Pierwsze kroki z aparatami i bibliotekami zdjęć

Utworzyłeś klasę RatingsView i dodałeś ją do ekranów Szczegóły restauracji i Formularz recenzji. Umożliwiłeś również użytkownikowi przesłanie recenzji za pomocą ekranu Formularz recenzji, chociaż przesłana recenzja jest na razie drukowana tylko w obszarze debugowania. Tu zakończysz implementację ekranu Filtr zdjęć, dzięki czemu będziesz mógł pobrać zdjęcie z aparatu lub biblioteki zdjęć i zastosować do niego filtr. Zaczniesz od zaimportowania pliku .plist zawierającego filtry, których chcesz użyć, a następnie utworzysz klasę obiektu filtru do przechowywania danych filtru i utworzysz klasę menedżera danych, aby odczytać plik .plist i wypełnić tablicę obiektów filtrów. Następnie utworzysz protokół z metodą stosowania filtrów do obrazów. Następnie utworzysz kontrolery widoku dla ekranu Photo Filter i widoku kolekcji w nim, zaimplementujesz protokół UIImagePickerDelegate, który umożliwia pobieranie zdjęć z aparatu lub biblioteki zdjęć, oraz zaimplementujesz metody zastosowania wybranego filtra do zdjęcie. Pamiętaj, że zdjęcie nie zostanie zapisane. W następnym rozdziale dowiesz się, jak zapisywać recenzje i zdjęcia. Pod koniec dowiesz się, jak importować zdjęcia do własnych aplikacji i jak stosować do nich filtry.

Zostaną omówione następujące tematy:

• Zrozumienie filtrów

• Tworzenie obiektów modelu dla ekranu Filtr zdjęć

• Tworzenie protokołu filtrowania obrazu

• Tworzenie klas dla ekranu Filtr zdjęć

• Implementacja protokołu delegata selektora obrazów

• Uzyskanie pozwolenia na korzystanie z aparatu lub biblioteki zdjęć

Zrozumienie filtrów

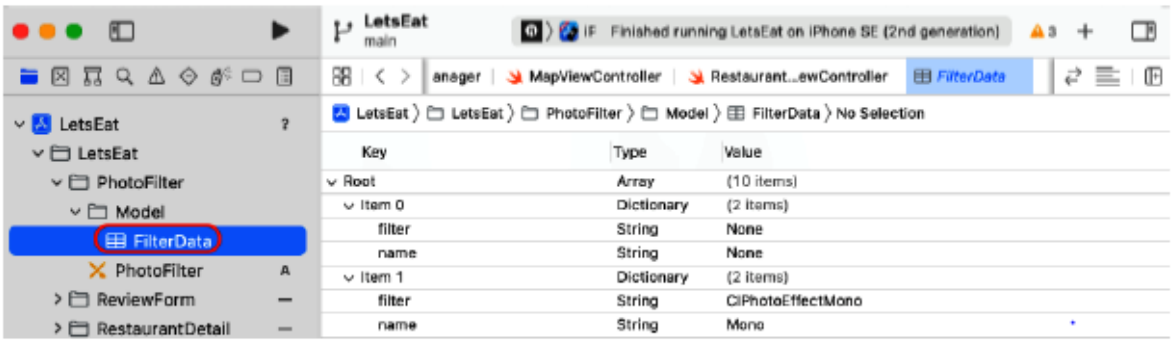
iOS ma szereg wbudowanych filtrów, których można używać do ulepszania zdjęć. Filtry te są dostępne w bibliotece Core Image. Core Image to technologia przetwarzania i analizy obrazu, która zapewnia wysoką wydajność przetwarzania obrazów nieruchomych i wideo. W Core Image dostępnych jest ponad 170 filtrów, które umożliwiają zastosowanie do zdjęć szerokiej gamy fajnych efektów. W przypadku tej aplikacji będziesz używać tylko 10 filtrów. Szczegóły tych filtrów znajdują się w pliku .plist. Zaimportuj ten plik do swojej aplikacji, wykonując następujące czynności:

1. Jeśli jeszcze tego nie zrobiłeś, pobierz i rozpakuj pakiet kodu. Plik FilterData.plist.

2. W nawigatorze projektów utwórz nową grupę w folderze PhotoFilter i nazwij ją Model.

3. Przeciągnij plik FilterData.plist do folderu Model. Upewnij się, że Kopiuj elementy w razie potrzeby jest zaznaczone i kliknij Zakończ.

4. Kliknij FilterData.plist w nawigatorze projektów, aby zobaczyć, co zawiera:



Jak widać, FilterData.plist to tablica słowników. Każdy słownik zawiera nazwę filtra i opisową etykietę. W następnej sekcji zobaczysz, jak możesz wykorzystać informacje z FilterData.plist w swojej aplikacji.

Tworzenie obiektów modelu dla ekranu Filtr zdjęć

Aby uzyskać informacje z FilterData.plist do swojej aplikacji, utworzysz strukturę FilterItem, która może przechowywać szczegółowe informacje o filtrze oraz klasę menedżera danych FilterManager, która załaduje plik FilterData.plist i utworzy tablicę wystąpień FilterItem . Jest to metoda podobna do metody używanej do wczytywania informacji o kuchni i lokalizacji do Twojej aplikacji. Zacznijmy od stworzenia struktury FilterItem. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Model w folderze PhotoFilter i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz Swift File i kliknij Dalej.

3. Nazwij ten plik FilterItem. Kliknij Utwórz. Plik FilterItem pojawi się w nawigatorze projektów.

4. Wewnątrz pliku FilterItem wpisz następujący kod po instrukcji import, aby zadeklarować i zdefiniować strukturę FilterItem:

```
struct FilterItem {

let filter: String?

let name: String?

init(dict: [String: String]) {

self.filter = dict["filter"]

self.name = dict["name"]

}

}
```

Ta struktura ma dwie właściwości i inicjator. Właściwość filter będzie przechowywać nazwy filtrów, a właściwość name będzie przechowywać krótki opis filtrów. Inicjator przyjmuje słownik jako parametr do ustawiania właściwości nazwy i filtra podczas tworzenia wystąpienia tej klasy. Teraz, po utworzeniu klasy FilterItem, utworzysz klasę menedżera danych, FilterDataManager. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Model w folderze PhotoFilter i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz Swift File i kliknij Dalej.

3. Nazwij ten plik FilterDataManager. Kliknij Utwórz. Plik FilterDataManager pojawi się w nawigatorze projektu.

4. Wewnątrz pliku FilterDataManager wpisz następujący kod po instrukcji import, aby zadeklarować i zdefiniować klasę FilterDataManager:

```
class FilterDataManager: DataManager {

func fetch() -> [FilterItem] {

var filterItems: [FilterItem] = []

for data in loadPlist(file: "FilterData") {

filterItems.append(FilterItem(dict:

data as! [String: String]))

}

return filterItems

}

}
```

Klasa FilterDataManager przyjmuje protokół DataManager utworzony wcześniej w rozdziale 16, Pierwsze kroki z MapKit. Wywołanie metody fetch() ładuje dane z pliku FilterData.plist, tworzy tablicę instancji FilterItem i zwraca ją.

W następnej sekcji utworzysz protokół z metodą zastosowania filtru do obrazu.

Tworzenie protokołu filtrowania obrazu

Potrzebujesz sposobu na zastosowanie filtra do obrazu. Stworzysz protokół ImageFiltering, który implementuje metodę apply(filter:to:), aby to zrobić. Każda klasa, która przyjmie ten protokół, będzie miała dostęp do tej metody, która stosuje określony filtr do obrazu. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem folder PhotoFilter i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz Swift File i kliknij Dalej.

3. Nazwij ten plik ImageFiltering. Kliknij Utwórz. Plik filtrowania obrazów pojawi się w nawigatorze projektów.

4. Zmodyfikuj kod w tym pliku, aby zadeklarować i zdefiniować protokół mageFiltering:

```
import UIKit

import CoreImage

protocol ImageFiltering {

func apply(filter: String, originalImage:

UIImage) -> UIImage

}

extension ImageFiltering {
```

```
func apply(filter: String, originalImage:
UIImage) -> UIImage {
let initialCIImage = CIImage(image:
originalImage, options: nil)
let originalOrientation =
originalImage.imageOrientation
guard let ciFilter = CIFilter(name:
filter) else {
print("filter not found")
return originalImage
}
ciFilter.setValue(initialCIImage, forKey:
kCIInputImageKey)
let context = CIContext()
let filteredCIImage =
(ciFilter.outputImage)!
let filteredCGImage =
context.createCGImage(filteredCIImage,
from: filteredCIImage.extent)
return UIImage(cgImage: filteredCGImage!,
scale: 1.0, orientation:
originalOrientation)
}
}
```

Rozbijmy to:

impor UIKit

Platforma UIKit zapewnia wymaganą infrastrukturę dla Twojej aplikacji na iOS. Importujesz UIKit zamiast Foundation, ponieważ obsługa klasy UIImage nie jest dostępna w Foundation.

import CoreImage

Core Image to technologia przetwarzania i analizy obrazu, która zapewnia wysoką wydajność przetwarzania obrazów nieruchomych i wideo. Importujesz CoreImage, ponieważ jest to wymagane, aby uzyskać dostęp do wbudowanych filtrów fotograficznych.

protocol ImageFiltering {

func apply(filter: String, originalImage:

UIImage) -> UIImage

}

Tutaj deklarujesz protokół o nazwie ImageFiltering. Protokół ten określa metodę apply(filter:originalImage:), która przyjmuje nazwę filtra i obraz jako parametry.

extension ImageFiltering {

func apply(filter: String, originalImage:

UIImage) -> UIImage {

To rozszerzenie protokołu ImageFiltering zawiera

implementacja metody apply(filter:originalImage:). Oznacza to, że każda klasa, która przyjmie protokół ImageFiltering, będzie mogła wykonać tę metodę.

let initialCIImage = CIImage(image:

originalImage, options: nil)

Ta instrukcja konwertuje oryginalny obraz na wystąpienie CIImage, dzięki czemu można zastosować do niego filtry i przypisuje go do initialCIImage.

let originalOrientation =

originalImage.imageOrientation

Ta instrucja przechowuje oryginalną orientację obrazu w originalOrientation.

guard let ciFilter = CIFilter(name: filter)

else {

print("filter not found")

return originalImage

}

Ta instrukcja Guard pobiera filtr o tej samej nazwie co filter i przypisuje go do ciFilter i zwraca oryginalny obraz, jeśli filtr nie zostanie znaleziony.

ciFilter.setValue(initialCIImage, forKey:

kCIInputImageKey)

let context = CIContext()

let filteredCIImage =

(ciFilter.outputImage)!

Te instrukcje stosują wybrany filtr do initialCIImage i przechowują wynik w filterCIImage.

let filteredCGImage =

context.createCGImage(filteredCIImage, from:

filteredCIImage.extent)

return UIImage(cgImage: filteredCGImage!,

scale: 1.0, orientation:

originalOrientation)

Te instrukcje konwertują wystąpienie CIImage przechowywane w filterCIImage z powrotem do wystąpienia UIImage i zwracają je.

To kończy implementację protokołu ImageFiltering i metody apply(filter:originalImage:). W tym momencie masz następujące:

• FilterData.plist, który zawiera dane filtrów zdjęć w Twojej aplikacji.

• FilterItem, klasa, która może przechowywać filtr i opis filtra.

• FilterDataManager, klasa menedżera danych, która ładuje dane z

FilterData.plist i generuje tablicę instancji FilterItem.

• ImageFiltering, protokół zawierający metodę apply(filter:originalImage:), która stosuje filtr do obrazu.

W następnej sekcji utworzysz klasy dla elementów interfejsu użytkownika na ekranie Photo Filter, co pozwoli Ci zarządzać tym ekranem i widokiem kolekcji wewnątrz niego.

Tworzenie klas dla ekranu Filtr zdjęć

Do tej pory zaimportowałeś plik FilterData.plist do swojej aplikacji, utworzyłeś klasy FilterItem i FilterDataManager oraz utworzyłeś protokół ImageFiltering. W tej sekcji skonfigurujesz klasy dla ekranu Filtra Zdjęć, który pozwoli Ci zarządzać tym ekranem i znajdującym się w nim widokiem kolekcji. Pamiętaj, że dodałeś plik scenorysu PhotoFilter do swojego projektu w rozdziale 16, Pierwsze kroki z MapKit. Zawiera scenę składającą się z dużego widoku obrazu, który będzie zawierał wybrane przez użytkownika zdjęcie oraz widok kolekcji, w którym będą wyświetlane podglądy filtrów. Poniższy zrzut ekranu pokazuje, jak to będzie wyglądać po zakończeniu implementacji:

Ten ekran działa w następujący sposób. Po dotknięciu przycisku Dodaj zdjęcie na ekranie szczegółów restauracji i wybraniu zdjęcia, pojawi się ekran Filtr zdjęć, pokazujący wybrane zdjęcie z przewijaną listą filtrów tuż pod nim. Każdy filtr na liście przewijanej jest wyświetlany w komórce widoku kolekcji. Dotknięcie filtra na przewijanej liście spowoduje zastosowanie wybranego filtra do zdjęcia. W następnej sekcji utworzysz i skonfigurujesz klasę do zarządzania komórkami widoku kolekcji. Każda komórka wyświetli miniaturowy podgląd tego, jak wygląda zdjęcie z zastosowanym filtrem.

Tworzenie klasy dla komórek widoku kolekcji

Ekran Filtr zdjęć zapewnia interfejs użytkownika, który pozwala użytkownikowi wybrać filtr, który ma zostać zastosowany do zdjęcia. Widok kolekcji wyświetli miniatury podglądu tego, jak wygląda zdjęcie z zastosowanym filtrem. Jeśli klikniesz plik scenorysu PhotoFilter w nawigatorze projektów, zobaczysz, że widok kolekcji jest już obecny w scenie kontrolera widoku, ale nie ma możliwości ustawienia zawartości komórek widoku kolekcji. Teraz utworzysz klasę, aby nimi zarządzać. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem folder PhotoFilter i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz klasę Cocoa Touch i kliknij Dalej.

3. Skonfiguruj plik w następujący sposób:

Class: FilterCell

Subclass: UICollectionViewCell

Also create XIB: Unchecked

Language: Swift

Click Next.

4. Kliknij Utwórz. Plik FilterCell pojawi się w nawigatorze projektów.

5. Dodaj następujący kod do tego pliku, aby zadeklarować i zdefiniować klasę FilterCell:

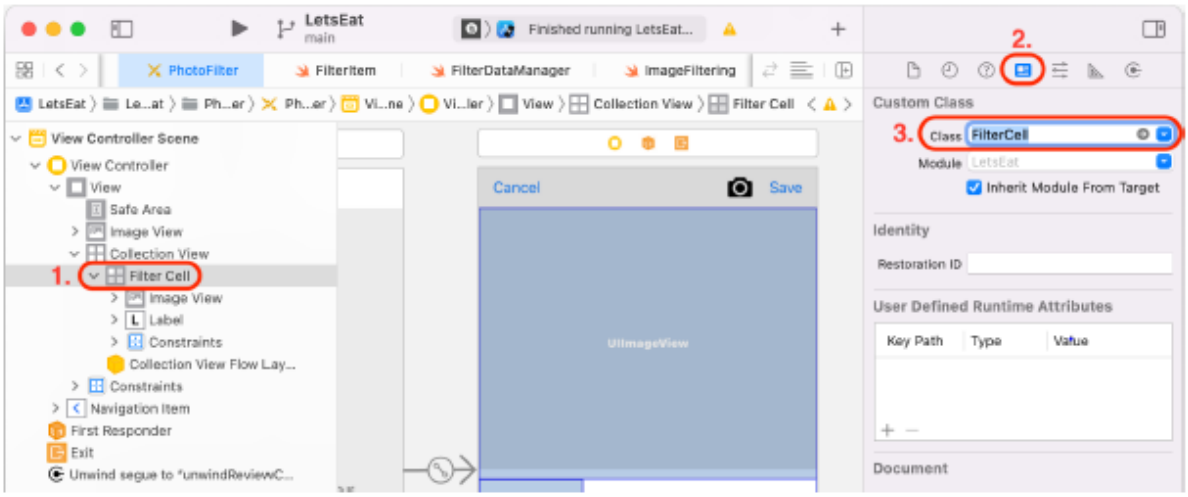
import UIKit

```
class FilterCell: UICollectionViewCell {
@IBOutlet var nameLabel: UILabel!
@IBOutlet var thumbnailImageView: UIImageView!
override func awakeFromNib() {
super.awakeFromNib()
thumbnailImageView.layer.cornerRadius = 9
thumbnailImageView.layer.masksToBounds = true
}
extension FilterCell: ImageFiltering {
func set(filterItem: FilterItem,
imageForThumbnail: UIImage) {
nameLabel.text = filterItem.name
if let filter = filterItem.filter {
if filter != "None" {
let filteredImage = apply(filter:
filter, originalImage:
imageForThumbnail)
thumbnailImageView.image =
filteredImage
} else {
thumbnailImageView.image =
imageForThumbnail
}
}
}
}
```

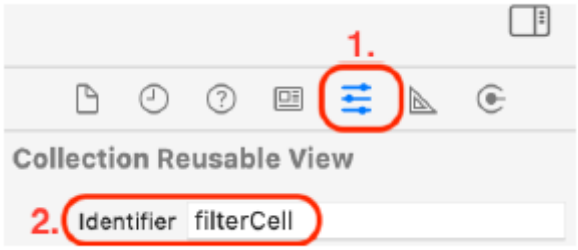
Klasa FilterCell ma dwie właściwości: etykietę nameLabel i widok obrazu thumbnailImageView. Na etykiecie zostanie wyświetlona nazwa filtra, a w widoku obrazu zostanie wyświetlony miniaturowy podgląd filtra. Ta klasa zawiera również dwie metody, awakeFromNib() i set(filterItem:imageForThumbnail:). Metoda awakeFromNib() jest wywoływana po załadowaniu instancji FilterCell, a dwie zawarte w niej instrukcje zaokrąglają rogi widoku obrazu. Metoda set(filterItem:imageForThumbnail:) przyjmuje instancje UIImage i FilterItem jako parametry, przypisuje właściwość name instancji FilterItem do nameLabel, stosuje filtr określony przez właściwość filter do instancji UIImage i przypisuje obraz z zastosowanym filtrem thumbnailImageView.

6. Kliknij plik scenorysu PhotoFilter w nawigatorze projektów.

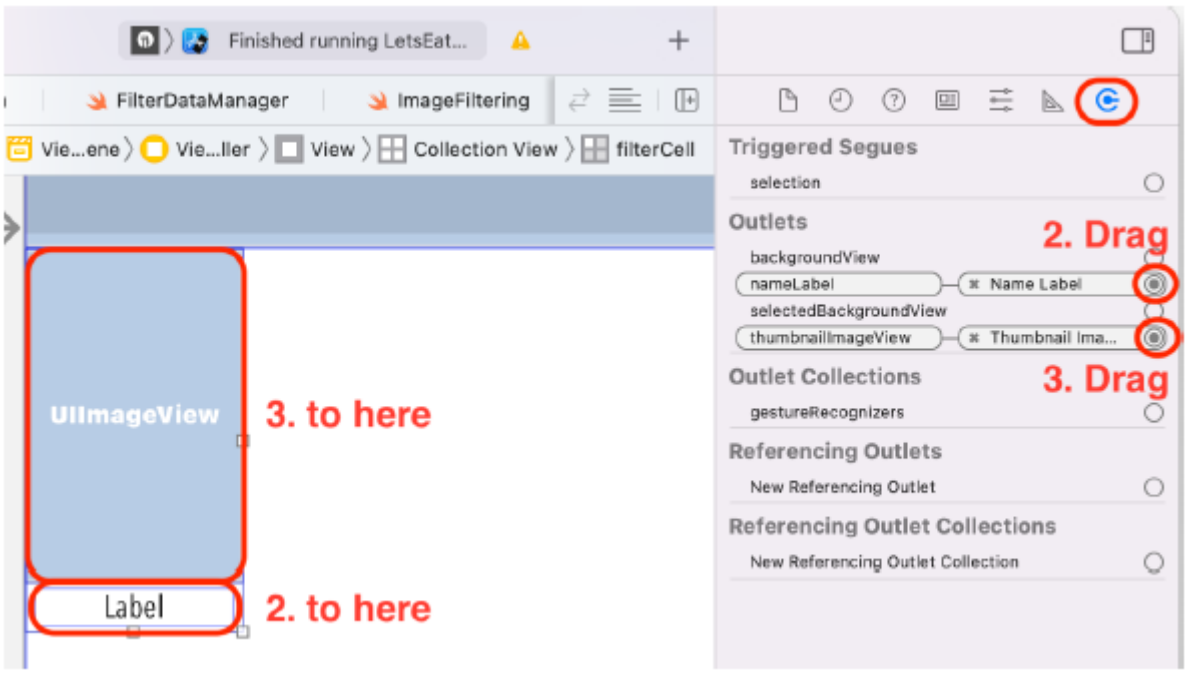
7. W konspekcie dokumentu wybierz Komórkę widoku kolekcji w scenie kontrolera widoku. Kliknij w przycisk Inspektora tożsamości. W sekcji Custom Class ustaw Class na FilterCell:



8. Kliknij przycisk Inspektora atrybutów. Ustaw identyfikator na filterCell:



9. Kliknij przycisk Inspektora połączeń. Połącz wyloty nameLabel i thumbnailImageView z odpowiadającymi im elementami interfejsu użytkownika, jak pokazano:



Zakończyłeś konfigurowanie komórek widoku kolekcji. W następnej sekcji utworzysz kontroler widoku dla ekranu Filtr zdjęć. To pozwoli Ci wybrać zdjęcie i wybrać filtr, który zostanie do niego zastosowany.

Tworzenie kontrolera widoku dla ekranu Filtr zdjęć

Do tej pory utworzyłeś klasę FilterCell do zarządzania komórkami widoku kolekcji na ekranie Filtr zdjęć. Teraz utworzysz kontroler widoku do zarządzania zawartością tego ekranu. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem folder PhotoFilter i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz klasę Cocoa Touch i kliknij Dalej.

3. Skonfiguruj plik w następujący sposób:

Class: PhotoFilterViewController

Subclass: UIViewController

Also create XIB: Unchecked

Language: Swift

Click Next.

4. Kliknij Utwórz. Plik PhotoFilterViewController pojawi się w nawigatorze projektów. Usuń cały kod wzorcowy po metodzie viewDidLoad().

5. Dodaj następujący kod do pliku, aby zadeklarować i zdefiniować klasę PhotoFilterViewController oraz jej właściwości:

```
import UIKit

import AVFoundation

class PhotoFilterViewController: UIViewController {
```

@IBOutlet var mainImageView: UIImageView!

@IBOutlet var collectionView: UICollectionView!

private let manager = FilterDataManager()

var selectedRestaurantID: Int?

private var mainImage: UIImage?

private var thumbnail: UIImage?

private var filters: [FilterItem] = []

override func viewDidLoad() {

super.viewDidLoad()

initialize()

}

}

Rozbijmy to:

import AVFoundation

To oświadczenie importuje strukturę AVFoundation. Ta struktura zawiera metody przechwytywania, przetwarzania, syntezowania, kontrolowania, importowania i eksportowania mediów audiowizualnych na platformach Apple.

class PhotoFilterViewController: UIViewController {

Ta instrukcja deklaruje klasę PhotoFilterViewController, podklasę klasy UIViewController.

@IBOutlet var mainImageView: UIImageView!

To jest wyjście dla widoku obrazu, który wyświetli wybrane przez użytkownika zdjęcie z zastosowanym filtrem.

@IBOutlet var collectionView: UICollectionView!

To jest wyjście dla widoku kolekcji, w którym będą wyświetlane miniatury podglądu każdego filtra.

private let manager = FilterDataManager()

Ta instrukcja przypisuje instancję klasy FilterDataManager do właściwości manager.

var selectedRestaurantID:Int?

Każda restauracja posiada unikalny identyfikator numeryczny. Ta właściwość służy do przechowywania tego identyfikatora. W następnym rozdziale zobaczysz, jak jest używany podczas przechowywania zdjęć przy użyciu danych podstawowych.

private var mainImage: UIImage?

Ta właściwość przechowuje zdjęcie wybrane przez użytkownika.

private var thumbnail: UIImage?

Ta właściwość przechowuje miniaturę zdjęcia wybranego przez użytkownika.

private var filters: [FilterItem] = []

Ta właściwość przechowuje tablicę wystąpień FilterItem dostarczonych przez menedżera.

override func viewDidLoad() {

super.viewDidLoad()

initialize()

}

Ta metoda wywołuje metodę initialize(), gdy instancja PhotoFilterViewController ładuje swój widok. Zauważ, że wygeneruje to błąd, ponieważ initialize() nie zostało jeszcze zaimplementowane.

6. Tak jak poprzednio, do uporządkowania kodu użyjesz rozszerzeń. Dodaj następujące prywatne rozszerzenie zawierające metodę initialize() po zamykającym nawiasie klamrowym:

// MARK: - Private Extension

private extension PhotoFilterViewController {

func initialize() {

setupCollectionView()

checkSource()

}

}

To rozszerzenie zawiera implementację metody initialize(), która wywołuje dwie inne metody. setupCollectionView() konfiguruje widok kolekcji używany do wyświetlania listy filtrów. checkSource() sprawdza status autoryzacji użytkownika do korzystania z kamery. Pamiętaj, że spowodują one wygenerowanie błędów, ponieważ nie zostały jeszcze zaimplementowane. Te metody zaimplementujesz w następnym kroku.

7. Zaimplementuj metody setupCollectionView() i checkSource() w rozszerzeniu prywatnym po metodzie initialize():

func setupCollectionView() {

let layout = UICollectionViewFlowLayout()

layout.scrollDirection = .horizontal

layout.sectionInset = UIEdgeInsets(top: 7,

left: 7, bottom: 7, right: 7)

layout.minimumInteritemSpacing = 0

layout.minimumLineSpacing = 7

collectionView.collectionViewLayout = layout

collectionView.dataSource = self

```
collectionView.delegate = self
}
func checkSource() {
let cameraMediaType = AVMediaType.video
let cameraAuthorizationStatus =
AVCaptureDevice.authorizationStatus(for:
cameraMediaType)
switch cameraAuthorizationStatus {
case .notDetermined:
AVCaptureDevice.requestAccess(for:
cameraMediaType) { granted in
if granted {
DispatchQueue.main.async {
self.showCameraUserInterface()
}
}
}
case .authorized:
self.showCameraUserInterface()
default:
break
}
}
```

Rozbijmy to:

setupCollectionView()

Ustawia widok kolekcji używany do wyświetlania podglądów miniatur filtrów. W tym miejscu tworzysz wystąpienie UICollectionViewFlowLayout, ustawiasz kierunek przewijania, wstawki sekcji, odstępy między elementami i odstępy między wierszami oraz przypisujesz je do widoku kolekcji. Następnie ustaw klasę PhotoFilterViewController jako delegata i źródło danych dla tego widoku kolekcji. Pamiętaj, że ustawiasz pełnomocnika i DataSource programowo, a nie za pomocą scenorysu; każde podejście jest akceptowalne. Nie przejmuj się błędami, pojawiają się one, ponieważ nie zostały jeszcze przyjęte protokoły UICollectionViewDataSource i UICollectionViewDelegate dla tej klasy. Naprawisz to później.

checkSource()

Sprawdza status autoryzacji użytkownika do korzystania z kamery. Możliwe przypadki są następujące:

.notDetermined oznacza, że użytkownik nie został poproszony o dostęp do kamery.

.authorized oznacza, że użytkownik wcześniej przyznał dostęp do kamery.

.restricted oznacza, że użytkownikowi nie można przyznać dostępu z powodu ograniczeń ustawionych na urządzeniu.

.denied oznacza, że użytkownik wcześniej odmówił dostępu do aplikacji z aparatu.

Jeśli stan to .notDetermined, aplikacja poprosi użytkownika o uprawnienie, a jeśli uprawnienie zostanie udzielone, wywoływana jest metoda showCameraUserInterface(). Jeśli stan to .authorized, wywoływana jest metoda showCameraUserInterface().

Zauważ, że spowoduje to wygenerowanie błędu, ponieważ funkcja showCameraUserInterface() nie została jeszcze zaimplementowana. Jeśli status to .restricted lub .denied, jest to wartość domyślna: case i metoda kończy działanie.

8. Wymaganych jest jeszcze kilka metod pomocniczych. Dodaj następujący kod do prywatnego rozszerzenia, aby zaimplementować je po metodzie checkSource():

```
func showApplyFilterInterface() {

filters = manager.fetch()

if let mainImage = self.mainImage {

mainImageView.image = mainImage

collectionView.reloadData()

}

}

@IBAction func onPhotoTapped(_ sender: Any) {

checkSource()

}
```

Rozbijmy to:

showApplyFilterInterface()

Ta metoda zostanie wywołana po wybraniu przez użytkownika zdjęcia z aparatu lub biblioteki zdjęć. Wywołuje metodę fetch() instancji FilterManager, która ładuje plik FilterData.plist i umieszcza jego zawartość w tablicy instancji FilterItem. Ta tablica jest następnie przypisywana do właściwości filters instancji PhotoFilterViewController, która później zostanie użyta do wypełnienia widoku kolekcji miniaturowymi podglądami filtrów. Następna instrukcja przypisuje właściwość mainImage instancji PhotoFilterViewController do mainImageView, która jest gniazdem dla widoku obrazu powyżej widoku kolekcji, jeśli ustawiono mainImage. Ostateczna instrukcja mówi widokowi kolekcji, aby sam się przerysował.

onPhotoTapped()

Ta metoda wywołuje zaimplementowaną wcześniej metodę checkSource(), która wywołuje metodę showCameraUserInterface(), jeśli autoryzacja została udzielona. Przypiszesz to później do przycisku aparatu w scenie kontrolera widoku filtra zdjęć.

Ważna informacja

Apple zastrzega, że aplikacje korzystające z aparatu lub biblioteki zdjęć muszą odpowiednio monitować użytkownika. Później zmodyfikujesz aplikację tak, aby wyświetlała okno dialogowe z prośbą o pozwolenie na korzystanie z aparatu lub biblioteki zdjęć, implementując klucze NSCameraUsageDescription i NSMicrophoneUsageDescription w pliku Info.plist.

9. Przyjmiesz protokół UICollectionViewDataSource i zaimplementujesz wymagane metody, aby widok kolekcji wyświetlał miniaturowe podglądy filtrów. Dodaj nowe rozszerzenie po rozszerzeniu prywatnym i zaimplementuj je w następujący sposób:

```
extension PhotoFilterViewController:

UICollectionViewDataSource {

func collectionView(_ collectionView:

UICollectionView, numberOfItemsInSection

section: Int) -> Int {

filters.count

}

func collectionView(_ collectionView:

UICollectionView, cellForItemAt indexPath:

IndexPath) -> UICollectionViewCell {

let cell = collectionView

.dequeueReusableCell

(withReuseIdentifier: "filterCell",

for: indexPath) as! FilterCell

let filterItem = filters[indexPath.row]

if let thumbnail = thumbnail {

cell.set(filterItem: filterItem,

imageForThumbnail: thumbnail)

}

return cell

}

}
```

Poniższe informacje powinny być Ci znajome, ponieważ zrobiłeś to wcześniej, ale omówmy to jeszcze raz:

collectionView(_:numberOfItemsInSection:)

Określa liczbę elementów, które ma wyświetlać widok kolekcji, która jest taka sama jak liczba FilterItems w tablicy filtrów instancji PhotoFilterViewController.

collectionView(_:cellForItemAt:)

Określa, co umieścić w każdej komórce. Tutaj otrzymujesz instancję FilterItem odpowiadającą pozycji komórki w widoku kolekcji i przekazujesz ją wraz z właściwością thumbnail instancji PhotoFilterViewController do metody set(filterItem:imageForThumbnail:), która ustawia obraz i etykietę dla komórki widoku kolekcji .

10. Widok kolekcji został skonfigurowany wcześniej przy użyciu instancji UICollectionViewFlowLayout. Teraz ustawisz rozmiar komórek widoku kolekcji. Dodaj następujące rozszerzenie po rozszerzeniu zawierającym metody źródła danych:

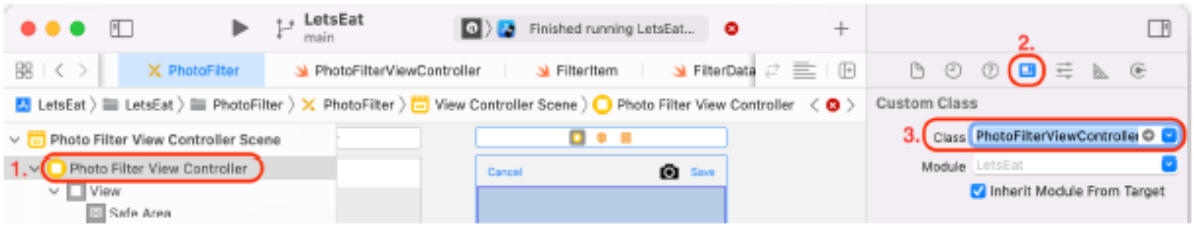
```
extension PhotoFilterViewController:

UICollectionViewDelegateFlowLayout {

func collectionView(_ collectionView:

UICollectionView, layout

collectionViewLayout:

UICollectionViewLayout, sizeForItemAt

indexPath: IndexPath) -> CGSize {

let collectionViewHeight =

collectionView.frame.size.height

let topInset = 14.0

let cellHeight = collectionViewHeight -

topInset

return CGSize(width: 150, height:

cellHeight)

}

}
```

collectionView(_:layout:sizeForItemAt:) zwraca rozmiar każdej komórki widoku kolekcji. Najpierw wysokość widoku kolekcji jest przypisywana do collectionViewHeight. Następnie wartość topInset jest ustawiana na 14,0 punktów. Wysokość komórki widoku kolekcji jest obliczana przez odjęcie topInset od collectionViewHeight. Powoduje to 14-punktową przerwę między górną częścią komórek widoku kolekcji a górną częścią widoku kolekcji. Na koniec instancja CGSize z szerokością ustawioną na 150 punktów i wysokością ustawioną na cellHeight jest zwracana jako rozmiar komórki widoku kolekcji. Wcześniej robiłeś to za pomocą Inspektora rozmiaru; teraz robisz to programowo. Teraz połączysz

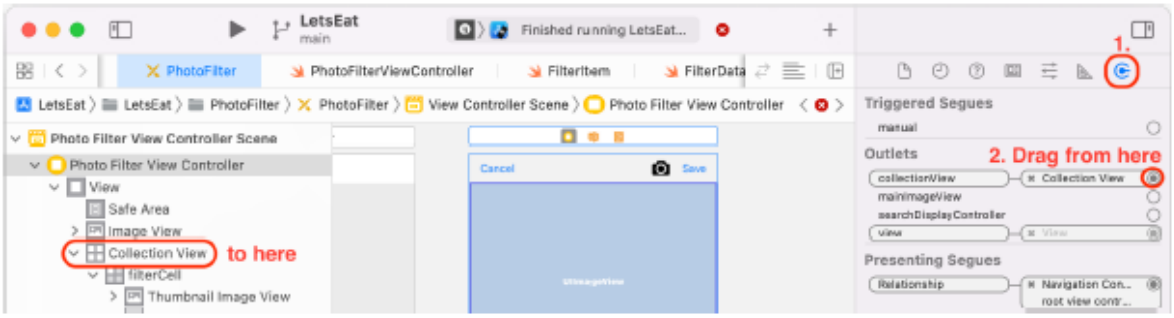
wyjścia i akcje w tej klasie z elementami interfejsu użytkownika w pliku scenorysu PhotoFilter. collectionView jest gniazdem widoku kolekcji, który wyświetla listę filtrów. mainImageView jest dla widoku obrazu tuż nad nim, który pokazuje obraz wybrany przez użytkownika. onPhotoTapped() jest dla przycisku aparatu na pasku nawigacyjnym. Skonfigurujesz również przycisk Anuluj, aby zamknąć ekran Filtr zdjęć.

Wykonaj następujące kroki:

1. Kliknij plik scenorysu PhotoFilter w nawigatorze projektów. Wybierz ikonę kontrolera widoku sceny kontrolera widoku w zarysie dokumentu. Kliknij w przycisk Inspektora tożsamości. W obszarze Klasa niestandardowa ustaw klasę na PhotoFilterViewController:

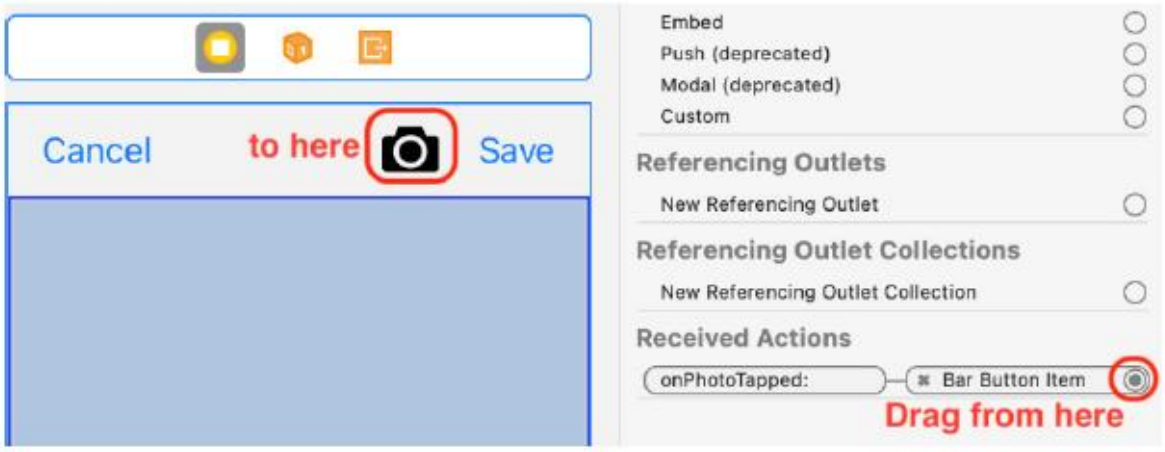


2. Wybierz inspektora połączeń. Kliknij i przeciągnij z gniazdka collectionView do widoku kolekcji w konspekcie dokumentu:

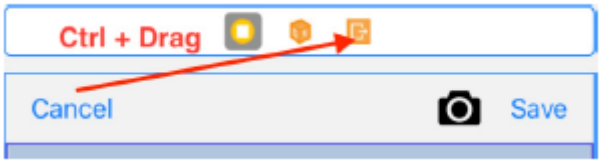


3. Kliknij i przeciągnij z wyjścia mainImageView do widoku obrazu w zarysie dokumentu:

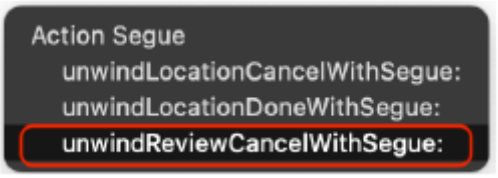
4. Kliknij i przeciągnij z akcji onPhotoTapped: na przycisk aparatu:



5. Przycisk Anuluj służy do wyjścia z tego ekranu, jeśli użytkownik nie chce dokonywać wyboru. Połączysz przycisk Anuluj z metodą rozwijania zaimplementowaną w poprzednim rozdziale, co spowoduje zamknięcie tego ekranu i powrót użytkownika do ekranu szczegółów restauracji. Ctrl + przeciągnij z przycisku Anuluj do ikony Zakończ w Docku sceny:



6. Wybierz unwindReviewCancelWithSegue: w wyskakującym menu:



Wszystkie wyjścia i akcje dla klasy PhotoFilterViewController zostały połączone. Następnie zaimplementujesz następujące metody:

• showCameraUserInterface(), metoda wyświetlająca widok z kamery urządzenia lub bibliotekę zdjęć w interfejsie selektora obrazów.

• Dwie metody protokołu UIImagePickerControllerDelegate, które będą wywoływane po wybraniu obrazu w interfejsie selektora obrazów lub kliknięciu przycisku Anuluj.

Aby zaimplementować metody showCameraUserInterface() i UIImagePickerControllerDelegate, kliknij plik PhotoFilterViewController w nawigatorze projektów i dodaj następujące rozszerzenie po rozszerzeniu UICollectionViewDelegateFlowLayout:

extension PhotoFilterViewController:

UIImagePickerControllerDelegate,

UINavigationControllerDelegate {

func showCameraUserInterface() {

```
let imagePicker = UIImagePickerController()
imagePicker.delegate = self
#if targetEnvironment(simulator)
imagePicker.sourceType =
UIImagePickerController.SourceType.photoLibrary
#else
imagePicker.sourceType =
UIImagePickerController.SourceType.camera
imagePicker.showsCameraControls = true
#endif
imagePicker.mediaTypes = ["public.image"]
imagePicker.allowsEditing = true
self.present(imagePicker, animated: true,
completion: nil)
}
func imagePickerControllerDidCancel(_ picker:
UIImagePickerController) {
picker.dismiss(animated: true, completion: nil)
}
func imagePickerController(_ picker:
UIImagePickerController,
didFinishPickingMediaWithInfo info:
[UIImagePickerController.InfoKey : Any]) {
if let selectedImage =
info[UIImagePickerController.InfoKey
.editedImage] as? UIImage {
self.thumbnail =
selectedImage.preparingThumbnail(of:
CGSize(width: 100, height: 100))
let mainImageViewSize =
mainImageView.frame.size
```

```
self.mainImage =

selectedImage.preparingThumbnail(of:

mainImageViewSize)

}

picker.dismiss(animated: true){

self.showApplyFilterInterface()

}

}

}
```

Porozmawiajmy najpierw o showCameraUserInterface(). Ta metoda jest uruchamiana po stuknięciu przycisku aparatu, co powoduje wyświetlenie selektora obrazu na ekranie. Ten selektor obrazów to standardowy selektor obrazów iOS, który pojawia się, gdy chcesz użyć obrazu – na przykład, aby dodać obraz do posta na Facebooku lub do tweeta. Rozbijmy to:

```
let imagePicker = UIImagePickerController()
```

Tworzy wystąpienie klasy UIImagePickerController i przypisuje je do imagePicker.

```
imagePicker.delegate = self
```

Ustawia właściwość delegata wystąpienia imagePicker na wystąpienie PhotoFilterViewController.

```
#if targetEnvironment(simulator)

imagePicker.sourceType =

UIImagePickerController.SourceType.photoLibrary

#else

imagePicker.sourceType =

UIImagePickerController.SourceType.camera

imagePicker.showsCameraControls = true

#endif
```

Ten blok kodu jest znany jako blok kompilacji warunkowej. Zaczyna się od dyrektywy kompilacji #if i kończy się dyrektywą kompilacji #endif. Jeśli korzystasz z symulatora, kompilowana jest tylko instrukcja ustawiająca właściwość sourceType wystąpienia imagePicker na bibliotekę zdjęć. Jeśli korzystasz z rzeczywistego urządzenia, instrukcje ustawiające właściwość sourceType instancji imagePicker na aparat i wyświetlające kontrolki aparatu są kompilowane.

```
imagePicker.mediaTypes = ["public.image"]
```

Ustawia interfejs aparatu na przechwytywanie obrazów nieruchomych.

```
imagePicker.allowsEditing = true
```

Wskazuje, że użytkownik może edytować wybrany obraz. self.present(imagePicker, animowany: prawda, uzupełnianie: zero)

Przedstawia imagePicker na ekranie.

Gdy na ekranie pojawi się selektor obrazów, możesz wybrać zdjęcie lub anulować. Jeśli anulujesz, imagePickerControllerDidCancel(_:) zostanie wyzwolony, a selektor obrazów zostanie odrzucony. Jeśli wybierzesz zdjęcie, imagePickerController(_:didFinishPickingMediaWithInfo:) jest wyzwolony, a zdjęcie zostanie zwrócone i przypisane do wybranego obrazu. Następnie metoda PrepareThumbnail(of:) instancji selectedImage zostanie użyta do utworzenia małego obrazu o szerokości i wysokości 100 punktów. Zostanie to następnie przypisane do właściwości miniatury. Następnie obraz o takim samym rozmiarze jak mainImageView zostanie utworzony z selectedImage przy użyciu metody PreparationThumbnail(of:). Zostanie to przypisane do właściwości mainImage, a selektor obrazów zostanie odrzucony.

Następnie zaimplementujesz filterMainImage(filterItem:), metodę do zastosowania filtru do obrazu w mainImageView. Dodaj rozszerzenie zawierające tę metodę po rozszerzeniu UIImagePickerControllerDelegate:

```
extension PhotoFilterViewController: ImageFiltering {

func filterMainImage(filterItem: FilterItem) {

if let mainImage = mainImage, let filter =

filterItem.filter {

if filter != "None" {

mainImageView.image =

self.apply(filter: filter,

originalImage: mainImage)

} else {

mainImageView.image = mainImage

}

}

}

}
```

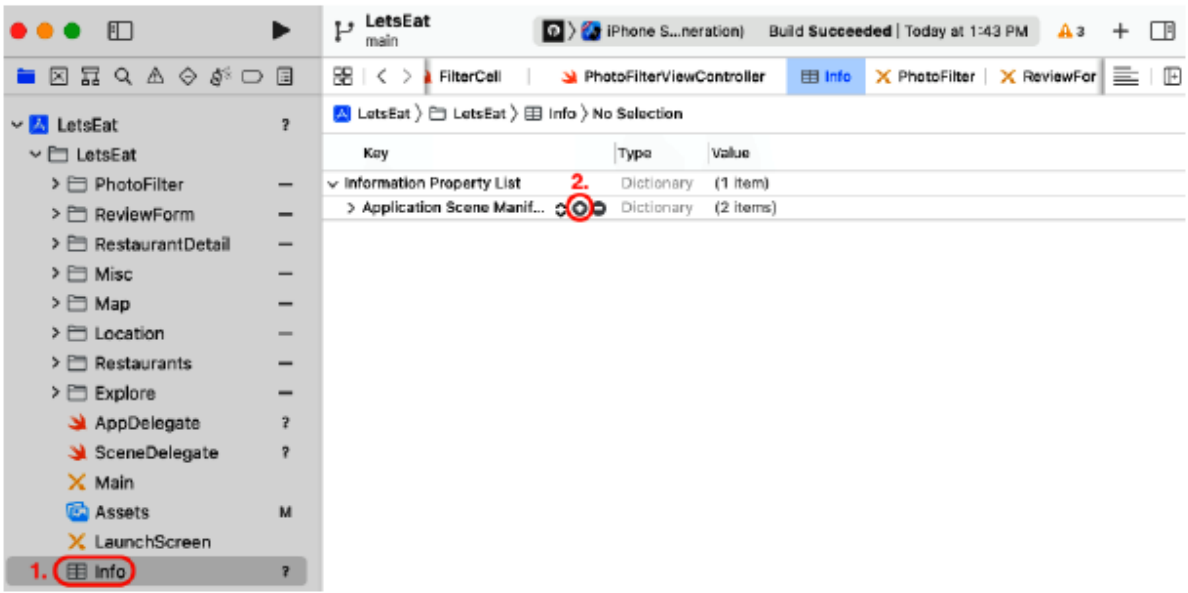
Dzięki temu klasa PhotoFilterViewController przyjmuje protokół ImageFiltering. Pamiętaj, że każda klasa, która to przyjmuje protokół pobiera metodę apply(filter:originalImage:). Metoda filterMainImage(filterItem:) używa tej metody do zastosowania wybranego filtru do zdjęcia przechowywanego we właściwości mainImage instancji PhotoFilterViewController, a wynik jest przypisywany do gniazda mainImageView, dzięki czemu jest widoczny na ekranie. Jeśli wybrałeś filtr Brak, mainImage jest przypisany do gniazdk mainImageView Nadal musisz wiedzieć, który filtr wybrał użytkownik, więc sprawisz, że klasa PhotoFilterViewController przyjmie protokół UICollectionViewDelegate i zaimplementuje metodę identyfikującą, która komórka w widoku kolekcji

została naciśnięta. Dodaj następujące rozszerzenie zawierające tę metodę po rozszerzeniu ImageFiltering:

```
extension PhotoFilterViewController:

UICollectionViewDelegate {

func collectionView(_ collectionView:

UICollectionView, didSelectItemAt

indexPath: IndexPath) {

let filterItem = self.filters[indexPath.row]

filterMainImage(filterItem: filterItem)

}

}
```

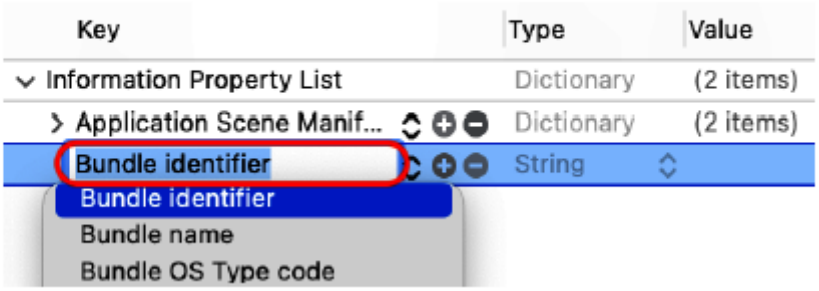
Metoda collectionView(_:didSelectItemAt:) jest wywoływana za każdym razem, gdy użytkownik naciśnie komórkę w widoku kolekcji. Element FilterItem odpowiadający komórce, która została naciśnięta, jest następnie przekazywany do filterMainImage(filterItem:). Implementacja klasy PhotoFilterViewController jest już zakończona, ale pamiętaj, że musisz poprosić o pozwolenie na korzystanie z aparatu lub dostęp do biblioteki zdjęć. Zmodyfikujesz plik Info.plist w projekcie, aby wiadomości były wyświetlane użytkownikowi, gdy aplikacja próbuje uzyskać dostęp do aparatu lub biblioteki zdjęć. Uzyskiwanie pozwolenia na korzystanie z aparatu lub biblioteki zdjęć Jak wspomniano wcześniej, Apple wymaga, aby Twoja aplikacja informowała użytkownika, jeśli chce uzyskać dostęp do aparatu lub biblioteki zdjęć. Jeśli tego nie zrobisz, Twoja aplikacja zostanie odrzucona i będzie niedozwolone w App Store. Zmodyfikujesz plik Info.plist w projekcie, aby aplikacja wyświetlała komunikaty, gdy próbuje uzyskać dostęp do aparatu lub biblioteki zdjęć. Wykonaj następujące kroki:

1. Kliknij plik Info.plist w Nawigatorze projektów, aby wyświetlić listę kluczy. Przesuń wskaźnik myszy na dowolny istniejący klawisz i kliknij przycisk +:



2. Powinno pojawić się pole umożliwiające wprowadzenie dodatkowego klucza:

2. A field should appear, allowing you to enter an additional key:

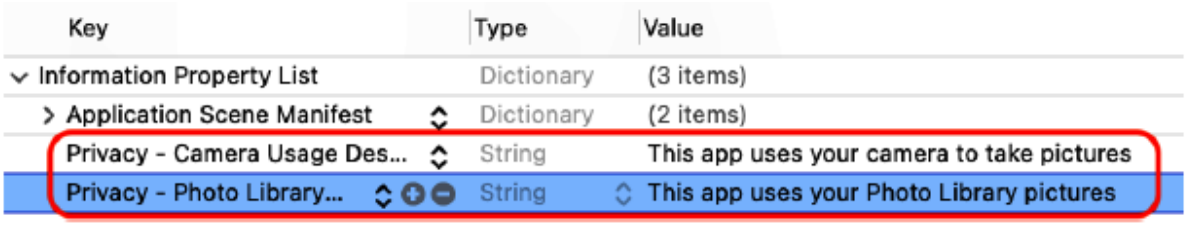


3. Wprowadź następujące klucze:

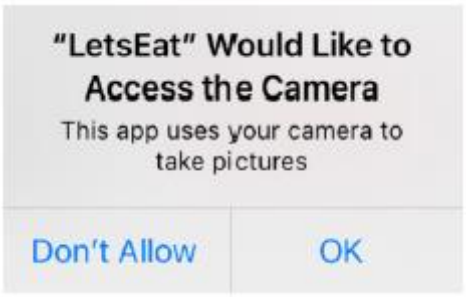
NSPhotoLibraryUsageDescription

NSCameraUsageDescription

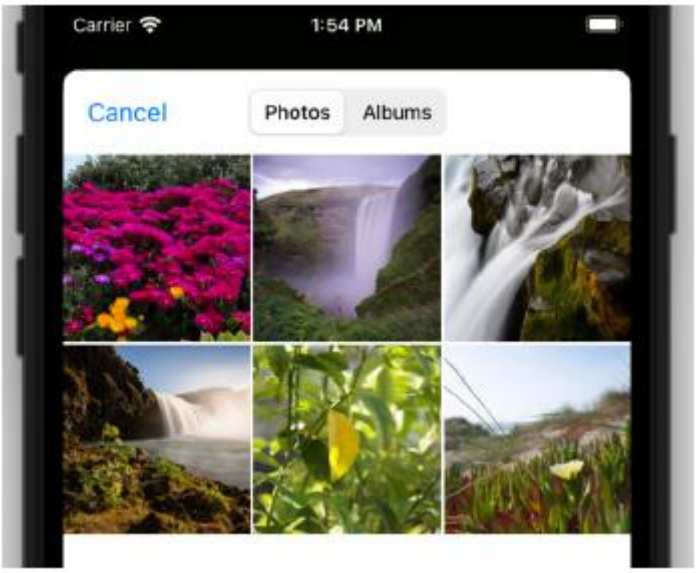
4. Dla wartości każdego klucza wprowadź ciąg, który wyjaśnia użytkownikowi, dlaczego chcesz korzystać z aparatu lub biblioteki zdjęć:



Zbuduj i uruchom projekt. Przejdź do ekranu szczegółów restauracji i dotknij przycisku Dodaj zdjęcie. Powinieneś zobaczyć następujący alert:



Dotknij OK. Pojawi się selektor obrazów:



Wybierz zdjęcie, a ekran Filtr zdjęć wyświetli zdjęcie i listę miniatur z zastosowanymi do nich różnymi filtrami. Dotknięcie filtra spowoduje zastosowanie jego efektu do zdjęcia:

Zmodyfikowałeś plik info.plist w swoim projekcie, a Twoja aplikacja prosi teraz o pozwolenie przed użyciem aparatu lub biblioteki zdjęć. Możesz użyć przycisku Anuluj, aby zamknąć ekran Filtr zdjęć i powrócić do ekranu szczegółów restauracji. Nie możesz jeszcze użyć przycisku Zapisz, zaimplementujesz jego funkcjonalność w następnym rozdziale.

Podsumowanie

Zakończyłeś implementację ekranu Photo Filter. Zaimportowano plik FilterData.plist, plik .plist zawierający filtry, których chcesz użyć, utworzono klasę FilterItem do przechowywania danych filtrów

i utworzono klasę menedżera danych FilterManager w celu odczytania pliku .plist i wypełnienia tablicy wystąpień FilterItem. Następnie utworzyłeś protokół ImageFiltering z metodą nakładania filtrów na obrazy. Następnie utworzono klasy FilterCell i PhotoFilterViewController w celu zarządzania komórkami widoku kolekcji i ekranem filtru zdjęć. Następnie utworzono klasę PhotoFilterViewController, która przyjęła protokół UIImagePickerDelegate i dodano metody, dzięki czemu można używać zdjęć z aparatu lub biblioteki zdjęć w aplikacji. Na koniec dodano kod do PhotoFilterViewController, aby zastosować wybrany filtr do obrazu. Możesz teraz pisać własne aplikacje, które importują zdjęcia z aparatu lub biblioteki zdjęć i stosować do nich filtry. Zwróć uwagę, że wybranego obrazu nie można zapisać.