



Introduction to SwiftUI Data Management

Girish Lukka

Topics

- App Bundle
- **Introduction to json**
- Mapping json object to swift struct
- Data persistence

History

- Web 1.0 – SGML, HTML
- Web 1.1 – XML, HTML XHTML
- XML provided a technology to store, communicate, and validate any kind of data, in a form that applications on any platform can easily read and process.
- The delivery of pre-rendered HTML pages to the browser was not scalable -
Web 1.2 AJAX
 - HTML (or XHTML) and CSS for presentation.
 - The Document Object Model (DOM) for dynamic display of, and interaction with, data.
 - XML for the interchange of data, and XSLT for its manipulation.
 - The XMLHttpRequest object for asynchronous communication.
 - JavaScript to bring these technologies together.

History

- Web 2.0 – Single-page Applications
- Emergence of json as "most popular data interchange format."
- In a nutshell:
- Json is lightweight, faster and compact
- Xml is more versatile, has presentation capability and is more secure.
- Good summary of the both – [Comparison](#)

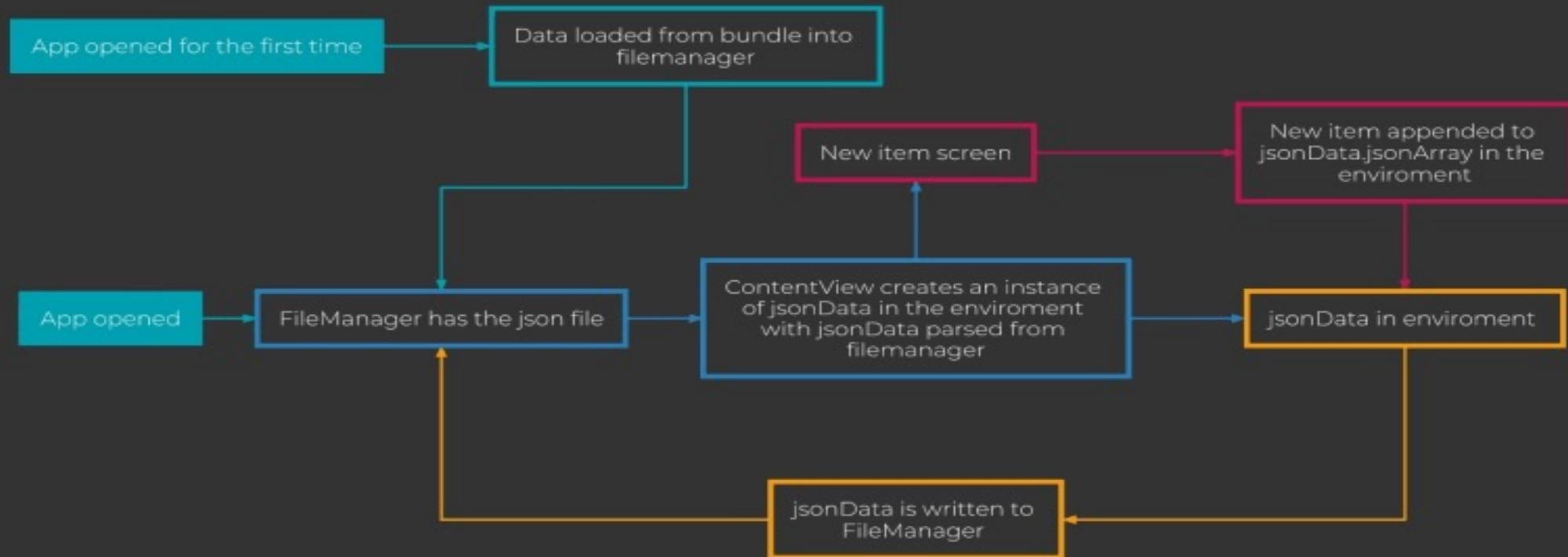
Data Serialization Languages		
JSON	YAML	XML
JavaScript Object Notation	YAML Ain't Markup Language	eXtensible Markup Language
Data Interchange	Data Interchange	Markup Language
2002	2006	1996
Easy to read	Easier to read	A little complex
Fast	Fast	Slow
Map Structure	Map Structure	Tree Structure
.json	.yaml	.xml

SwiftUI and json

- Most data exchanges between the apps and servers will be as json packets
- You can always use Wireshark to capture the data frames and see the format.
- It is also necessary to load data from files when installing apps or working locally though CoreData is better suited for that.

Dataflowdata

Dataflow



What is JSON?

JSON stands for **JavaScript Object Notation**.

JavaScript started out as a new language in 1995 as a way for internet browsers to run code.

When calling services to get data over the internet there needed to be a way to **express objects as text**.

Example:

```
{  
    "field1": "value1",  
    "field2": "value2"  
}
```

Notation means "a system of writing used to represent numbers, amounts, or elements in something". Such as shown here by representing data in a structured way with text.

JSON encloses data with **braces**.

The braces usually specify a single object which we would recognize as a class or struct.

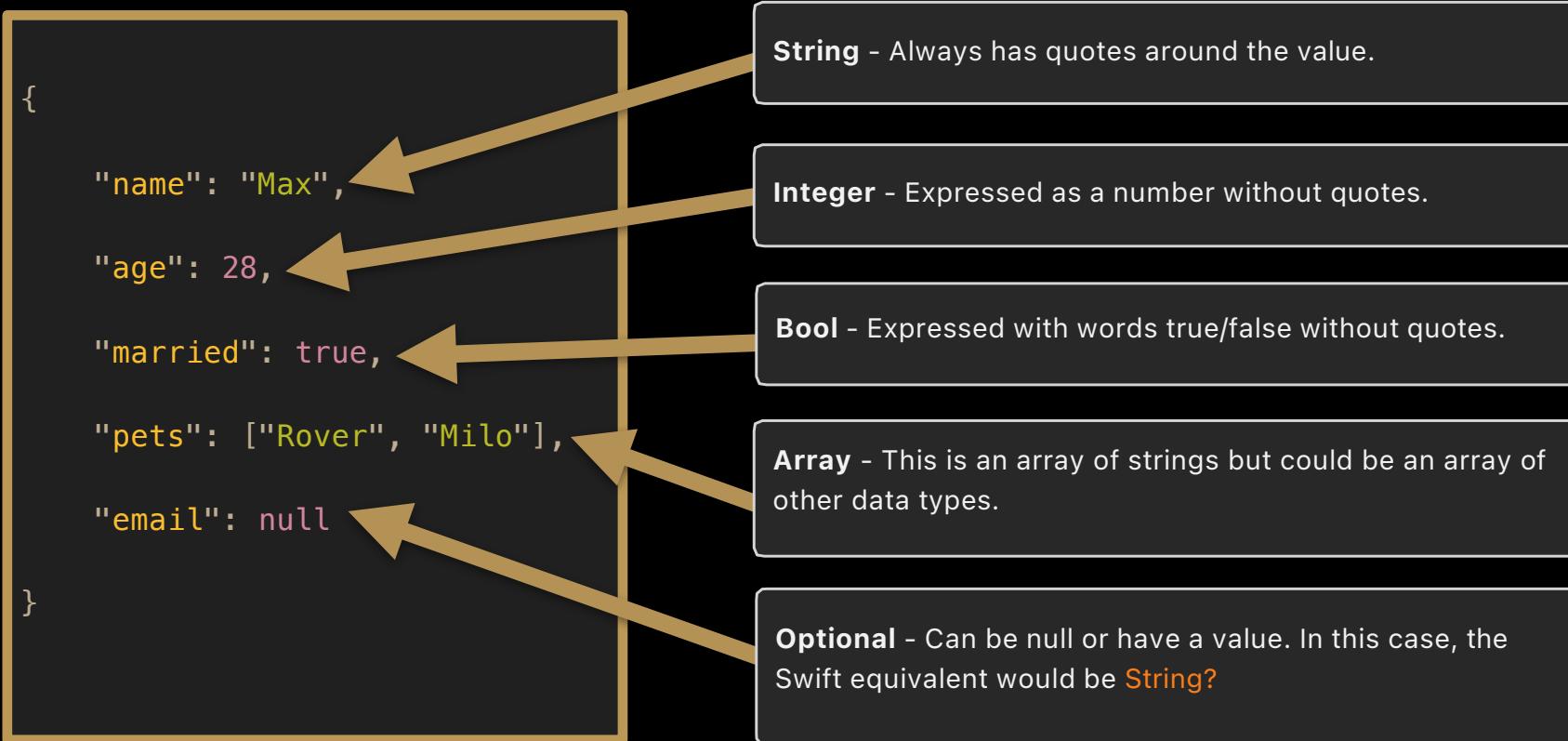
Data is identified with a field or property name.

```
{"firstName": "Joe",  
 "lastName": "Smith"}
```

The value comes after the field/property name and a colon.

Data Types

Here are some ways JSON expresses different data types.



The goal is for you to be able to look at the JSON you are receiving and create a matching struct or class to easily move this data into by using "decoding". JSON is the "coded" part. When you **decode**, you transfer values from JSON to a Swift object.

Decodable

Here is the struct to represent the JSON we have.

You use Decodable to indicate that you will be decoding JSON into this object.

JSON

```
{  
    "name": "Max",  
    "age": 28,  
    "married": true,  
    "pets": ["Rover", "Milo"],  
    "email": null  
}
```

Swift

```
struct Json_User: Decodable {  
    var name = ""  
    var age = 0  
    var married = false  
    var pets: [String] = []  
    var email: String?  
}
```

Because the email field can be null, we want to make it optional in Swift.

Decoding (Observable Object)

Decoding JSON into a Swift object will most likely happen in your observable object. Here is an example of how this might happen.

```
var jsonString = """"  
{  
    "name": "Max",  
    "age": 28,  
    "married": true,  
    "pets": ["Rover", "Milo"],  
    "email": null  
}""""  
  
class Json_Decoding00: ObservableObject  
{ @Published var user = Json_User()  
@Published var jsonError: Error?  
  
func fetch() {  
    let jsonData = jsonString.data(using: .utf8)!  
  
    let decoder = JSONDecoder()  
  
    do {  
        user = try decoder.decode(Json_User.self, from: jsonData)  
    } catch {  
        jsonError = error  
    }  
}  
}
```

First, you want to convert your JSON to data if it's not already in this format.

What is utf8? See next page.

You will also need a JSONDecoder to do the work of transferring the data from JSON to your object.

This is where the actual decoding happens. You specify the destination type and the JSON data to decode.

Sometimes errors can happen if you don't set up the destination type correctly to match the JSON.

What is UTF-8?

UTF stands for “Unicode Transformation Format”.

What is “Unicode”?

Unicode is an international standard of representing characters, letters and even emoji's using a specified code. (This code will eventually be converted into ones and zeroes.)

Example:

Character	Unicode	Binary
A	U+0041	01010101 00101011 00110000 00110000 00110100 00110001
Ä	U+00C4	01010101 00101011 00110000 00110000 01000011 00110100
😊	U+1F603	01010101 00101011 00110001 01000110 00110110 00110000 00110011

Binary is what is used by the computer (processor).

8

Why is it utf-8 though?
The 8 specifies how “bits” (1 or 0) will be used.

If you look at the Binary column, you will notice there are groups of 8 bits.

You convert your JSON string into Unicode first before you can decode it into a struct or class:
`let jsonData = jsonString.data(using: .utf8)!`

Json structure

```
{  
    "id": "1010",  
    "name": "girish",  
    "mark": "100",  
    "status": "active",  
    "pic": "image/002.jpg"  
},  
{  
    "id": "1020",  
    "name": "mario",  
    "mark": 60,  
    "status": "active",  
    "pic": "image/003.jpg"  
},  
{  
    "id": "1030",  
    "name": "andy",  
    "mark": 70,  
    "status": "active",  
    "pic": "image/004.jpg"  
}
```

A collection of name/value pairs. Different programming languages support this data structure in different names. Like object, record, struct, dictionary, hash table, keyed list, or associative array.
It can also be nested

```
3  let jsonData = """  
4  {  
5      "id": 1,  
6      "name": "Leanne Graham",  
7      "username": "Bret",  
8      "email": "Sincere@april.biz",  
9      "address": {  
10         "street": "Kulas Light",  
11         "suite": "Apt. 556",  
12         "city": "Gwenborough",  
13         "zipcode": "92998-3874",  
14     }  
15 }  
16  
17 """ .data(using: .utf8)  
18
```

Json parsing

- Json keys are usually written as snake_case
- Swift is camelCase
- Map json object to a swift object

Coding Keys

Coding keys are necessary when:

- A Swift variable/property name must not start with a number. If a key does start with a number you have to specify a compatible CodingKey to be able to decode the key at all.
- You want to use a different property name.
- You want to exclude keys from being decoded for example an id property which is not in the JSON and is initialized with an UUID constant.
-

Flattening a json - demo

```
3 let jsonData = """
4 {
5     "id": 1,
6     "name": "Leanne Graham",
7     "username": "Bret",
8     "email": "Sincere@april.biz",
9     "address": {
10         "street": "Kulas Light",
11         "suite": "Apt. 556",
12         "city": "Gwenborough",
13         "zipcode": "92998-3874",
14     }
15 }
16 """
17 """".data(using: .utf8)
```

```
21 struct User: Decodable {
22     var id: Int?
23     var name: String = ""
24     var userName: String = ""
25     var email: String = ""
26     var street: String = ""
27     var suite: String = ""
28     var city: String = ""
29     var zipCode: String = ""
30
31     //coding keys
32     private enum UserKeys: String, CodingKey{
33         case id
34         case name
35         case userName = "username"
36         case email
37         case address
38     }
39
40     private enum AddressKeys: String, CodingKey{
41         case street
42         case suite
43         case city
44         case zipCode = "zipcode"
45     }
46 }
47
```

Flattening a json

```
47
48     init(from decoder: Decoder) throws {
49         if let userContainer = try? decoder.container(keyedBy: UserKeys.self){
50             self.id = try userContainer.decode(Int.self, forKey: .id)
51             self.name = try userContainer.decode(String.self, forKey: .name)
52             self.userName = try userContainer.decode(String.self, forKey: .userName)
53             self.email = try userContainer.decode(String.self, forKey: .email)
54
55             if let addressContainer = try?userContainer.nestedContainer(keyedBy: AddressKeys.self, forKey: .address){
56                 self.street = try addressContainer.decode(String.self, forKey: .street)
57                 self.suite = try addressContainer.decode(String.self, forKey: .suite)
58                 self.city = try addressContainer.decode(String.self, forKey: .city)
59                 self.zipCode = try addressContainer.decode(String.self, forKey: .zipCode)
60
61             }
62         }
63     }
64 }
65 if let user = try? JSONDecoder().decode(User.self, from: jsonData!){
66     print(user.name)
67     print(user.city)
68     print(user.email)
69     print(user)
70
71 }
```

Using a nested scheme - demo

```
3 let jsonData = """
4 {
5     "id": 1,
6     "name": "Leanne Graham",
7     "username": "Bret",
8     "email": "Sincere@april.biz",
9     "address": {
10         "street": "Kulas Light",
11         "suite": "Apt. 556",
12         "city": "Gwenborough",
13         "zipcode": "92998-3874",
14     }
15 }
16 """
17 """.data(using: .utf8)
```

```
18 struct User: Decodable {
19
20     let id: Int
21     let name: String
22     let username: String
23     let email: String
24     let address: Address
25
26
27 struct Address: Codable {
28
29
30     let street: String
31     let suite: String
32     let city: String
33     let zipcode: String
34
35 }
36
37
38 }
```

PizzaStore - demo

```
8 import Foundation
9
10
11 struct Pizza: Identifiable, Decodable, Encodable{
12     var id = UUID()
13     var name: String
14     var ingredients: String
15     var imageName: String
16     var thumbnailName: String
17     var type: String
18 }
```

Loading json file – from user defaults or file

```
5 // Created by GirishALukka on 14/02/2023.
6 //
7
8 import Foundation
9 class PizzaStore: ObservableObject {
10     @Published var pizzas: [Pizza] = []
11     //{
12     //    didSet {
13     //        savePizzasToUserDefaults()
14     //    }
15     //}
16
17     init() {
18         let userDefaults = UserDefaults.standard
19
20         // Try to read the pizzas data from user defaults
21         if let pizzasData = userDefaults.data(forKey: "pizzas"),
22             let pizzas = try? JSONDecoder().decode([Pizza].self, from: pizzasData) {
23             self.pizzas = pizzas
24         }
25         // If pizzas data is not available in user defaults, load it from the bundled pizzas.json file
26         else if let url = Bundle.main.url(forResource: "pizzas", withExtension: "json") {
27             do {
28                 let data = try Data(contentsOf: url)
29                 let decoder = JSONDecoder()
30                 self.pizzas = try decoder.decode([Pizza].self, from: data)
31                 // Save the loaded pizzas data to user defaults
32                 let pizzasData = try JSONEncoder().encode(self.pizzas)
33                 userDefaults.set(pizzasData, forKey: "pizzas")
34             } catch {
35                 print("Error decoding pizza data: \(error)")
36             }
37         }
38     }
39 }
```

Saving and deleting

```
40     // Function to save the pizza data to user defaults
41     func savePizzasToUserDefaults() {
42         let userDefaults = UserDefaults.standard
43         do {
44             let pizzasData = try JSONEncoder().encode(self.pizzas)
45             userDefaults.set(pizzasData, forKey: "pizzas")
46         } catch {
47             print("Error encoding pizza data: \(error)")
48         }
49     }
50
51     func deletePizza(_ pizza: Pizza) {
52         if let index = pizzas.firstIndex(where: { $0.id == pizza.id }) {
53             pizzas.remove(at: index)
54             savePizzasToUserDefaults()
55         }
56     }
57 }
```

ContentView – tab navigation and filter by type

```
8 import SwiftUI
9
10 struct ContentView: View {
11     @EnvironmentObject var pizzaStore: PizzaStore
12     @State private var isPresentingAddPizzaModal = false
13     @State private var newPizza = Pizza(name: "", ingredients: "", imageName: "", thumbnailName: "", type: "")
14
15     var body: some View {
16
17         NavigationStack{
18             TabView {
19                 PizzaList(pizzas: pizzaStore.pizzas)
20                     .tabItem {
21                         Label("All Pizzas", systemImage: "cart")
22                     }
23
24                 PizzaList(pizzas: pizzaStore.pizzas.filter { $0.type == "meat" })
25                     .tabItem {
26                         Label("Meat Pizzas", systemImage: "circle")
27                     }
28
29                 PizzaList(pizzas: pizzaStore.pizzas.filter { $0.type == "vegetarian" })
30                     .tabItem {
31                         Label("Vegetarian Pizzas", systemImage: "leaf")
32                     }
33             }
34             .navigationBarItems(trailing:
35                 Button(action: {
36                     isPresentingAddPizzaModal = true
37                 }) {
38                     Image(systemName: "plus")
39                 }
40             )
41             .sheet(isPresented: $isPresentingAddPizzaModal) {
42                 AddPizzaView(pizza: $newPizza, isPresented: $isPresentingAddPizzaModal)
43                     .environmentObject(pizzaStore)
44             }
45         }
46     }
47 }
```

PizzaListView

```
5 // Created by GirishALukka on 14/02/2023.
6 //
7
8 import SwiftUI
9
10 struct PizzaList: View {
11     let pizzas: [Pizza]
12
13     var body: some View {
14
15         NavigationView{
16             List(pizzas) { pizza in
17                 NavigationLink(destination: PizzaDetailView(pizza: pizza)) {
18                     HStack {
19                         Image(pizza.thumbnailName)
20                             .resizable()
21                             .frame(width: 50, height: 50)
22                         Text(pizza.name)
23
24                     }
25                 }
26             }
27             .navigationTitle("Pizza Menu")
28         }
29     }
30 }
```

PizzaListView

```
5 // Created by GirishALukka on 14/02/2023.
6 //
7
8 import SwiftUI
9
10 struct PizzaList: View {
11     let pizzas: [Pizza]
12
13     var body: some View {
14
15         NavigationView{
16             List(pizzas) { pizza in
17                 NavigationLink(destination: PizzaDetailView(pizza: pizza)) {
18                     HStack {
19                         Image(pizza.thumbnailName)
20                             .resizable()
21                             .frame(width: 50, height: 50)
22                         Text(pizza.name)
23
24                     }
25                 }
26             }
27             .navigationTitle("Pizza Menu")
28         }
29     }
30 }
```

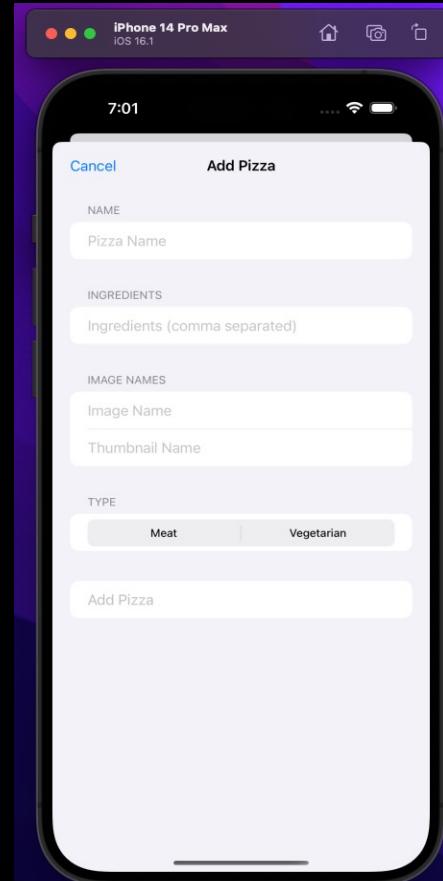
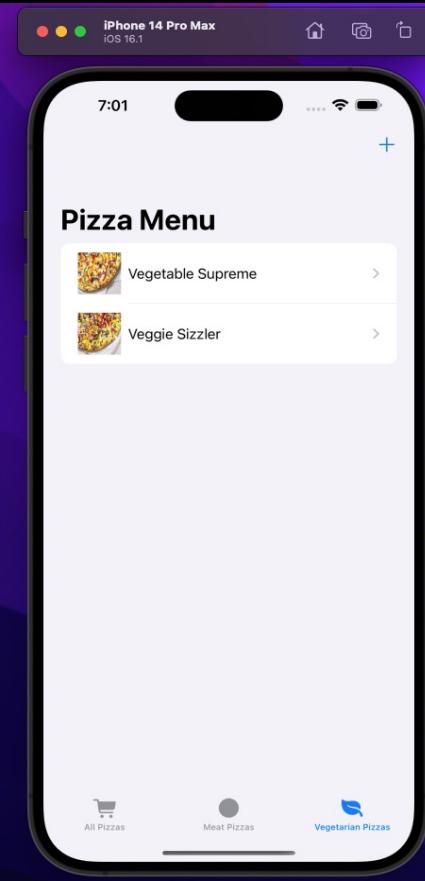
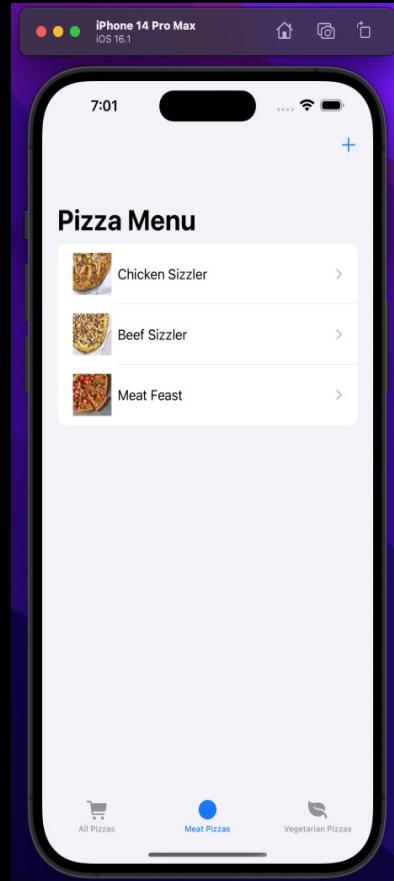
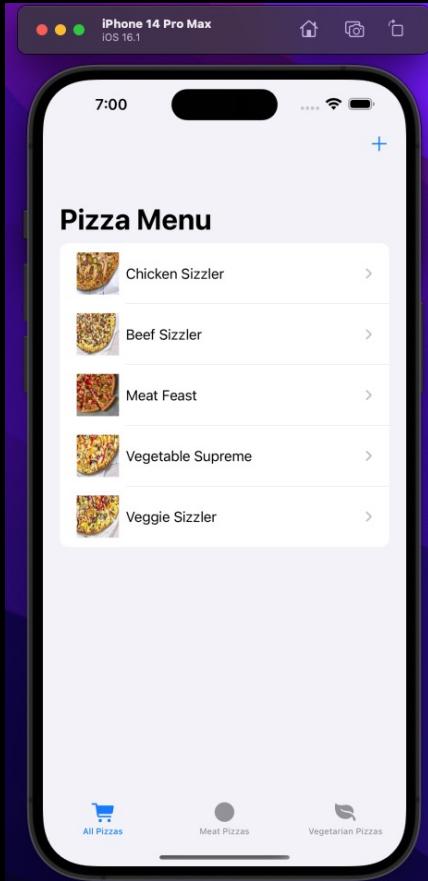
AddPizzaView

```
8 import SwiftUI
9
10 struct AddPizzaView: View {
11     @Environment(\.presentationMode) private var presentationMode
12     @EnvironmentObject var pizzaStore: PizzaStore
13     @Binding var pizza: Pizza
14     @Binding var isPresented: Bool
15
16     var body: some View {
17         NavigationView {
18             Form {
19                 Section(header: Text("Name")) {
20                     TextField("Pizza Name", text: $pizza.name)
21                 }
22                 Section(header: Text("Ingredients")) {
23                     TextField("Ingredients (comma separated)", text: $pizza.ingredients)
24                 }
25                 Section(header: Text("Image Names")) {
26                     TextField("Image Name", text: $pizza.imageName)
27                     TextField("Thumbnail Name", text: $pizza.thumbnailName)
28                 }
29                 Section(header: Text("Type")) {
30                     Picker(selection: $pizza.type, label: Text("Type")) {
31                         Text("Meat").tag("meat")
32                         Text("Vegetarian").tag("vegetarian")
33                     }
34                     .pickerStyle(SegmentedPickerStyle())
35                 }
36                 Section {
37                     Button(action: {
38                         pizzaStore.pizzas.append(pizza)
39                         presentationMode.wrappedValue.dismiss()
40                         pizzaStore.savePizzasToUserDefaults()
41                     // save each time new one is added as alternative to didSet in class
42                     }) {
43                         Text("Add Pizza")
44                     }
45                     .disabled(pizza.name.isEmpty || pizza.ingredients.isEmpty || pizza.imageName.isEmpty || pizza.thumbnailName.isEmpty)
46                 }
47             }
48         }
49     }
50 }
```

AddPizzaView - continued

```
40         pizzaStore.savePizzasToUserDefaults()
41     // save each time new one is added as alternative to didSet in class
42     })
43     Text("Add Pizza")
44 }
45 .disabled(pizza.name.isEmpty || pizza.ingredients.isEmpty || pizza.imageName.isEmpty || pizza.thumbnailName.isEmpty)
46 }
47 }
48 .navigationBarTitle(Text("Add Pizza"), displayMode: .inline)
49 .autocapitalization(.none)
50 .navigationBarItems(leading: Button(action: {
51     presentationMode.wrappedValue.dismiss()
52 }) {
53     Text("Cancel")
54 })
55 }
56 }
57 }
```

PizzaStoreApp Screens



Modal View Demo

