

Beta 2.0

# Full Stack iOS Entwicklung mit Swift

WPF im MIM - WS 17/18  
Alexander Dobrynin, M.Sc.

# Heute

3rd-Party Libraries  
“Märkte” für 3rd-Party Libraries  
Beispiele  
Tools für iOS (insb. Swift)

Demo  
Assignment

# 3rd-Party Libraries

the good

- Es existieren bereits viele fertige Lösungen für (wiederkehrende) Probleme
- Teilweise sind die Lösungen sogar elegant und effizient implementiert. Manchmal sogar in einem Maße, wie man es selber niemals schaffen würde/möchte
- Aus ökologischer Sicht kann man sich als Entwickler den “wichtigeren” Problemen widmen, die für die App einzigartig sind. Man muss nicht für “jede Kleinigkeit” das Rad neu erfinden
- Einige 3rd-Party Libraries verdienen ihr Geld damit, indem sie das beste und ausgereifteste Framework für einen bestimmten Anwendungsfall sind. Gerade hier sind Kosten/Nutzen und Vertrauen sehr attraktiv
- Als Entwickler kann man von anderen Entwicklern viel lernen, weil die Implementierungen offen sind
- Zudem kann man zur Weiterentwicklung eines geliebten Projektes beitragen, indem Pull-Requests eingereicht werden. Das Partizipieren an einem Open-Source Projekt macht einen selbst zu einem besseren Softwareentwickler
- Die Verwendung guter APIs lehrt einem was eine gute API ausmacht, wodurch man selbst anfängt gute APIs zu designen
- 3rd-Party Libraries sind oftmals die dominierende Antwort für eine bestimmte Frage auf Stackoverflow, was allerdings auch seine Gefahren mit sich bringt

# 3rd-Party Libraries

the bad

- Tendenz, dass jedes Problem, ob klein oder groß, mit einer noch größeren Library erschlagen wird
- Verständnis der darunterliegenden Konzepte geht verloren. Gleichzeitig geht das Verständnis der iOS Platform als solches immer weiter verloren
- Grad an Anpassbarkeit ist von der 3rd-Party Library abhängig. Falls eine Anpassbarkeit möglich ist, muss man sich meistens in ungewohnte Strukturen, neue Coding Styles und im Zweifel sogar in neue Paradigmen einarbeiten. Die Zeit kann man auch zum Einarbeiten in die System-API nutzen, um auf die Abhängigkeit zu verzichten
- Jede 3rd-Party Library ist eine Abhängigkeit zu einem Dritten, die man sich ins Projekt holt. Zudem entsteht ein implizites Vertrauen, dass der Code aktualisiert und weiterentwickelt wird, denn es wird ~jedes Jahr eine neue iOS und eine neue Swift Version veröffentlicht
  - Die Abhängigkeit im Projekt kann je nach Distribution der App rechtliche Implikationen haben (Software Lizenzen)
  - Rule of Thumb: wenn die 3rd-Party Library von heute auf morgen verschwindet, kann ich die Abhängigkeit alleine warten?
- Das App-Bundle wird größer. Die App wird komplexer und unübersichtlicher

# Choose an open source license

{ Which of the following best describes your situation? }



**I want it simple and permissive.**

The **MIT License** is a permissive license that is short and to the point. It lets people do anything they want with your code as long as they provide attribution back to you and don't hold you liable.

**Babel**, **.NET Core**, and **Rails** use the MIT License.



**I'm concerned about patents.**

The **Apache License 2.0** is a permissive license similar to the MIT License, but also provides an express grant of patent rights from contributors to users.

**Elasticsearch**, **Kubernetes**, and **Swift** use the Apache License 2.0.



**I care about sharing improvements.**

The **GNU GPLv3** is a copyleft license that requires anyone who distributes your code or a derivative work to make the source available under the same terms, and also provides an express grant of patent rights from contributors to users.

**Ansible**, **Bash**, and **GIMP** use the GNU GPLv3.

{ What if none of these work for me? }

**My project isn't software.**

There are licenses for that.

**I want more choices.**

More licenses are available.

**I don't want to choose a license.**

You don't have to.

# Heute

3rd-Party Libraries

“Märkte” für 3rd-Party Libraries

Beispiele

Tools für iOS (insb. Swift)











Demo

Assignment

# “Märkte” für 3rd-Party Libraries

- GitHub ist der Markt für nahezu jede bekannte oder unbekannte 3rd-Party Library
- Häufig gelangt man über Umwege auf die entsprechenden Repositories, indem man über eine Suche auf Google zunächst auf Stackoverflow und anschließend auf GitHub gelangt
- Für eine gezieltere Suche ist das GitHub Repository “[Awesome iOS](#)” eine gute Anlaufstelle
  - aktuell (24.01.18) werden dort 677 Swift und 683 Obj-C Libraries und Frameworks gelistet
  - diese sind zudem in Kategorien wie Analytics, Cache, Charts, CoreData, Database, Layout, Networking, Parsing, Security, UI, etc. unterteilt
  - außerdem verfügt die Seite über weitere Informationen rund um “awesome iOS” wie Getting Started, Dependency / Package Manager, Tools, Deployment / Distribution, Xcode, Good Websites, Podcast, Books, etc.
- Indikator für die Qualität einer 3rd-Party Library sind
  - Stars, Forks und die Qualität der Implementierung
  - Anzahl der Contributors und Issues
  - Frequenz von Pull-Requests, Merges, Releases und Commits
  - Getting Stared in der README.md, API-Dokumentation im Wiki, Demo, Sample Code und Tests

## Database

- [Realm](#) - The alternative to CoreData and SQLite: Simple, modern and fast.
- [YapDatabase](#) - YapDatabase is an extensible database for iOS & Mac.
- [Couchbase Mobile](#) - Couchbase document store for mobile with cloud sync.
- [FMDB](#) - A Cocoa / Objective-C wrapper around SQLite.
- [FCModel](#) - An alternative to Core Data for people who like having direct SQL access.
- [Zephyr](#) - Effortlessly synchronize NSUserDefaults over iCloud. 
- [Prephirences](#) - Prephirences is a Swift library that provides useful protocols and convenience methods to manage application preferences, configurations and app-state. 
- [Storez](#) - Safe, statically-typed, store-agnostic key-value storage (with namespace support). 
- [SwiftUserDefaults](#) - Statically-typed NSUserDefaults. 
- [SugarRecord](#) - Data persistence management library written in Swift 2.0 
- [SQLite.swift](#) - A type-safe, Swift-language layer over SQLite3. 
- [GRDB.swift](#) - A versatile SQLite toolkit for Swift, with WAL mode support 
- [Fluent](#) - Simple ActiveRecord implementation for working with your database in Swift. 
- [ParseAlternatives](#) - A collaborative list of Parse alternative backend service providers.
- [TypedDefaults](#) - TypedDefaults is a utility library to type-safely use NSUserDefaults. 
- [realm-cocoa-converter](#) - A library that provides the ability to import/export Realm files from a variety of data container formats. 
- [YapDatabaseExtensions](#) - YapDatabase extensions for use with Swift 
- [RealmGeoQueries](#) - RealmGeoQueries simplifies spatial queries with Realm Cocoa. In the absence of and official functions, this library provide the possibility to do proximity search. [e]
- [SwiftMongoDB](#) - A MongoDB interface for Swift 
- [ObjectiveRocks](#) - An Objective-C wrapper of Facebook's RocksDB - A Persistent Key-Value Store for Flash and RAM Storage.
- [OHMySQL](#) - An Objective-C wrapper of MySQL C API.
- [SwiftStore](#) - Key-Value store for Swift backed by LevelDB 
- [OneStore](#) - A single value proxy for NSUserDefaults, with clean API. 
- [MongoDB](#) - A Swift wrapper around the mongo-c client library, enabling access to MongoDB servers. Part of the [Perfect](#) project, but stand-alone. SPM and Swift 3 support.
- [SQLite](#) - A Swift wrapper around the SQLite 3 client library, enabling access to SQLite servers. Part of the [Perfect](#) project, but stand-alone. SPM and Swift 3 support.
- [MySQL](#) - A Swift wrapper around the MySQL client library, enabling access to MySQL servers. Part of the [Perfect](#) project, but stand-alone. SPM and Swift 3 support.
- [Redis](#) - A Swift wrapper around the Redis client library, enabling access to Redis. Part of the [Perfect](#) project, but stand-alone. SPM and Swift 3 support.
- [PostgreSQL](#) - A Swift wrapper around the libpq client library, enabling access to PostgreSQL servers. Part of the [Perfect](#) project, but stand-alone. SPM and Swift 3 support.
- [FileMaker](#) - A Swift wrapper around the FileMaker XML Web publishing interface, enabling access to FileMaker servers. Part of the [Perfect](#) project, but stand-alone. SPM and Swift 3 support.
- [Nora](#) - Nora is a Firebase abstraction layer for working with FirebaseDatabase and FirebaseStorage. 
- [PersistentStorageSerializable](#) - Swift library that makes easier to serialize the user's preferences (app's settings) with system User Defaults or Property List file on disk. 
- [WCDB](#) - WCDB is an efficient, complete, easy-to-use mobile database framework for iOS, macOS.
- [StorageKit](#) - Your Data Storage Troubleshooter 
- [UserDefaults](#) - Simple, Strongly-Typed UserDefaults for iOS, macOS and tvOS 



[Code](#)
[Issues 31](#)
[Pull requests 15](#)
[Projects 1](#)
[Insights](#)













## Elegant HTTP Networking in Swift

[networking](#)
[urlsession](#)
[urlrequest](#)
[httpurlresponse](#)
[request](#)
[response](#)
[swift](#)
[xcode](#)
[certificate-pinning](#)
[public-key-pinning](#)
[parameter-encoding](#)
[alamofire](#)
[cocoapods](#)
[carthage](#)
[swift-package-manager](#)

[1,227 commits](#)
[12 branches](#)
[52 releases](#)
[150 contributors](#)
[MIT](#)

[Branch: master](#)
[New pull request](#)
[Create new file](#)
[Upload files](#)
[Find file](#)
[Clone or download](#)

 **cnoon** committed with **jshier** Added radar to the README for the URLSessionTaskMetrics issue on watc... [...](#) Latest commit 098a420 on Dec 21, 2017

|   |  |              |
|---|--|--------------|
|  <a href="#">Alamofire.xcodeproj</a>    | 4.6.0 Release. (#2374)   | 2 months ago |
|  <a href="#">Alamofire.xcworkspace</a> | Remove WorkspaceSettings.  | 7 months ago |
|  <a href="#">Documentation</a>         | Fix #session-manager link (#2373)                                      | 2 months ago |
|  <a href="#">Example</a>               | 4.6.0 Release. (#2374)   | 2 months ago |
|  <a href="#">Source</a>                | 4.6.0 Release. (#2374)   | 2 months ago |
|  <a href="#">Tests</a>                 | Add Error mapping functions to Response types (#2361)                  | 2 months ago |
|  <a href="#">docs</a>                  | 4.6.0 Release. (#2374)   | 2 months ago |
|  <a href="#">.gitignore</a>            | Updated the .gitignore file with the latest ignore flags for Swift.    | 2 years ago  |
|  <a href="#">.jazzy.yaml</a>           | Fix dash download (#2258x2)  | 5 months ago |
|  <a href="#">.ruby-gemset</a>          | [PR #2250] Added Jazzy docs for the release to work with GitHub Pages. | 5 months ago |
|  <a href="#">.ruby-version</a>         | Update Travis Usage (#2302)  | 4 months ago |
|  <a href="#">.swift-version</a>        | Bumped the swift-version file to Swift 3.2.                            | 5 months ago |

# ALAMOFIRE

## *Elegant Networking in Swift*


build [passing](#) pod [v4.6.0](#) Carthage [compatible](#) platform [ios](#) | [osx](#) | [tvos](#) | [watchos](#) twitter [@AlamofireSF](#) chat [on gitter](#)

Alamofire is an HTTP networking library written in Swift.

- [Features](#)
- [Component Libraries](#)
- [Requirements](#)
- [Migration Guides](#)
- [Communication](#)
- [Installation](#)
- [Usage](#)
  - [Intro](#) - [Making a Request](#), [Response Handling](#), [Response Validation](#), [Response Caching](#)
  - [HTTP](#) - [HTTP Methods](#), [Parameter Encoding](#), [HTTP Headers](#), [Authentication](#)
  - [Large Data](#) - [Downloading Data to a File](#), [Uploading Data to a Server](#)
  - [Tools](#) - [Statistical Metrics](#), [cURL Command Output](#)
- [Advanced Usage](#)
  - [URL Session](#) - [Session Manager](#), [Session Delegate](#), [Request](#)
  - [Routing](#) - [Routing Requests](#), [Adapting and Retrying Requests](#)
  - [Model Objects](#) - [Custom Response Serialization](#)
  - [Connection](#) - [Security](#), [Network Reachability](#)
- [Open Radars](#)
- [FAQ](#)
- [Credits](#)
- [Donations](#)
- [License](#)

## Features

- ✓ [Chainable Request / Response Methods](#)

|   |   |                                      |
|---|---|--------------------------------------|
|  naeemshaikh90 committed with jshier Fix #session-manager link (#2373) |   | Latest commit 1478094 on Dec 4, 2017 |
| ..  |   |                                      |
| <a href="#">AdvancedUsage.md</a>  | Refactor README (#2360)   | 2 months ago                         |
| <a href="#">Alamofire 2.0 Migration Guide.md</a>  | Updated the migration guide with the parameter encoding updates.  | 2 years ago                          |
| <a href="#">Alamofire 3.0 Migration Guide.md</a>  | Update Alamofire 3.0 Migration Guide.md                           | 2 years ago                          |
| <a href="#">Alamofire 4.0 Migration Guide.md</a>  | [PR #2212] Fixed update requirements in the AF 4 migration guide. | 5 months ago                         |
| <a href="#">Usage.md</a>  | Fix #session-manager link (#2373)                                 | 2 months ago                         |



naeemshaikh90 Fix #session-manager link (#2373)

1478094 on Dec 4, 2017

2 contributors

690 lines (501 sloc) 27.3 KB

Raw

Blame

History



## Usage

### Making a Request

```
import Alamofire

Alamofire.request("https://httpbin.org/get")
```

### Response Handling

Handling the `Response` of a `Request` made in Alamofire involves chaining a response handler onto the `Request`.

```
Alamofire.request("https://httpbin.org/get").responseJSON { response in
    print("Request: \(String(describing: response.request))") // original url request
    print("Response: \(String(describing: response.response))") // http url response
    print("Result: \(response.result)") // response serialization result


    if let json = response.result.value {
        print("JSON result: \(json)") // original json response
    }
}
```

```

28 class MasterViewController: UITableViewController {
29
30     @IBOutlet weak var titleImageView: UIImageView!
31
32     var detailViewController: DetailViewController? = nil
33     var objects = NSMutableArray()
34
35     // MARK: - View Lifecycle
36
37     override func awakeFromNib() {
38         super.awakeFromNib()
39
40         navigationItem.titleView = titleImageView
41     }
42
43     override func viewDidLoad() {
44         super.viewDidLoad()
45
46         if let split = splitViewController {
47             let controllers = split.viewControllers
48
49             if
50                 let navigationController = controllers.last as? UINavigationController,
51                 let topViewController = navigationController.topViewController as? DetailViewController
52             {
53                 detailViewController = topViewController
54             }
55         }
56     }
57
58     // MARK: - UIStoryboardSegue
59
60     override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
61         if
62             let navigationController = segue.destination as? UINavigationController,
63             let detailViewController = navigationController.topViewController as? DetailViewController
64         {
65             func requestForSegue(_ segue: UIStoryboardSegue) -> Request? {
66                 switch segue.identifier! {
67                     case "GET":
68                         detailViewController.segueIdentifier = "GET"
69                         return Alamofire.request("https://httpbin.org/get")
70                     case "POST":
71                         detailViewController.segueIdentifier = "POST"
72                         return Alamofire.request("https://httpbin.org/post", method: .post)
73                     case "PUT":
74                         detailViewController.segueIdentifier = "PUT"
75                         return Alamofire.request("https://httpbin.org/put", method: .put)
76                     case "DELETE":
77                         detailViewController.segueIdentifier = "DELETE"
78                         return Alamofire.request("https://httpbin.org/delete", method: .delete)
79                     case "DOWNLOAD":
80                         detailViewController.segueIdentifier = "DOWNLOAD"
81                         let destination = DownloadRequest.suggestedDownloadDestination(
82                             for: .cachesDirectory,
83                             in: .userDomainMask
84                         )
85                         return Alamofire.download("https://httpbin.org/stream/1", to: destination)
86                     default:
87                         return nil
88                 }

```



|   |  |                                       |
|---|--|---------------------------------------|
|  <b>jshier</b> Add Error mapping functions to Response types (#2361) ... |  | Latest commit 0a34b44 on Nov 27, 2017 |
| ..  |  |                                       |
| Resources   | Added download response serialization tests for all serializer types.    | a year ago                            |
| AFError+AlamofireTests.swift  | Fixed the copyright in all the source files.                             | 5 months ago                          |
| AuthenticationTests.swift   | Fixed the copyright in all the source files.                             | 5 months ago                          |
| BaseTestCase.swift  | Fixed the copyright in all the source files.                             | 5 months ago                          |
| CacheTests.swift  | Fixed the copyright in all the source files.                             | 5 months ago                          |
| DownloadTests.swift   | Add Error mapping functions to Response types (#2361)                    | 2 months ago                          |
| FileManager+AlamofireTests.swift  | Fixed the copyright in all the source files.                             | 5 months ago                          |
| Info.plist  | Refactored frameworks, tests and example app to compile against Swift... | 3 years ago                           |
| MultipartFormDataTests.swift  | Fixed the copyright in all the source files.                             | 5 months ago                          |
| NetworkReachabilityManagerTest...   | Fixed the copyright in all the source files.                             | 5 months ago                          |
| ParameterEncodingTests.swift  | Fixed the copyright in all the source files.                             | 5 months ago                          |
| RequestTests.swift  | Fixed the copyright in all the source files.                             | 5 months ago                          |
| ResponseSerializationTests.swift  | Fixed the copyright in all the source files.                             | 5 months ago                          |
| ResponseTests.swift   | Add Error mapping functions to Response types (#2361)                    | 2 months ago                          |
| ResultTests.swift   | Fixed the copyright in all the source files.                             | 5 months ago                          |
| ServerTrustPolicyTests.swift  | Fixed the copyright in all the source files.                             | 5 months ago                          |
| SessionDelegateTests.swift  | Fixed the copyright in all the source files.                             | 5 months ago                          |

# Heute

3rd-Party Libraries

“Märkte” für 3rd-Party Libraries

Beispiele

Tools für iOS (insb. Swift)

Demo

Assignment

# Awesome Cache

build **passing** pod **v5.0** Carthage **compatible** Swift **3.0** platform iOS | watchOS | tvOS

Delightful on-disk cache (written in Swift). Backed by NSCache for maximum performance and support for expiry of single objects.

## Usage

```
do {
    let cache = try Cache<NSString>(name: "awesomeCache")

    cache["name"] = "Alex"
    let name = cache["name"]
    cache["name"] = nil
} catch _ {
    print("Something went wrong :(")
}
```

## Sync by design

AwesomeCache >= 3.0 is designed to have a sync API, making it easy to reason about the actual contents of the cache. This decision has been made based on [feedback from the community](#), to keep the API of AwesomeCache small and easy to use.

The internals of the cache use a concurrent dispatch queue, that syncs reads and writes for thread safety. In case a particular caching operation blocks your main thread for too long, consider offloading the read and write operations to a different thread.

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0)) {
    cache["name"] = "Alex"
}
```

## Cache expiry

Objects can also be cached for a certain period of time.

```
cache.setObject("Alex", forKey: "name", expires: .never) // same as cache["name"] = "Alex"
cache.setObject("Alex", forKey: "name", expires: .seconds(2)) // name expires in 2 seconds
cache.setObject("Alex", forKey: "name", expires: .date(Date(timeIntervalSince1970: 1428364800))) // name expires at 2015-02-03 12:00:00
```

If an object is accessed after its expiry date, it is automatically removed from the cache and deleted from disk. However, you are responsible to delete expired objects regularly by calling `removeExpiredObjects` (e.g. on app launch).

## Awesome API Caching

API responses are usually cached for a specific period of time. AwesomeCache provides an easy method to cache a block of asynchronous tasks.

```
cache.setObject(forKey: "name", cacheBlock: { success, failure in
    // Perform tasks, e.g. call an API
})
```



- 🕒 **Correct:** Using Foundation API correctly, Timepiece helps to calculate dates correctly without deep understanding.
- 📦 **Small:** Timepiece has only 4 file. You can read the inside of this easily.

## Requirements

- Swift 3.0+
- iOS 8.0+ / macOS 10.9+ / tvOS 9.0+ / watchOS 2.0+

## Usage

### Initialization

```
Date(era: 235, year: 26, month: 8, day: 14, hour: 20, minute: 25, second: 43, nanosecond: 0, on: Calendar(identif:
Date(year: 2014, month: 8, day: 14, hour: 20, minute: 25, second: 43, nanosecond: 0)
Date(year: 2014, month: 8, day: 14, hour: 20, minute: 25, second: 43)
Date(year: 2014, month: 8, day: 14)

Date.today()
Date.yesterday()
Date.tomorrow()
```

### Calculation

```
now + 1.year
now - 2.months
now + (3.weeks - 4.days + 5.hours)

1.year.later
1.year.ago
```

### Change

```
now.changed(year: 2014)
now.changed(weekday: 1)
now.truncated([.minute, .second, .nanosecond])
now.truncated(from: .day)
```

### Formating

```
now.stringIn(dateStyle: .long, timeStyle: .medium)
now.dateString(in: .medium)
now.timeString(in: .short)

3.days.string(in: .full)
```

### Parsing

```
"2014/8/14".date(inFormat: "yyyy/MM/dd")
"2014-08-14T20:25:43+0900".dateInISO8601Format()
```

## Usage

```
import SQLite

let db = try Connection("path/to/db.sqlite3")

let users = Table("users")
let id = Expression<Int64>("id")
let name = Expression<String?>("name")
let email = Expression<String>("email")

try db.run(users.create { t in
    t.column(id, primaryKey: true)
    t.column(name)
    t.column(email, unique: true)
})
// CREATE TABLE "users" (
//     "id" INTEGER PRIMARY KEY NOT NULL,
//     "name" TEXT,
//     "email" TEXT NOT NULL UNIQUE
// )

let insert = users.insert(name <- "Alice", email <- "alice@mac.com")
let rowid = try db.run(insert)
// INSERT INTO "users" ("name", "email") VALUES ('Alice', 'alice@mac.com')

for user in try db.prepare(users) {
    print("id: \(user[id]), name: \(user[name]), email: \(user[email])")
    // id: 1, name: Optional("Alice"), email: alice@mac.com
}
// SELECT * FROM "users"

let alice = users.filter(id == rowid)

try db.run(alice.update(email <- email.replace("mac.com", with: "me.com")))
// UPDATE "users" SET "email" = replace("email", 'mac.com', 'me.com')
// WHERE ("id" = 1)

try db.run(alice.delete())
// DELETE FROM "users" WHERE ("id" = 1)

try db.scalar(users.count) // 0
// SELECT count(*) FROM "users"
```

SQLite.swift also works as a lightweight, Swift-friendly wrapper over the C API.

```
let stmt = try db.prepare("INSERT INTO users (email) VALUES (?)")
for email in ["betty@icloud.com", "cathy@icloud.com"] {
    try stmt.run(email)
}

db.totalChanges // 3
db.changes      // 1
db.lastInsertRowid // 3

for row in try db.prepare("SELECT id, email FROM users") {
    print("id: \(row[0]), email: \(row[1])")
    // id: Optional(2), email: Optional("betty@icloud.com")
    // id: Optional(3), email: Optional("cathy@icloud.com")
}
```

# CSStickyHeaderFlowLayout

## Contributors

For anyone who'd like to be a contributor to the repository, please read the [Contribution Guideline](#)



# Heute

3rd-Party Libraries

“Märkte” für 3rd-Party Libraries

Beispiele

Tools für iOS (insb. Swift)

Demo

Assignment

# Tools für iOS (insb. Swift)

- Je nach Support seitens der 3rd-Party Library existieren maximal 4 Möglichkeiten, um diese in sein eigenes Xcode-Projekt einzubinden
  - Manuell, indem die 3rd-Party Library in Xcode geöffnet und das jeweilige Target gebaut wird. Das daraus resultierende .framework wird in das eigene Projekt kopiert und in den “Build Phases” gelinked. Ggf. müssen die Abhängigkeiten des Frameworks ebenfalls gelinked werden
  - Cocoapods, welches als Ruby Gem einmalig installiert wird. Im Root jedes Xcode-Projektes wird eine Podfile erzeugt, welche die 3rd-Party Libraries als Einzeiler enthält. Danach werden die Abhängigkeiten mit einem Befehl heruntergeladen. Cocoapods modifiziert das eigene Xcode-Projekt und erstellt ein Workspace, um alle Frameworks und Binaries korrekt zu linkern. Anschließend wird das Projekt über die .xcworkspace-Datei geöffnet
  - Carthage, welcher ein in Swift geschriebener Dependency-Manager ist und bswp. über Homebrew einmalig installiert wird. Ähnlich zu Cocoapods wird eine Cartfile erstellt, welche die 3rd-Party Libraries als Einzeiler enthält. Auch hier werden die Abhängigkeiten mit einem Befehl heruntergeladen. Allerdings muss man die Frameworks und Binaries selbst einbetten und linkern
  - Swift Package Manager, ein Tools welches seit Swift 3 mit dem Compiler ausgeliefert wird. Hier sind zunächst einige Installationen und Konfigurationen notwendig. Ansonsten wird auch hier eine Package.swift erstellt und mit den 3rd-Party Libraries gefüllt. Mit einem Befehl werden die Abhängigkeiten heruntergeladen

Scope von FSIOs

# Tools für iOS (insb. Swift)

- Die drei Dependency-Management Tools haben den Vorteil, dass alle verwendeten 3rd-Party Libraries mit einem Befehl aktualisiert werden können
- Zudem nehmen sie gegenüber dem manuellen Hinzufügen viel Arbeit ab und machen das Dependency-Management bequemer (und damit auch verlockender, Vorsicht!)
- Cocoapods ist das älteste, verbreitetste und einfachste Tools. Es nimmt jede Arbeit ab und hat nahezu keine Barrieren. Allerdings konfiguriert das Tool das eigene Xcode-Projekt, und zwar in einem Maße, das man nicht nachvollziehen kann
- Letzteres ist der größere Unterschied zu Carthage, welches sich damit selbst als leichtgewichtiges und flexibleres Tools bezeichnet:

*“CocoaPods is a long-standing dependency manager for Cocoa. So why was Carthage created?”*

*Firstly, CocoaPods (by default) automatically creates and updates an Xcode workspace for your application and all dependencies. Carthage builds framework binaries using xcodebuild, but leaves the responsibility of integrating them up to the user. CocoaPods’ approach is easier to use, while Carthage’s is flexible and unintrusive.”*

- Der Swift Package Manager ist gegenüber Cocoapods und Carthage relativ jung und nicht so weit verbreitet
- Nachdem die 3rd-Party Library mit dem eigenen Projekt gebaut werden kann, muss das jeweilige Framework an jener Stelle importiert (import Foo) werden, wo es benutzt wird
- Falls man in seinem Swift Projekt eine 3rd-Party Library verwendet die in Objective-C geschrieben ist, muss ein “Swift Bridging Header” erstellt und verlinkt werden. In der Datei wird die Header-Datei des jeweiligen Obj-C Frameworks importiert (z.B. #import <Foo/Foo.h>)
- Tipp: ein Commit (git) vor der Integration einer 3rd-Party Library ermöglicht einen vollständigen und schmerzfreien Rollback

## Installation

### CocoaPods

[CocoaPods](#) is a dependency manager for Cocoa projects. You can install it with the following command:

```
$ gem install cocoapods
```

CocoaPods 1.1+ is required to build Alamofire 4.0+.

To integrate Alamofire into your Xcode project using CocoaPods, specify it in your `Podfile` :

```
source 'https://github.com/CocoaPods/Specs.git'
platform :ios, '10.0'
use_frameworks!

target '<Your Target Name>' do
  pod 'Alamofire', '~> 4.5'
end
```

Then, run the following command:

```
$ pod install
```

### Carthage

[Carthage](#) is a decentralized dependency manager that builds your dependencies and provides you with binary frameworks.

You can install Carthage with [Homebrew](#) using the following command:

```
$ brew update
$ brew install carthage
```

To integrate Alamofire into your Xcode project using Carthage, specify it in your `Cartfile` :

```
github "Alamofire/Alamofire" ~> 4.5
```

Run `carthage update` to build the framework and drag the built `Alamofire.framework` into your Xcode project.

### Swift Package Manager

The [Swift Package Manager](#) is a tool for automating the distribution of Swift code and is integrated into the `swift` compiler. It is in early development, but Alamofire does support its use on supported platforms.

Once you have your Swift package set up, adding Alamofire as a dependency is as easy as adding it to the `dependencies` value of your `Package.swift` .

#### Swift 3

```
dependencies: [
  .Package(url: "https://github.com/Alamofire/Alamofire.git", majorVersion: 4)
```

## Swift 4

```
dependencies: [  
  .package(url: "https://github.com/Alamofire/Alamofire.git", from: "4.0.0")  
]
```

## Manually

If you prefer not to use any of the aforementioned dependency managers, you can integrate Alamofire into your project manually.

### Embedded Framework

- Open up Terminal, `cd` into your top-level project directory, and run the following command "if" your project is not initialized as a git repository:

```
$ git init
```

- Add Alamofire as a git `submodule` by running the following command:

```
$ git submodule add https://github.com/Alamofire/Alamofire.git
```

- Open the new `Alamofire` folder, and drag the `Alamofire.xcodeproj` into the Project Navigator of your application's Xcode project.

It should appear nested underneath your application's blue project icon. Whether it is above or below all the other Xcode groups does not matter.

- Select the `Alamofire.xcodeproj` in the Project Navigator and verify the deployment target matches that of your application target.
- Next, select your application project in the Project Navigator (blue project icon) to navigate to the target configuration window and select the application target under the "Targets" heading in the sidebar.
- In the tab bar at the top of that window, open the "General" panel.
- Click on the `+` button under the "Embedded Binaries" section.
- You will see two different `Alamofire.xcodeproj` folders each with two different versions of the `Alamofire.framework` nested inside a `Products` folder.

It does not matter which `Products` folder you choose from, but it does matter whether you choose the top or bottom `Alamofire.framework`.

- Select the top `Alamofire.framework` for iOS and the bottom one for OS X.

You can verify which one you selected by inspecting the build log for your project. The build target for `Alamofire` will be listed as either `Alamofire iOS`, `Alamofire macOS`, `Alamofire tvOS` or `Alamofire watchOS`.

- And that's it!

The `Alamofire.framework` is automagically added as a target dependency, linked framework and embedded framework in a copy files build phase which is all you need to build on the simulator and a device.



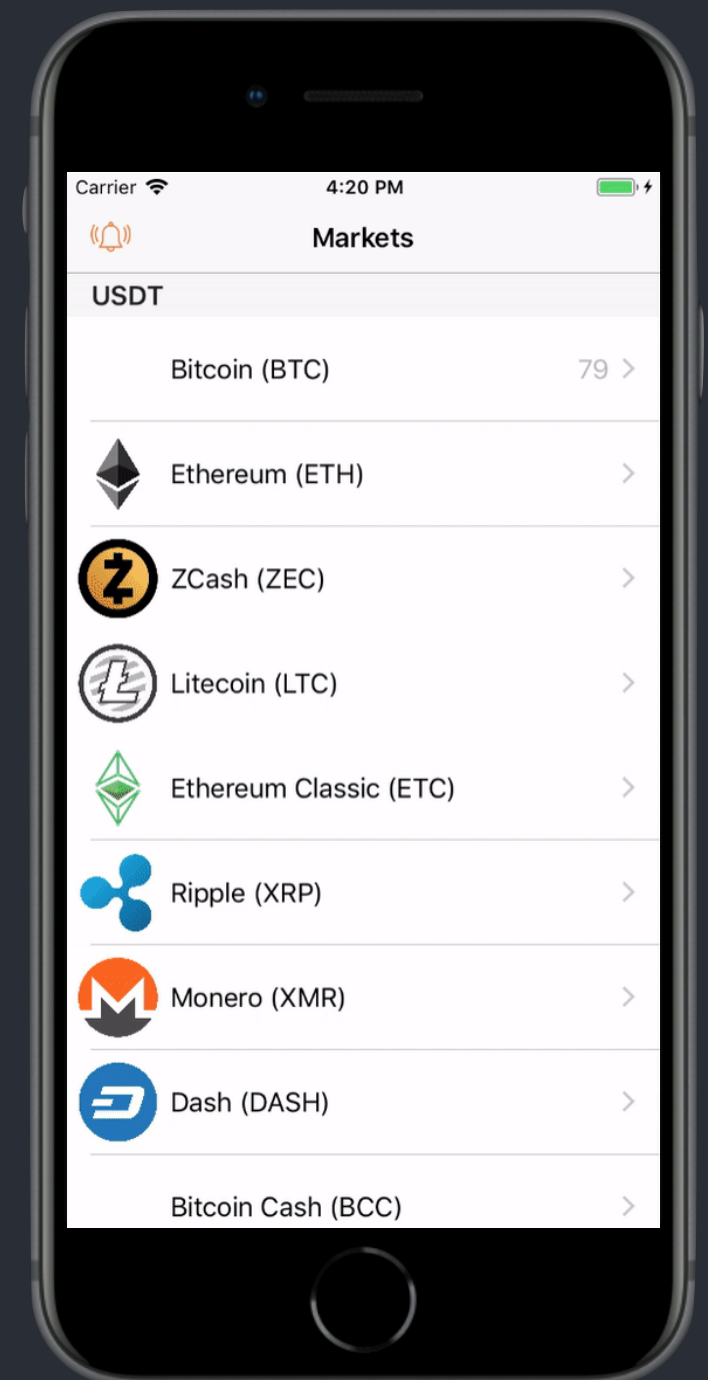
# Heute

3rd-Party Libraries  
“Märkte” für 3rd-Party Libraries  
Beispiele  
Tools für iOS (insb. Swift)

Demo  
Assignment

# Cryptomarket - Demo

- ContainerViewController
- Outlets und Action im Code definieren
- Refactoring im Storyboard
- Bestehende ViewController wiederverwenden
- Einbindung einer 3rd-Party Library mit Cocoapods
- Fremden Code (3rd-Party) lesen, verstehen, verwenden und modifizieren



# Heute

3rd-Party Libraries  
“Märkte” für 3rd-Party Libraries  
Beispiele  
Tools für iOS (insb. Swift)

Demo  
Assignment

# \$Project - Assignment

