Sample Code

Using JSON with Custom Types

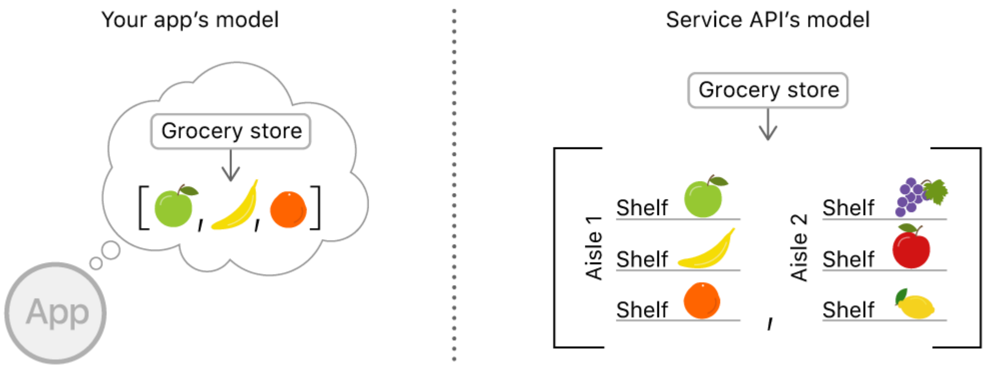
Encode and decode JSON data, regardless of its structure, using Swift's JSON support.

Download

Xcode 10.0+

Overview

JSON data you send or receive from other apps, services, and files can come in many different shapes and structures. Use the techniques described in this sample to handle the differences between external JSON data and your app's model types.



This sample defines a simple data type, `GroceryProduct`, and demonstrates how to construct instances of that type from several different JSON formats.

```swift
struct GroceryProduct: Codable {
    var name: String
    var points: Int
    var description: String?
}
```

Read Data from Arrays

Use Swift's expressive type system to avoid manually looping over collections of identically structured objects. This playground uses array types as values to see how to work with JSON that's structured like this:

```
[
    {
        "name": "Banana",
        "points": 200,
        "description": "A banana grown in Ecuador."
    }
]
```

Change Key Names

Learn how to map data from JSON keys into properties on your custom types, regardless of their names. For example, this playground shows how to map the `"product_name"` key in the JSON below to the `name` property on `GroceryProduct`:

```
{
    "product_name": "Banana",
    "product_cost": 200,
    "description": "A banana grown in Ecuador."
}
```

Custom mappings let you to apply the Swift API Design Guidelines to the names of properties in your Swift model, even if the names of the JSON keys are different.

Access Nested Data

Learn how to ignore structure and data in JSON that you don't need in your code. This playground uses an intermediate type to see how to extract grocery products from JSON that looks like this to skip over unwanted data and structure:

```
[
    {
        "name": "Home Town Market",
        "aisles": [
            {
                "name": "Produce",
                "shelves": [
                    {
                        "name": "Discount Produce",
                        "product": {
                            "name": "Banana",
                            "points": 200,
                            "description": "A banana that's perfectly ripe."
                        }
                    }
                ]
            }
        ]
    }
]
```

Merge Data at Different Depths

Combine or separate data from different depths of a JSON structure by writing custom implementations of protocol requirements from `Encodable` and `Decodable`. This playground shows how to construct a `GroceryProduct` instance from JSON that looks like this:

```
{
    "Banana": {
        "points": 200,
        "description": "A banana grown in Ecuador."
    }
}
```

See Also

JSON

`class JSONEncoder`

An object that encodes instances of a data type as JSON objects.

`class JSONDecoder`

An object that decodes instances of a data type from JSON objects.

`class JSONSerialization`

An object that converts between JSON and the equivalent Foundation objects.