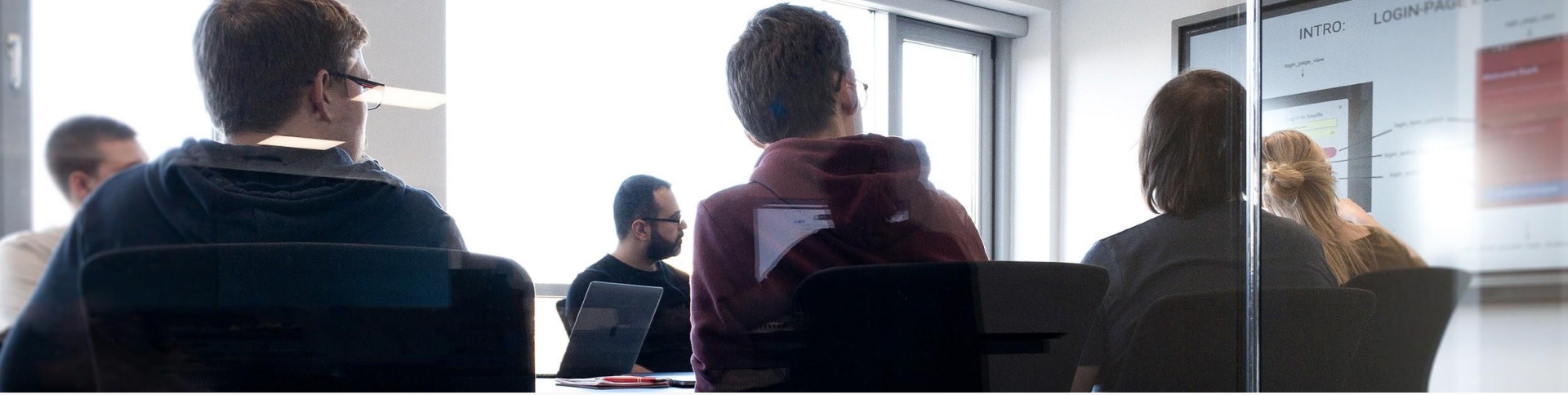

amsiq

Today

- Amsiq
- Presentation
  - Realm (Local database)
  - RxSwift (Reactive programming)
- Exercises



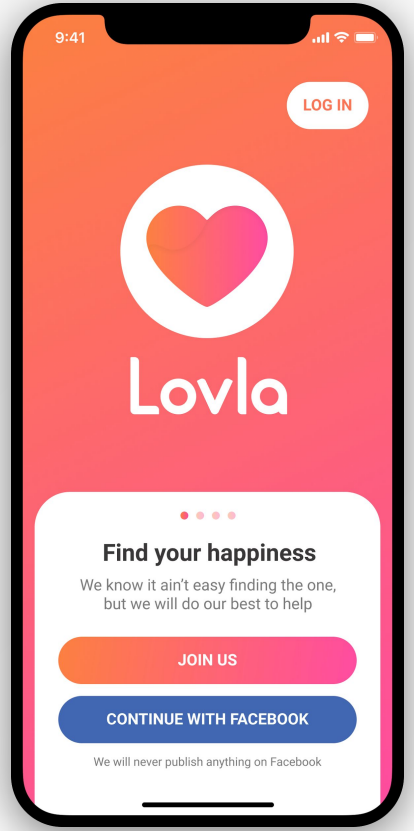
About us

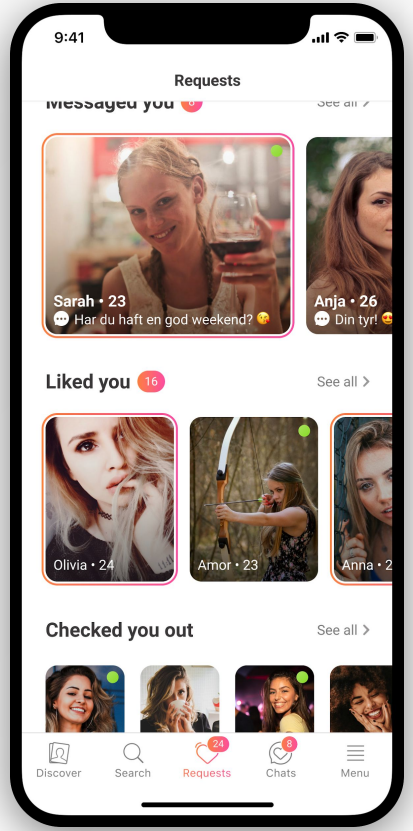
Software house founded in 2009
Jernbanepladsen 1, 2800 Kongens Lyngby

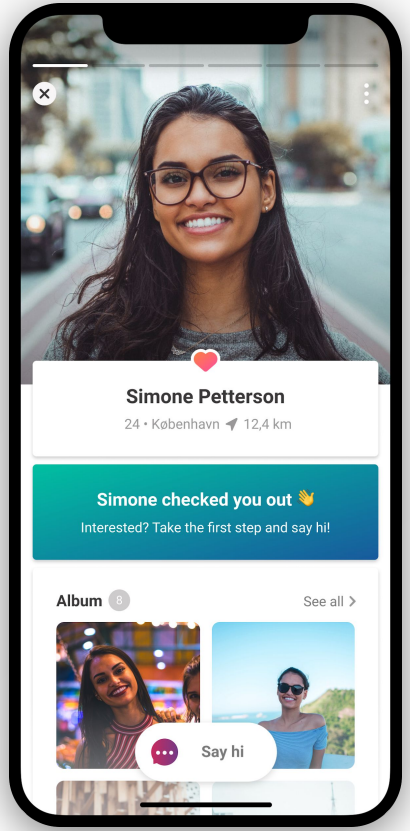
Platforms: iOS, Android, Web & Backend

Soudfa

The future at Amsiq!







Job opportunities

Internships

Student-assistant

Full-time

Find us here: https://www.amsiq.com

Realm

Object database

Supports Cocoapod & Carthage

Alternative to Core-data

Usage

Documentation: https://realm.io/docs/swift/latest
Github: https://github.com/realm/realm-cocoa
Read more about pros and cons: https://medium.com/@andy_41059/coredata-vs-realm-in-swift-826a647ddeb7

Realm

Flow

- Install Realm
- Setup Realm
- Define Realm models
- I/O operations
  - Write
  - Update (Primary keys - Duplication in DB + Override existing object)
  - Read
- Browser
- Extensions
- Relationships
- Kotlin comparison

Demo-project can be found on:
https://github.com/amsiq/demo-ios.git

Realm - Installation

Via Cocoapods:

1. Add ‘pod RealmSwift’ to your Podfile
2. Run ‘pod install’ in your terminal

Realm - Setup

1. File location

a. Default location
b. Shared location
c. Two databases

2. Migration strategy

Changing models

a. Delete
b. Custom migration

3. Compression

Compress database if over 100 MB etc.

Find the example at: https://realm.io/docs/swift/latest/

Type	Non-optional	Optional
Bool	`@objc dynamic var value = false`	`let value = RealmOptional<Bool>()`
Int	`@objc dynamic var value = 0`	`let value = RealmOptional<Int>()`
Float	`@objc dynamic var value: Float = 0.0`	`let value = RealmOptional<Float>()`
Double	`@objc dynamic var value: Double = 0.0`	`let value = RealmOptional<Double>()`
String	`@objc dynamic var value = ""`	`@objc dynamic var value: String? = nil`
Data	`@objc dynamic var value = Data()`	`@objc dynamic var value: Data? = nil`
Date	`@objc dynamic var value = Date()`	`@objc dynamic var value: Date? = nil`
Object	`n/a: must be optional`	`@objc dynamic var value: Class?`
List	`let value = List<Type>()`	`n/a: must be non-optional`
LinkingObjects	`let value = LinkingObjects(fromType: Class.self, property: "property")`	`n/a: must be non-optional`

Realm

Models

Realm - Write

Flow

1. Create an instance of Realm
2. Create the model
3. Execute the write transaction

Realm - Browser

Database overview

1. Install Realm Studio
    a. Use brew: ‘brew cask install realm-studio’
    b. Manual: https://realm.io/products/realm-studio/
2. Find Realm file
    a. Locate using lldb ‘po Realm.Configuration.defaultConfiguration.fileURL’
        i. Simulator: Placed on computer in folder: /Users/’user’/Library/Developer/CoreSimulator/Devices/
        ii. Device: Placed on device in folder: /var/mobile/Containers/Data/Application/’id’/Documents/
3. Open file through Realm Studio
    a. Simulator: Navigate to path and open
    b. Device: Download file from device. Xcode -> Window -> Devices -> Download container -> Show Package Contents -> AppData/Documents/’realm_path’

How to find a Realm file: https://medium.com/@agungsantoso/how-to-find-realm-file-3ecdce39a57b

Realm - Update (Primary keys)

Flow

1. Define primary key in the model
2. Allow updates in write transaction

Realm - Read

Flow

1. Find object (could use primary id)

Realm - Extensions

Remove (force try)/(individual catch handling)

1. Create instance
2. Write transaction

Realm - Parse and persist

Different way of parsing and persisting

Manual

```
class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let realm = Realm.create()

        let person = Person.parse(json: personJSON)
        realm.safeWrite {
            _ = person.persist(realm: realm)
        }
    }
}
```

```
class Person: Object {
    override static func primaryKey() -> String? {
        return "id"
    }

    @objc dynamic var id: Int = 0
    @objc dynamic var firstName: String = ""
    @objc dynamic var lastName: String = ""
    @objc dynamic var age: Int = 0

    class func parse(json: JSON) -> Person {
        let person = Person()
        person.id = json["id"].int!
        person.firstName = json["firstName"].string!
        person.lastName = json["lastName"].string!
        person.age = json["age"].int!
        return person
    }

    func persist(realm: Realm) {
        _ = realm.create(Person.self, value: self, update: true)
    }
}
```

Realm - Parse and persist

Different way of parsing and persisting

Realm

```
class Person: Object {
    override static func primaryKey() -> String? {
        return "id"
    }

    @objc dynamic var id: Int = 0
    @objc dynamic var firstName: String = ""
    @objc dynamic var lastName: String = ""
    @objc dynamic var age: Int = 0

    class func parseAndPersist(realm: Realm, json: JSON) -> Person {
        return realm.create(Person.self, value: json.dictionaryObject!, update: true)
    }
}
```

```
class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let realm = Realm.create()
        realm.safeWrite {
            let person = Person.parseAndPersist(realm: realm, json: personJSON)
        }
    }
}
```

Realm - Relationship

Types:

1. One-to-one
2. One-to-many (Many-to-one)
3. Many-to-many

1. A person with one animal
2. A person with a list of animals
3. A list of persons with a list of animals

Realm - Comparison

Swift

```swift
import RealmSwift
import SwiftyJSON

class Person: Object {
    override static func primaryKey() -> String? {
        return "id"
    }

    @objc dynamic var id: Int = 0
    @objc dynamic var firstName: String = ""
    @objc dynamic var lastName: String = ""
    @objc dynamic var age: Int = 0

    let animals = List<Animal>()

    class func load(realm: Realm, id: Int) -> Person? {
        return realm.objects(Person.self).first(where: { $0.id == id })
    }

    class func parseAndPersist(json: JSON, realm: Realm) -> Person {
        let dic = json.dictionaryObject! // handle optional?
        return realm.create(Person.self, value: dic, update: true)
    }
}
```

```swift
import UIKit
import RealmSwift
import SwiftyJSON

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let realm = Realm.create()

        // Create
        let json = self.getPersons()
        let personsJson = json["persons"]

        // Persist
        realm.safeWrite {
            let persons = personsJson.arrayValue.map({ Person.parseAndPersist(json: $0, realm: realm) })

            persons.forEach {
                debugPrint("Saved: \($0.firstName) \($0.lastName) with animals: \(Array($0.animals.map({ $0.name })))")
            }
        }
    }


    private func getPersons() -> JSON {
        let path = Bundle.main.path(forResource: "Persons", ofType: "json")!
        let jsonString = try! String(contentsOfFile: path, encoding: .utf8)

        return JSON(parseJSON: jsonString)
    }
}
```

Kotlin

```kotlin
import io.realm.*
import io.realm.annotations.PrimaryKey
import org.json.JSONObject

open class Person : RealmObject() {
    @PrimaryKey
    var id: Long = 0

    var firstName: String = ""
    var lastName: String = ""
    var age: Int = 0
    var animals: RealmList<Animal>? = RealmList()

    companion object {
        fun load(realm: Realm, id: Long?): Person? {
            return realm.where(Person::class.java).equalTo("id", id).findFirst()
        }

        fun parseAndPersist(realm: Realm, json: JSONObject): Person? {
            return realm.createOrUpdateObjectFromJson(Person::class.java, json)
        }
    }
}
```

```kotlin
import com.google.gson.JsonObject
import com.soudfa.devutilities.extensions.safeWrite
import io.realm.Realm
import org.androidannotations.annotations.*

@EViewGroup(R.layout.view_notification_native)
open class ViewWithRealm(context: Context, attrs: AttributeSet?): FrameLayout(context, attrs) {

    @AfterViews
    fun afterViews() {
        val realm = Realm.getDefaultInstance()

        // Create
        val json = getPersons()
        val personsJson = json["persons"]

        // Persist
        realm.safeWrite {
            val persons = personsJson.asJsonArray.map { json -> Person.parseAndPersist(realm, json.asJsonObject) }

            persons.forEach { person ->
                Log.d("", "Saved: ${person?.firstName} ${person?.lastName} with animals: ${person?.animals?.map { animal -> animal.name }}")
            }
        }
    }

    private fun getPersons(): JsonObject {
        // Load json file
        return JsonObject()
    }
}
```

Realm - Nice to know

Encryption

Syncing

Realm notifications (Change listeners)

Questions?

Coffee break!

Juhu!

Network - Presetup

Allow HTTP requests

Network manager

Request builder

JSON pretty printer

RxSwift

What is Rx?

Multi-platform standard (Web, Android & iOS)

Difficult asynchronously code becomes easier to write and more logical to read

Handle concurrent tasks (Like user-input & network calls etc)

RxSwift

What is Rx?

A sequence of events, which you subscribe on

- Can emit zero or more events through lifetime
- Emits events to onNext(), onError() and onCompleted()
- subscribeOn()
- doOn()

Read more at: https://medium.com/ios-os-x-development/learn-and-master-%EF%B8%8F-the-basics-of-rxswift-in-10-minutes-818ea6e0a05b

RxSwift

What is Rx?

Disposebag

- Cancellation of observables
- Automatic cancellation of observables on deinit of bag

Transform (Map)

Filter (Debounce)

RxSwift

Intro

- RxSwift
- Simple printer
- Schedulers
- RxAlamofire

RxSwift - Observable vs. Delegates

Implementation with delegates

```
class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        PersonsAPI.get(delegate: self)
    }
}

extension ViewController: PersonsApiDelegate {

    func personsFetchSucceeded(json: JSON) {
        debugPrint(json.prettyPrintedString())
    }

    func personsFetchFailed(error: Error) {
        debugPrint(error)
    }
}
```

```
protocol PersonsApiDelegate {
    func personsFetchSucceeded(json: JSON)
    func personsFetchFailed(error: Error)
}

class PersonsAPI {

    class func get(delegate: PersonsApiDelegate) {
        let urlComponents = RequestBuilder.getApiComponents(path: "/persons", queryItems: nil)

        Alamofire.request(urlComponents.url!).validate().responseJSON { response in
            switch response.result {
            case .success:
                delegate.personFetchSucceeded(json: JSON(response.data!))

            case .failure(let error):
                delegate.personFetchFailed(error: error)
            }
        }
    }
}
```

RxSwift - Observable vs. Delegates

Implementation with observables

```
class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        _ = PersonsAPI.get()
            .subscribe(onNext: { (json) in
                debugPrint(json.prettyPrintedString())
            }, onError: { (error) in
                debugPrint(error)
            })
    }
}
```

```
class PersonsAPI {

    class func get() -> Observable<JSON> {
        let urlComponents = RequestBuilder.getApiComponents(path: "/persons", queryItems: nil)

        let manager = NetworkManager.shared.manager
        return manager.rx.request(.get, urlComponents.url!, parameters: nil, encoding: JSONEncoding.default, headers: nil)
            .map { data in
                return JSON(data)
            }
    }
}
```

RxSwift - Installation

Via Cocoapods:

1. Add ‘pod RxSwift’ + ‘pod RxAlamofire’ to your Podfile
2. Run ‘pod install’ in your terminal

RxSwift - API

Fetch data from server

PersonsAPI
GET http://localhost:7000/persons

RxSwift - Usage

RxSwift - Nice to know

Life cycle (DisposeBag)

Threading

Retry

Error-handling

Debounce (UI input etc.)

Delay subscriptions

Combine: Merge, Concat & Zip

Exercises

1. Download demo project, run server and application on simulator
2. Realm (Use Realm-branch)
    a. Save a new person to Realm
    b. Add an optional variable of type ‘Bool, Int, Float or Double’ to the person model, save, update and read the value from a person
    c. Implement a new list of objects inside the person model
2. RxSwift (Use Rx-branch)
    a. Extend the API to include a PUT/PATCH, and change a specific person name
    b. Handle error received from the server, when inserting a ‘$ ‘in personsAPI GET method.
3. Realm + RxSwift
    a. Fetch data from the server, and save it into the database. Visualise the data in the application

Thanks for your attention