

# Persistenz, Core Data, Tests, Frameworks

Programmieren für iOS



# Inhalt

- Persistenz (Fokus auf Core Data)
- Tests
- Inter-App Kommunikation
- Frameworks
  - User Notifications
  - Swift Charts

# Persistenz

# Persistenz

- Persistenz = “Daten über Programmlaufzeit erhalten”
- Grundsätzlich in einer Datei oder DB
  - Evt. via Web, z.B. Backend, iCloud etc. (würde für Vorlesung zu weit gehen)
- iOS-Mechanismen für lokale Persistenz:
  - Dateisystem (schon gesehen)
  - UserDefaults (schon gesehen)
  - Keychain (nur kurz)
  - SQLite & Core Data (neu)

# Dateisystem

- Zugriff auf's Dateisystem wird aus Sicherheits-Überlegungen stark limitiert
- iOS hat keinen Finder / File Explorer
- "Every app is an island" → Jede App hat ihre Sandbox und keinen Lese- / Schreibzugriff auf andere Orte (mit kontrollierten Ausnahmen, z.B. dokumentbasierte Apps)
- Innerhalb der Sandbox kann ziemlich beliebig mit Dateien gearbeitet werden:
  - Lesen / Schreiben
  - Verzeichnisse, Dateien erstellen, löschen etc.



# Dateisystem

- Bevorzugter Weg, um mit dem Filesystem zu interagieren: Klasse **FileManager** (mit Singleton-Instanz **FileManager.default**)
- URL für Documents Directory:

```
func getDocumentsDirectory() -> URL {  
    let paths = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask)  
    let documentsDirectory = paths[0]  
    return documentsDirectory  
}
```

- FileManager hat Methoden für viele gängige Operationen:
  - Verzeichnis erstellen / löschen
  - Prüfen, ob Datei existiert
  - Datei erstellen / schreiben / auslesen

# Dateisystem

- Für generische Objekte: **Codable**-Conformance → mit **JSONEncoder** & **JSONDecoder** encoden / decoden, speichern als JSON-Objekte
- Schreiben:

```
struct Object: Codable {  
    let value: String  
}  
  
let o = Object(value: "Hello")  
let data: Data = try! JSONEncoder().encode(o)  
let success = FileManager.default.createFile(atPath: path, contents: data)
```

- Lesen:

```
let data = FileManager.default.contents(atPath: path)!  
let o = try! JSONDecoder().decode(Object.self, from: data)
```

# UserDefaults

- System-Mechanismus zum Persistieren kleiner Datenmengen.  
Geeignet u.a. für:
  - Benutzereinstellungen
  - App-Zustand
  - App-Metainformationen (z.B. `isFirstAppStart` o.ä.)
- Achtung: Wird unverschlüsselt gespeichert, darum nie für sensitive Daten verwenden!
- Sehr einfach zu benutzende API, dafür Datentypen beschränkt
  - Wobei `Data` möglich → somit mittels `Codable` beliebige Datentypen



# UserDefaults – API

- Schreiben & lesen:

```
UserDefaults.standard.set(42, forKey: "intValue")
let intValue = UserDefaults.standard.integer(forKey: "intValue")

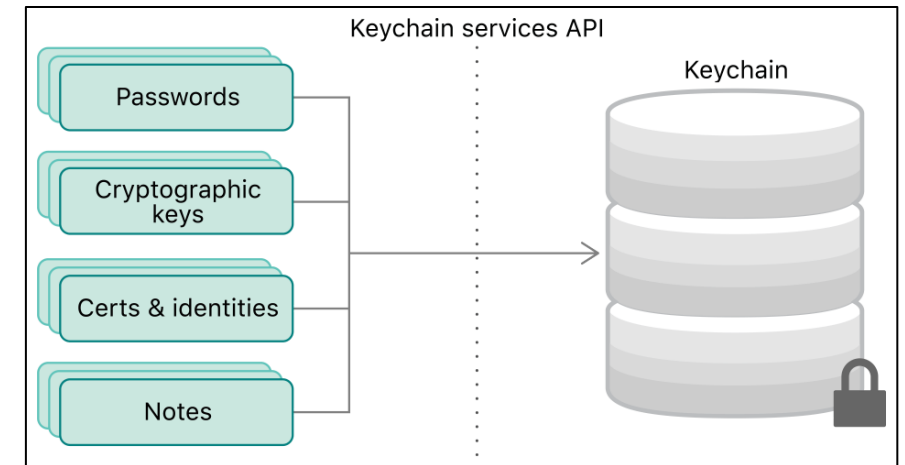
UserDefaults.standard.set(true, forKey: "boolValue")
let boolValue = UserDefaults.standard.bool(forKey: "boolValue")
```

- Für SwiftUI: @AppStore Property Wrapper

```
@AppStorage("editCount") var editCount: Int = 0
```

# Keychain

- Keychain: Kleine Mengen an Userdaten in verschlüsselter DB speichern
- V.a., aber nicht nur für Passwörter
- DB wird zwar (verschlüsselt) auf Filesystem gespeichert, Schlüssel dafür in der Secure Enclave
  - *Secure Enclave in a Nutshell: Dediziertes Hardware-Subsystem, das getrennt vom Main-Prozessor lebt. Dort gespeicherte Keys verlassen die Secure Enclave nie.*

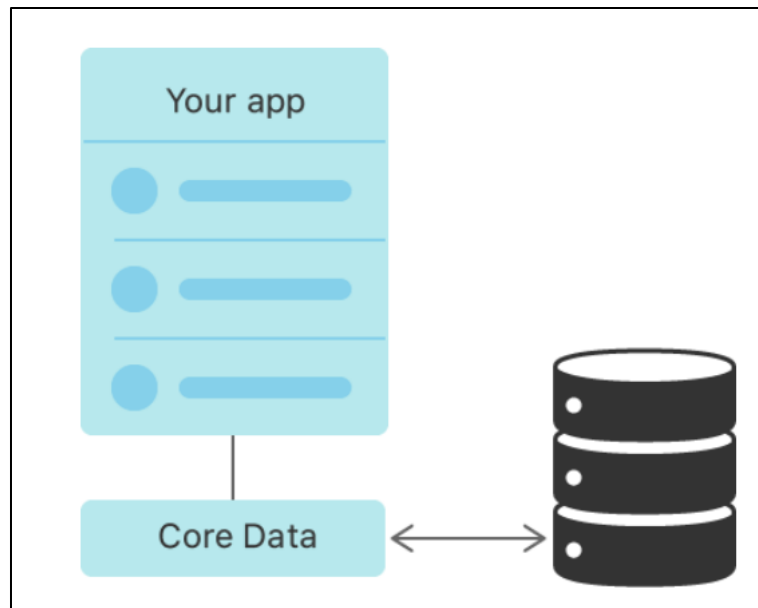


# SQLite

- SQLite: Schlanke SQL-Engine
  - <https://www.sqlite.org/>
  - Open Source
  - Läuft auf jedem Smartphone und fast allen Computern
- Auf iOS ebenfalls verfügbar
  - Gebraucht in diversen Standard-iOS-Apps
  - Wird von Core Data intern ebenfalls verwendet
- Direkt-Verwendung ist theoretisch möglich (C-Library), aber nicht üblich/empfohlen
  - Bevorzugt über Swift-Wrapper-Frameworks verwenden (z.B. Core Data, aber auch andere → GitHub), z.B. wegen Type Safety

# Core Data

- Offizielles Apple-Framework, um beliebige (auch grosse) Datenmengen zu persistieren, cachen und synchronisieren (via iCloud)
- Zusätzliche Abstraktionsebene über klassische Datenbank
  - Intern wird DB benutzt, aber als Framework-User interagiert man nicht direkt damit

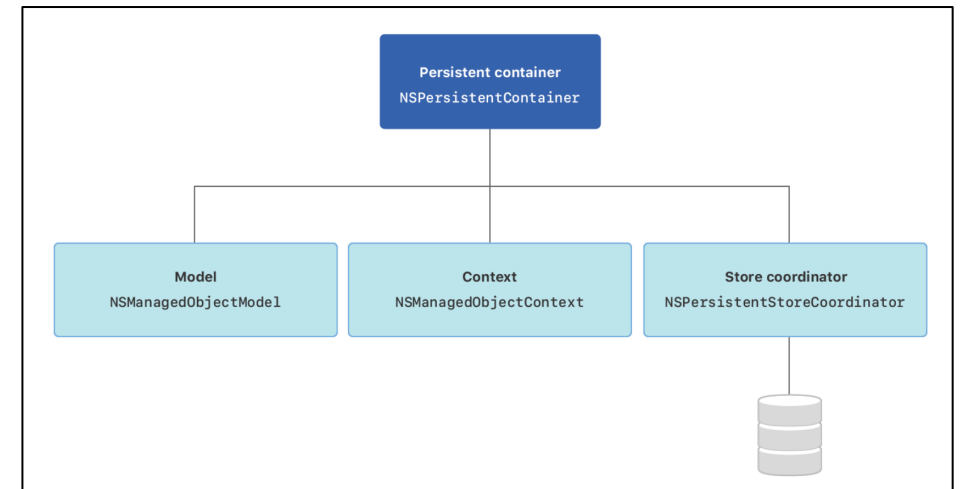


# Core Data

- Unterstützung für diverse Tasks:
  - Undo/Redo-Operationen
  - Batch Changes / Rollbacks
  - Synchronisation mit UI-Layer (Integrationen für diverse Views in UIKit oder SwiftUI), z.B.
    - `NSFetchedResultsController` (UIKit)
    - `@FetchRequest` (SwiftUI)
- Versionierung & Migration
  - Bei Änderung von Datenmodell: Einfache Migrationen werden wenn möglich automatisch gemacht
  - Komplexere Migrationen: Manuelle Unterstützung notwendig (Mapping-Model)

# Core Data – Aufbau

- **NSManagedObjectModel**: Repräsentation der .xcdatamodeld-Datenmodelle
- **NSManagedObjectContext**: Koordiniert Changes in den Models (primäre Interaktionen passieren auf dieser Klasse: insert, update, save, rollback, ...)
- **NSPersistentStoreCoordinator**: Verantwortlich für Speichern und Lesen von Objekten aus einem Store
- **NSPersistentContainer**: Wrapper, der sich um all diese Objekte kümmert



# Core Data – Features (1/2)

Core Data drastically decreases the amount of code you write to support the model layer. This is primarily due to the following built-in features that you do not have to implement, test, or optimize:

- **Change tracking** and built-in management of **undo and redo** beyond basic text editing.
- Maintenance of **change propagation**, including maintaining the **consistency of relationships** among objects.
- **Lazy loading** of objects, partially materialized futures (faulting), and copy-on-write data sharing to reduce overhead.
- **Automatic validation** of property values. Managed objects extend the standard key-value coding validation methods to ensure that individual values lie within acceptable ranges, so that combinations of values make sense.

## Core Data – Features (2/2)

- **Schema migration** tools that simplify schema changes and allow you to perform efficient in-place schema migration.
- Optional integration with the application's controller layer to support user **interface synchronization**.
- **Grouping, filtering, and organizing** of data in memory and in the user interface.
- Automatic support for **storing objects in external data repositories**.
- Sophisticated **query compilation**. Instead of writing SQL, you can create complex queries by associating an **NSPredicate** object with a fetch request.
- **Version tracking and optimistic locking** to support automatic multiwriter conflict resolution.

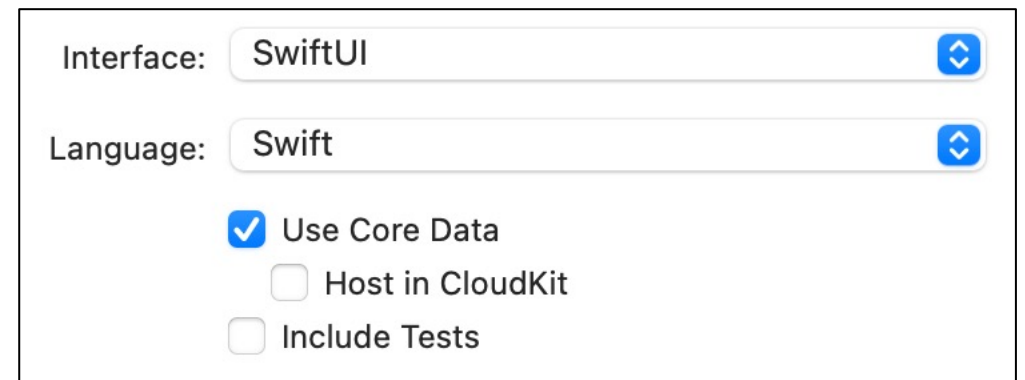


# Was Core Data *nicht* ist

- Core Data ist keine relationale Datenbank oder ein *Relational Database Management System (RDBMS)*
  - *Core Data provides an infrastructure for change management and for saving objects to and retrieving them from storage. It can use SQLite as one of its persistent store types. It is not, though, in and of itself a database. ...*
- “Core Data is not a silver bullet”
  - Core Data does not remove the need to write code. Although it is possible to create a sophisticated application solely using the Xcode data modeling tool and Interface Builder, for more real-world applications you will still have to write code.

# Core Data: Xcode Template

- iOS App als Template wählen, “Use Core Data” Häkchen setzen
- Generiert ziemlich viel (Boilerplate-)Code und Objekte
  - PersistentController
  - Datenmodell-Datei (.xcdatamodeld)
  - @FetchRequest PropertyWrapper, um Model mit (SwiftUI-)View zu connecten
    - Optional mit Predicate und SortDescriptor



The image shows a screenshot of the Xcode interface for selecting an iOS app template. It features two dropdown menus: 'Interface' set to 'SwiftUI' and 'Language' set to 'Swift'. Below these are three checkboxes: 'Use Core Data' (checked), 'Host in CloudKit' (unchecked), and 'Include Tests' (unchecked).


Interface:	SwiftUI
Language:	Swift
<input checked="" type="checkbox"/>	Use Core Data
<input type="checkbox"/>	Host in CloudKit
<input type="checkbox"/>	Include Tests

# Datenmodell

- Datei mit Endung .xcdatamodeld
- Visueller Editor für Models
- Primär wichtig: Entities mit...
  - Attributen (Name & Typ, plus zusätzliche Attribute)
  - Relationships (Beziehungen zwischen Entities)
- Swift-Klassen für diese Entities werden automatisch generiert

**Attribute**

Name



Type  

☒ Optional ☐ Transient

☐ Derived

☐ Allows Cloud Encryption

Default Value ☐ Default String


Validation    

Min Length Max Length

Reg. Ex.

Advanced ☐ Index in Spotlight

☐ Preserve After Deletion

Relationships		
Relationship	Destination	Inverse
 teams	Team	members

# Core Data – Take-aways

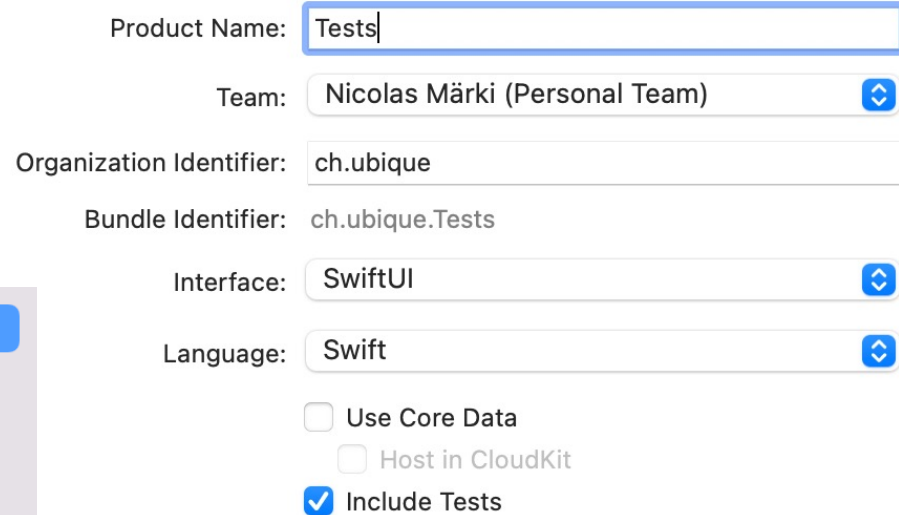
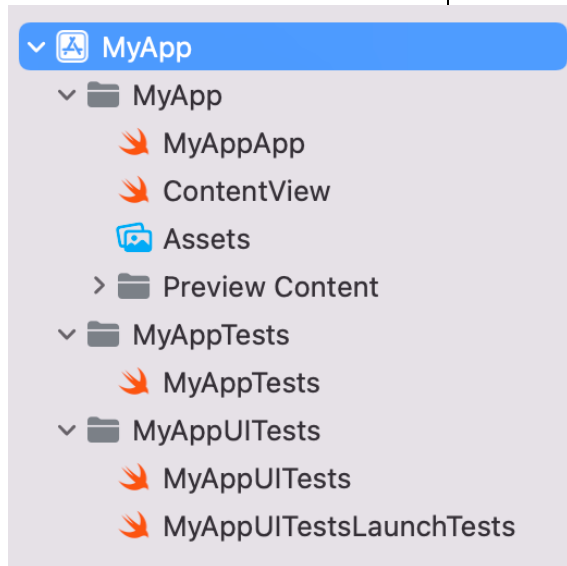
- Core Data ist definitiv keine Entry-Level Technologie. Sehr mächtiges Tool, das aber Zeit braucht, um gelernt zu werden
  - That being said: Mit Projekt-Template und einem nicht allzu komplexen Use-Case durchaus gute Alternative (weil auch andere DB-ähnliche Alternativen nicht unbedingt trivial...)
- Grosser Funktionsumfang, kann interessant sein für diverse Use-Cases
- Core Data ist verhältnismässig “altes” Framework
  - → Gibt relativ viel Support & Dokumentation
  - → Sehr gut getestet

# Core Data – Demo

# Testing

# Testing

- 2 Test-Typen:
  - Unit Tests
  - UI Tests

A screenshot of the 'New Project' dialog in Xcode. The 'Product Name' field is set to 'Tests'. The 'Team' is 'Nicolas Märki (Personal Team)'. The 'Organization Identifier' is 'ch.ubique'. The 'Bundle Identifier' is 'ch.ubique.Tests'. The 'Interface' is 'SwiftUI' and the 'Language' is 'Swift'. At the bottom, there are three checkboxes: 'Use Core Data' (unchecked), 'Host in CloudKit' (unchecked), and 'Include Tests' (checked).

# Unit Tests mit XCTest

- Xcode bietet Unit Testing an: Prüfen korrektes Funktionieren einzelner Code-Einheiten (“Units”)
  - Projekt-Templates können automatisch Tests inkl. Target erstellen
- <https://developer.apple.com/documentation/xctest>



# Unit Tests

- Klassische Unit-Tests: Test von einzelnen Methoden, Datenstrukturen, deren Zusammenspiel, ...
- Werden losgelöst von der App ausgeführt
  - → d.h., es gibt z.B. kein AppDelegate
  - → Evt. Mock-Objekte nötig
- Tests laufen lassen: Product → Test (oder direkt via Play-Button im Editor)

```
8 import XCTest
9
10 final class HSLUCoreDataUITests: XCTestCase {
11
12     override func setUpWithError() throws {
13         // Put setup code here. This method is
14
15         // In UI tests it is usually best to st
16         continueAfterFailure = false
17     }
```

# Test-Methoden

- Methoden-Name beginnt mit **test**
- Rückgabetyp: void
- **setup** & **tearDown**:
  - Diese Methoden werden vor resp. nach jeder Test-Methode ausgeführt

# Assertions

- Diverse Assertions, um Werte zu prüfen
  - XCTAssertTrue
  - XCTAssertEqual
  - XCTAssertNotNil

```
24
❌ func testFunction() {
26     XCTAssertEqual(multiply(x: 3, y: 5), 14)
27 }
28
```

❌ testFunction(): XCTAssertEqual failed: ("15") is not equal to ("14")

# UI Tests

- Basierend auf Accessibility API
- Automatisierung von UI Interaktionen
  - Elemente finden
  - Mit der UI interagieren (z.B. Button klicken)
  - Eigenschaften und Status von Elementen validieren
- UI Recording: Quick Start für UI Tests

# Unit vs. UI Tests

- “Interne” vs. “externe” Sicht
- Unit Tests:
  - Zugriff auf Methoden, Funktionen, Variablen und Status der App
- UI Tests:
  - Simulieren von UI-Interaktionen aus User-Sicht in einem separaten externen Prozess

# Unit Tests – Demo

# Inter-App Kommunikation

# Inter-App Kommunikation

- Bereits behandelt: Jede App ist eine Insel (“Sandbox”-Konzept)
  - → Wie funktioniert Austausch mit anderen Apps?
- Verschiedene Möglichkeiten, z.B.:
  - Andere App starten → URL Schemes
  - Document Interaction
    - Z.B. PDF-Dateien
  - Universal Links



# URL Schemes

- System-Methode, um beliebige URLs zu öffnen
- System prüft, ob URL von bestimmter App geöffnet werden kann/soll

```
let url = URL(string: "mailto:ios@hslu.ch")!  
UIApplication.shared.open(url)
```

# URL Schemes

- URLs für System-Apps:
  - http(s)
  - mailto
  - tel
  - sms
  - youtube
  - itunes
  - maps
- Gibt's auch für Apps von Drittanbietern
  - Z.B. SBB, Facebook, ...
  - Inoffizielle Listen im Internet

# URL Schemes

- Probleme / Nachteile:

## Warning

URL schemes offer a potential attack vector into your app, so make sure to validate all URL parameters and discard any malformed URLs. In addition, limit the available actions to those that don't risk the user's data. For example, don't allow other apps to directly delete content or access sensitive information about the user. When testing your URL-handling code, make sure your test cases include improperly formatted URLs.

## Note

If multiple apps register the same scheme, the app the system targets is undefined. There's no mechanism to change the app or to change the order apps appear in a Share sheet.

# URL Schemes – Demo

# Lösung: Universal Links

- Öffnen von “normalen” http(s) Links mit der eigenen App
- Meta-File auf dem Server notwendig
  - → Sicherheitsmechanismus, um sicherzustellen, dass nur “eigene” URLs geöffnet werden können
- Bevorzugt gegenüber URL Schemes
- <https://developer.apple.com/documentation/xcode/allowing-apps-and-websites-to-link-to-your-content>

# User Notifications

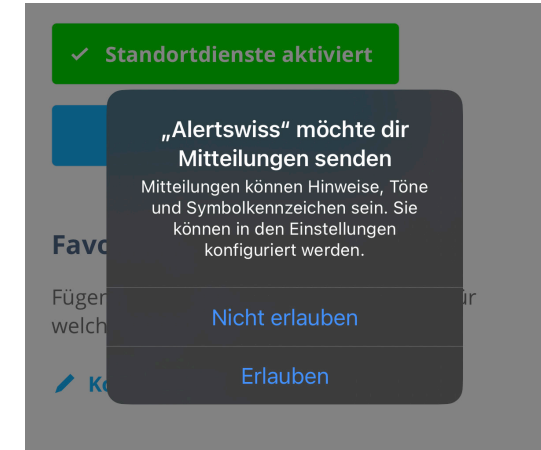
# User Notifications

- Framework für Benachrichtigungen
- Wird verwendet für zwei Arten von Benachrichtigungen:
  - Push-Benachrichtigungen von einem Server
  - Lokale, direkt von der App generierte Benachrichtigungen



# User Notifications – Grundlagen

- Benachrichtigungen können wichtiger Kanal für eine App sein, um Informationen zu kommunizieren
- Informieren den User, auch wenn die App gerade nicht verwendet wird
- Brauchen explizite Berechtigung → viele User sind eher zurückhaltend (wollen nicht mit Benachrichtigungen “zugespammed” werden)



- HIG zu Notifications: <https://developer.apple.com/design/human-interface-guidelines/components/system-experiences/notifications/>

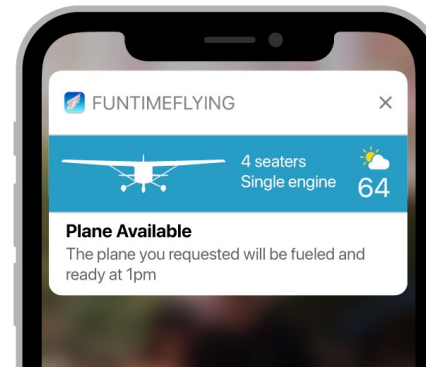


# User Notifications – zusätzliche Möglichkeiten

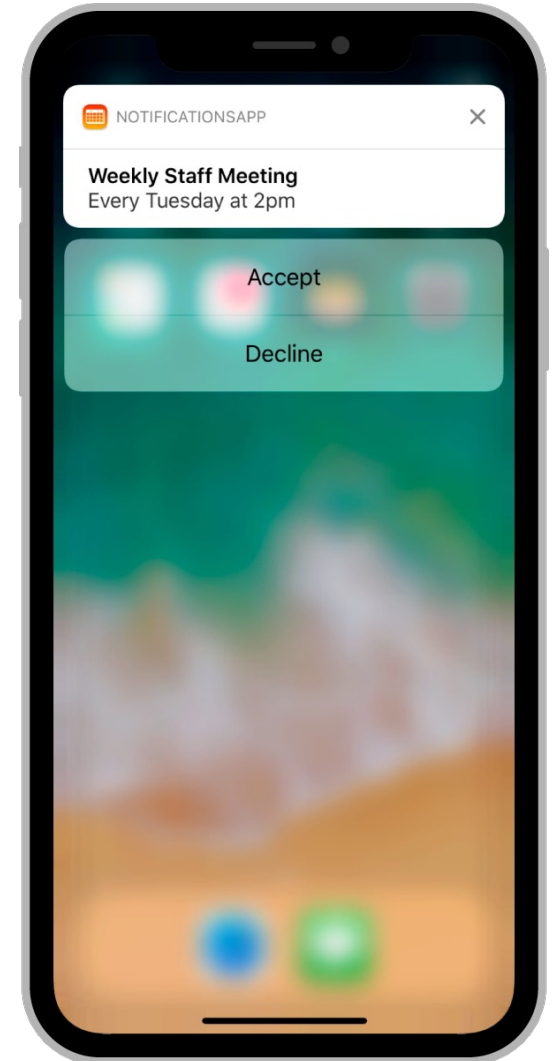
- Man kann Actions definieren → User kann reagieren, ohne App öffnen zu müssen
- Man kann (mittels Extension) Custom UI implementieren (UNNotificationContentExtension)
- Ebenfalls mittels Extension: Möglichkeit, vom Server erhaltene Push Notification zu ändern (UNNotificationServiceExtension)



Default UI



Custom UI



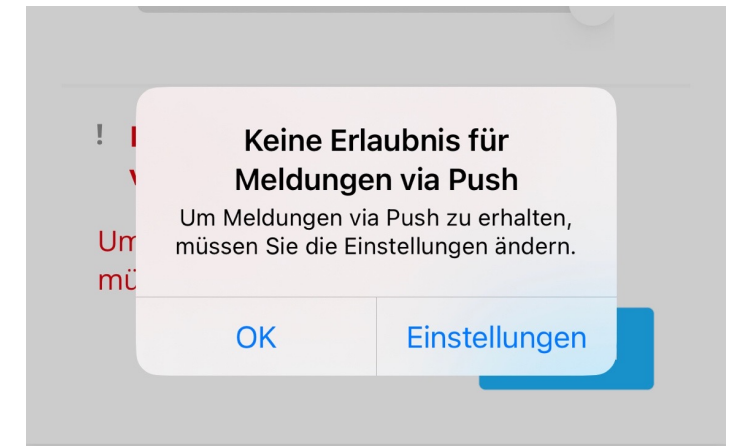
# User Notifications – Permission

- Grundsatz: Eine App darf 1x nach Erlaubnis fragen, danach nicht mehr
- User kann “Entscheidung” jederzeit in den Einstellungen anpassen
- Häufiger Flow: Fehlermeldung, falls Permission abgelehnt → inkl. Absprung in die Einstellungen

```
let center = UNUserNotificationCenter.current()
center.requestAuthorization(options: [.alert, .sound, .badge]) { granted, error in

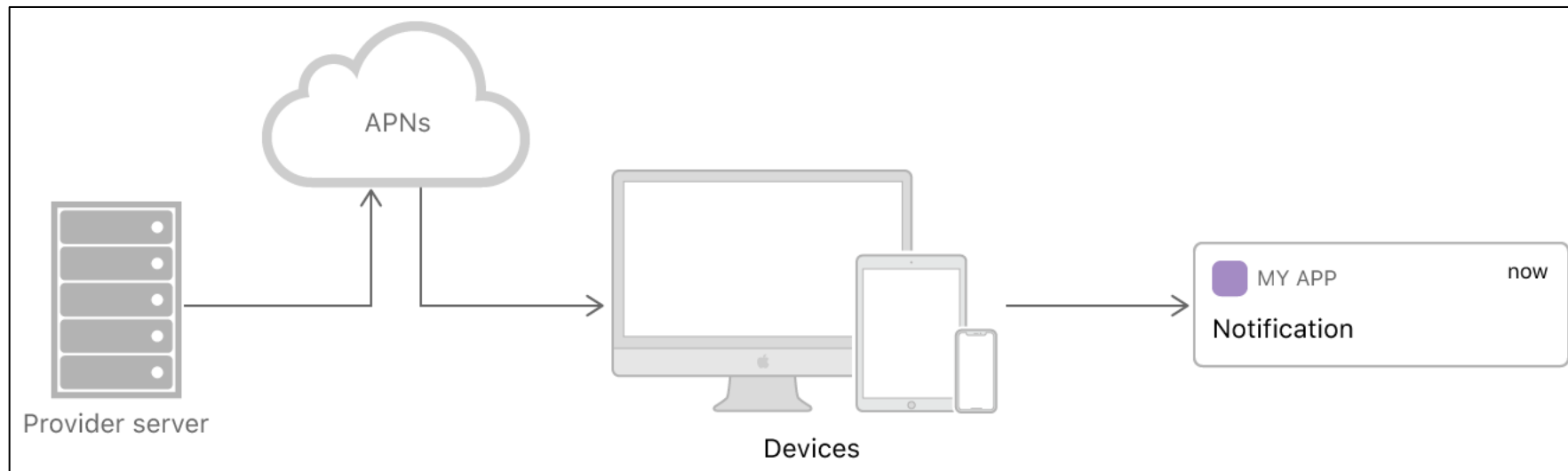
    if let error = error {
        // Handle the error here.
    }

    // Enable or disable features based on the authorization.
}
```

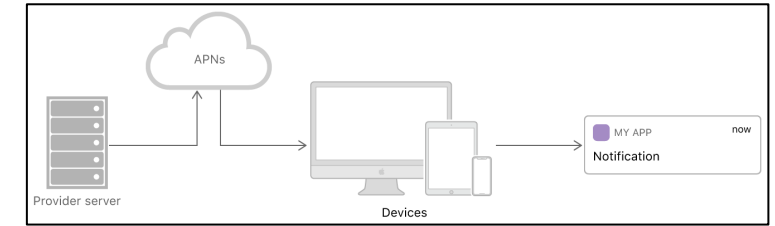


# User Notifications – Remote

- Architektur:
  - App Server: Verantwortlich für Business-Logik (wann muss Push an welche User versendet werden?)
  - APNs – Apple Push Notification service: Verantwortlich für's Ausstellen von Push Tokens sowie Versenden von Push-Nachrichten an Devices



# User Notifications – Remote



- Flow:

- App erlangt Permission für Benachrichtigungen vom User
- App fragt nach Token via System-Call zu APNs

```
UIApplication.shared.registerForRemoteNotifications()
```

- App erhält Token und schickt es an App-Server

```
func application(_ application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data) {  
    // Send token to app server...  
}
```

- App-Server merkt sich Token (z.B. in DB)
  - mit User verknüpft, z.B. via Account, Unique ID, ...
- Push-Benachrichtigung senden: App-Server schickt Inhalt und Token zu APNs  
→ Device erhält Push “direkt von Apple”

# User Notifications – Local

- Aus Usersicht kein Unterschied erkennbar → Unterschied zu remote Notifications: Zeitpunkt & Inhalt werden direkt von der App gemanaged
- Notifications werden gescheduled, entweder für sofortige Delivery oder optional mit Trigger
- 3 Arten von Triggers:
  - `UNCalendarNotificationTrigger`
  - `UNTimeIntervalNotificationTrigger`
  - `UNLocationNotificationTrigger` (Achtung: braucht wiederum Location-Permission...)
- Geschedulete Notifications können auch gecanceled werden

# User Notifications – Demo

# Exkurs: Silent Push Notifications

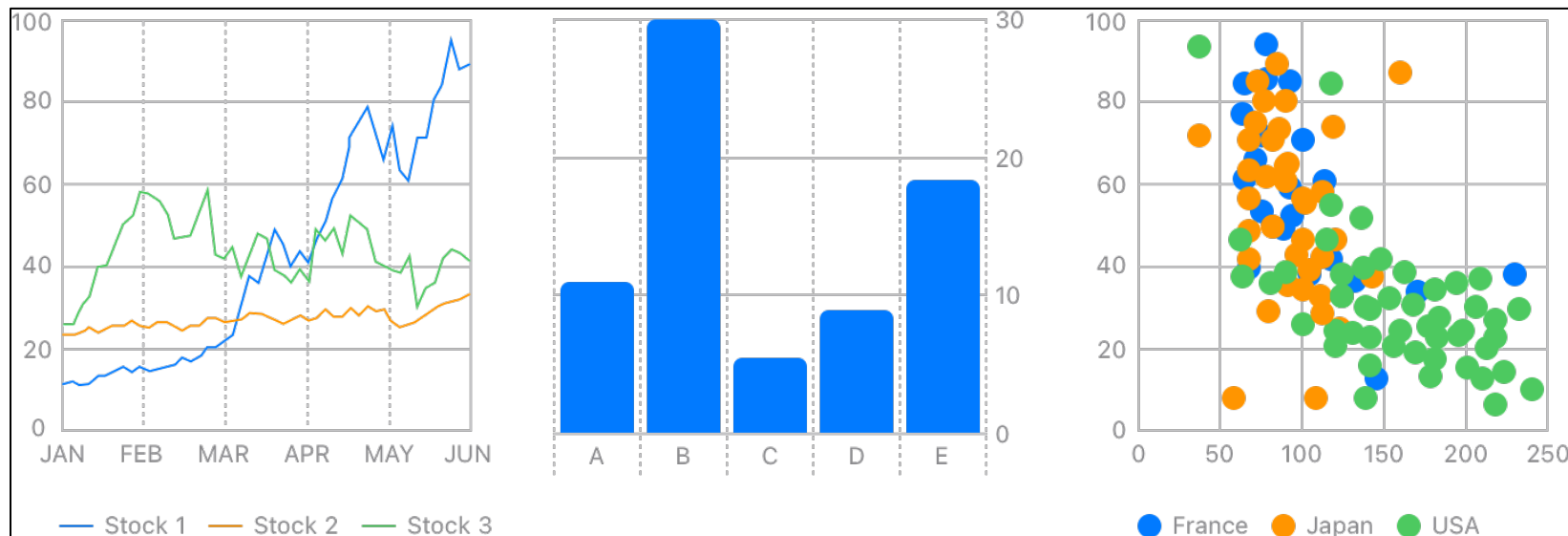
- Server kann Notifications auch “silent” schicken → App bekommt Push-Nachricht, ohne dass User etwas mitbekommt
- Beispiel UZH: Entscheiden, ob User sich tatsächlich in für Alarm relevanter Region befindet → Locations müssen nicht an Server gesendet werden
  - Alertswiss hat ähnliche Logik für Meldungen

# Swift Charts



# Swift Charts

- Neues SwiftUI-Framework (WWDC 2022, iOS 16)
- Sehr intuitiv, SwiftUI-ige deklarative Syntax
- Interessant für einfache, aber auch komplexere Daten-Visualisierungen



# Swift Charts – Einführung

- Wichtig bei Daten-Visualisierung:
  - Welche Daten habe ich?
  - Was möchte ich kommunizieren?
  - Was ist die geeignete Darstellung dafür?
- Wäre wohl eigene Vorlesung wert...

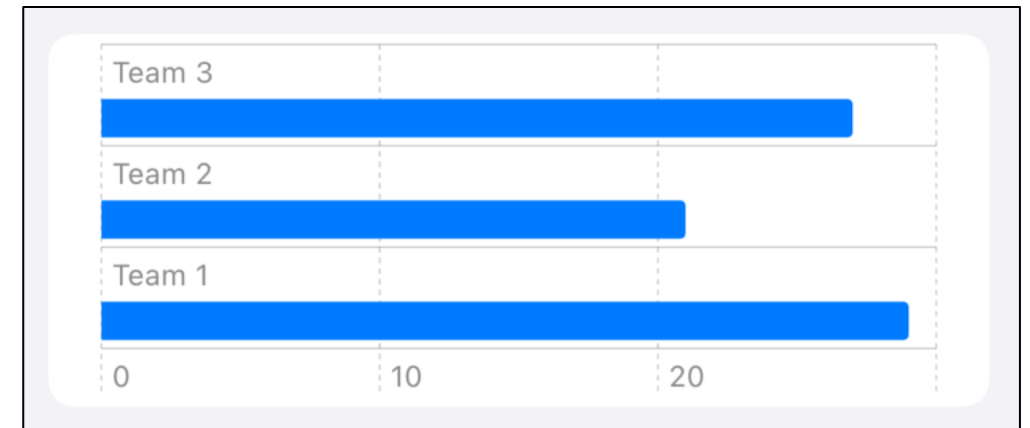
# Swift Charts – Syntax

- **Chart** als Wrapper mit verschiedenen “Marks”
  - BarMark, LineMark, PointMark, RectangleMark, AreaMark, ...
- Meistens mit `ForEach`, kann aber auch explizit mit einzelnen Marks erstellt werden
  - Oder Syntax-Vereinfachung:

```
Chart {  
    ForEach(teams, id: \.id) { team in  
        BarMark(x: .value("Points", team.points),  
                y: .value("Name", team.name))  
    }  
}
```

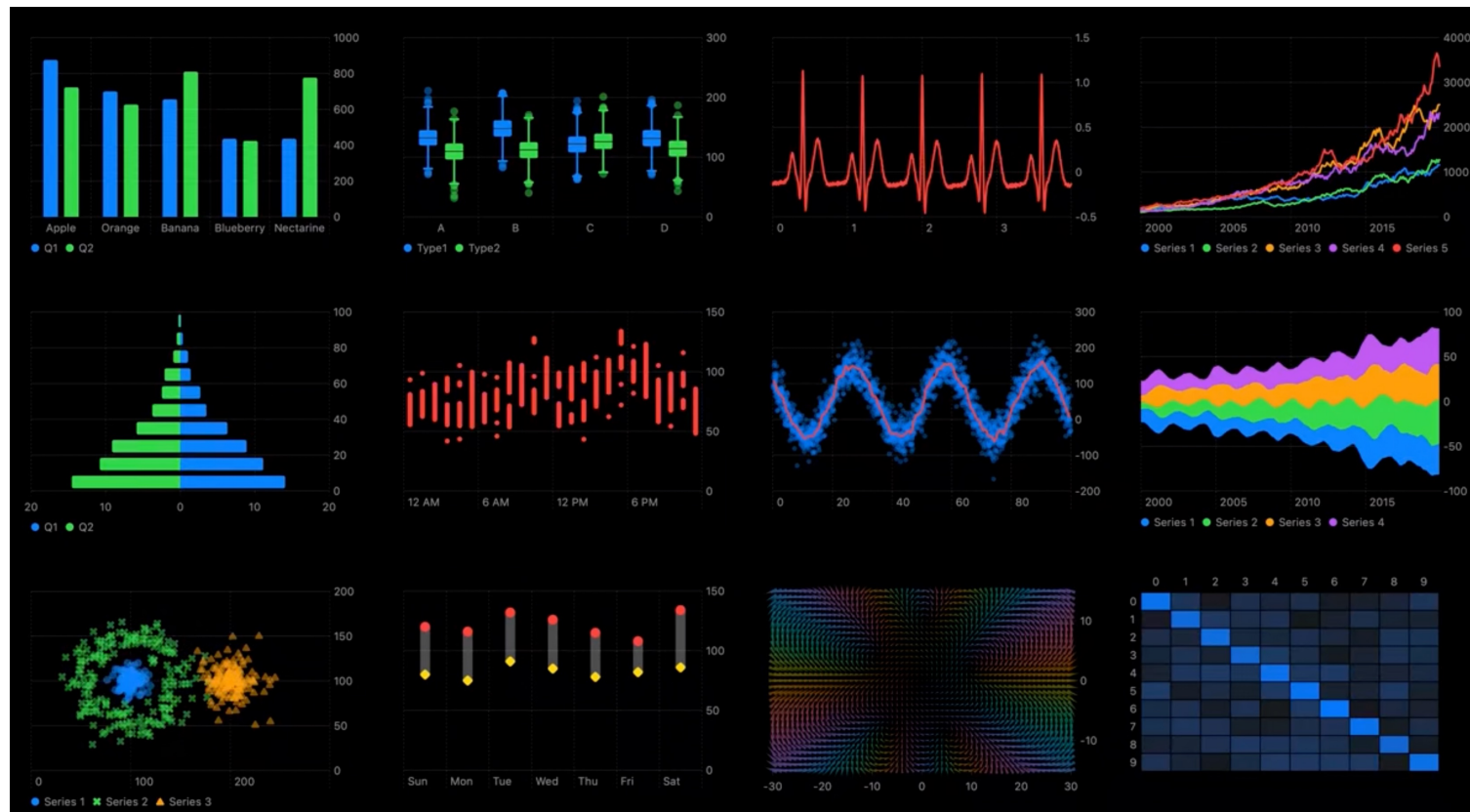


```
Chart(teams) { team in  
    BarMark(x: .value("Points", team.points),  
            y: .value("Name", team.name))  
}
```



# Swift Charts – Syntax

- Diverse Modifiers, um Style und Funktionalität anzupassen
  - Achsen, Legende, Farben, Styles, interaktive Elemente, ...



# Swift Charts – Ressourcen

- <https://developer.apple.com/documentation/charts>
- Sample-Projekt mit vielen Beispiel-Charts:  
[https://developer.apple.com/documentation/charts/visualizing\\_your\\_app\\_s\\_data](https://developer.apple.com/documentation/charts/visualizing_your_app_s_data)
- WWDC-Videos:
  - <https://developer.apple.com/videos/play/wwdc2022/10136/>
  - <https://developer.apple.com/videos/play/wwdc2022/10137/>

# Swift Charts – Demo

# Frameworks: Round-Up

- Es gibt eine Vielzahl spannender Frameworks / Technologien im iOS-Universum
- <https://developer.apple.com/documentation/technologies>
- Alle kennen: praktisch unmöglich
  - Viele sind sehr spezifisch / advanced, haben klaren Use Case
- Liste kann evt. als Inspirationsquelle für Gruppenprojekt dienen
  - Wenn möglich: Sample-Projekt downloaden und rumspielen