SwiftData

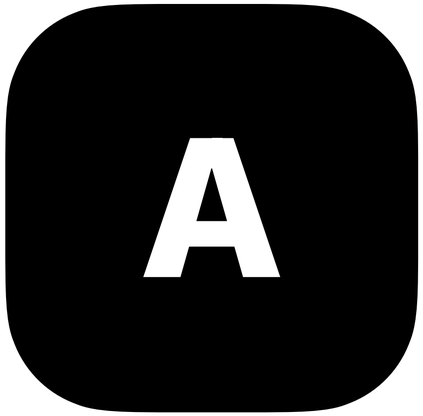
A New Way to Model Your Data

CocoaHeads Aachen 28.9.2023 - Lukas Woyke

How to make app persistent?

CoreData

SwiftData



AppStorage

(UserDefaults)

increasing functionality

SwiftData Integration

1. Create the app model using SwiftData
2. Use models in SwiftUI
3. Extending the model for new app version release

FlashCards App

1 Prepare The App Model

```
import SwiftData
```

```
@Model
```

```
import Foundation

class Drink {
    var name: String
    var symbol: String
    let id = UUID()
}
```

1 Prepare The App Model

```
import SwiftData
```

```
@Model
```

```
import SwiftData

@Model
class Drink {
    var name: String
    var symbol: String
    let id = UUID()
}
```

That’s it!

1 Prepare The App Model

`import SwiftData` `@Model`

```
import SwiftData

@Model
class Drink {
    @Attribute(.unique) var name: String
    @Attribute(.orginialName: “drinkSymbol”) var symbol: String
    @Transient let id = UUID()

    @Transient(.cascade)
    var drunkWith: [Friend]
}
```

- **Macros** allow to **customize schema:**
  - `@Attribute` - e.g. makes attribute unique
  - `@Transient` - excludes certain Properties
  - `@Relationship` - model Relationships
  - More available (see SwiftData Docs)

1 Prepare The App Model

```
import SwiftData
```

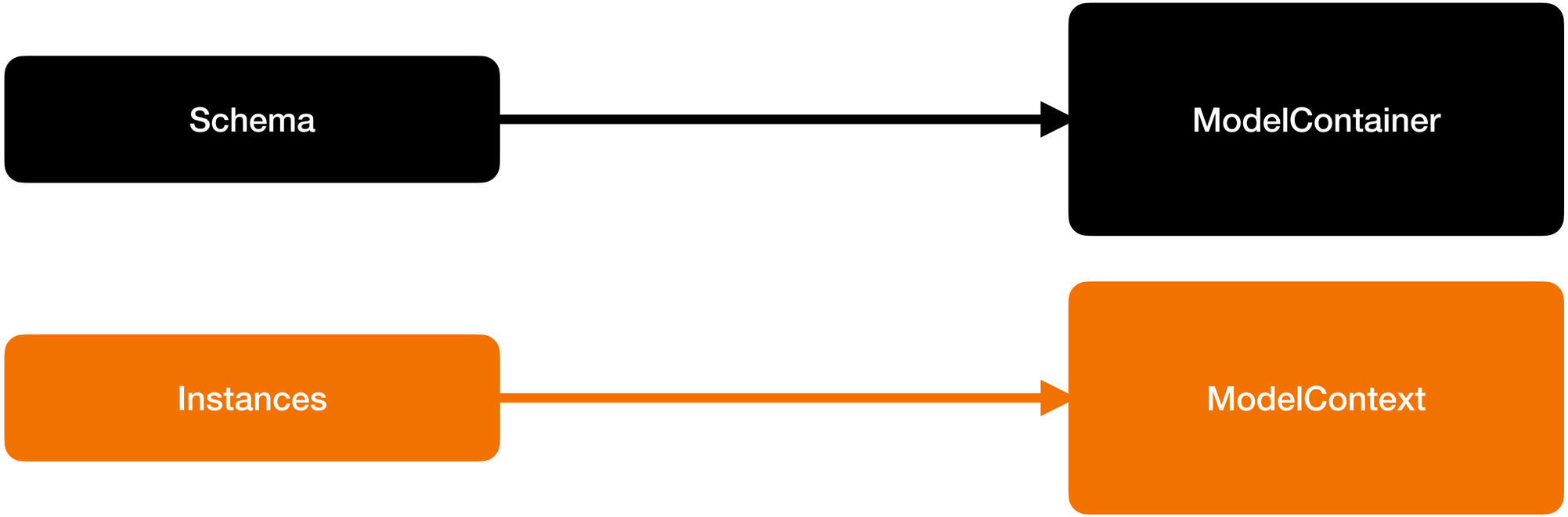
```
@Model
```

```
import SwiftData

@Model
class Drink {
    @Attribute(.unique) var name: String
    @Attribute(.orginialName: “drinkSymbol”) var symbol: String
    @Transient let id = UUID()

    @Transient(.cascade)
    var drunkWith: [Friend]
}

@Model
struct Friend {
    var name: String
}
```

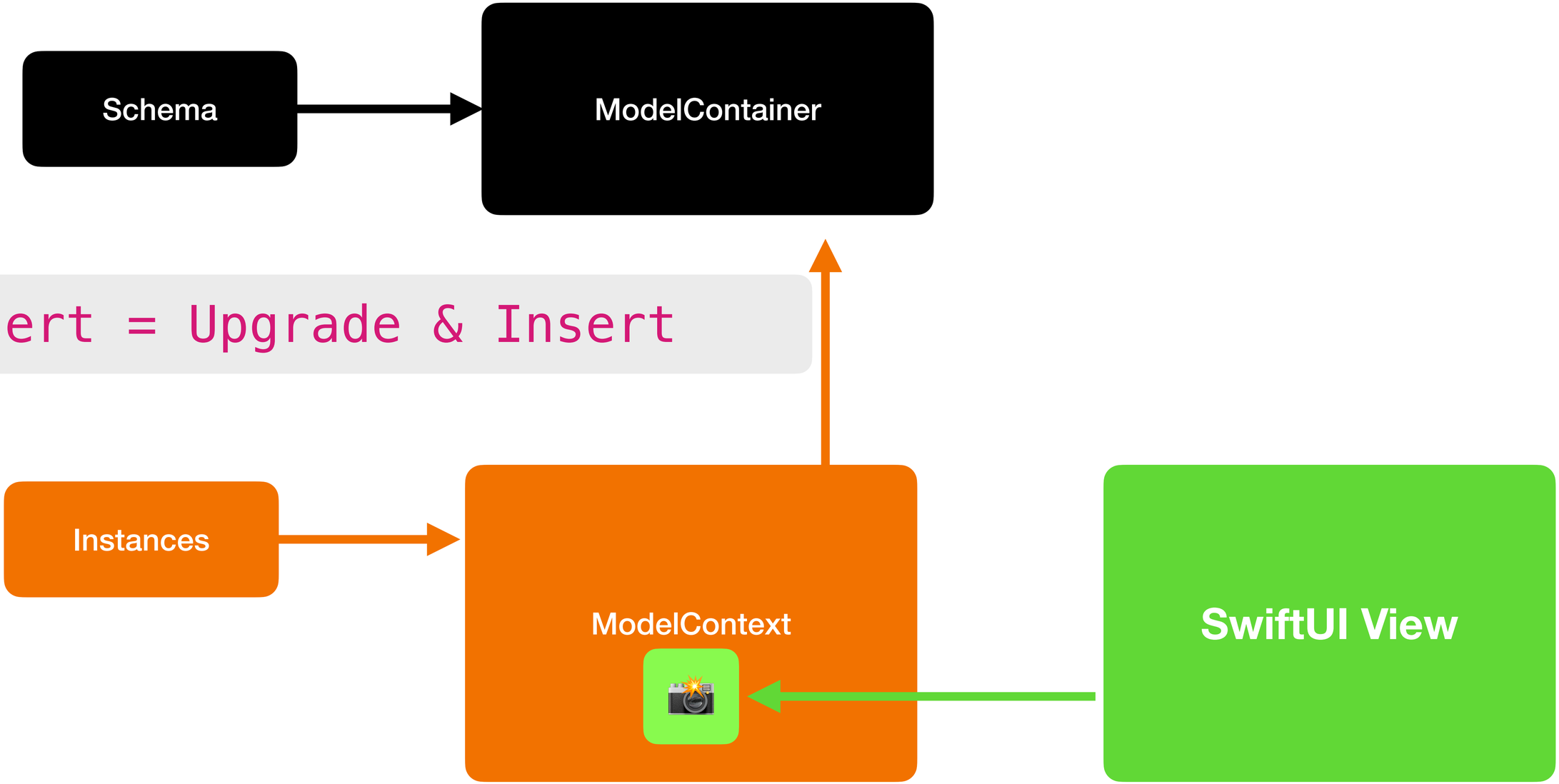
2 Use Models in SwiftUI

`ModelContainer`

`ModelContext`



2 Use Models in SwiftUI

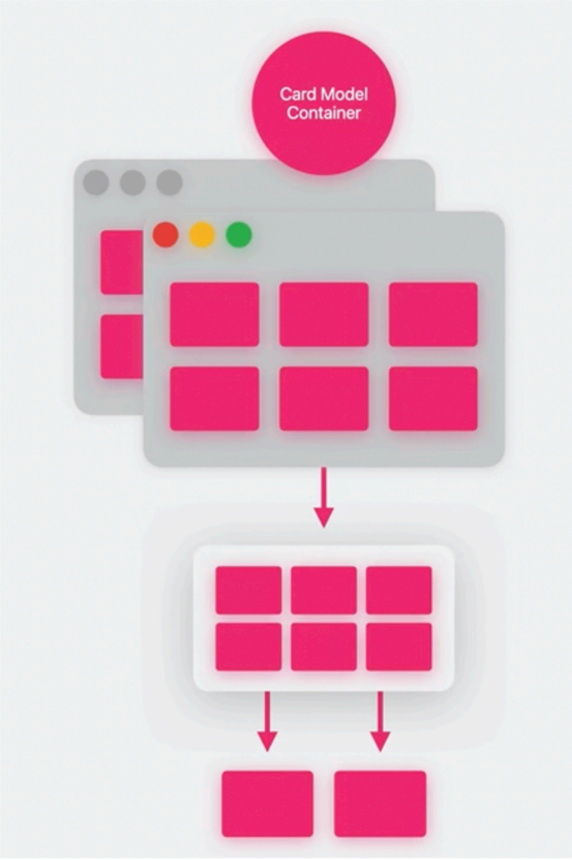



```
Upsert = Upgrade & Insert
```

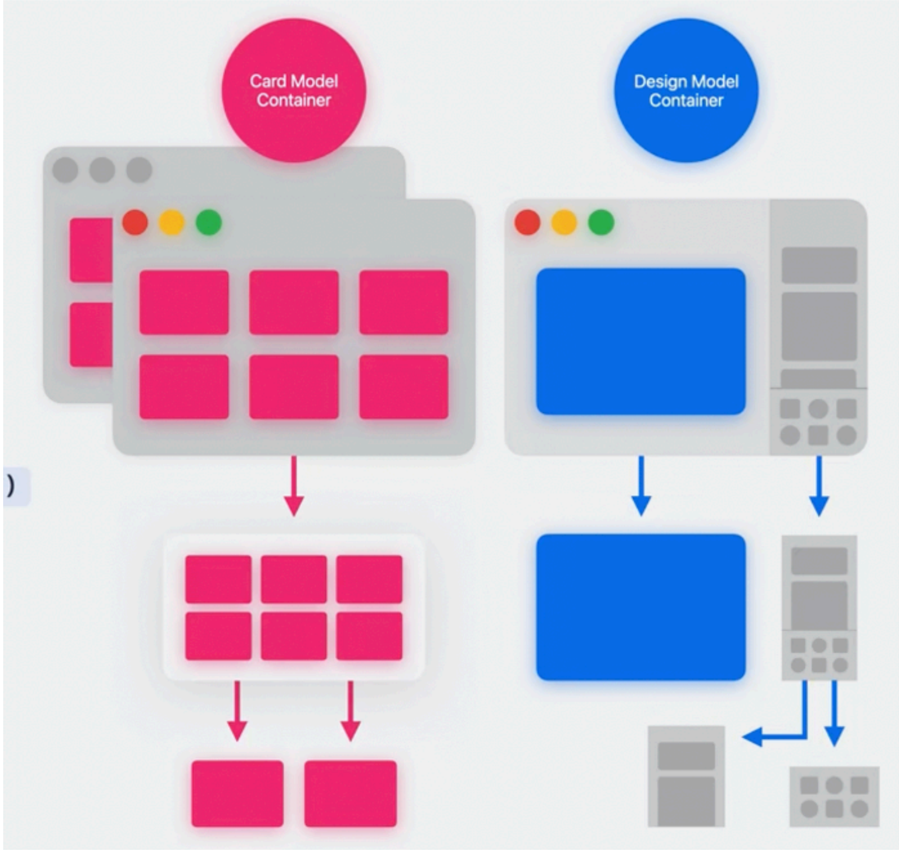
2 Use Models in SwiftUI

Multiple Model Containers 3 Options

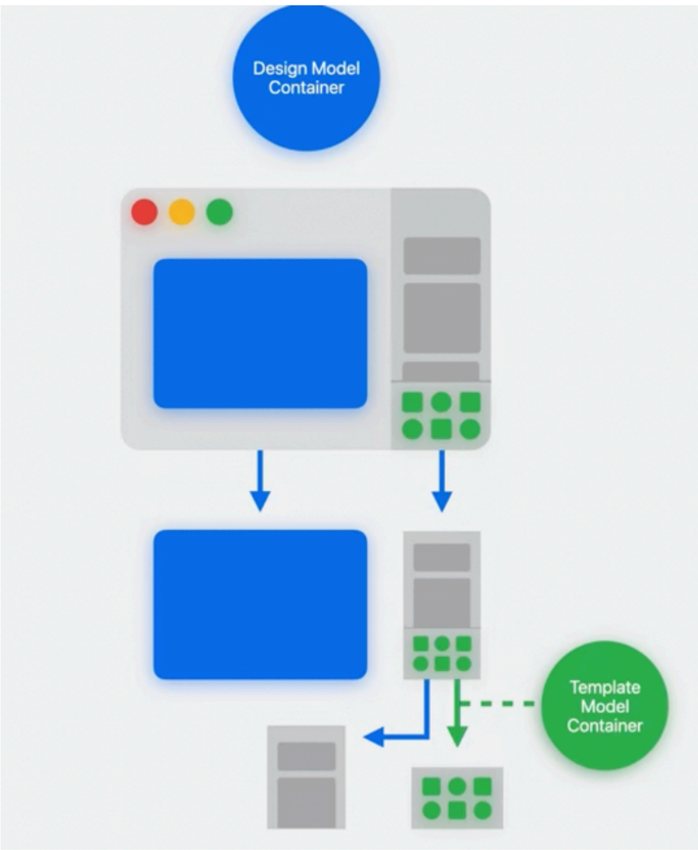
Single



Multiple



Granular



2 Use Models in SwiftUI

A `import SwiftData`

B Setting up the container, using `.modelContainer`

C Replace `@ObservableObject` with `@Bindable` in subviews

It allows the text fields to bind directly to the property

C Query the data from Swift Data Storage using `@Query`

D Accesing the new Model Context `@Environment(\.modelContext)`

2 Use Models in SwiftUI

E Update the content in the ModelContext

```
modelContext.insert(object: …)
```

3 Schema Migration: Example (src: developer.apple.com)

Extending the model for **new app version release**
Problem: Schema might be different now
→ need to handle conversion

A Define `VersionSchema` for each schema version v1,v2,...

A Define `VersionSchema` for each schema version v1 & v2

```swift
enum SampleTripsSchemaV1: VersionedSchema {
    static var models: [any PersistentModel.Type] {
        [Trip.self, BucketListItem.self, LivingAccommodation.self]
    }

    @Model
    final class Trip {
        var name: String
        var destination: String
        var start_date: Date
        var end_date: Date

        var bucketList: [BucketListItem]? = []
        var livingAccommodation: LivingAccommodation?
        // ...
    }

    // Define the other models in this version...
}
```

```swift
enum SampleTripsSchemaV2: VersionedSchema {
    static var models: [any PersistentModel.Type] {
        [Trip.self, BucketListItem.self, LivingAccommodation.self
    }

    @Model
    final class Trip {
        @Attribute(.unique) var name: String
        var destination: String
        var start_date: Date
        var end_date: Date

        var bucketList: [BucketListItem]? = []
        var livingAccommodation: LivingAccommodation?
        // ...
    }

    // Define the other models in this version...
}
```

B Define `MigrationPlan`

2 Types of Changes: `Lightweight` `Custom`

e.g. **`.cascade`** e.g. **`.unique`**

```
enum SampleTripsMigrationPlan: SchemaMigrationPlan {
    static var schemas: [any VersionedSchema.Type] {
        [SampleTripsSchemaV1.self, SampleTripsSchemaV2.self, SampleTripsSchemaV3.self]
    }

    static var stages: [MigrationStage] {
       []
    }

    // ...
}
```

B Define `MigrationPlan` `Lightweight` e.g. **.cascade**

```
enum SampleTripsMigrationPlan: SchemaMigrationPlan {
    static var schemas: [any VersionedSchema.Type] {
        [SampleTripsSchemaV1.self, SampleTripsSchemaV2.self, SampleTripsSchemaV3.self]
    }

    static var stages: [MigrationStage] {
       [migrateV1toV2, migrateV2toV3]
    }

    // ...

    static let migrateV2toV3 = MigrationStage.lightweight(
        fromVersion: SampleTripsSchemaV2.self,
        toVersion: SampleTripsSchemaV3.self
    )
}
```

B Define `MigrationPlan` `Custom` e.g. `.unique`

```
enum SampleTripsMigrationPlan: SchemaMigrationPlan {
    static var schemas: [any VersionedSchema.Type] {
        [SampleTripsSchemaV1.self, SampleTripsSchemaV2.self, SampleTripsSchemaV3.self]
    }

    static var stages: [MigrationStage] {
       [migrateV1toV2]
    }

    static let migrateV1toV2 = MigrationStage.custom(
        fromVersion: SampleTripsSchemaV1.self,
        toVersion: SampleTripsSchemaV2.self,
        willMigrate: { context in
            let trips = try? context.fetch(FetchDescriptor<SampleTripsSchemaV1.Trip>())
            // De-duplicate Trip instances here...
            try? context.save()
        }, didMigrate: nil
    )
}
```

C Perform the actual migration, by adding info to the ModelContainer

```
struct TripsApp: App {
    let container = ModelContainer(
        for: Trip.self,
        migrationPlan: SampleTripsMigrationPlan.self
    )

    var body: some Scene {
        WindowGroup {
            ContentView()
        }
        .modelContainer(container)
    }
}
```

3 Summary: Schema Migration

Extending the model for **new app version release**
Problem: Schema might be different now
→ need to handle conversion

A Define `VersionSchema` for each schema version v1,v2,...

B Define `MigrationPlan`

2 Types of Changes: `Lightweight` `Custom`

C Perform the actual migration, by adding info to the ModelContainer

SwiftData Wrap-Up

Features

- Entirely uses Swift code
- Autosave, undo and redo provided without additional
- **Coexistence** to CoreData
  - → both writing/ modifying objects on the same database
- Works with iCloud ☁️
- offers integration with swift, for background activities, server sync and batch processing

Downsides

- Less documentation & less customizability then CoreData
- Only on **iOS 17+** and **macOS Sonoma** available

Demo

FlashCards App

(src: developer.apple.com)

References

- developer.apple.com
- https://medium.com/ios-os-x-development/10-core-principles-to-use-coredata-without-blowing-your-head-off-5ed11c623c6b
- medium.com