SWIFT REALMS:-

A DEEP DIVE INTO REALM DATABASE

Presented by: Amitesh Mani

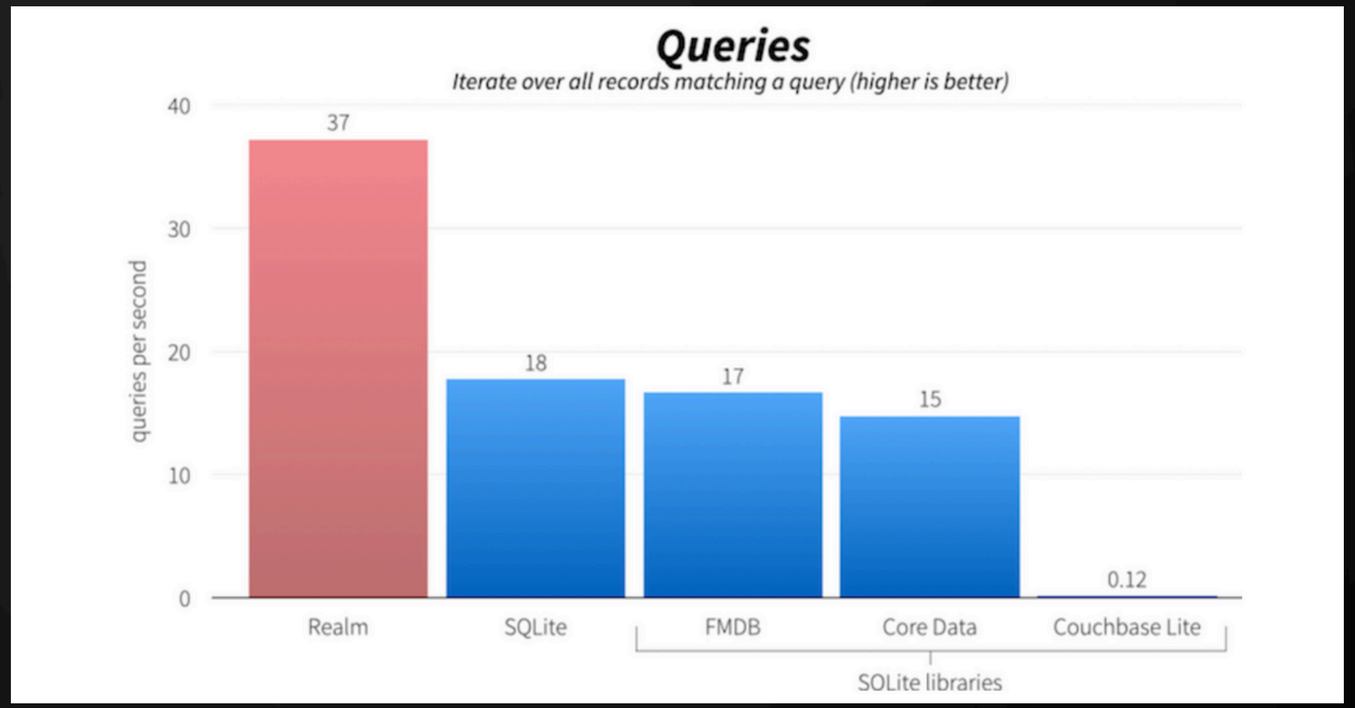
Software Engineer (iOS Team) @Upstox📈 || 35+ Non Tech/Tech Talks🎶 || 9000 Students Impacted, 21U21 Award Winner

Overview

- *Why Realm?*
- Realm's Architecture
- *Setting Up Realm in an iOS Project*
- *Basic CRUD Operations*
- *Advanced Techniques*
- *Best Practices*
- *Power Up Your Startup with Realm Swift*
- *Q&A*

Why Realm?

- A mobile database designed for simplicity and speed.
- Cross-platform support.
- High performance and low memory footprint
- Easy-to-use APIs
- Seamless data persistence for offline applications.



Realm's Architecture

- Object-Oriented: Store data as Swift objects, not tables. Easier to manage and aligns with Swift's nature.
- Live Objects: Always up-to-date. Changes in the database reflect instantly in your app.
- Zero-Copy Architecture: Fast and efficient. Maps data directly from disk, reducing memory usage.
- Multi-Version Concurrency: Handles multiple reads/writes without conflicts.
- Reactive Programming: Get notified of data changes. Easy UI updates and integration with reactive frameworks.

Setting Up Realm in an iOS Project

CocoaPods

- Add **pod 'RealmSwift'** to your **Podfile**.
- Run **pod install** to install the Realm frameworks.
- Open the **.xcworkspace** file to start using Realm.

Swift Package Manager

- Open your project in Xcode.
- Go to File > Add Packages....
- Enter https://github.com/realm/realm-cocoa in the search bar.
- Choose the appropriate version and add the package to your project.

Carthage

- Add github "realm/realm-cocoa" to your Cartfile.
- Run carthage update to build the frameworks.
- Drag the built Realm.framework and RealmSwift.framework into your Xcode project.

Basic CRUD Operations

Create:

```swift
import RealmSwift

class Member: Object {
    @objc dynamic var name = ""
    @objc dynamic var email = ""
    @objc dynamic var joinedDate = Date()
}

// Create a new Member object
let newMember = Member()
newMember.name = "Amitesh Mani"
newMember.email = "amiteshn.tiwari@speaker.com"

// Get the default Realm
let realm = try! Realm()

// Add the member to the Realm
try! realm.write {
    realm.add(newMember)
}
```

Read:

```swift
// Fetch all Member objects from the Realm
let members = realm.objects(Member.self)

// Print out the names of all the members
for member in members {
    print(member.name)
}
```

Update:

```swift
// Fetch the member to update
if let attende = realm.objects(Member.self).filter("name == 'Amitesh Mani'"
    ).first {
    // Update the member's email
    try! realm.write {
        attende.email = "amiteshn.newemail@example.com"
    }
}
```

Delete:

```swift
// Fetch the member to delete
if let attendeDelete = realm.objects(Member.self).filter("name == 'Amitesh
    Mani'").first {
    // Delete the member from the Realm
    try! realm.write {
        realm.delete(attendeDelete)
    }
}
```

Advanced Techniques

Relationships:

```
52  // Create a group and add members
53  let swiftGroup = Group()
54  swiftGroup.name = "Swift Developers"
55  swiftGroup.members.append(newMember)
56
57  // Add the group to the Realm
58  try! realm.write {
59      realm.add(swiftGroup)
60  }
```

Migrations:

```
let config = Realm.Configuration(
    schemaVersion: 1,
    migrationBlock: { migration, oldSchemaVersion in
        if oldSchemaVersion < 1 {
            // Perform any necessary migration steps
            migration.enumerateObjects(ofType: Member.className()) {
                oldObject, newObject in
                // Example: Renaming a property
                newObject!["fullName"] = oldObject!["name"]
            }
        }
    }
)

// Apply the new configuration
Realm.Configuration.defaultConfiguration = config

// Access the Realm with the updated schema
let realm = try! Realm()
```

Notifications:

```
// Observe Realm notifications
let token = realm.observe { notification, realm in
    // React to changes in the Realm
    print("Realm data changed")
}

// Later, when you no longer need to observe changes
token.invalidate()
```

Best Practices for Using Realm (Swift Bengaluru Community Database)

Efficient Querying:

```
// Fetch members who joined in the last year and sort by joined date
let lastYear = Calendar.current.date(byAdding: .year, value: -1, to: Date
    ())!
let recentMembers = realm.objects(Member.self).filter("joinedDate > %@",
    lastYear).sorted(byKeyPath: "joinedDate", ascending: false)
```

Memory Management

- **Autoreleasepool:**

```
DispatchQueue.global(qos: .background).async {
    autoreleasepool {
        let realm = try! Realm()
        let allMembers = realm.objects(Member.self)
        // Process large datasets efficiently
    }
}
```

Concurrency

- **Thread Safety:**

```
DispatchQueue(label: "background").async {
    let realm = try! Realm()
    let members = realm.objects(Member.self)
    // Perform operations on background thread
}
```

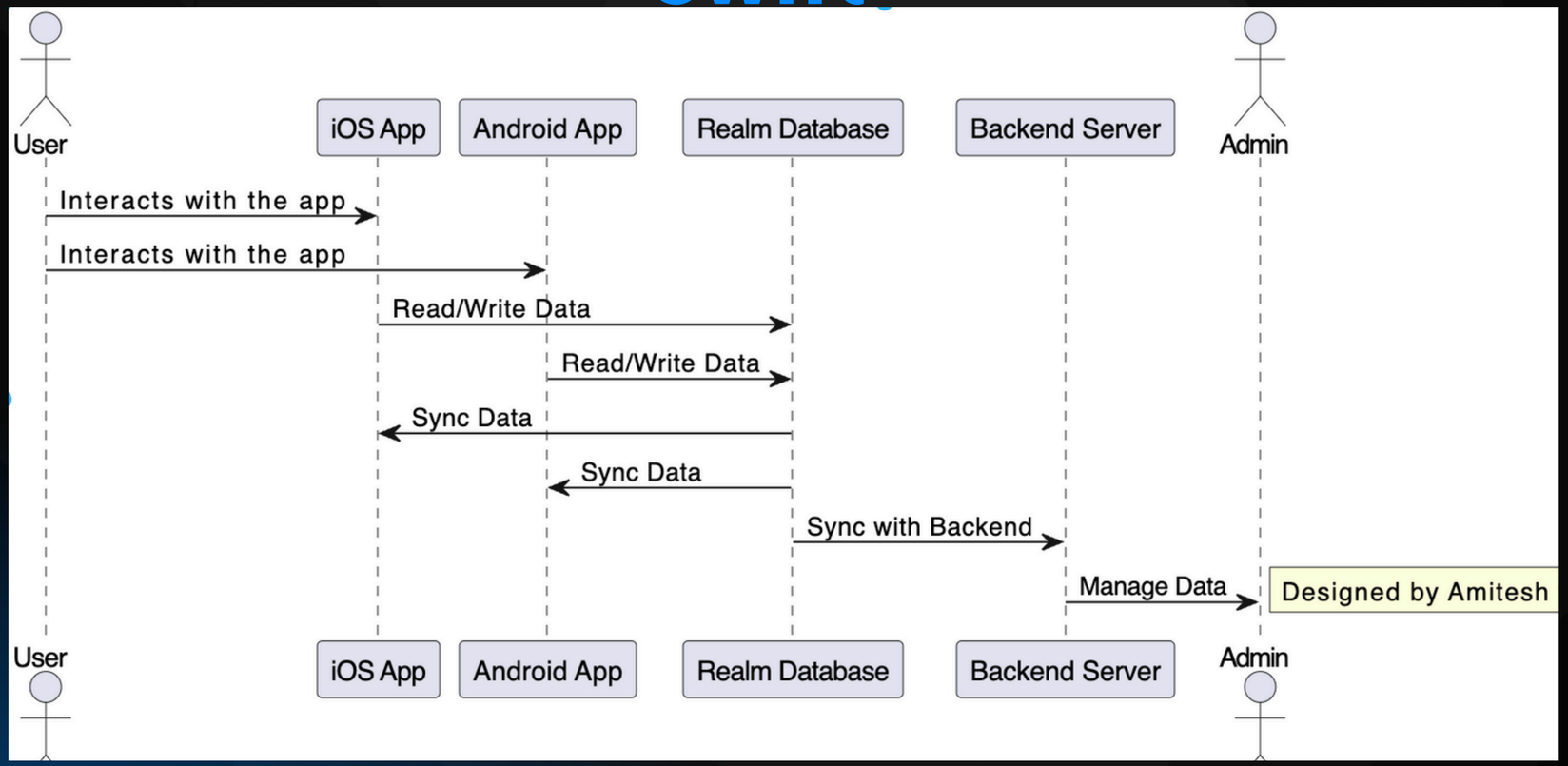
**Realm instances are not thread-safe.
Use separate instances for background threads.

Using Realm Notifications

```
// Observe changes to the Member objects
let notificationToken = realm.objects(Member.self).observe { changes in
    switch changes {
    case .initial(let members):
        // Initial data loaded
        print("Loaded \(members.count) members")
    case .update(let members, let deletions, let insertions, let
        modifications):
        // Handle updates
        print("Inserted: \(insertions), Deleted: \(deletions), Modified: \
            (modifications)")
    case .error(let error):
        // Handle error
        print("Error: \(error)")
    }
}

// Invalidate the token when done observing
notificationToken.invalidate()
```

Power Up Your Startup with Realm Swift



Q&A

THANK YOU

For watching this presentation

+91 9354615057

amiteshmanitiwari1997@gmail.com

https://github.com/swiftwithamitesh