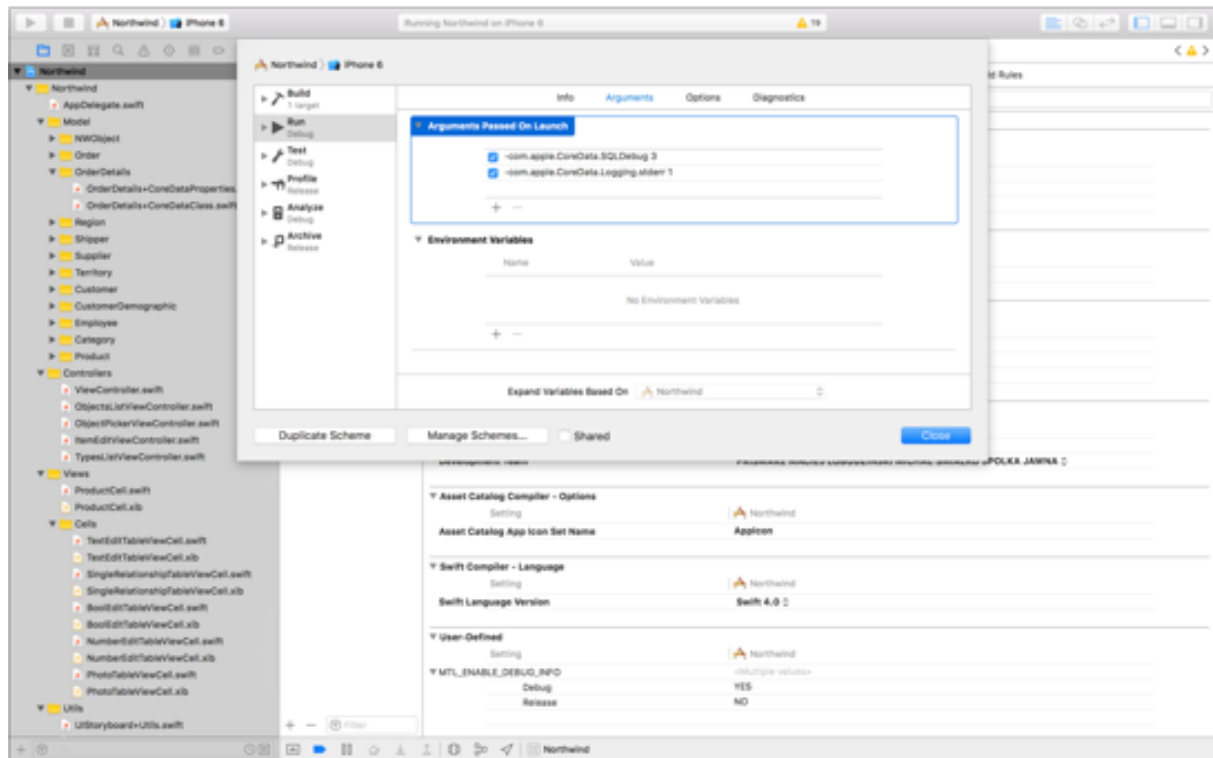
```

Logging

By zobaczyć zapytania SQLite które wykonuje CoreData, musimy włączyć logowanie CoreData. By to zrobić otwieramy Run scheme i dodajemy dwa argumenty które będą podawane po uruchomieniu.

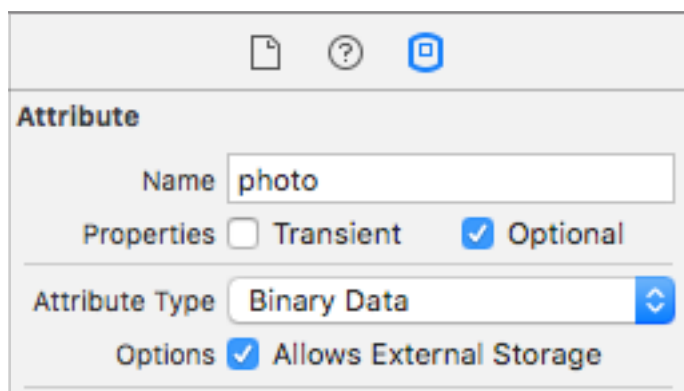


```
-com.apple.CoreData.SQLDebug 3  
-com.apple.CoreData.Logging.stderr 1
```



Going deeper - External Storage

As you probably know, storing large binary data in SQL database is not sufficient. However, it would be really helpful to store images and videos in our database just like any other piece of data, rather than manually managing file paths, deletions, redos etc. Core Data External Storage comes to the rescue! If one of your entity's attributes has type `Binary Data`, you will see an option in Attributes Inspector called `Allows External Storage`.



When checked, Core Data will transparently decide whether this piece of data should be stored in the SQLite database or as a external file on disk and keep reference to it. All of that happens transparently. Don't believe? Let's check.

We've checked that option on photo attribute in Employee entity. Then, we downloaded app data and inspected sqlite database. Here's how it looks like:

```
Documents $ ls
SingleViewCoreData.sqlite  SingleViewCoreData.sqlite-shm  SingleViewCoreData.sqlite-wal
Documents $ sqlite3 SingleViewCoreData.sqlite
SQLite version 3.19.3 2017-06-27 16:48:08
Enter ".help" for usage hints.
sqlite> .tables
ZMWOBJECT      Z_STERRITORIES  Z_MODELCACHE
Z_3DEMOGRAPHICS  Z_METADATA      Z_PRIMARYKEY
sqlite> SELECT * FROM ZMWOBJECT;
1|5|3|0||0|||last name|||31201550-4597-4808-90D0-19F9EB52EF7B
2|5|2|0||0|||F0785587-EFD1-4DED-843D-FDB238A95C39
3|5|2|0||0|||civicicic|||39A7FA2B-D90C-476B-8C47-18932E493308
4|6|1|10991|||1727|906|||38.51|Taucherstraße 10|Cunewalde|Germany|QUICK-Stop|01307|NUL
L|||
5|0|1|16|||0|0|10|29|0|522|1243|||17.45|||Pavlova|||
6|7|1|10645|||15|||0|10|||
7|7|1|10812|||20|||0|13|||
```

You will notice that the last value in the row is some kind of identifier. We've never set it so where does it come from?

When you list all the files in the directory where database is located, you will find a hidden directory. Let's see what's inside it.

```
Documents $ ls -al
total 1512
drwxr-xr-x  6 michal  staff   192 Jan 18 22:35 .
drwxr-xr-x  6 michal  staff   192 Jan 18 22:35 ..
drwxr-xr-x  3 michal  staff    96 Jan 18 22:35 .SingleViewCoreData_SUPPORT
-rw-r--r--  1 michal  staff  94208 Jan 18 22:35 SingleViewCoreData.sqlite
-rw-r--r--  1 michal  staff 32768 Jan 18 22:35 SingleViewCoreData.sqlite-shm
-rw-r--r--  1 michal  staff 646872 Jan 18 22:35 SingleViewCoreData.sqlite-wal
Documents $ ls -al .SingleViewCoreData_SUPPORT/_EXTERNAL_DATA/
total 23504
drwxr-xr-x  5 michal  staff   160 Jan 18 22:35 .
drwxr-xr-x  3 michal  staff    96 Jan 18 22:35 ..
-rw-r--r--  1 michal  staff 6055092 Jan 18 22:35 31201550-4597-4808-90D0-19F9EB52EF7B
-rw-r--r--  1 michal  staff 3525409 Jan 18 22:35 39A7FA2B-D90C-476B-8C47-18932E493308
-rw-r--r--  1 michal  staff 1842182 Jan 18 22:35 F0785587-EFD1-4DED-843D-FDB238A95C39
Documents $
```

Surprise! We've just found some binary files saved there using the same identifiers. If you change extensions of those files to eg .png, you will find out that those are exactly the same images we set as an employee's photo :)

Przewodnik programowania w Swift i CoreData

W tej części chcielibyśmy zilustrować jak należy programować elementy systemu bazodanowego - na przykładzie naszego projektu i bazy Northwind.

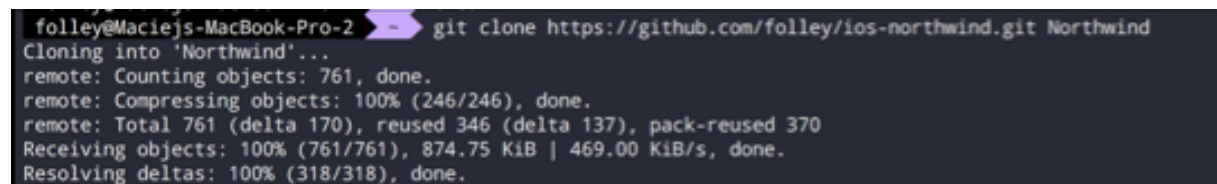
Do uruchomienia projekt wymaga:

1. Systemu operacyjnego mac OS (testowane na wersji 10.13.2 macSO High Sierra)
2. IDE Xcode minimum wersji 9.0 (Testowane na wersji 9.2 9C40b)
3. Aplikacja działa na systemie operacyjnym iOS 11.1 (Deployment target = 11.1)
4. Do przetestowania funkcjonowania systemu potrzeba jednego urządzenia. Istnieje możliwość 'mockowania' iPhone'ów przez symulatory w IDE Xcode.

Setup projektu

1. By pobrać projekt należy w terminalu użyć komendy git clone

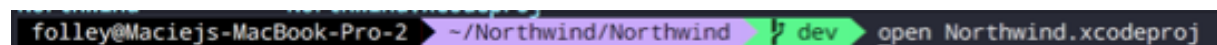
```
git clone https://github.com/folley/ios-northwind.git
```



```
folley@Maciej's-MacBook-Pro-2 ~ % git clone https://github.com/folley/ios-northwind.git Northwind
Cloning into 'Northwind'...
remote: Counting objects: 761, done.
remote: Compressing objects: 100% (246/246), done.
remote: Total 761 (delta 170), reused 346 (delta 137), pack-reused 370
Receiving objects: 100% (761/761), 874.75 KiB | 469.00 KiB/s, done.
Resolving deltas: 100% (318/318), done.
```

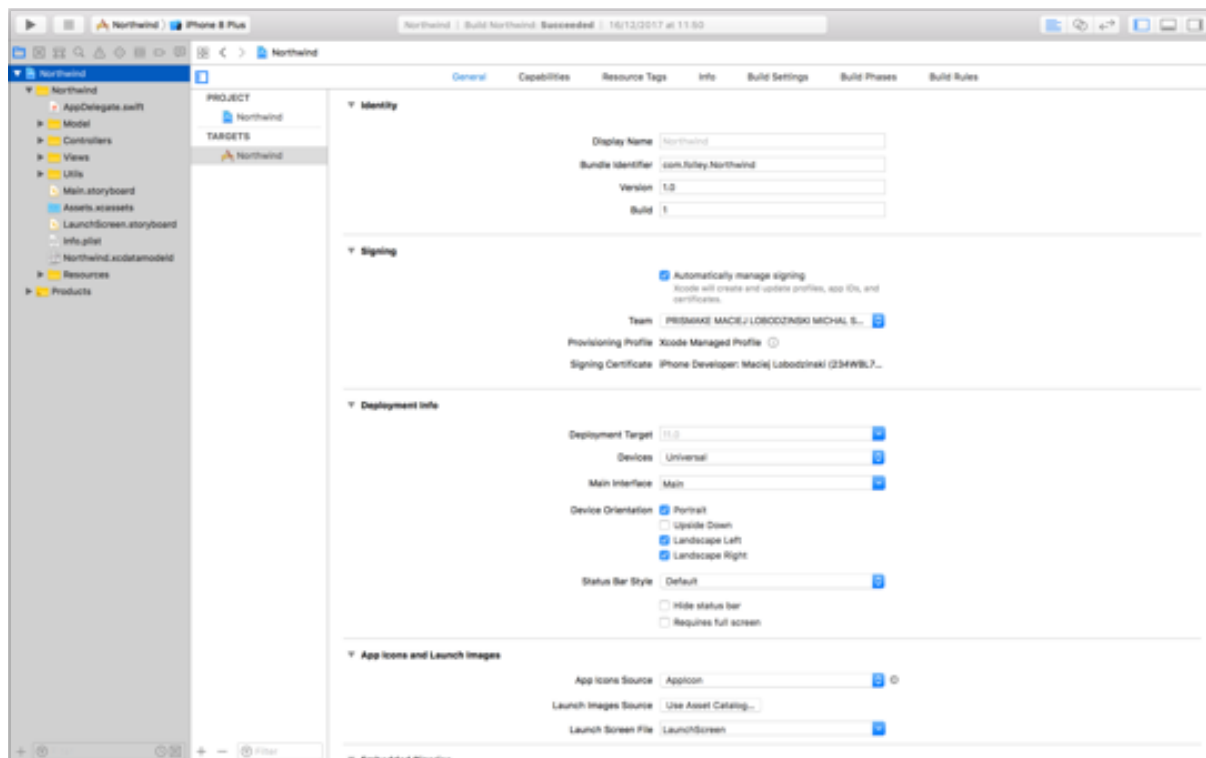
2. By otworzyć projekt

```
open Northwind.xcodeproj
```



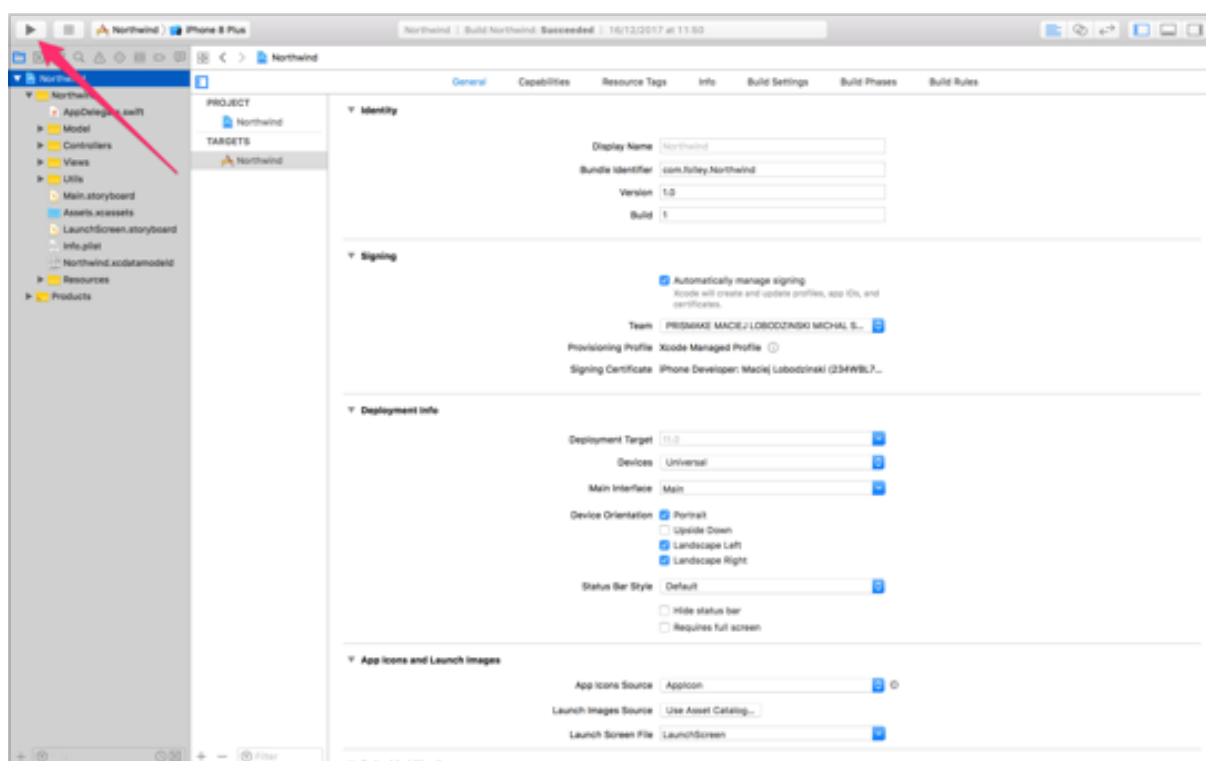
```
folley@Maciej's-MacBook-Pro-2 ~/Northwind/Northwind % open Northwind.xcodeproj
```

3. Co spowoduje otwarcie IDE Xcode



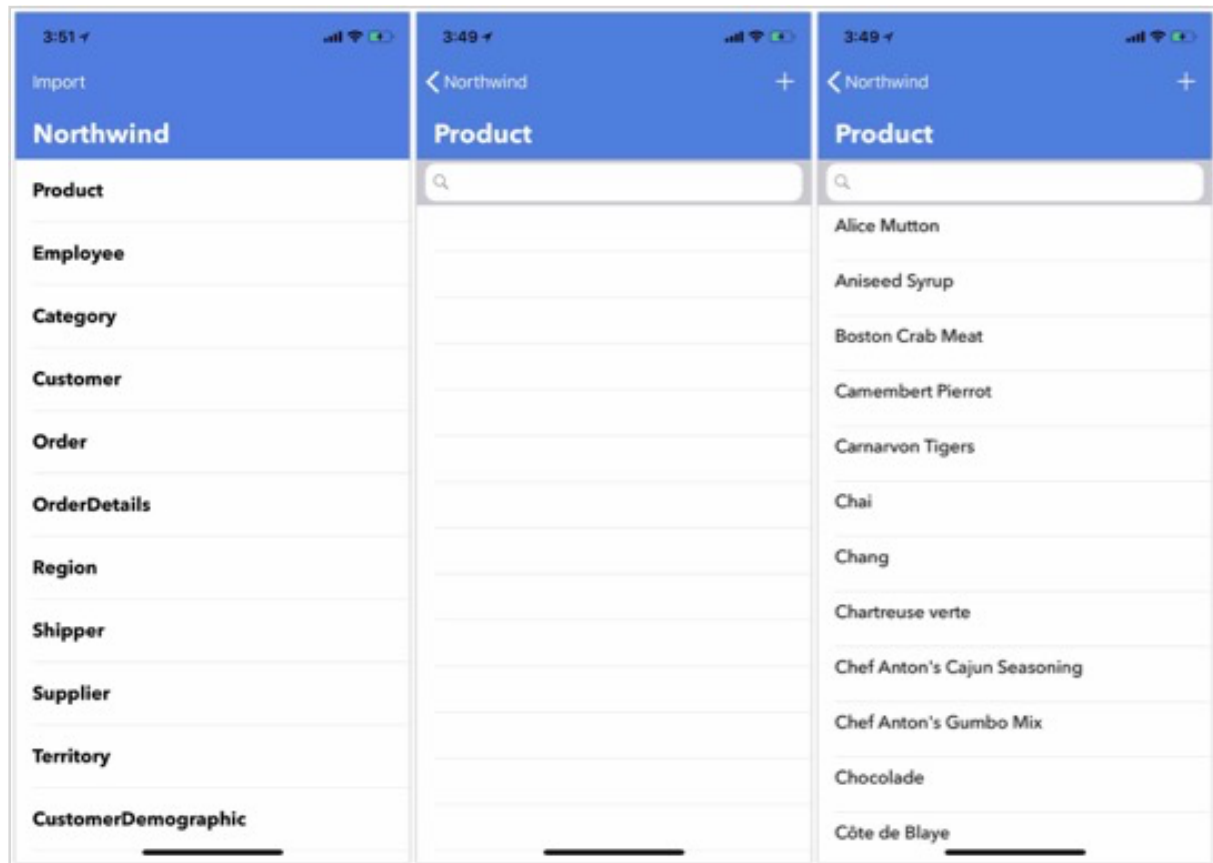
5. By zbudować projekt należy kliknąć przycisk play lub użyć skrótu klawiszowego `command +`

B



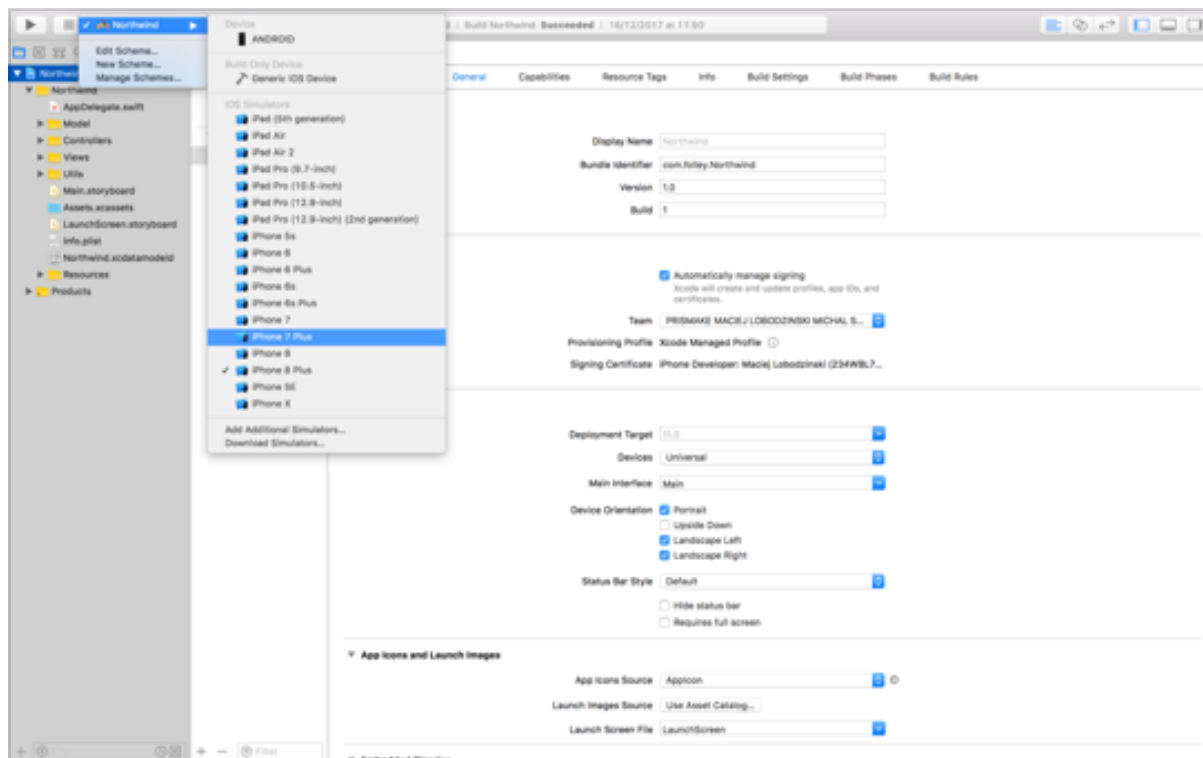
6. Powinniśmy zaobserwować zmieniający się pasek ładowania stanu z napisem "Building Northwind" → "Launching Northwind" → "Running Northwind on iPhone 6".
7. Automatycznie uruchomi się zewnętrzny program "Symulator" z uruchomioną aplikacją Northwind.

8. Aplikacja włączy się ekranem startowym, tak zwanym **Launch screen**, który jest pokazywany w czasie ładowania zasobów do pamięci podręcznej telefonu.
9. Pierwszym ekranem który pokazujemy po faktycznym starcie aplikacji jest ekran wyboru kategorii. Dla celów projektu dodaliśmy w lewym górnym rogu przycisk *import*

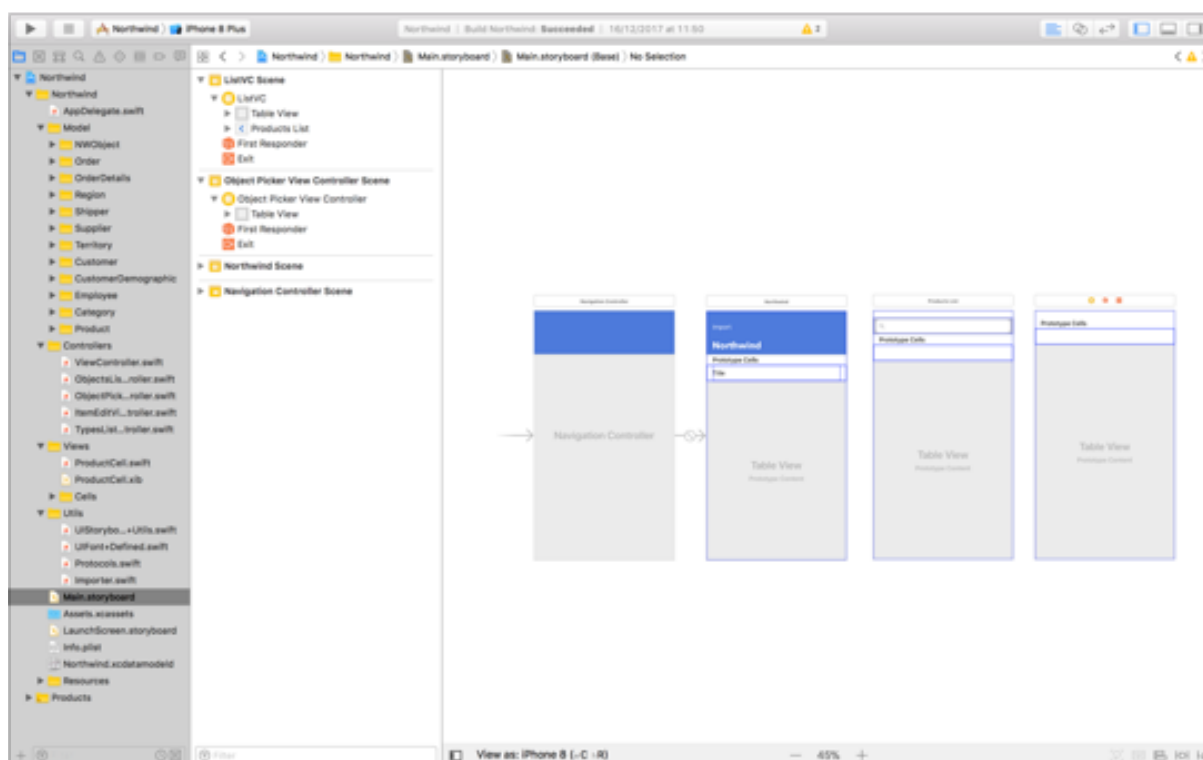


Opis IDE Xcode

1. Po lewej stronie widzimy hierarchię plików
2. W lewym górnym rogu znajduje się panel do budowania projektu i wyboru urządzenia na którym będziemy testować



3. Plik Main.storyboard zawiera widoki zaprogramowane w programie Interface Builder zapewnianym przez firmę Apple w swoim IDE Xcode. Jest to graficzna forma prezentowania XMLA, którą w czasie rzeczywistym można edytować i wizualnie wyświetlać w takiej formie, w jakiej user zobaczy ją finalnie w aplikacji.

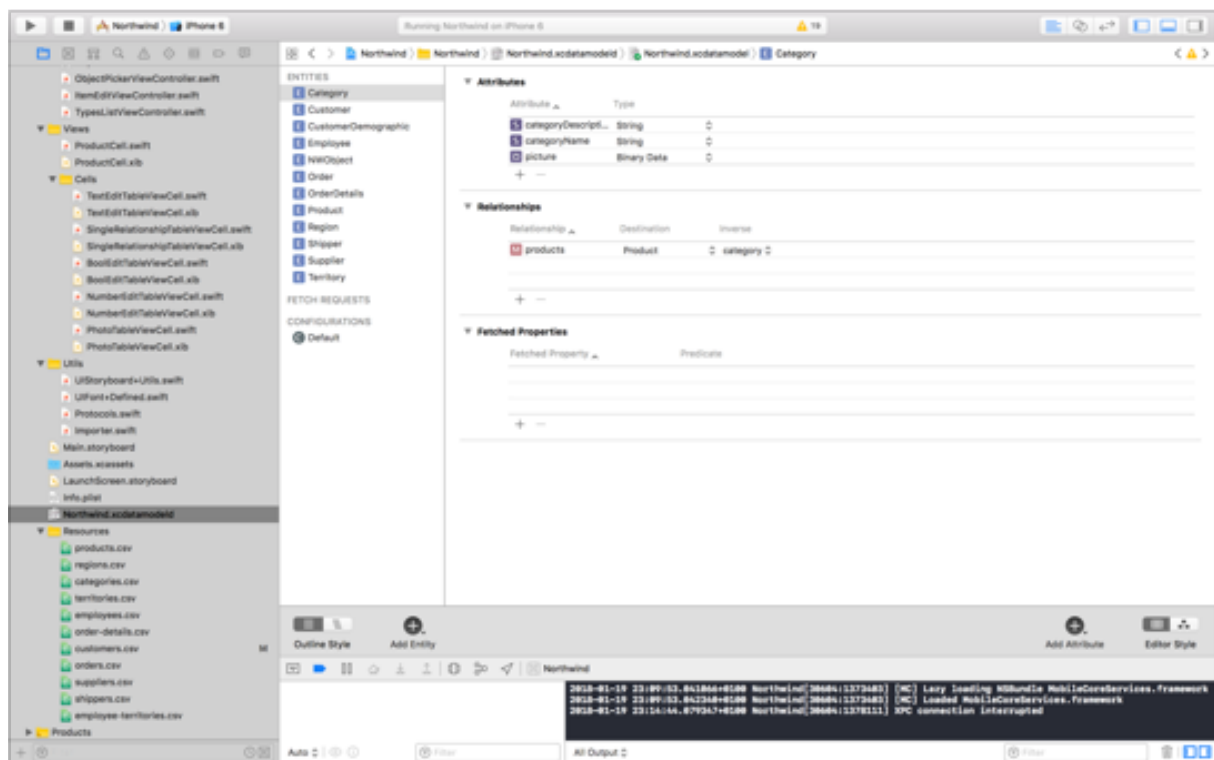


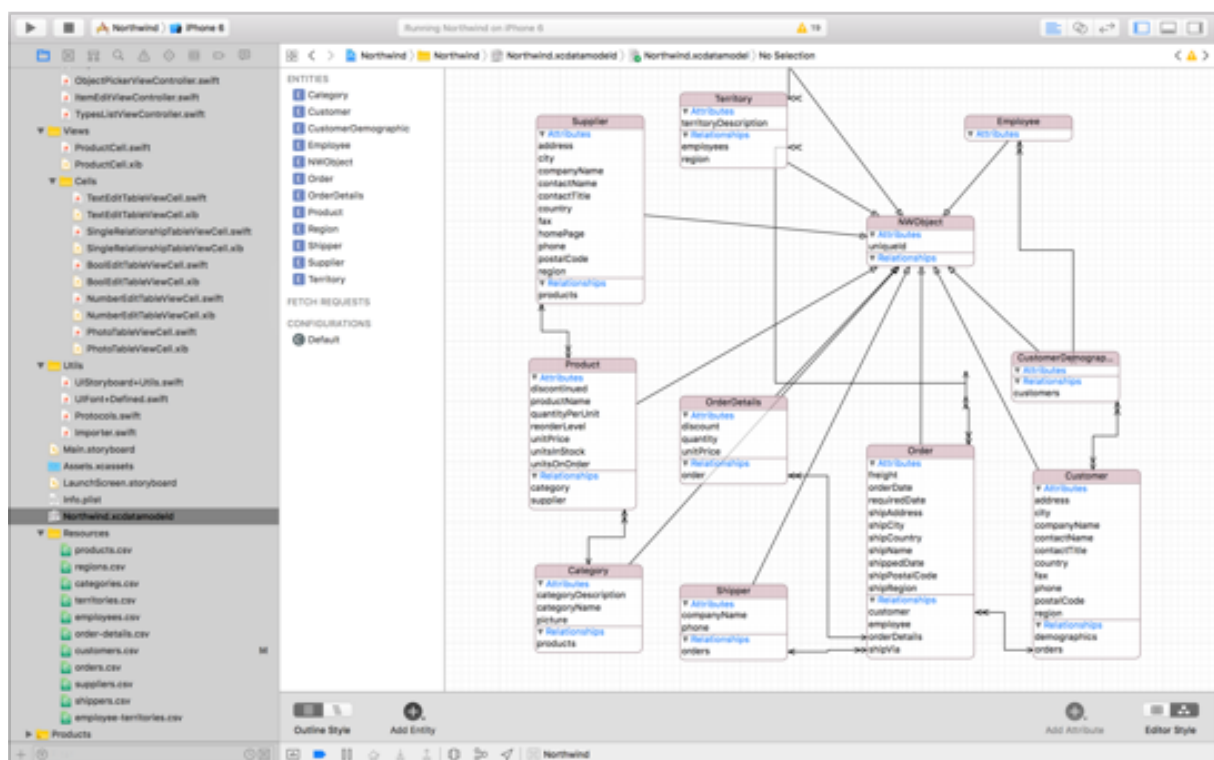
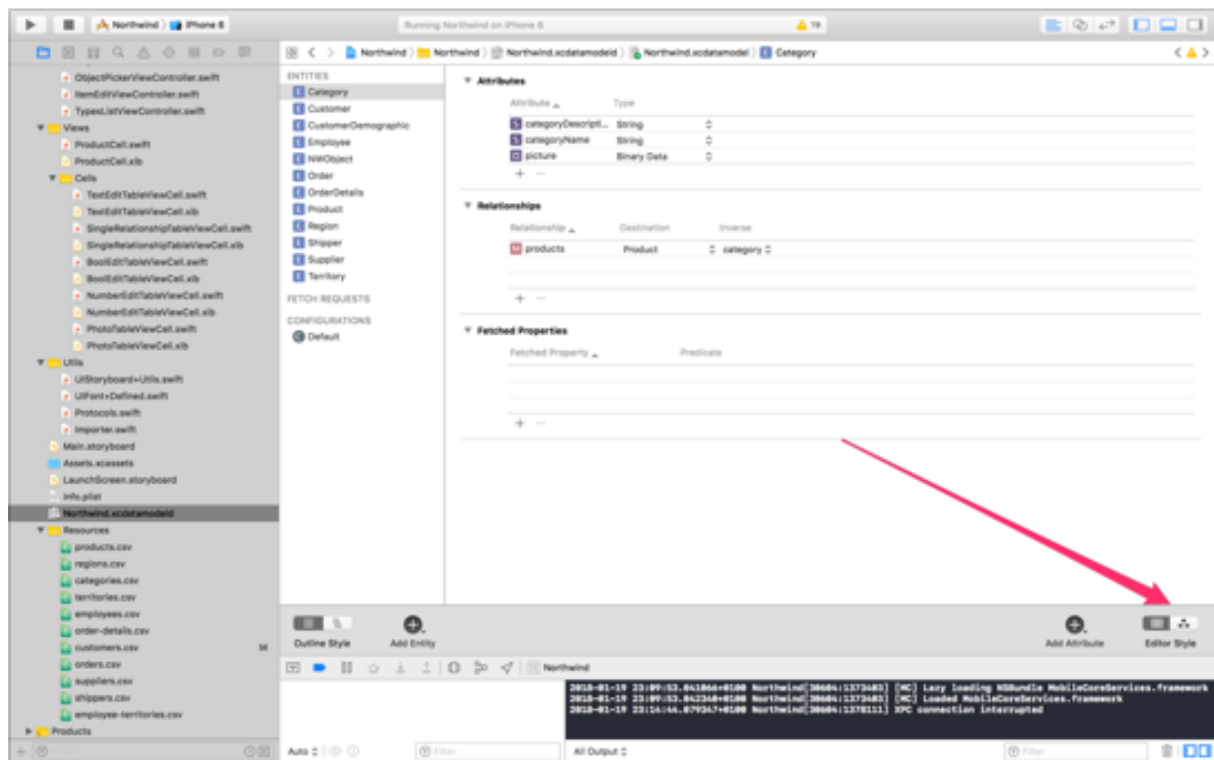
AppDelegate to pierwszy plik który jest wykonywany po uruchomieniu aplikacji. Tam ładowany jest pierwszy widok, ustawiany jest `tintColor` czyli kolor przewodni aplikacji oraz między innymi tworzona jest baza danych. W tym miejscu też inicjalizujemy potrzebne zewnętrzne biblioteki.

Baza danych

Baza danych jest trzymana w pliku wygenerowanym przez CoreData o rozszerzeniu `.xcdatamodeld`.

W projekcie możemy zlokalizować plik `Northwind.xcdatamodeld` by zobaczyć listę Entities oraz Relationships (związki) między nimi.

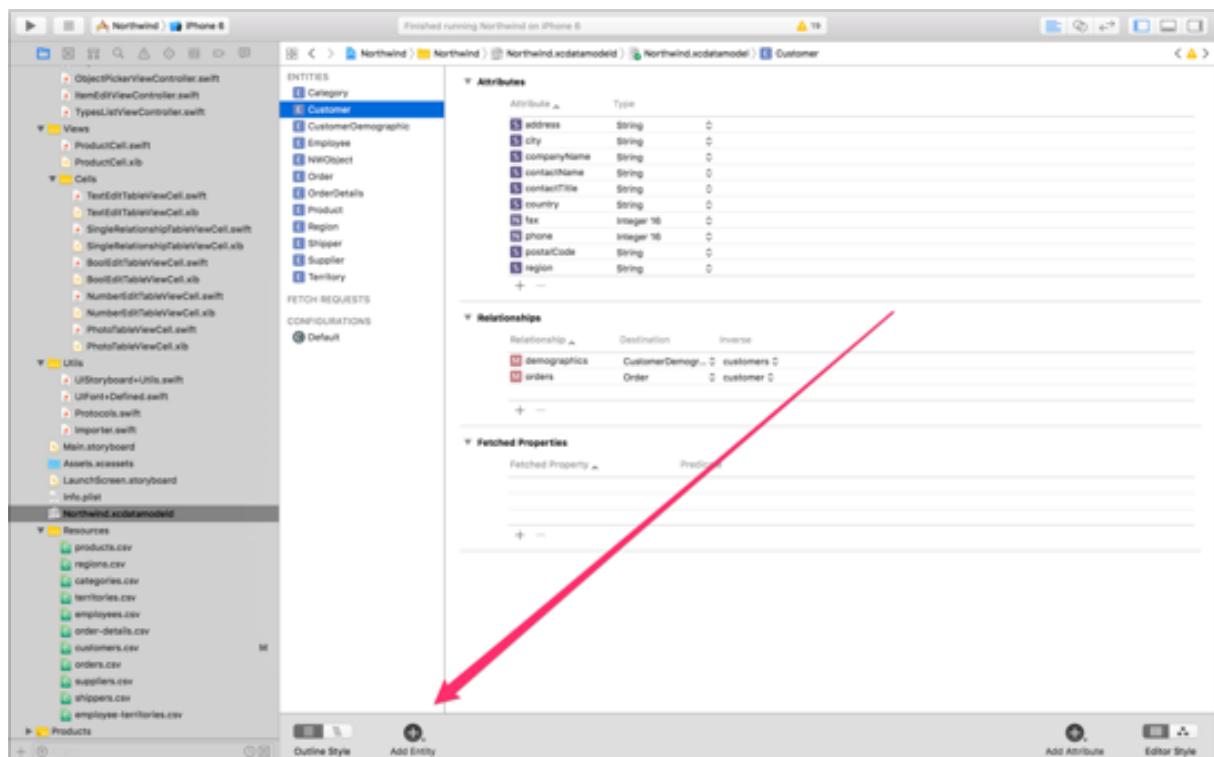




W pliku AppDelegate.swift* Core Data automatycznie generuje Stack kodu który tworzy persistentContainer oraz zapisuje dane.

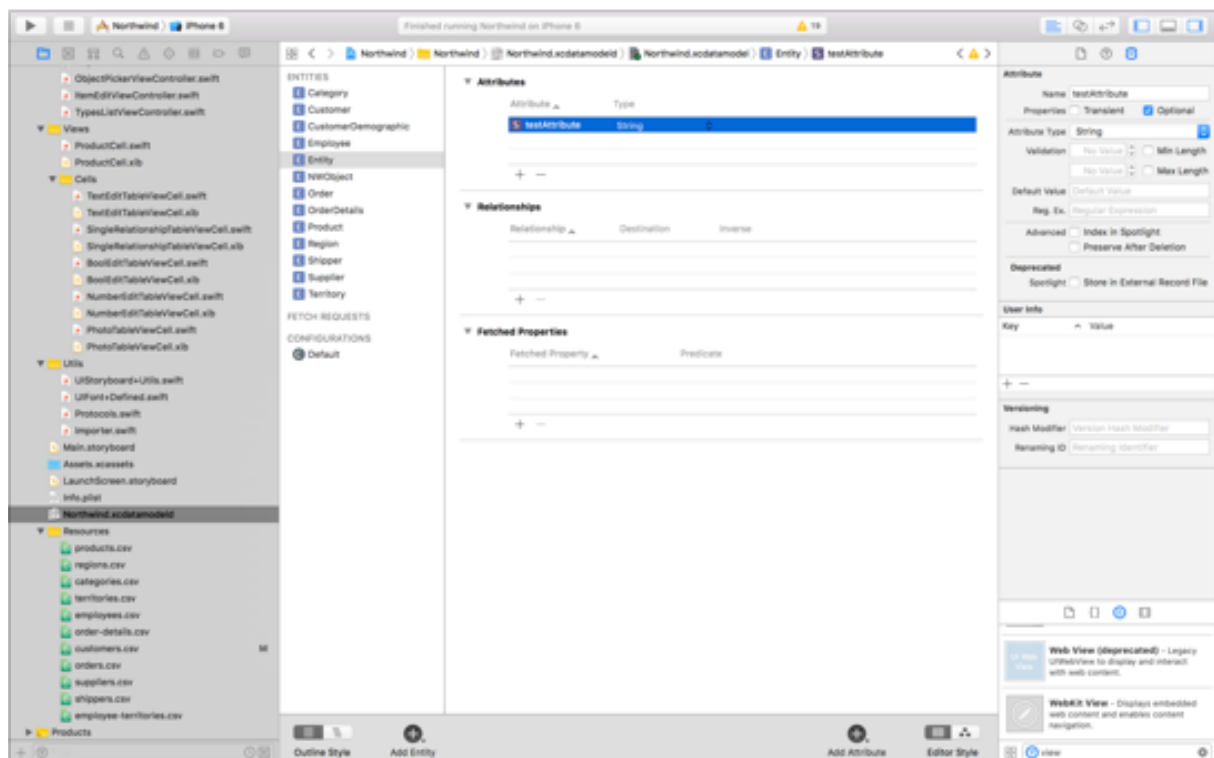
AppDelegate to pierwszy plik który jest wykonywany po uruchomieniu aplikacji.

Dodawanie Entity do bazy danych



Co doda Entity do listy Entities. Klikając plus w liście attributes możemy dodać atrybut (pamiętając, że nazwa powinna zaczynać się z małej listery) oraz wybrać jego typ.

Po prawej stronie znajduje się panel dodatkowych ustawień.



Usuwanie elementu z bazy danych

Poprzez kliknięcie ikonki kosza na śmieci zlokalizowanej w Navigation Barze Navigation Controllera uruchamiamy fragment kodu:

```
@objc private func didTapDelete(_ sender: Any) {  
    let object = configuration.object  
    object.managedObjectContext?.delete(object)  
    object.managedObjectContext?.northwindSave()  
    self.navigationController?.popViewController(animated: true)  
}
```