

# Fragmentierung, Mobile Usability, HIG, Lokalisierung, Widgets

Programmieren für iOS



# Inhalt

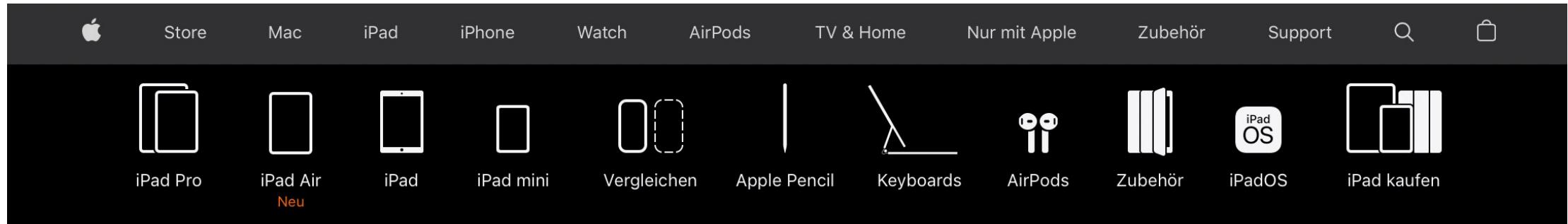
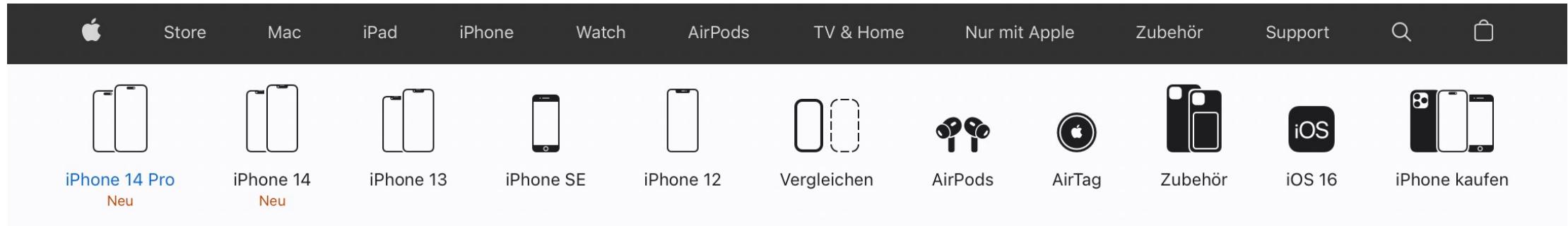
- Fragmentierung: iOS-Geräte und Eigenschaften
- App-Design & Human Interface Guidelines (HIG)
- Mobile Usability und User Experience (UX)
- Internationalisierung & Lokalisierung
- Widgets
- App-Veröffentlichung

# Die iOS-Gerätefamilie



# Die iOS-Gerätefamilie

... wächst jedes Jahr:



# Geräte-Eigenschaften

- Touch-Bildschirm (verschiedene Größen)
- Kamera(s) mit verschiedenen Spezifikationen
  - Front & Back
- Audio
- (Satelliten-)Telefonie
- 4G/5G, GPS, WiFi, Bluetooth, ...
- Viele Sensoren: Kompass, Beschleunigung, TouchID/FaceID, Helligkeit, Gyroskop, NFC, ...
- Weitere spezifische Chips: GPU, U1, Lidar, ...

# Bildschirm-Auflösungen

- <https://iosref.com/res>

## Resolution by iOS device

[iPhone](#) [iPad](#) [Apple Watch](#) [iPod touch](#) [Further reading](#)

All tables are ordered by release date, then logical resolution height, with some exceptions.

### iPhone

Device	Diagonal size	Logical resolution	Scale factor	Actual resolution	Aspect ratio	PPI
iPhone 14 Pro	6.1"	393 × 852	@3x	1179 × 2556	9 : 19.5	460
iPhone 14 Pro Max	6.7"	430 × 932		1290 × 2796		
iPhone 14 Plus and 13 Pro Max and 12 Pro Max		428 × 926		1284 × 2778		
iPhone 14 and 13 / 13 Pro and 12 / 12 Pro	6.1"	390 × 844		1170 × 2532		
iPhone 13 mini and 12 mini	5.4"	375 × 812	@2.88–3x <sup>1</sup>	1080 × 2340	476	458
iPhone 11 Pro Max	6.5"	414 × 896	@3x	1242 × 2688		

# Bildschirm-Auflösungen

- 3 “Versionen” von Pixels:
  1. “Points” → Im Code verwendete Einheit. Abstraktion von physikalischen Pixels.
  2. “Rendered Pixels” → Bilder werden hochskaliert mit bestimmtem Faktor, meistens 1x / 2x / 3x.
  3. “Physical Pixels” → Rendered Pixels werden auf physische Pixels im Display gemapped. Meistens 1:1, kann aber auch != 1 sein (siehe Tabelle).
- Dazu: PPI (Pixels-per-Inch) → Sagt aus, wieviele Pixels in einem Inch Platz haben und somit, wie gross das Display am Schluss in der echten Welt ist

<https://www.paintcodeapp.com/news/iphone-6-screens-demystified>

# Bildschirm-Auflösungen

- Im Code: Wir haben fast immer nur mit **Points** zu tun
- Falls nötig: Scale Factors können ausgelesen und verwendet werden.

```
/// The natural scale factor associated with the screen.
```

```
let scale = UIScreen.main.scale
```

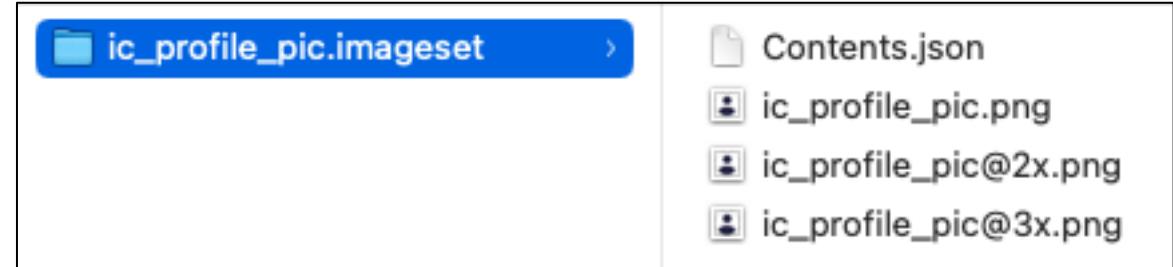
```
/// The native scale factor for the physical screen.
```

```
let nativeScale = UIScreen.main.nativeScale
```

- NativeScale kann mehrheitlich ignoriert werden, kann wichtig sein in Grafik-lastigen Anwendungen (z.B. Games)

# Exkurs: Bilder

- Bilder werden in 3 Auflösungen gespeichert
- App wählt richtige Auflösung für jeweiligen Screen-Scalefaktor
- Wenn richtig benennt: Drag and drop in Xcode
- Viele Design-Tools haben mittlerweile Export-Funktion mit richtigem Format

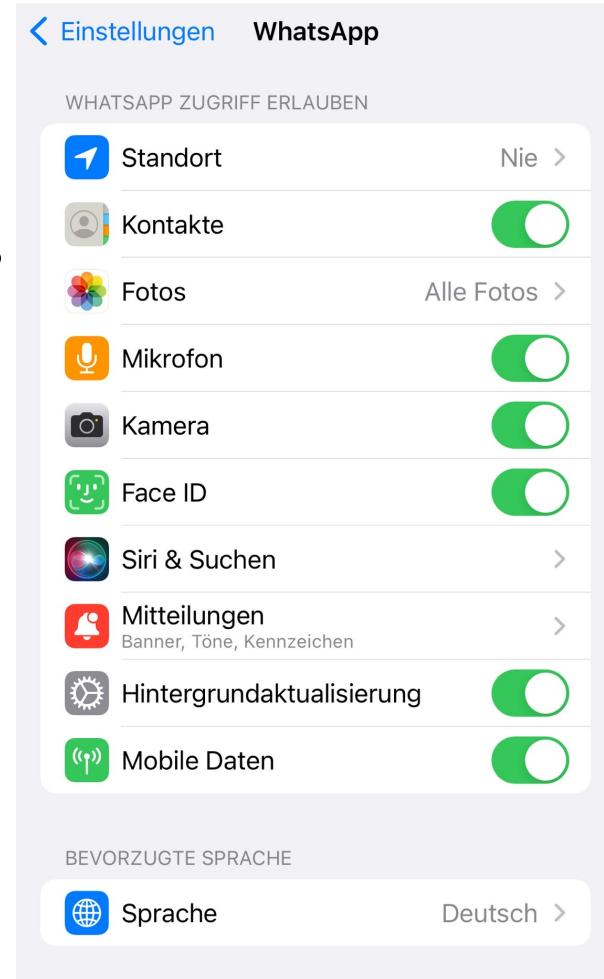


# Bildschirm-Auflösungen

- Grundsatz für UI-Programmierung: So wenig wie möglich mit hardcoded Point-/Pixel-Werten arbeiten → Relatives Layout
- Mit SwiftUI nochmals besser geworden: Man sagt, was man möchte (z.B [List](#)), System sorgt für sinnvolle Darstellung, sogar auf verschiedenen Devices (iPhone, iPad, Watch)
- Wenn fixe Größen nötig sind: Kleinstes / grösstes Gerät im Auge behalten (+ Displayzoom, Dynamic Type etc.)

# Geräte: Möglichkeiten / Einschränkungen

- Immer mehr neue Sensoren & Berechtigungen
- System “schützt” Nutzer vor Apps:
  - Sandbox → Apps haben nur sehr beschränkten Zugriff auf's Filesystem oder auf Daten anderer Apps
  - Gute Kontrolle darüber, welche Daten an welche App gegeben werden
  - Aber: Permission Handling aus Entwickler-Sicht kann mühsam werden. Viele Spezialfälle möglich (z.B. kein Background-Fetch, nur “ungefähre Location” freigegeben)



# Interaktions-Limitierungen

- Was es standardmässig auf iOS nicht gibt:
    - Maus
    - USB-Anschluss (für externe Keyboards oder andere Accessories)
  - Dafür gibt's:
    - Software-Keyboards (flexibler, z.B. optimiert für Inputfeld)
    - Multitouch → Gesten
    - Zugriff auf Sensoren
    - iPad: Apple Pencil
- Interessante, neue Mensch-Computer-Interaktionen!

# App-Design & Human Interface Guidelines

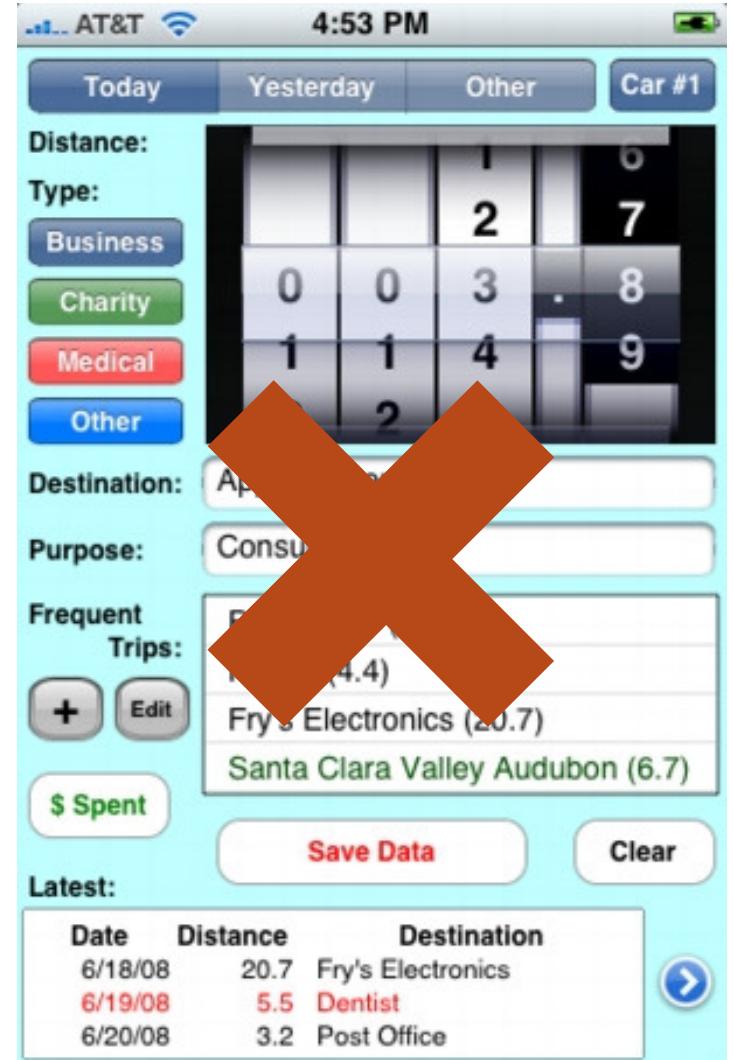
# App-Design



# App-Design

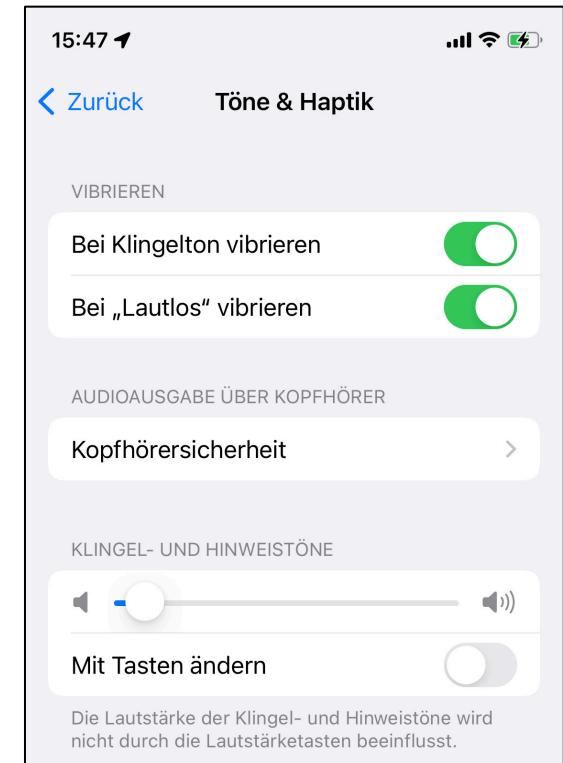
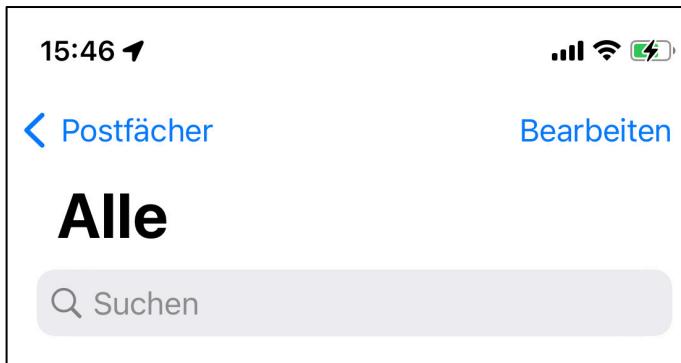
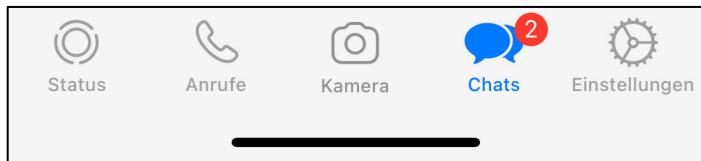
- Warum sehen viele iOS-Apps einheitlich und “schön” aus?

→ Apple macht Regeln!



# Standard-UI

- System-Frameworks (UIKit & SwiftUI) enthalten viele Standard-UI-Elemente:
  - Navigation Bars
  - Tab Bars
  - Buttons, Toggles, Sliders, ...



# Human Interface Guidelines

- Guide zur Erstellung von Apps für Apple-Plattformen
- Instruktionen & Hinweise, wie die Standard-UI-Elemente eingesetzt werden sollten
- Ausführliche Sammlung von Best Practices (“The Apple Way”)

**Im Grossen und Ganzen sinnvoll – aber nicht als einzige Wahrheit zu sehen!**

- Grosser Vorteil: Apps fühlen sich einheitlich und vertraut an, User müssen weniger “lernen”

# Human Interface Guidelines

Demo

<https://developer.apple.com/design/human-interface-guidelines/guidelines/overview/>

# Design, Usability & UX

# UX – Warum?



# UX – Warum?

Birthday\*:

January ▾ 1 ▾ 2017 ▾

Primary phone number\*:

0 ▾ 0 ▾ 0 ▾ 0 ▾ 0 ▾ 0 ▾ 0 ▾ 0 ▾  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9

Secondary phone number\*:

0 ▾ 0 ▾ 0 ▾ 0 ▾ 0 ▾ 0 ▾ 0 ▾  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9

Fields marked with an asterisk (\*) are required. Make sure your details are correct before you submit the application and will email you on the progress.

Make sure you have read and understood the terms and conditions before you submit the application in cases outlined in terms and conditions.

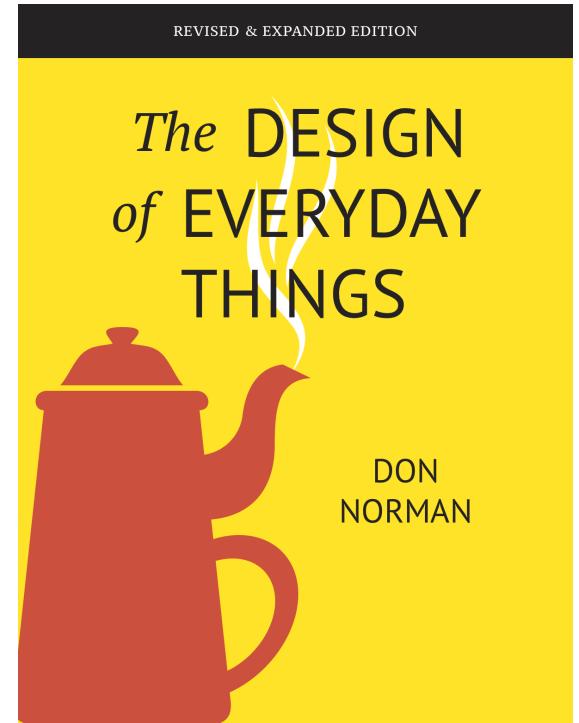
reject yes

# Definitionen

- Design: Spezifischer Zweck, Absicht, wie etwas zu gebrauchen ist / auszusehen hat.  
Kommunikation zwischen Objekt und User.
- Usability: “Gebrauchtstauglichkeit”, der Grad, die Qualität der Benutzbarkeit von etwas
  - Wie gut kann ein **User** in einem bestimmten **Kontext** ein bestimmtes **Ziel** erreichen
  - Viele Aspekte: nützlich, zuverlässig, sicher, übersichtlich, barrierefrei, ...
- User Experience: Wie einfach und angenehm etwas zu benutzen ist
  - Ziel: Glückliche User :-)

# Design-Prinzipien

- “The Design of Everyday Things”, Donald Norman (1988)
- Was macht gutes & schlechtes Design aus?
- Vorschlag verschiedener **Design-Prinzipien**



# Design-Prinzipien

- **Affordances**: Mögliche Handlungen, die ein Objekt erlaubt.  
Bsp.: Ein Stuhl erlaubt es, darauf zu sitzen (ein Tisch aber auch!).
- **Signifiers**: Helfen, die Affordances eines Objektes zu kommunizieren, v.a. dort wo sie nicht offensichtlich sind  
Bsp.: Labels, Schilder
- **Constraints**: Limitierungen, die die Benutzung eines Objektes einschränken
- **Mappings**: Beziehungen zwischen Objekt und Resultat. Können z.B. örtlich oder zeitlich sein.
- **Feedback**: Kommunikation der Resultate einer Handlung für den User.

# Affordance – Beispiele

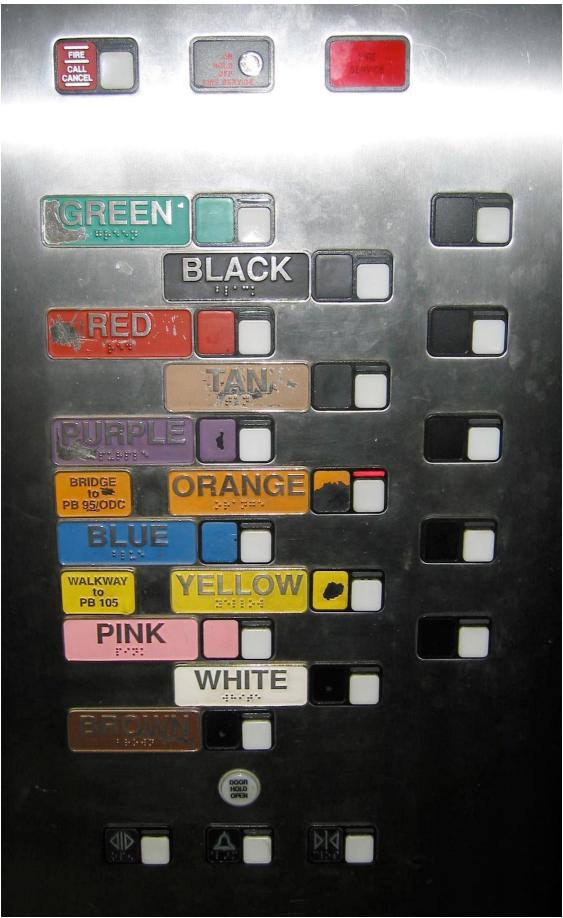


# Affordance – Beispiele

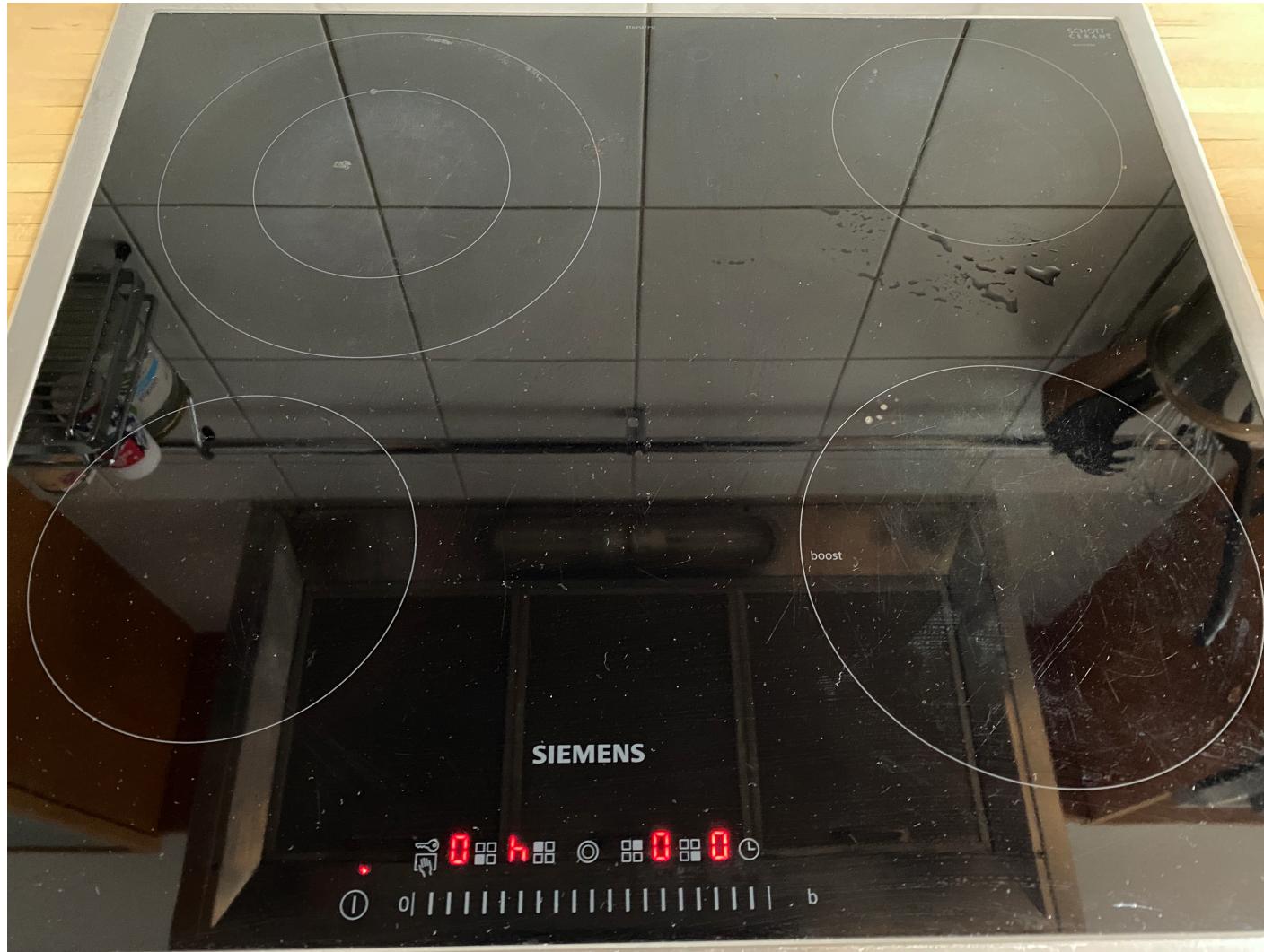
## ***Norman door (n.):***

- 1. A door where the design tells you to do the opposite of what you're actually supposed to do.**
- 2. A door that gives the wrong signal and needs a sign to correct it.**

# Mappings – Beispiele







# Mappings – Beispiele

- Besseres Mapping, weil Beziehung zur realen Welt sofort erkennbar ist
- Ausserdem auch Affordance sehr einfach erkennbar: Label und Button in einem → man weiss intuitiv, wo klicken



# UX-Design für Mobile Apps

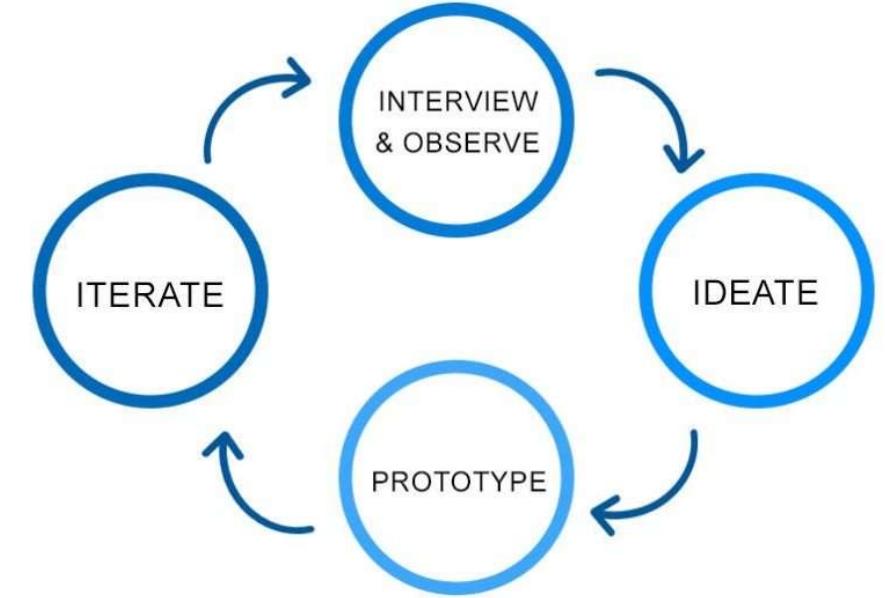
- Informations-Architektur: Wie ist der Inhalt strukturiert?
  - Navigations-Elemente: Tabbar, Navigationbar, Sheets, ...
  - Übersichten, Listen, Details, ...
- Design
  - Einfachheit
  - Einheitlichkeit (Design-System – Farben, Icons, Schriften, Spacing, ...)
- Interaktion
  - Feedbacks (z.B. Touch-States)
  - Loading & Error States
  - Sinnvolle Transitions / Animationen

# UX – Erfahrungen

- UX ist entscheidend für eine gute App
- UX ist auf allen Ebenen wichtig:
  - Konzeption
  - Design
  - Implementierung
  - ...
- Alle beteiligten Personen sollten UX-Fokus haben
- UX is hard!!

# Prototyping

- Ziel: Iterativ zum Ziel kommen, Ideen & Designs früh validieren
- Verschiedene Möglichkeiten:
  - Paper Prototyping
  - Software-Lösungen (Wireframes)
- Mit SwiftUI & Previews kann man ebenfalls sehr schnell iterieren!



# UX/Usability-Testing

- Personen von “ausserhalb” die App testen lassen → selber wird man oft blind für UX, wenn man zu tief im Projekt ist
- Keine vollständige App nötig, kann z.B. mit Papier-Prototypen oder Click-Through-Prototype gemacht werden
- Deckt UX-Probleme oft sehr schnell auf
- Ganz wichtig: Qualitatives statt quantitatives Feedback! (kleine Sample-Grösse, z.B. 2-4 Personen genügt völlig)

# Lokalisierung

# Internationalisierung

- **Internationalization – i18n:**

**Internationalization** is the process of making your app able to adapt to different languages, regions, and cultures. Because a single language can be used in multiple parts of the world, your app should adapt to the regional and cultural conventions of where a person resides. An internationalized app appears as if it is a native app in all the languages and regions it supports.

- **Localization – l10n**

**Localization** is the process of translating your app into multiple languages. But before you can localize your app, you internationalize it.

- **Achtung: Vor allem, aber nicht nur Sprache!**

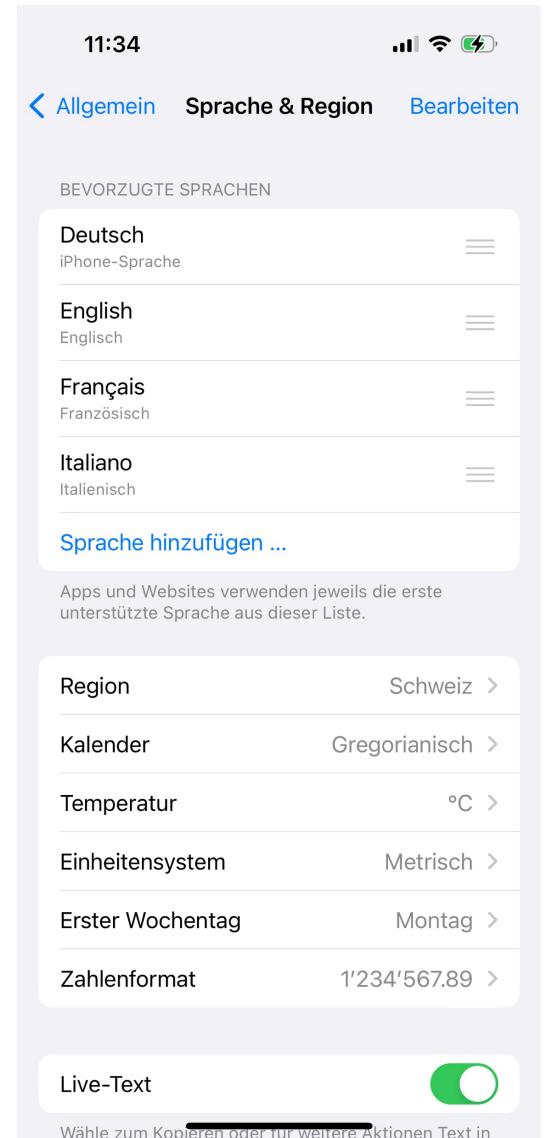
- Layout- und Text-Ausrichtung (right-to-left)
- Datums- / Zeitformate
- Zahlen
- Icons / Farben

<https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/BPInternational/Introduction/Introduction.html>

# Lokalisierung in iOS

- iOS gibt die Sprache vor
  - Einstellungen → Allgemein → Sprache & Region
- Apps verwenden die erste unterstützte Sprache aus dieser Liste
  - User kann pro App eine andere Sprache einstellen

Wie reagieren wir in der App auf diese Einstellungen?



# Lokalisierung in iOS

- **Locale**

Information about linguistic, cultural, and technological conventions for use in formatting data for presentation.

- Informationen über die aktuelle Konfiguration (Region, Sprache etc.)
  - Nützlich z.B. für Formatierung (Datum, Währung, Zahlen etc.)

```
func getButtonTitle() -> String {  
    let languageCode = Locale.preferredLanguages[0]  
    switch languageCode {  
        case "en": return "Register"  
        case "fr": return "Enregistrer"  
        default: return "Registrieren"  
    }  
}
```

??

# Exkurs: ISO-Abkürzungen

- ISO 639-1: 2-Buchstaben-Kürzel für jede Sprache
  - Deutsch = de
  - Englisch = en
  - Französisch = fr
  - etc...
- ISO 3166-1: 2-Buchstaben-Kürzel für jede Region
  - Schweiz: CH
  - USA: US
  - Australien: AU
- Kann kombiniert werden, z.B. **en\_US** vs. **en\_AU**

# Lokalisierung in Xcode

- Xcode erlaubt es, Lokalisierung auf Datei-Ebene vorzunehmen
  - 1 Datei pro unterstützte Sprache/Region
- Hilfreich, um lokalisierte Inhalte von Implementation zu trennen  
→ Bessere Wartbarkeit (z.B. Arbeit mit Übersetzern)
- Funktioniert mit verschiedenen Dateitypen
  - Storyboards/Interface Builder Files
  - Texte
  - Bilder
  - Sounds
  - ...
- Im folgenden: Fokus auf Texte

# Lokalisierung in Xcode

PROJECT  
HSLUDemo

TARGETS  
HSLUDemo

Deployment Target  
iOS Deployment Target 16.1

Configurations

Name	Based on Configuration File
> Debug	No Configurations Set
> Release	No Configurations Set

Use Release for command-line builds

Parallelize build for command-line builds (does not apply when using schemes)

Localizations

Localization	Resources
Base	0 Files Localized
English — Development Localization	0 Files Localized

Use Base Internationalization

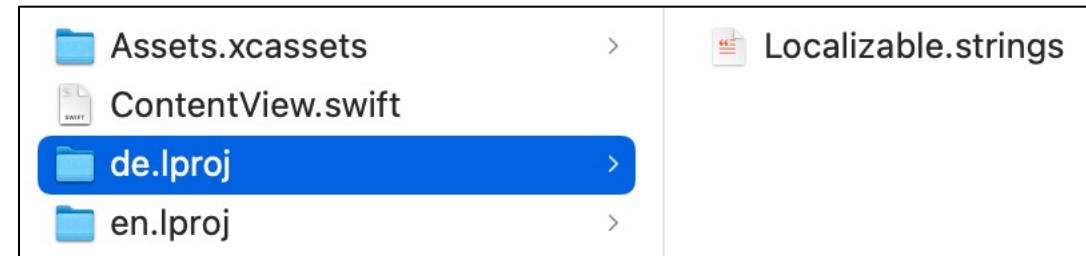
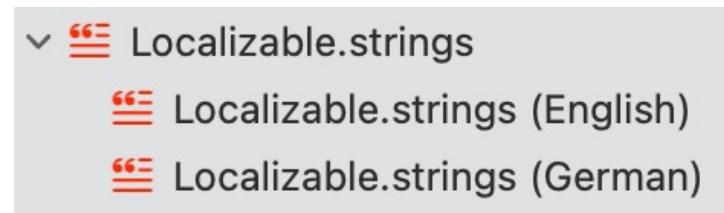
# Localizable.strings

- Dateiformat, um sprachabhängige Strings im Projekt zu verwalten
- 1 Datei pro Sprache
  - Localizable.strings
    - Localizable.strings (English)
    - Localizable.strings (German)
- Format: Key-value Pairs
  - Key = String, der im Code für Lookup verwendet wird
  - Value = übersetzter Text in jeweiliger Sprache
  - Semikolon nicht vergessen!

```
"welcome_message" = "Willkommen";
```

# Localizable.strings

- Auf dem Filesystem: 1 Folder pro lokalisierte Sprache



# NSLocalizedString

- Globale Funktion, um String-Lookup zu machen in .strings-Dateien
  - Per Default wird Localizable.strings verwendet (tableName leer lassen)
- UIKit:

```
let label = UILabel()  
label.text = NSLocalizedString("welcome_message", comment: "")
```

```
func NSLocalizedString(  
    _ key: String,  
    tableName: String? = nil,  
    bundle: Bundle = Bundle.main,  
    value: String = "",  
    comment: String  
) -> String
```

# SwiftUI

- Ebenfalls möglich mit NSLocalizedString:

```
Text(NSLocalizedString("welcome_message", comment: ""))
```

- Noch besser / einfacher:

```
Text("welcome_message")
```

- Warum funktioniert das?

# SwiftUI

- Text hat verschiedene Initializers. Wenn ein String-literal übergeben wird, interpretiert der Compiler dies als LocalizedStringKey

## Creating a text view from a string

`init(LocalizedStringKey, tableName: String?, bundle: Bundle?, comment: StaticString?)`

Creates a text view that displays localized content identified by a key.

`init(LocalizedStringResource)`

Creates a text view that displays a localized string resource.

`init<S>(S)`

Creates a text view that displays a stored string without localization.

`init(verbatim: String)`

Creates a text view that displays a string literal without localization.

# SwiftUI – Text

```
struct ContentView: View {  
    let text = "welcome_message"  
    let text2: LocalizedStringKey = "welcome_message"  
  
    var body: some View {  
        VStack {  
            Text("welcome_message") // ?  
            Text(verbatim: "welcome_message") // ?  
            Text(text) // ?  
            Text(text2) // ?  
        }  
    }  
}
```

# SwiftUI – Text

```
struct ContentView: View {  
    let text = "welcome_message"  
    let text2: LocalizedStringKey = "welcome_message"  
  
    var body: some View {  
        VStack {  
            Text("welcome_message") // ?  
            Text(verbatim: "welcome_message") // ?  
            Text(text) // ?  
            Text(text2) // ?  
        }  
    }  
}
```



Welcome  
welcome\_message  
welcome\_message  
Welcome

# Lokalisierung – Demo

# Widgets

# iOS Widgets

- Apple hat's (nicht) erfunden ;-)
- Seit iOS 14 (2020): Homescreen Widgets
  - Ebenfalls auf Today View (wer braucht den...?)
- Seit iOS 16 (2022): Lockscreen Widgets



# iOS Widgets – Merkmale

- Kleiner Ausschnitt einer App auf dem Homescreen
- Ist immer Teil einer App (können nicht ohne zugehörige App installiert werden)
- iOS erlaubt Widget-Stapel (Widgets wechseln manuell oder intelligent)
- Konfiguration einzelner Widgets möglich
- Verschiedene Größen: Small, Medium, Large, Extra-Large (iPad)
- Limitierte Interaktion möglich (keine Animationen etc.)
- → HIG zu Widgets: <https://developer.apple.com/design/human-interface-guidelines/components/system-experiences/widgets/>

# iOS Widgets – Konfiguration



# iOS Widgets – Dokumentation

- WidgetKit: Entsprechendes Framework  
<https://developer.apple.com/documentation/widgetkit/>
- Creating a Widget Extension  
<https://developer.apple.com/documentation/widgetkit/creating-a-widget-extension>

# iOS Widgets: Timeline

- Widgets haben sehr beschränkte Reload-Möglichkeiten
  - Abhängig davon, wie oft das Widget angeschaut/benutzt wird
  - Grundsätzlich: Keine Garantien, sondern Apple Magic...
  - Größenordnung für oft benutztes Widget: Reload alle 30 min sollte möglich sein

A widget's budget applies to a 24-hour period. WidgetKit tunes the 24-hour window to the user's daily usage pattern, which means the daily budget doesn't necessarily reset at exactly midnight. For a widget the user frequently views, a daily budget typically includes from 40 to 70 refreshes. This rate roughly translates to widget reloads every 15 to 60 minutes, but it's common for these intervals to vary due to the many factors involved.

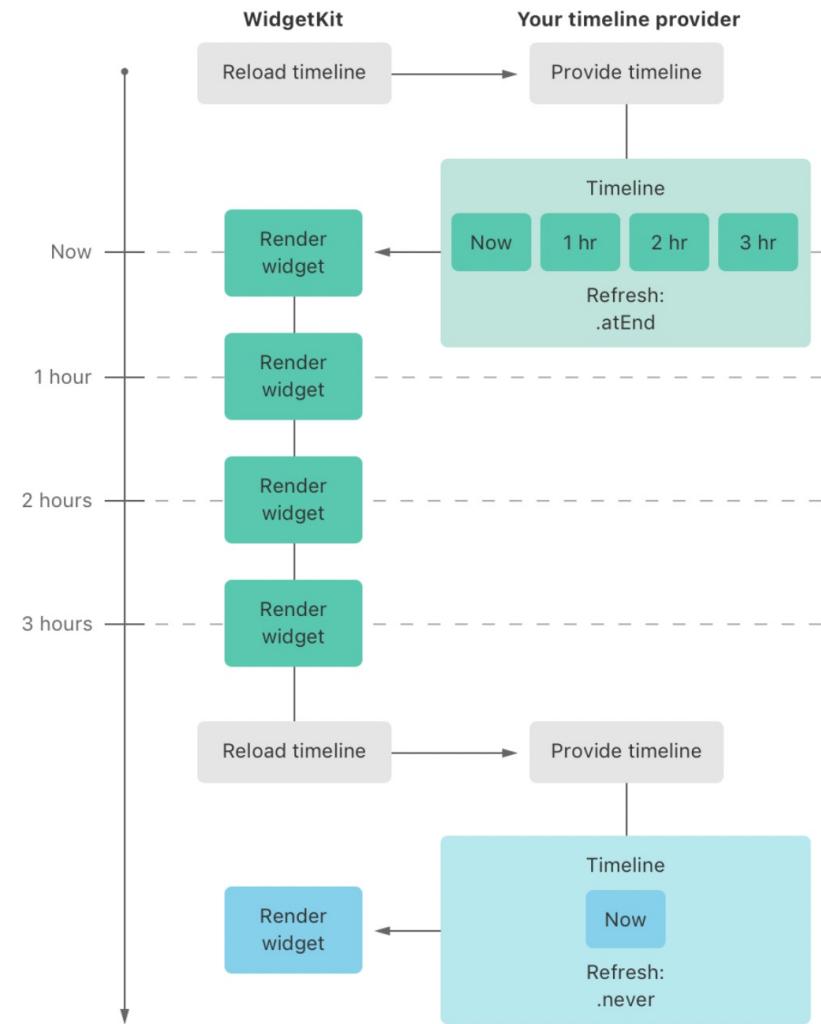
## Note

The system takes a few days to learn the user's behavior. During this learning period, your widget may receive more reloads than normal.

# iOS Widgets: Timeline

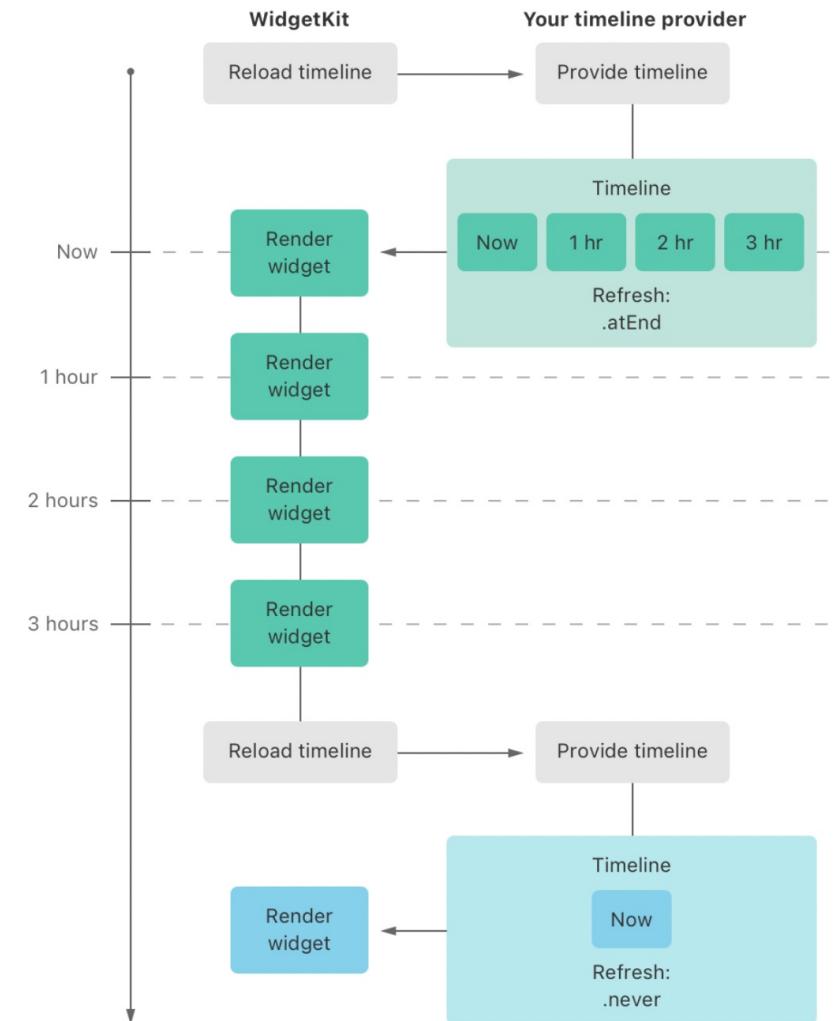
- Das System fragt nach einer Timeline
- Mittels **TimelineProvider** gibt man **TimelineEntries** an mit allen Infos, um das Widget zu rendern
  - Refresh-Policy: `.atEnd` / `.after(Date)` / `.never`
- Nach Ablauf fragt das System erneut
- Man kann Timeline-Reload auch manuell triggern (Achtung: keine Garantie!)

```
WidgetCenter.shared.reloadAllTimelines()
```



# iOS Widgets: Timeline Entries

- 2 Properties:
  - `date`: Zeitpunkt, wann das Widget gerendert werden soll
  - `relevance`: Momentane Relevanz des Inhalts für den User (optional)
- Relevanz wird benutzt, damit iOS lernt, wann Widget in Smart Stacks gezeigt werden soll
  - Bsp.: Wetter-Widget könnte Relevanz höher setzen, wenn schlechtes Wetter oder aktive Wetter-Warnungen angezeigt werden.



# Demo: Widget

# Exkurs: Sync von Daten mit App

- Widget ist eine “Extension”. Extensions laufen unabhängig von der App (eigener Prozess)
- Konsequenz: Daten-Austausch zwischen App und Widget ist nicht ganz trivial
- Einigermassen einfache Lösung: Widget & App in gemeinsamer App Group → können gemeinsame **UserDefaults** verwenden:

```
let userDefaults = UserDefaults(suiteName: "app.group.identifier")!
```

- App Group einrichten ist etwas mühsam... Für Gruppenprojekt aber sicher möglich!

# App-Installation & Provisioning

# App-Provisioning

- Grundsatz: Apple möchte möglichst viel Kontrolle darüber, welche Apps auf iOS-Geräten laufen
  - Dieser Ansatz hat offensichtlich Vor- und Nachteile
- Mit Xcode und Simulator kann man ohne Barriere entwickeln (sofern man einen Mac hat...), aber um App auf einem physischen Gerät laufen zu lassen, ist Developer-Account nötig (mittlerweile gratis)

# App-Provisioning

- **Certificates**: Zertifikat, das auf dem Mac in der Keychain gespeichert wird und benutzt wird, um die App zu signieren
- **Identifiers**: App IDs, die eine App eindeutig identifizieren. Damit sind gewisse Capabilities/App Services verbunden
- **Devices**: Geräte, auf denen man Apps installieren will, müssen ebenfalls registriert werden
- **Profiles**: Provisioning Profiles verknüpfen all diese Infos (Certificate, App ID, Device IDs) und werden mit der App auf dem Gerät installiert. Nur wenn alles korrekt ist, kann die App installiert und geöffnet werden.

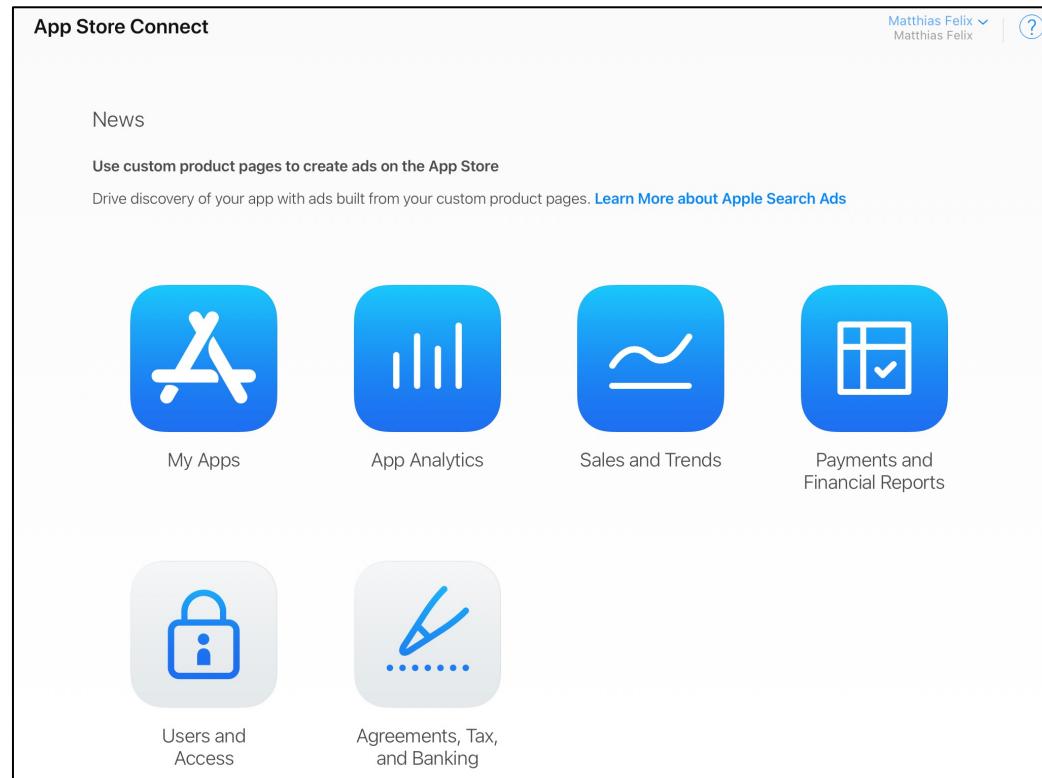
# App-Provisioning

- Prozess war früher sehr mühsam, heute macht Xcode im Idealfall (fast) alles selber
- Man muss für die meisten Schritte nicht mehr ins Developer Portal abspringen. Xcode kann u.a.
  - Zertifikate erstellen und in der Keychain speichern
  - App IDs erstellen
  - Geräte registrieren
  - Neue Capabilities hinzufügen
  - etc...

# App-Veröffentlichung

# App-Veröffentlichung

- Apps werden grundsätzlich über den App Store veröffentlicht
- Dafür nötig: Kostenpflichtiger Developer-Account (ca. 100 CHF/Jahr)
- Verwaltung dieses Prozesses passiert in AppStore Connect:  
<https://appstoreconnect.apple.com>



# AppStore Connect – Demo

# App-Veröffentlichung – Prozess

- App muss für Veröffentlichung mit Distribution Certificate signiert werden
- Release-Konfiguration im Xcode (Build wird nicht für Debugging, sondern für Release optimiert)
- Hochladen zu Apple (via Xcode möglich)
- Automatisiertes Processing
- Je nach Distribution Channel manuelles Review

# App-Veröffentlichung – Testing

- Apple stellt für Beta-Testing eine Plattform zur Verfügung: TestFlight
- Integriert in AppStore Connect
- Erlaubt es, Builds vor der Veröffentlichung an interne oder externe Tester zu verteilen
  - Interne Tester: Auf AppStore Connect registrierte Benutzer, z.B. Teammitglieder etc. (höchstens 100)
  - Externe Tester: Können via Emailadresse oder public link eingeladen werden (bis zu 10'000) → Achtung: App muss durch App Review!
- Installation auf Geräten über TestFlight-App



# App-Veröffentlichung – App Review

- Jede neue App bzw. jedes Update wird von Apple geprüft
- Grundsätzliche Funktionalität; grobe Verstöße aufdecken
- Human Interface Guidelines
- In App Purchases werden immer sehr genau geprüft → Schutz vor Betrug
- App Review Prozess: Sowohl automatisiert als auch manuell
  - Bsp. automatisch: Covid-Strings in einer App
  - Bsp. manuell: Demo ;-)
- Meistens Antwort innerhalb weniger Tage, manchmal sogar Stunden
  - Leider teilweise viel länger, gibt keine Garantie