Basic iOS interview questions and answers

Thursday, 29 May 2025 07:26

1. რა არის Coredata

Coredata - არის buildin framework, რომელიც გამოიყენება მონაცემების offline რეჟიმში გამოყენებისთვის, ანუ Data-ს cache-ში შენახვისთვის.
მაგ: ქსელი მოხდა [ინტერნეტიდან დაკავშირებული ობიექტების დატა სერვერიდან] ანუ [დატა მონაცემები], რაც რომ ის მოგვიანებით არ იყოს ჩატვირთული შენახულიყო.
თუ აქამდე Coredata-ს გარეშე დატა მონაცემების ვიღებდით ჩვეულებრივ ინტერნეტით და გვქონდა ინტერნეტი, core data გვიჩვენებს ბოლო ჩატვირთულს, რომ ჩვეულებრივ შემოსული მონაცემები, რომლებიც ნაჩვენები სერვერიდან იყო, და როცა აღარ სერვერი გვქონდა ჩვენ ბოლო ჩატვირთულს.

Coredata - ობიექტ-გრაფის მართვის ფრეიმვორკი და მონაცემთა შენახვის ფუნქციონალობა, რომელიც ხშირად ეხმარება ხელით გარეშე ფუნქციონალობას: caching, object relationship management, undo-redo ფუნქციები, versioning. ასევე Xcode-ში არსებობს Data Model Editor ვიზუალური მოდელის შექმნის და ვიზუალური ხედის.

მას რამდენიმე ნაკლი აქვს.

- core data-ს მთავარი ნაკლი არის ის, რომ ... cache data-ზე მთლიანად არ არის დამოკიდებული.
- თუ ბევრი რიგი ... „record", Core data-ს „record"-ის შენახვისას ... ჩატვირთვისას.
- Core Data-ს გამოყენება მოსალოდნელია მხოლოდ thread-ზე. [ასევე გამოყენება ... Multithreading გამოყენება CoreData-სთვის რთულია]
- თუ აპლიკაციას აქვს მარტივი Object graph, Core Data არის ... (მაგ: ...), ... შენახვისთვის UserDefaults.

NSManagedObject - მონაცემების ობიექტი (მაგ: User, Product.)

NSManagedObjectContext - ადგილი სადაც ქმნი და შეცვლი ობიექტს

NSPersistentContainer - მთელი Core Data სისტემის მართვა და მონაცემების შენახვა.

Core Data code example
აქ არის [playground code]

```swift
import Foundation
import CoreData

// Step 1: Define the model programmatically
class Person: NSManagedObject {
    @NSManaged var name: String
}
```

→ შევქმენით კლასი Person, რომელიც მემკვიდრეობით იღებს NsManagedObject-ს ანუ მისი მემკვიდრეა, რაც ნიშნავს რომ Person კლასი გახდა Core Data Object.

→ NsManaged-ნიშნავს, რომ Core Data დააგენერირებს setter/getter-ს Name - property-სთვის

```swift
// Step 2: Set up Core Data stack (in-memory for Playground)
let model = NSManagedObjectModel()

let personEntity = NSEntityDescription()
personEntity.name = "Person"
personEntity.managedObjectClassName = NSStringFromClass(Person.self)

let nameAttribute = NSAttributeDescription()
nameAttribute.name = "name"
nameAttribute.attributeType = .stringAttributeType
nameAttribute.isOptional = false

personEntity.properties = [nameAttribute]
model.entities = [personEntity]

let container = NSPersistentContainer(name: "PlaygroundModel", managedObjectModel: model)

let description = NSPersistentStoreDescription()
description.type = NSInMemoryStoreType
container.persistentStoreDescriptions = [description]
```

→ შევქმენით Core Data Model ობიექტი. [ჩვეულებრივ პროექტებში ასეთი იქმნება .xcdatamodeld ფაილი playground-ში, მას ხელით ვქმნით]

→ შევქმენით Core Data Entity "person" [როგორც .xcdatamodeld] შევქმენით Entity Person კლასის დაკავშირებით NSStringFromClass-ის გამოყენებით.

→ დავამატეთ "name" ატრიბუტი
- isOptional = Fals (ნიშნავს, რომ სახელი არ შეიძლება იყოს nil.

→ - დავამატეთ "name" ატრიბუტი Person Entity-ს
- დავამატეთ Person Entity მთლიან Data Model-ს.

შევქმენით კონტეინერი, რომელიც იჭერს Stack-ს.

დავაყენეთ მეხსიერება „სწრაფი" მეხსიერებაში [რაც იდეალურია playground ტესტირებისთვის]
- ანუ ყველაფერი დროებით ინახება, არ მუდმივად დისკზე ... მეხსიერებაში.

```swift
// Load persistent stores
container.loadPersistentStores { (desc, error) in
    if let error = error {
        fatalError("Failed to load store: \(error)")
    }
}
```

→ store ანუ საცავის ჩატვირთვა, შეცდომების დამუშავება.

```swift
// Step 3: Create and save data
let context = container.viewContext
```

→ ეს არის სამუშაო კონტექსტი რომელიც ვიყენებთ ობიექტების შექმნის, და fetch-ისთვის.
- წარმოიდგინე მაგიდა ისე, რომ ჩანიშნავენ დროებითი workspace RAM-ში CoreData ობიექტებისთვის.

```swift
let person = Person(entity: personEntity, insertInto: context)
person.name = "Nini"

do {
    try context.save()
    print("Saved successfully ✅")
} catch {
    print("Failed to save ❌: \(error)")
}

// Step 4: Fetch and print data
let fetchRequest = NSFetchRequest<Person>(entityName: "Person")

do {
    let results = try context.fetch(fetchRequest)
    for person in results {
        print("👤 Name: \(person.name)")
    }
} catch {
    print("Fetch failed: \(error)")
}
```

→ შევქმენით ახალი Person ობიექტი, დავაინსერტეთ context-ში set Name = „Nini"

→ context.save - ანუ ვცდილობთ და ვინახავთ შენახვა Memory-ში

→ დავაყენეთ Fetching request-ი „Person"-ის ყველა ობიექტის მისაღებად

→ ვბეჭდავთ fetch-შედეგებს

Concept	Meaning
NSManagedObjectModel	Defines the structure of the data model in memory
NSEntityDescription	Defines one "table"/entity (like a class)
NSAttributeDescription	Defines one property (like a column in a table)
NSPersistentContainer	Core Data engine that manages context and storage
NSManagedObjectContext	Workspace where you add/edit/delete Core Data objects
.save()	Saves changes to storage (in-memory or disk)
NSFetchRequest	Allows you to read (fetch) objects from Core Data