Realm & Reactive

Patrick Kaalund & Magnus Hansen

Today

Introduction

Presentation

- Realm (Local database)
- RxSwift (Reactive programming)

Exercises

Introduction

About us

Patrick

- iOS developer (4 years)
- Cardlay

Magnus

1. Backend + Database developer
2. iOS development (1 year)
3. Amsiq

Both:

1. DTU
2. OverallApps

Realm

Object database

Supports Cocoapod & Carthage & SPM

Alternative to Core-data

Usage

- Amazon
- Dropbox
- Netflix
- Ebay

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
- Realm Browser
- Extensions
- Relationships

Realm - Installation

Via Cocoapods:

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

Via Carthage:

1. Add ‘github "realm/realm-cocoa"’ in your Cartfile
2. Run ‘carthage update --platform iOS --cache-builds’ in your terminal

Via SPM:

1. Add ‘https://github.com/realm/realm-cocoa.git’ in your Swift Packages under project settings

Realm - Setup

1. File location

a. Default location
b. Shared location
c. Two databases
d. In memory

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/

Realm Models

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 - Write

Flow

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

- Managed objects
    - Changing data directly in database
    - Not shareable across threads

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

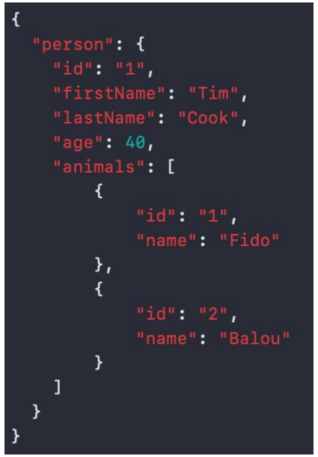
Remove (force try)/(individual catch handling)

1. Create instance
2. Write transaction

Realm - Parse and persist

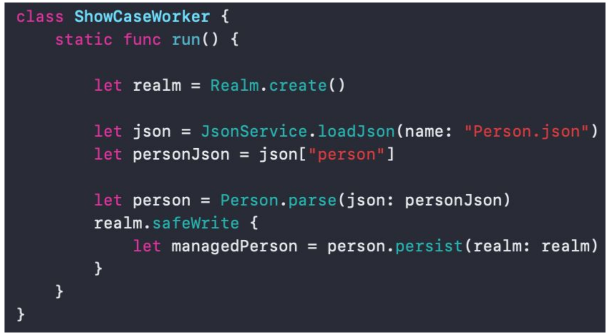
Different way of parsing and persisting

Manual through SwiftyJSON (Alternative Codeable)



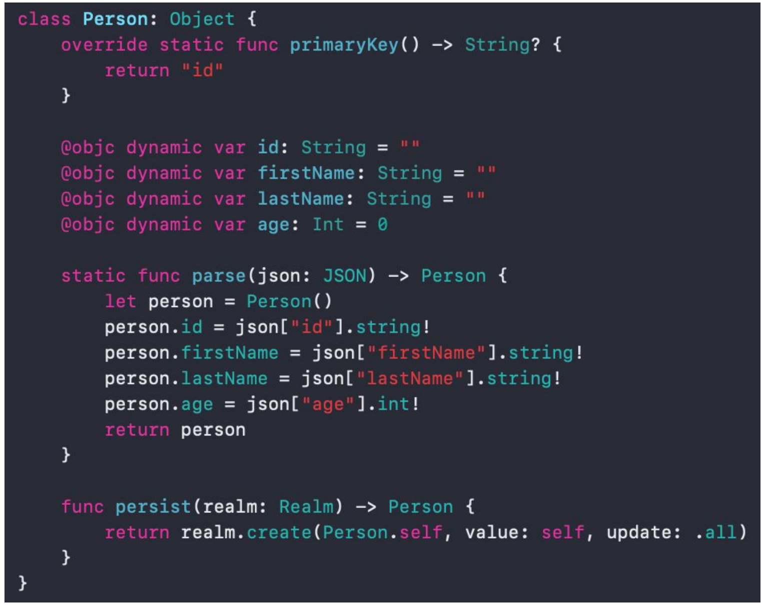
```json
{
  "person": {
    "id": "1",
    "firstName": "Tim",
    "lastName": "Cook",
    "age": 40,
    "animals": [
        {
            "id": "1",
            "name": "Fido"
        },
        {
            "id": "2",
            "name": "Balou"
        }
    ]
  }
}
```

Person.json



```swift
class ShowCaseWorker {
    static func run() {

        let realm = Realm.create()

        let json = JsonService.loadJson(name: "Person.json")
        let personJson = json["person"]

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

realm controller



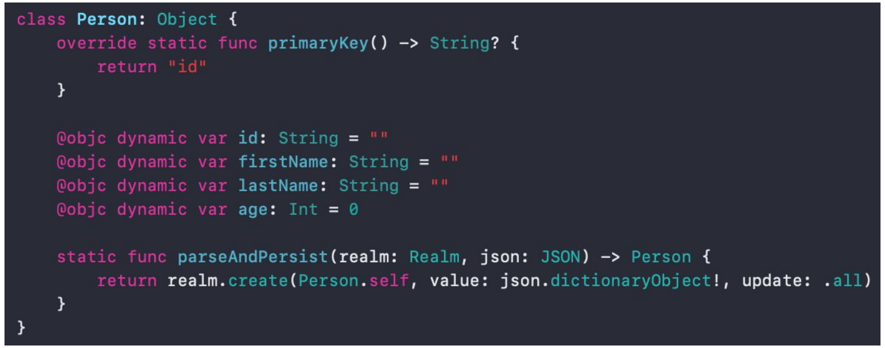
```swift
class Person: Object {
    override static func primaryKey() -> String? {
        return "id"
    }

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

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

    func persist(realm: Realm) -> Person {
        return realm.create(Person.self, value: self, update: .all)
    }
}
```

realm model

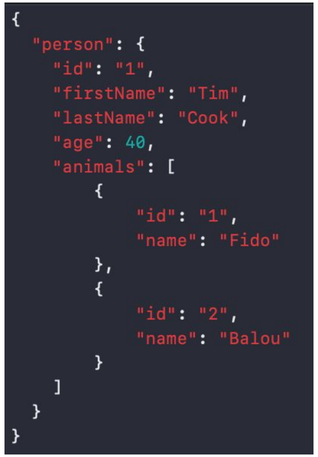
Realm - Parse and persist

Different way of parsing and persisting

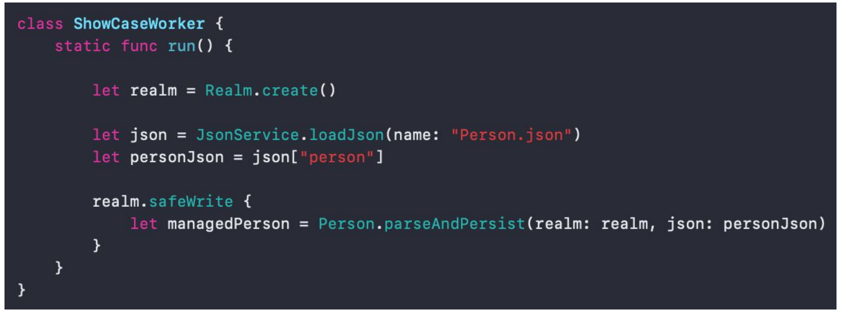
Realm



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

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

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



```json
{
  "person": {
    "id": "1",
    "firstName": "Tim",
    "lastName": "Cook",
    "age": 40,
    "animals": [
        {
            "id": "1",
            "name": "Fido"
        },
        {
            "id": "2",
            "name": "Balou"
        }
    ]
  }
}
```



```swift
class ShowCaseWorker {
    static func run() {

        let realm = Realm.create()

        let json = JsonService.loadJson(name: "Person.json")
        let personJson = json["person"]

        realm.safeWrite {
            let managedPerson = Person.parseAndPersist(realm: realm, json: personJson)
        }
    }
}
```

Person.json

realm controller

realm model

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 - Nice to know

Supporting Android (Java/Kotlin), .NET, JS.

Encryption

Realm sync

Realm notifications (Change listeners)

Questions?

Exercises & Coffee break!

Exercises 1/2

1. Download demo project and run application on simulator
2. Realm (Use Realm-tag) *‘git checkout Realm’*
   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 (one-to-many relationship)

Demo project: https://gitlab.com/overallapps/demo-ios.git

Network - Presetup

Network manager

- Alamofire SessionManager
- Request builder

RxSwift

Intro

- RxSwift
- Simple printer
- Schedulers
- RxAlamofire

Rx Documentation: http://reactivex.io/

RxSwift

What is Rx?

Multi-platform standard (Web, Android & iOS)

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

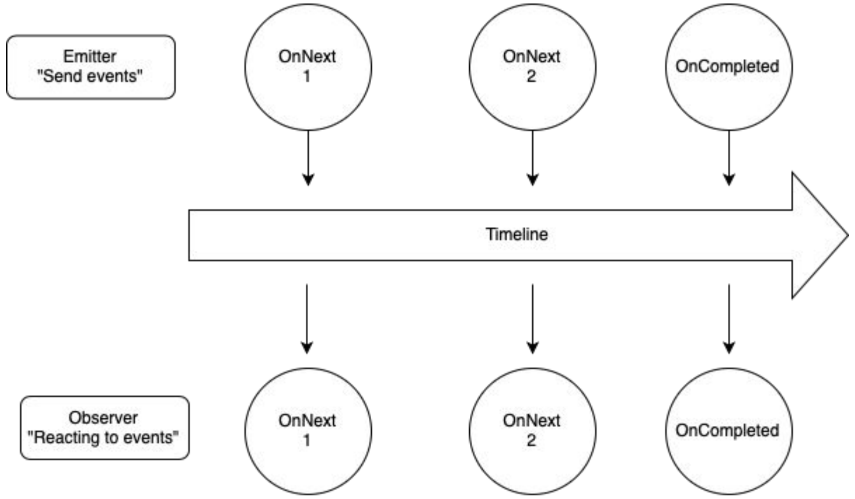
Handle concurrent tasks (Like user-input & network calls etc)

RxSwift

What is Rx?

A sequence of events in a timeline (observer lifetime), which you subscribe on

- Can emit zero or more events
- Emits three types of events:
  a. onNext()
  b. onError()
  c. onCompleted()
- React to events
  a. subscribeOn()
  b. doOn()
- Printer demo



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

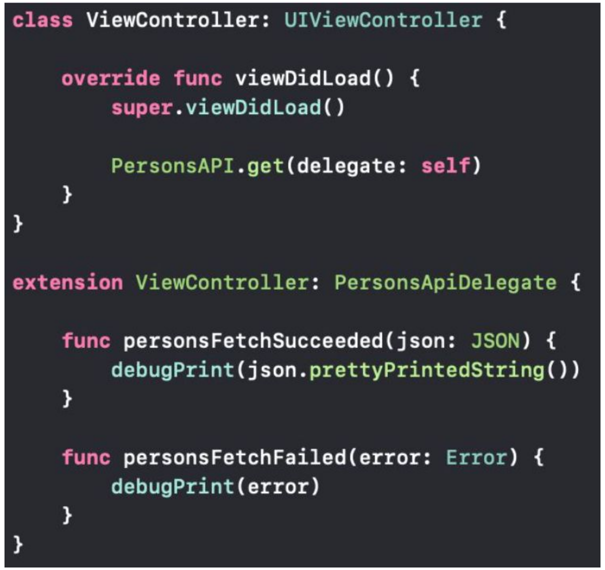
- Cancellation of observables
- Automatic cancellation of observables on deinit of bag

Data manipulation

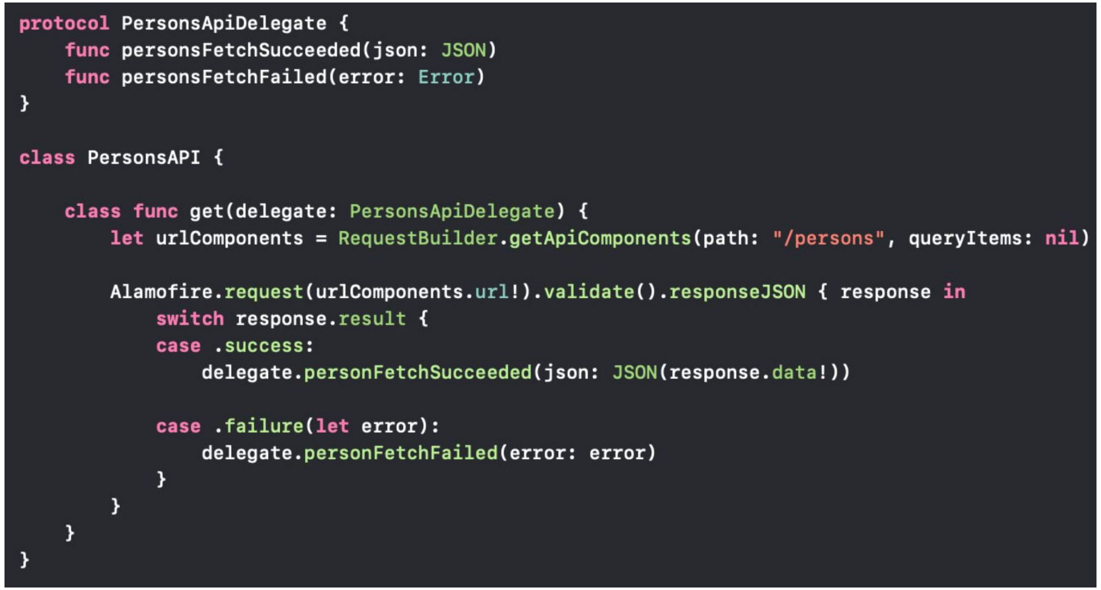
- Transform (Map)
- Filter

RxSwift - Observeable vs. Delegates

Implementation with delegates



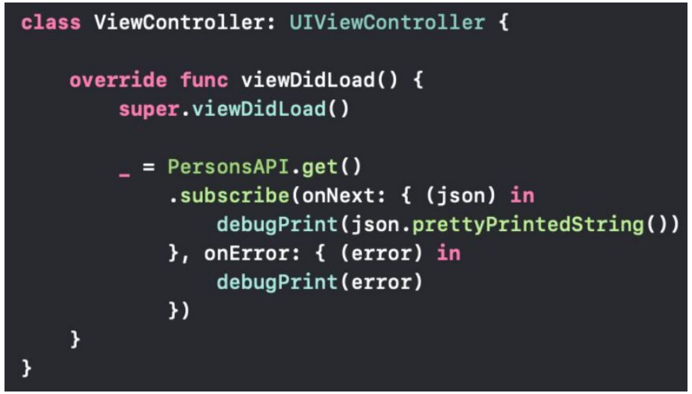
```swift
class ViewController: UIViewController {

    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)
    }
}
```



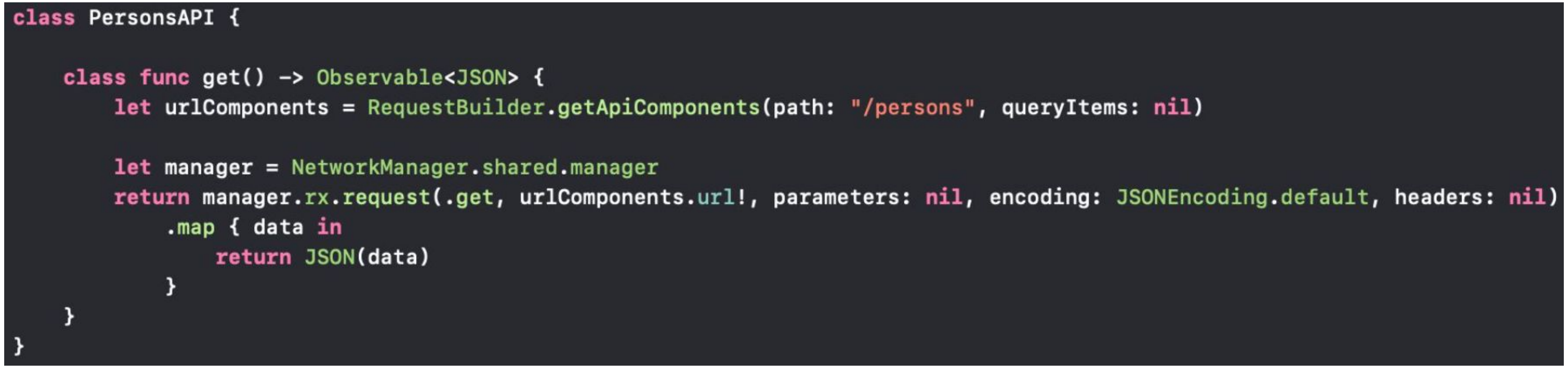
```swift
protocol PersonsApiDelegate {
    func personsFetchSucceeded(json: JSON)
    func personsFetchFailed(error: Error)
}

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 - Observeable vs. Delegates

Implementation with observables



```swift
class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

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



```swift
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’ to your Podfile
2. Add ‘pod RxAlamofire’ to your Podfile
3. Run ‘pod install’ in your terminal

Via Carthage:

1. Add ‘github "ReactiveCocoa/ReactiveSwift"’ in your Cartfile
2. Add ‘github "RxSwiftCommunity/RxAlamofire"’ in your Cartfile
3. Run ‘carthage update --platform iOS --cache-builds’ in your terminal

Via SPM:

1. Add ‘https://github.com/ReactiveX/RxSwift.git’ in your Swift Packages under project settings
2. Add https://github.com/RxSwiftCommunity/RxAlamofire.git’ in your Swift Packages under project settings

RxSwift - Usage

1. Implementation of person API (RxAlamofire)
2. Use person API, and save to Realm

Examples of Rx libraries:

1. RxAlamofire
2. RxRealm
3. RxRouter (RxFlow)
4. RxDataSource
5. RxBinding

See more at https://community.rxswift.org/

RxSwift - Nice to know

Life cycle (DisposeBag)

Threading

Retry-handling

Error-handling

Debounce (UI input etc.)

Merge, Concat & Zip (https://rxmarbles.com/#merge)

Types of observable: Completable, Single etc.

Map / FlatMap

Swift Combine Framework

RxSwift - Exercises preparation

1. Generate api key: `curl --header "Content-Length: 0" -X POST https://demo.overallapps.com/apikey`
2. GET https://demo.overallapps.com/persons

Swagger documentation on server: https://demo.overallapps.com/docs

Exercises 2/2

1. Download demo project and run application on simulator
2. Realm (Use Realm-tag) *'git checkout Realm'*
    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 (one-to-many relationship)
3. RxSwift (Use Rx-tag)
    a. Extend the PersonsAPI to include a PUT, and change a specific person name on the server
    b. Extend the PersonsAPI to include a POST, and add a new person on the server
    c. Create AnimalsAPI which can add an animal on the server
    d. Extend the PersonsAPI to add a relationship between a person and an animal
    e. Present alert with error message, when receiving a 404 error from the server.
    Trigger 404 by inserting '$ 'in the path in personsAPI GET method. Trigger 401 by deleting api key.
4. Realm + RxSwift
    a. Fetch animals from the server, and save it into the database.
    b. Voluntary: Visualise animals in the application through 'realm.objects(Animals.self).observe'

Demo project: https://gitlab.com/overallapps/demo-ios.git
Swagger documentation: https://demo.overallapps.com/docs