currently i have a recipe strcut that conforms to codable if i want to save this data using swiftdata i need to make my recipe conform to @Model final class, but how do i also make it conform to codable ? struct Recipe: Codable, Hashable, Identifiable{
let id: Int
let name: String
let ingredients: [String]
let instructions: [String]
let image: String
let difficulty: String
let rating: Double
let cuisine: String
let prepTimeMinutes: Int
let cookTimeMinutes: Int
let reviewCount: Int
}

struct RecipesResponse: Codable {
let recipes: [Recipe]
}

When using **SwiftData**, your `Recipe` struct needs to be converted into a **class** that conforms to `@Model`. However, since `@Model` classes do not natively support `Codable`, you need to **manually implement** encoding and decoding.

Solution: Convert Recipe to a SwiftData `@Model` While Keeping Codable

To make your `Recipe` conform to **both** `@Model` and `Codable`, you need to:

1. Convert `Recipe` from a `struct` to a `final class`.
2. Use `@Model` to make it work with SwiftData.
3. Store arrays (`ingredients` and `instructions`) as `String` (JSON-encoded).
4. Implement `Codable` manually.

Updated `Recipe` Model

swift Copy Edit

```swift
import Foundation
import SwiftData

@Model
final class Recipe: Codable, Hashable, Identifiable {
    @Attribute(.unique) var id: Int
    var name: String
    var ingredientsData: String // Store JSON-encoded ingredients
    var instructionsData: String // Store JSON-encoded instructions
    var image: String
    var difficulty: String
    var rating: Double
    var cuisine: String
    var prepTimeMinutes: Int
    var cookTimeMinutes: Int
    var reviewCount: Int

    // Computed properties for ingredients and instructions
    var ingredients: [String] {
        get {
            (try? JSONDecoder().decode([String].self, from: Data(ingredientsData.utf8
        }
        set {
            if let data = try? JSONEncoder().encode(newValue),
               let jsonString = String(data: data, encoding: .utf8) {
                ingredientsData = jsonString
            }
        }
    }

    var instructions: [String] {
        get {
            (try? JSONDecoder().decode([String].self, from: Data(instructionsData.utf
        }
        set {
            if let data = try? JSONEncoder().encode(newValue),
               let jsonString = String(data: data, encoding: .utf8) {
                instructionsData = jsonString
            }
        }
    }

    // MARK: - Initializer
    init(id: Int, name: String, ingredients: [String], instructions: [String], image:
        self.id = id
        self.name = name
        self.image = image
        self.difficulty = difficulty
        self.rating = rating
        self.cuisine = cuisine
        self.prepTimeMinutes = prepTimeMinutes
        self.cookTimeMinutes = cookTimeMinutes
        self.reviewCount = reviewCount
        self.ingredientsData = (try? JSONEncoder().encode(ingredients)).flatMap { Str
        self.instructionsData = (try? JSONEncoder().encode(instructions)).flatMap { S
    }

    // MARK: - Codable Conformance
    enum CodingKeys: String, CodingKey {
        case id, name, ingredients, instructions, image, difficulty, rating, cuisine,
    }

    required convenience init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        let id = try container.decode(Int.self, forKey: .id)
        let name = try container.decode(String.self, forKey: .name)
        let ingredients = try container.decode([String].self, forKey: .ingredients)
        let instructions = try container.decode([String].self, forKey: .instructions)
        let image = try container.decode(String.self, forKey: .image)
        let difficulty = try container.decode(String.self, forKey: .difficulty)
        let rating = try container.decode(Double.self, forKey: .rating)
        let cuisine = try container.decode(String.self, forKey: .cuisine)
        let prepTimeMinutes = try container.decode(Int.self, forKey: .prepTimeMinutes
        let cookTimeMinutes = try container.decode(Int.self, forKey: .cookTimeMinutes
        let reviewCount = try container.decode(Int.self, forKey: .reviewCount)

        self.init(id: id, name: name, ingredients: ingredients, instructions: instruc
    }

    func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        try container.encode(id, forKey: .id)
        try container.encode(name, forKey: .name)
        try container.encode(ingredients, forKey: .ingredients)
        try container.encode(instructions, forKey: .instructions)
        try container.encode(image, forKey: .image)
        try container.encode(difficulty, forKey: .difficulty)
        try container.encode(rating, forKey: .rating)
        try container.encode(cuisine, forKey: .cuisine)
        try container.encode(prepTimeMinutes, forKey: .prepTimeMinutes)
        try container.encode(cookTimeMinutes, forKey: .cookTimeMinutes)
        try container.encode(reviewCount, forKey: .reviewCount)
    }

    // MARK: - Hashable Conformance
    static func == (lhs: Recipe, rhs: Recipe) -> Bool {
        return lhs.id == rhs.id
    }

    func hash(into hasher: inout Hasher) {
        hasher.combine(id)
    }
}
```

Why This Works

✅ **SwiftData Compatibility (`@Model`)**

- The class is now marked with `@Model`, allowing it to work with **SwiftData**.

✅ **Codable Compatibility (`Codable`)**

- Manual implementation of `init(from:)` and `encode(to:)` ensures it **still works with Codable**.

✅ **Storing Arrays (`ingredients` & `instructions`) in CoreData**

- Stored as **JSON strings** (`ingredientsData` and `instructionsData`).
- Computed properties handle encoding and decoding.

✅ **Hashable & Identifiable Support**

- `id` is used for identity.

How to Use This in Swift