CoreData

Agenda

- Was ist CoreData?
- CoreData Stack
- CoreData Basisimplementierung
- Modellierung von Daten
- Daten speichern mit CoreData
- Daten laden mit CoreData
- Demo

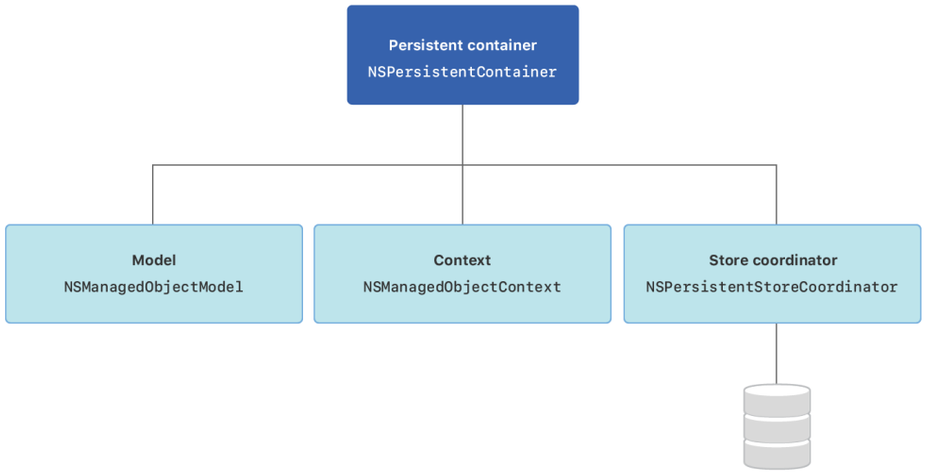
Was ist CoreData?

„Core Data is a framework that you use to manage the model layer objects in your application. It provides generalized and automated solutions to common tasks associated with object life cycle and object graph management, including persistence.“

–Apple

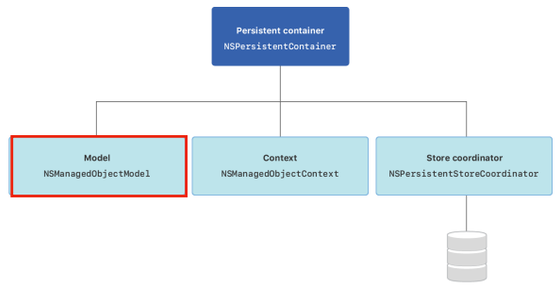
- Framework: Keine Datenbank, sondern Framework zur Verwaltung eines Objektgraphen
- object graph: Viele Objekte, die auf verschiedene Weise miteinander verbunden sind
- object life cycle: Von Erstellung und Veränderung bis zur Zerstörung eines Objektes (CoreData bietet entsprechende Methoden zur Manipulation des Objektgraphen)
- Model Layer: enthält Businesslogik, repräsentiert Zustand und Organisation von Daten
- Persistenz optional

CoreData Stack



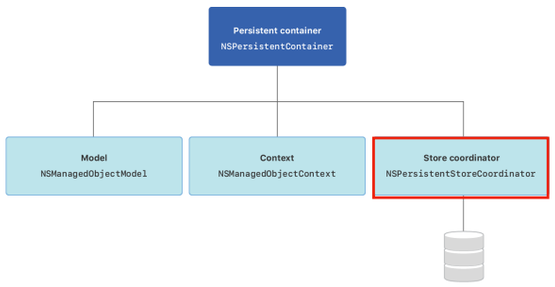
CoreData Stack: Model

- beschreibt die Daten der Applikation
- arbeitet nur mit Objekten (=> NSManagedObject)
- Inhalte der Objekte (=> Entity) müssen vorab definiert werden
  - meist Standard-Datentypen (String, Int, Bool)



CoreData Stack: Store coordinator

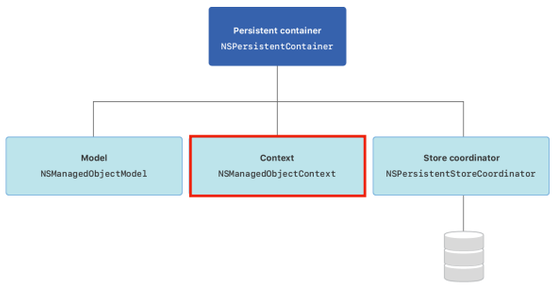
- koordiniert und verwaltet die Persistierung der Daten
- arbeitet mit Persistent Store (dauerhafter Speicher)
  - erzeugt Objekt-Instanzen aus Entitäten im Model
  - holt Objekt-Instanzen von Entitäten aus dem Persistent Store



- Persistent Store unter iOS meistens SQLite, auf Mac auch XML möglich

CoreData Stack: Context

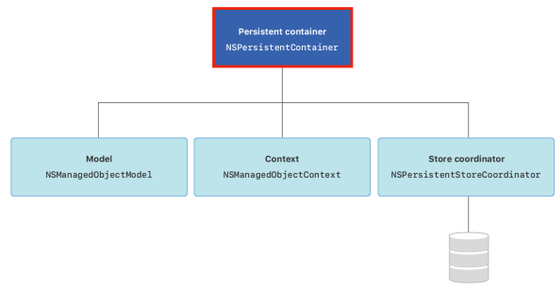
- positioniert im Arbeitsspeicher (=> sehr schnell)
- verfolgt alle Änderungen im Objektgraphen
- bietet Methoden zum Speichern/Laden/Ändern/Löschen von Objekten (=> CRUD)
- „scratch pad“
- alle ManagedObjects gehören zu einem Context



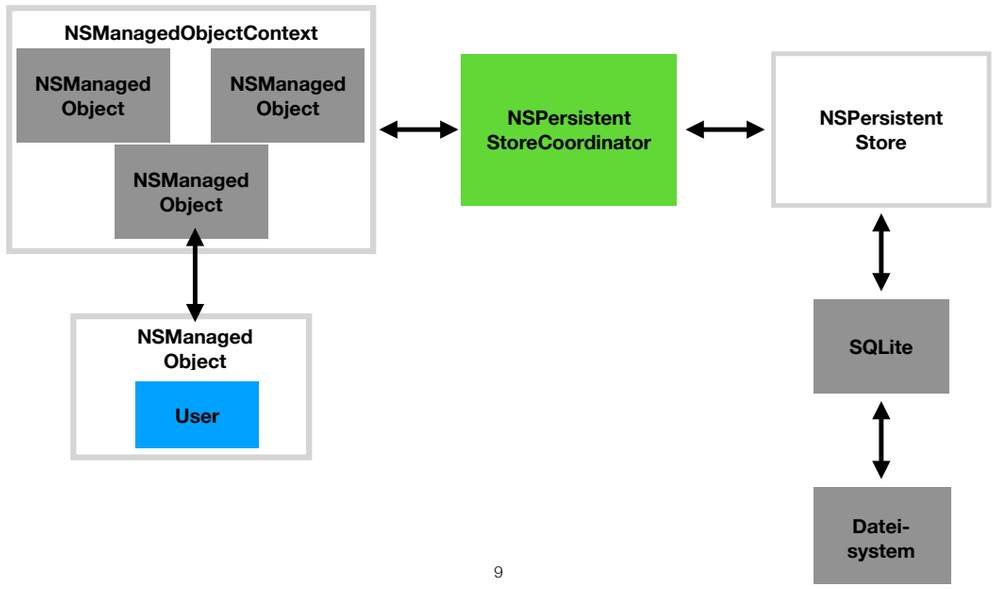
- scratch pad: Objekte aus dem Persistent Store sind verfügbar als temporäre Kopien auf dem Context und formen einen Objektgraphen (viele Objekte irgendwie miteinander verbunden). Können beliebig modifiziert werden, aber Manipulationen werden wieder im Persistent Store gespeichert wenn Methode saveContext() ausgeführt wurde
- Context sorgt für Integrität (valide Bezeichnungen, Beziehungen), bietet zusätzliche Funktionen (z.B. Undo/Redo)

CoreData Stack: Persistent Container

- Wrapper
  - Model
  - Context
  - Store coordinator
  - Persistent Store



CoreData Stack: Verknüpfungen



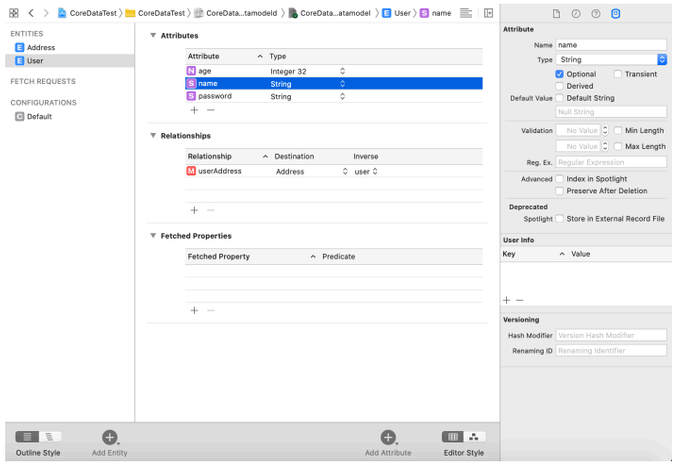
CoreData Basisimplementierung

- Projekt erstellen mit „Include CoreData“ erzeugt:
  - neue Methoden in der AppDelegate
  - Methode in der SceneDelegate
  - .xcdatamodeld-Datei zur Datenmodellierung

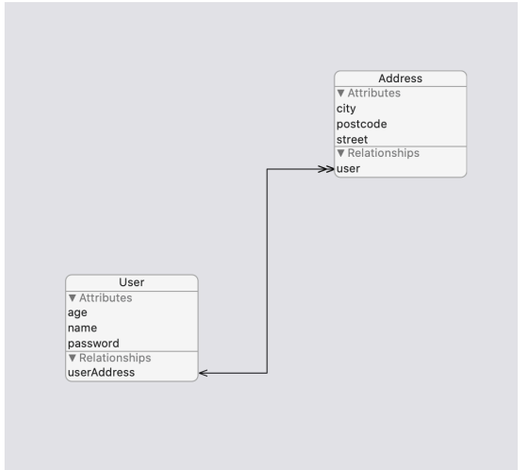
```
// MARK: - Core Data stack

lazy var persistentContainer: NSPersistentContainer = {
    /*
     The persistent container for the application. This implementation
     creates and returns a container, having loaded the store for the
     application to it.
    */
    let container = NSPersistentContainer(name: "CoreDataTest")
    container.loadPersistentStores(completionHandler: { (storeDescription, error) in
        if let error = error as NSError? {
            fatalError("Unresolved error \(error), \(error.userInfo)")
        }
    })
    return container
}()

// MARK: - Core Data Saving support

func saveContext () {
    let context = persistentContainer.viewContext
    if context.hasChanges {
        do {
            try context.save()
        } catch {
            let nserror = error as NSError
            fatalError("Unresolved error \(nserror), \(nserror.userInfo)")
        }
    }
}
```

```
func sceneDidEnterBackground(_ scene: UIScene) {
    // Called as the scene transitions from the foreground to the background.
    // Use this method to save data, release shared resources, and store enough
       scene-specific state information
    // to restore the scene back to its current state.

    // Save changes in the application's managed object context when the application
       transitions to the background.
    (UIApplication.shared.delegate as? AppDelegate)?.saveContext()
}
```

Modellierung von Daten



Modellierung von Daten in XCode



Dazugehöriger Objektgraph

Daten speichern mit CoreData

1. AppDelegate nach dem Kontext fragen
2. Neues ManagedObject erstellen und in den Kontext einfügen
    1. Entity Description verbindet die Entität mit der Instanz eines ManagedObject zur Laufzeit
3. Mit Key-Value Coding die Attribute setzen, dabei sehr genau sein
4. Änderungen „committen“ und Objekt zum Interface hinzufügen

```
guard let appDelegate =
    UIApplication.shared.delegate as? AppDelegate else {
    return
  }
  // 1
  let managedContext =
    appDelegate.persistentContainer.viewContext
  // 2
    let entity = NSEntityDescription.entity(forEntityName: "User",
                                            in: managedContext)!
      let user = NSManagedObject(entity: entity,
                                 insertInto: managedContext)
      // 3
      user.setValue(name, forKeyPath: "name")

      // 4
      do {
        try managedContext.save()
        [redacted]
      } catch let error as NSError {
        print("Could not save. \(error), \(error.userInfo)")
      }
}
```

Daten laden mit CoreData

1. AppDelegate nach dem Context fragen

2. FetchRequest aufsetzen mit Rückgabetyp (hier: NSManagedObject, auch z.B. nur Anzahl der Objekte möglich)

    1. mehrere optionale Filter: NSPredicate, NSSortDescriptor…

3. Fetch-Methode vom Context aufrufen und Ergebnis speichern

```swift
//1
  guard let appDelegate =
    UIApplication.shared.delegate as? AppDelegate else {
      return
  }

  let managedContext =
    appDelegate.persistentContainer.viewContext

  //2
  let fetchRequest =
    NSFetchRequest<NSManagedObject>(entityName: "User")

  //3
  do {
    users = try managedContext.fetch(fetchRequest)
  } catch let error as NSError {
    print("Could not fetch. \(error), \(error.userInfo)")
  }
```

Daten laden mit CoreData - Predicates

```
let name = "Peter";
fetchRequest.predicate = NSPredicate(format: "name == %@", name);
```

```
fetchRequest.predicate = NSPredicate(format: "age >= 30")
```

- Es lassen sich auch mehrere Predicates miteinander verbinden (z.B. name=Peter AND age>=30) => CompoundPredicates

Daten laden mit CoreData - SortDescriptors

```
fetchRequest.sortDescriptors = [NSSortDescriptor(key: "name", ascending: true)]
```

- auch Definition von mehreren SortDescriptors möglich

```
let sortDescriptor1 = NSSortDescriptor(key: "name", ascending: true)
let sortDescriptor2 = NSSortDescriptor(key: "date", ascending: true)
fetchRequest.sortDescriptors = [sortDescriptor1, sortDescriptor2]
```

Demo

Übung

- Aufgabe 1: Übung aus GitHub laden (https://github.com/tolja/fsios-workshop-coredata). Datenmodell definieren. Markierte Aufgaben in der UserTableViewController.swift erledigen
- Bonus-Aufgabe 2: Mit Alert-Fenster Nutzereingaben abfragen
- Bonus-Aufgabe 3: Mit Predicate nur Benutzer mit dem Namen „Herbert“ anzeigen lassen

Literatur

- Apple „Core Data Programming Guide“ (https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CoreData/index.html)
- Apple WWDC 2019 Video zu Core Data (https://developer.apple.com/videos/play/wwdc2019/230/)
- Ray Wenderlich: Core Data by Tutorials (https://store.raywenderlich.com/products/core-data-by-tutorials)
- Stanford Kurs (https://www.youtube.com/watch?v=3wkM4Ml7p4o&list=PLprb6BoXapmVu8XlveDbua6J0_tRE14WX&index=11&t=0s und https://www.youtube.com/watch?v=UL_aFn9-sQI&list=PLprb6BoXapmVu8XlveDbua6J0_tRE14WX&index=11)
- Udemy: Swift 5 iOS 13 App Entwickler Kurs (https://www.udemy.com/course/swift-5-ios-13-der-ultimative-ios-13-app-entwickler-kurs/)