No-Code/Low-Code: Anwendbarkeit im Vergleich zur klassischen Entwicklung

Bachelorarbeit

eingereicht von: **Manuel Mischak**
Matrikelnummer: 51905480

im Fachhochschul-Bachelorstudiengang Wirtschaftsinformatik (0470)
der Ferdinand Porsche FernFH

zur Erlangung des Akademischen Grades eines
Bachelor of Arts in Business

Betreuung und Beurteilung: Dipl.-Ing. Dr. Werner Toplak

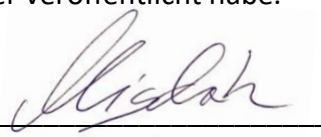
Wiener Neustadt, Mai 2022

Ehrenwörtliche Erklärung

Ich versichere hiermit,

1. dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Inhalte, die direkt oder indirekt aus fremden Quellen entnommen sind, sind durch entsprechende Quellenangaben gekennzeichnet.
2. dass ich diese Bachelorarbeit bisher weder im Inland noch im Ausland in irgendeiner Form als Prüfungsarbeit zur Beurteilung vorgelegt oder veröffentlicht habe.

Wien, 17.05.2022



Unterschrift

Die Kapitel 2 bis 3 der vorliegenden Bachelorarbeit wurden im Rahmen der Lehrveranstaltung „Bachelor Seminar 1“ eingereicht und am 07.02.2022 als Bachelorarbeit 1 angenommen.

Abstract

When developing new software, the question is always asked which method or which tool is best suited to implement the given requirements. Now a days, NoCode and LowCode tools are more and more becoming a good option or alternative, so that the question is getting even more complex. The aim of this bachelor thesis is to find out under which criteria and circumstances NoCode tools are better suited for the creation of IT applications and when it is better to fall back on traditional, code-based development.

Basically, this work is divided into two parts, starting off with a foundation chapter, which is giving context and introduces the basics. Followed by a second chapter that shows how NoCode and LowCode Tools are used in real life business cases. The second part focused on answering the research question, by using and applying Prototyping / proof-of-concept as a methodical approach.

Kurzfassung

Steht man vor der Erstellung einer neuen Software kommt immer wieder die Frage auf, mit welchen Tool bzw. welchen Vorgehensweise, die gewünschten Anforderungen bestmöglich implementiert werden können. LowCode und NoCode Tools spielen immer mehr eine alternative Rolle und sind für so manchen Anwendungsfall sehr gut geeignet. Diese Bachelorarbeit soll herausfinden, wann bzw. unter welchen Kriterien NoCode Tools für die Entwicklung neuer Applikationen besser geeignet ist, und wann man eher auf eine klassische codebasierte Entwicklung zurückgreifen sollte.

Grundsätzlich gliedert sich diese Arbeit in zwei Teile, beginnend mit einem Grundlagenkapitel, welches den Kontext liefert und in die Grundlagen einführt. Danach folgt ein zweites Kapitel, das zeigt, wie NoCode und LowCode Tools in realen Geschäftsfällen eingesetzt werden. Der zweite Teil konzentriert sich auf die Beantwortung der Forschungsfrage, indem Prototyping / Proof-of-Concept als methodischer Ansatz verwendet und angewendet wird.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Bei der Entwicklung einer neuen Software steht man immer vor der Frage, welche Methode, oder welches Tool am besten geeignet ist, um die gegebenen Anforderungen umzusetzen. Hierbei ergibt sich häufig die Problematik, vorab schwer abzuschätzen, ob das gewählte Tool Set für die Aufgabenstellung optimal gewählt wurde. Durch den ständigen technischen Fortschritt und der steigenden Wichtigkeit von digitaler Transformation in Unternehmen ergeben sich immer neue Methoden und Möglichkeiten, welche es einfacher, schneller oder performanter machen Software zu erstellen. Im Laufe dieses digitalen Wandels sind in den letzten Jahren immer wieder neue disruptive Technologien, wie Cloud-Services versstärkt aufgetreten, welche für große Schritte in der Art wie wir Applikationen nutzten oder auch entwickeln gesorgt haben. Selten hat man auch die Zeit verschiedene Ansätze auf die Machbarkeit der Anforderungen zu überprüfen.

Da nun auch NoCode/LowCode Tools immer öfters eine geeignete Option darstellen und stetig verbessert werden, erweitert sich die Auswahlmöglichkeit an Tools und die zuvor angesprochene Fragestellung erschwert sich weiter. „Ab dem Jahr 2024 sollen laut Gartner 65% aller App-Development Aktivitäten von NoCode bzw. LowCode Tools ausgehen.“[1] Was genau NoCode oder LowCode Tools sind und wie sie verwendet werden können, wird in den nachfolgenden Kapiteln ausführlich beschrieben.

Sehr häufig werden neue Möglichkeiten und Technologien nicht in Betracht gezogen, oder man entscheidet sich im Endeffekt für eine klassische Entwicklung, in dem man Softwareentwickler nutzt, um die Applikation händisch zu programmieren. Damit ist gemeint, dass neuer Code geschrieben werden muss, was natürlich einiges an Zeit und Fachpersonal benötigt. Gründe für das Umgehen von neuen Tools wie NoCode oder LowCode sind zum einen das Fehlende Fachwissen zu diesen Anwendungen, die Angst von einem Anbieter abhängig zu sein, oder auch die Unsicherheit, ob die Anforderungen mit einem NoCode/LowCode Tool umgesetzt werden können. [2] Auf der anderen Seite gibt es aber auch viele Applikationen welche rein über NoCode/LowCode Tools erstellt wurden, wo häufig die Steigerung der digitalen Innovation oder die Reaktionsfähigkeit auf das Geschäftsumfeld für die schlussendliche Endscheidung ausschlaggebend war. [3]

Wie man also sieht, gibt es unterschiedliche Kriterien und Gründe, warum auf NoCode/LowCode gesetzt wird und warum nicht.

1.1. Ziel der Arbeit

Ziel dieser Bachelorarbeit soll es sein, herauszufinden unter welchen Kriterien NoCode Tools für die Erstellung von eigenen IT-Applikationen besser geeignet sind und wann eher zu einer traditionellen, code-basierten Entwicklung zurückgegriffen werden soll.

[1] Harvard Business Review, „Empower Your Team with No-Code/Low-Code Software Applications“.
[2] OutSystems, „The State of Application Development“.
[3] OutSystems.

Da der Begriff IT-Applikationen einen großen Bereich abdeckt und man unzählige Anwendungsfälle betrachten kann, wird im genaueren auf die Entwicklung einer iOS App eingegangen, um hier das Spektrum einzuschränken. Es soll somit am Ende dieser Arbeit ersichtlich sein welche Anforderungskriterien bei der Erstellung, sprich Entwicklung einer Applikation, genauer gesagt einer iOS App, für die Wahl eines NoCode/LowCode Tools als Plattform der Umsetzung spricht und was für einen traditionellen Ansatz spricht.

1.1.1. Forschungsfrage

Unter welchen Kriterien ist die Entwicklung einer iOS App mit NoCode Tools besser geeignet als die Entwicklung mit einer code-basierten Herangehensweise über Xcode?

1.1.2. Hypothese

Bei der Entwicklung einer iOS App über NoCode ergibt sich im Wesentlichen ein zeitlicher Vorteil. Bei einem zeitlichen Kriterium ist somit die NoCode Variante besser geeignet. Gegenüber der programmierten Variante ergibt sich aber der Nachteil, dass man vor allem in der visuellen Gestaltung eingeschränkter ist. So ist bei dem Kriterium, welches genaue Designvorgaben beinhaltet, die programmierte Variante besser geeignet.

1.2. Aufbau und methodisches Vorgehen

Im grundlegenden ist diese Arbeit in zwei Teile geteilt, im nachfolgenden Kapitel wird auf Grundlagen der behandelten Thematik eingegangen und unter anderem der aktuelle Stand der Forschung dargestellt. Beginnend mit einem kurzen Rückblick über die Entstehung und Vergangenheit von NoCode Tools, soll ein erster Überblick gegeben werden. Da man auch häufig die Begriffe NoCode und LowCode zusammen hört, wird auf die konkreten Unterschiede der beiden Ausdrücke eingegangen. Danach wird genauer erläutert, wie NoCode oder LowCode Tools funktionieren, sprich was bei der Erstellung einer Applikation im Hintergrund passiert und wie es möglich ist ohne Code zu schreiben eine funktionstüchtige Lösung für verschiedenste Anwendungsfälle zu entwickeln. Die zuvor angesprochene klassische code-basierten Herangehensweise wird im Kapitel Grundlagen ebenfalls behandelt, da auch diese Variante im methodischen Vorgehen verwendet wird. Im Detail wird auf die aktuellen Standards in der iOS App Entwicklung eingegangen und welche Plattformen hierbei notwendig sind.

Im dritten Kapitel wird geschildert wie sich NoCode/LowCode Tools aktuell in realen Anwendungen verhält. Es werden Punkte wie Adaption und Zufriedenheit von diesen Tools betrachtet und welches Feedback von NoCode/LowCode Anwender gegeben wird. Auch wird gezeigt, wie disruptiv diese Plattformen in der realen Wirtschaft schon sind, bzw. andeuteten welchen disruptiven Effekten sie noch haben können. Des Weiteren wird in diesem Kapitel auf Use & Business Cases, sprich Anwendungsfälle, geschaut und in welcher Weise hierzu NoCode/LowCode Tools ihren Beitrag leisten können.

Um die Forschungsfrage zu beantworten und die Hypothese zu bewerten wird in einem zweiten Teil Prototyping bzw. Proof-of-Concept als methodischer Ansatz verwendet und angewandt. Dieser Ansatz ist im Wesentlichen so aufgebaut, dass von einer gemeinsamen Ausgangslage aus sprich, mit denselben Anforderungen, zwei Applikationen (iOS Apps) entwickelt wird.

Diese Anforderungen beziehen sich zum einen auf funktionale Punkte, sowie zum anderen auf die grafischen und optischen Eigenschaften, die das Endprodukt aufweisen soll. Um den Vergleich von NoCode und klassischer Entwicklung durchführen zu können, wird eine der beiden Apps rein mit Hilfe von NoCode Tools erstellt wird und eine zweite App rein über die klassische Programmierung mit den dazu bereitgestellten Tools, welche in Kapitel 2 genau beschrieben werden.

2. Grundlagen

In diesem Kapitel wird auf die Grundlagen von NoCode/LowCode Tools und deren Hauptbestandteile eingegangen, sowie auf die aktuellen Standards in der iOS App Entwicklung. Hier wird auch die Technische Funktionsweise beschrieben und was hinter diesen Tools steht.

Was bedeutet aber jetzt der Begriff NoCode bzw. LowCode Tools?
Kurz gesagt sind damit Entwicklungsumgebungen oder Plattformen gemeint, die es einem ermöglichen neue Software und Anwendungen über eine grafische Oberfläche zu erstellen und dabei ohne Code, also ohne Anwendung einer Programmiersprache auskommen. Im Gegensatz zur konventionellen Softwareentwicklung, wo erfahrene Programmierer viele Zeilen von Code schreiben, um Features und Funktionalitäten ihrer Computer-Programme zu erstellen, bieten NoCode Umgebungen die Möglichkeit dieselben Funktionalitäten über visuelle wiederverwendbare Komponenten herzustellen, ohne selbst Erfahrung in der klassischen Softwareentwicklung zu haben. Es bietet so zu sagen eine niedrige Eintrittshürde für Personen, die ihre eigene Software entwickeln möchten, aber vielleicht noch aufgrund einer technischen Barriere der Zugang zu diesem doch komplexen Thema fehlte.

Grundsätzlich können verschiedene Arten von Anwendungen mit Hilfe von NoCode/LowCode Tools erstellt werden. Maßgeblich dabei sind aber Mobile Applikationen und Webapplikationen, welche neben Integrationsanwendungen und sonstigen Applikationen mit Abstand am häufigsten erstellt bzw. entwickelt werden.[4]

Eine häufige Zielgruppe dieser Plattformen, die auch oft in Unternehmen zu finden ist, sind die sogenannten „Citizen Developers“, was ins Deutsche übersetzt so viel wie "ziviler Entwickler" oder "Fachbereichsentwickler" bedeutet. Sie sind oft technisch versierte Mitarbeiter ohne Programmier- oder IT-Kenntnisse, die mit Hilfe dieser unterstützenden Tools selbstständig IT-Anwendungen erstellen können. Durch den Einsatz dieser speziellen Fachbereichsentwickler verkürzt sich die Umsetzungszeit von der eigentlichen Idee bis hin zur fertigen Anwendung. Da Citizen Developer meist vom Fach sind und sich mit der Thematik auskennen, sind sie mit den Problemen, Aufgaben, Zielen und den Prozessen ihrer eigenen Abteilung vertraut. Was auch im Hinblick auf die digitale Transformation eins Unternehmens sehr entscheidend sein kann. Natürlich besteht auch hierbei ein gewisser Bedarf an fachkundigen IT-Personal, denn es muss eine Umgebung geschaffen werden, in der Citizen Developer frei umsetzen bzw. „entwickeln“ können, sei es in Form einer detaillierten Einführung oder Integration in die Gesamtarchitektur.[5]

2.1. Entstehung und Vergangenheit von NoCode

In diesem Abschnitt wird rückblickend auf die Entstehung von NoCode/LowCode Plattformen eingegangen, sowie die ersten Ansätze, mit denen das Modell NoCode für den Endbenutzer möglich war.

[4] Luo u. a., „Characteristics and Challenges of Low-Code Development“.
[5] Khorram, Mottu, und Sunyé, „Challenges & Opportunities in Low-Code Testing“.

Die Grundidee von NoCode ist nicht neu. Die ersten Anwendungen, die so ein Prinzip verfolgten, waren eigentlich schon die ersten Tabellenbearbeitungsprogramme. Genauer gesagt, war das erste „NoCode"-Programm Microsoft Excel, welches diesen Ansatz schon mitgeführt hat. Mit diesen Anwendungen könnte man ohne Programmierkenntnisse komplexe Berechnungen durchführen und diese „Programmierung" dann wiederverwenden oder auch an andere weitere geben. Mit Tools wie Excel, Access, oder Google Sheets konnten Business User (nicht IT-affine Benutzer) programmier-ähnliche Aufgaben durchführen.

Anfang der 2000er kamen dann neue, schon eher an den heutigen Ansatz von NoCode verwandte, Plattformen auf den Markt. Als das Internet und die Online-Welt einen großen Boom erlebten, war es für Viele wichtig sich auch in Form einer Website im Internet zu repräsentieren. Ein Tool, das so einige verholfen hat, eine Website zu erstellen und dass ganz, ohne programmieren zu müssen, bzw. zu können, war WordPress. Den meisten ist WordPress auch heute noch ein Begriff, denn mit einem Marktanteil von immer noch 65.1% (Stand Okt.2021[6]) im Bereich der Online Content Management Lösungen. Eine weitere nennenswerte Plattform ist Shopify. Mit Shopify ist es einfach möglich Online-Shops zu erstellen und diese auch zu verwalten. Auch hier wird der Ansatz und die Idee verfolgt, den Benutzern die Möglichkeit zu geben, einen Shop zu erstellen, ohne eine Zeile Code zu schreiben. Das geschieht meist anhand von vorgefertigten Templates und Designs.

Gesamtheitlich für alle NoCode/LowCode Tools, sowie die zuvor genannten Anwendung, kann man sagen, dass sie aus der Idee heraus entstanden sind Nutzer mit Hilfe einer intuitiven grafischen Oberfläche eine Möglichkeit zu geben, ihre Pläne und Visionen selbst umzusetzen.

2.2. **Unterschied LowCode/NoCode**

Im Laufe der Arbeit wurden die Begriffe LowCode und NoCode oft im gemeinsamen Kontext verwendet. In diesem Abschnitt wird darauf eingegangen, ob zwischen den beiden Fachbegriffen Unterschiede liegen und wie diese Unterschiede ausschauen. LowCode und NoCode Anwendungen bieten im grundlegenden die gleichen fundamentalen Ansätze. Sie sprechen aber mit ihren Features und Anwendungsmöglichkeiten immer unterschiedliche Personengruppen an. Es kann aber auch beispielsweise innerhalb der NoCode Welt Anwendungen, bzw. Produkte geben, die für eine ganz bestimmte Zielgruppe ausgelegt ist. Ein theoretisches Beispiel könnte ein Tool sein, mit dem man Chatbots erstellen und auf verschiedenste Arten einsetzten kann, welches vielleicht eher für Personen geeignet ist, die im Kundenmanagement tätig sind und Know-how von dieser Branche mitbringen.

LowCode spricht zum einen eher IT-erfahrene Personen, die vielleicht schon Code schreiben können und mit den Grund Modellen der Softwareentwicklung vertraut sind. Es kann daher vorkommen, dass in LowCode Anwendungen die Möglichkeit geboten, wird an gewissen Stellen selbst Code zu schreiben und das Endprodukt mit den eigenen Programmierkenntnissen noch weiter anzupassen oder komplexere Aufgaben zu lösen. Es ist aber in der Regel so, dass man nach kurzer Einarbeitungszeit und überschaubaren Aufwand auch als Person ohne

[6] https://w3techs.com/ zugegriffen am 11.11.2021

Programmierkenntnissen mit komplexen Aufgaben zurechtkommt und weitere technische Anpassung selbst vornehmen kann.[7]

NoCode Anwendungen dagegen zielt auf nicht technische Benutzer, welche die verschiedensten Anforderungen abdecken möchten und dabei auf einfache und intuitive Benutzeroberfläche zurückgreifen können. Hier spricht man klar von der zuvor erwähnten Zielgruppe „Citizen Developer“, die mit NoCode Tools wesentlich schneller und einfacher umgehen können als mit den LowCode Tools.[8]

Die Anwendungsfälle der beiden Tools können sich auch in gewisser Weise unterschieden. Ein typischer Anwendungsbereich für LowCode Tools ist zum Beispiel die Prozessautomatisierung, vor allem wenn zwischen mehreren Systemen kommuniziert wird, weil hierbei häufig ein technisches Verständnis für den Datentransfer von Nöten ist. NoCode Tools sind eher in Bereichen aufzufinden, wo ein selbstständiges, meist anwenderbezogenes und mit grafischer Bedienung versehenes, Endprodukt das Ziel ist. Wobei aber die Linie zwischen den beiden Methoden nicht immer einfach zu ziehen und eher fließend ist, so ist eigentlich NoCode Teil eines jeden LowCode Tools, da man bei LowCode ein fertiges Produkt auch nur mit den Graphischen Komponenten vollständig erstellen kann.

2.3. **Wie funktionieren NoCode Tools?**

In diesem Kapitel wird der technische Hintergrund von NoCode und LowCode Tools betrachtet und darauf eingegangen wie solche Plattformen hinter dem Blickwinkel von den Endbenutzern funktioniert.

Grundsätzlich ist es bei allen NoCode/LowCode Plattformen so, dass die Modellierung der Applikation über eine graphische Oberfläche durchgeführt wird. Der englische Fachbegriff dafür ist GUI, damit gemeint ist ein „Graphical User Interface“ oder auch nur User Interface, kurz UI genannt. Über dieses User Interface kann der Benutzer verschiedenste Komponenten mit unterschiedlichsten Funktionalitäten auf seine Arbeitsfläche ziehen was zumeist mittels Drag & Drop funktioniert. Dieser Arbeitsbereich soll in gewisser Weise das Endprodukt darstellen und soll zeigen wie die App oder Website, in den meisten Fällen, ausschauen wird, bzw. zeigt dieser Bereich auch den Datenfluss und Verknüpfungen von verschiedenen Seiten der Applikation an.

Am besten lasst sich das Ganze an einem Beispiel, in diesem Fall von dem NoCode Tool Adalo, zeigen. In Abbildung 1 sieht man auf der rechten Seite des Bildschirmes die einzelnen Komponenten, für unterschiedliche Aktionen, sieht und in der Mitte den zuvor angesprochenen Bereich:

[7] Mary K., „What Are Low-Code and No-Code Development Platforms?“
[8] Khorram, Mottu, und Sunyé, „Challenges & Opportunities in Low-Code Testing“.

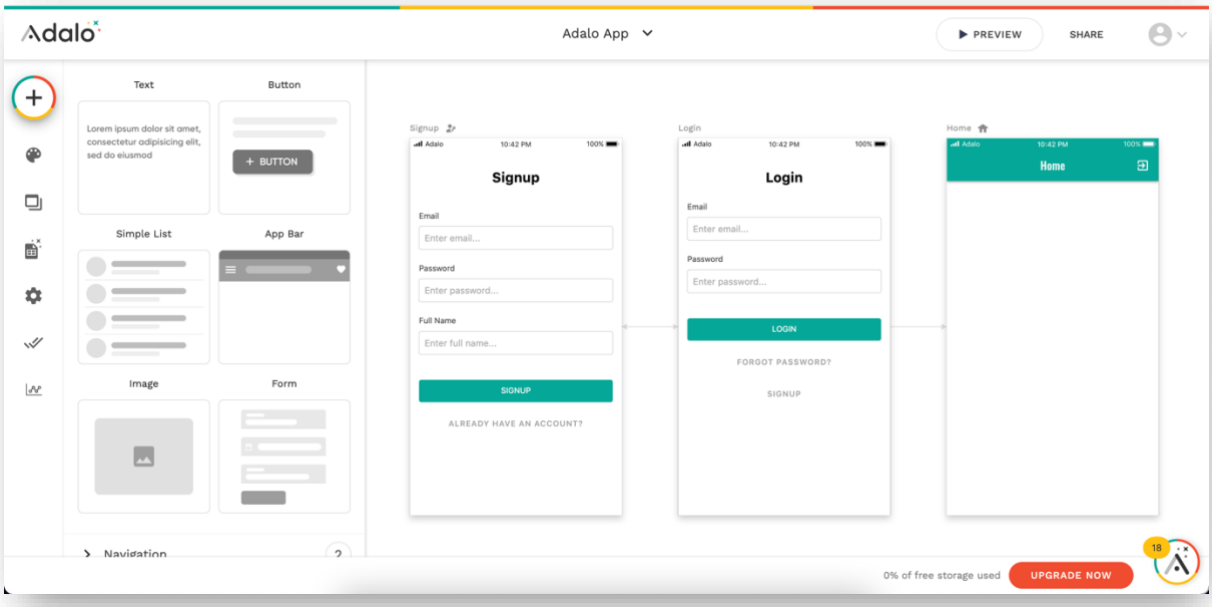


Abbildung 1: User Interface NoCode Tool[9]

Hinter jeder Komponente, hinter jedem Baustein, der auf dem Bildschirm platziert werden kann, liegt auch ein Code-Block. Dieser Code-Block ist nichts anderes als ein wiederverwendbares Stück Code, sprich einige zusammengehörige Zeilen Code die beliebig oft von Benutzer eingesetzt werden können, um das gewünschte Verhalten zu erreichen. Wenn man sich nun dieses Konzept genauer überlegt, kommt man recht schnell zur Tatsache, dass für eine NoCode Anwendung, welche es den Benutzer ermöglichen soll, eine Applikation zu erstellen, ohne programmieren zu müssen, allerdings sehr viel Programmieraufwand an sich notwendig ist. Allerdings ist nicht nur das Programmieren der Komponenten selbst aufwändig, sondern auch die Einbindung vom User in die bestehende App, was meistens über Drag & Drop passiert. Denn es ist fast in allen NoCode Tools der Fall, dass man Komponenten an jeder beliebigen Stelle platzieren kann und nicht an vorgefertigte Lücken, wo schlussendlich auch der Code platziert wird.

Um sich unter den Komponenten etwas genaueres vorstellen zu können und welche Funktionalitäten damit abgedeckt werden können, hier eine kurze Einführung:

2.3.1. **Navigation**

Zum einen gibt es Bausteine, mit deren Hilfe man sich durch die App navigieren kann. Der Ersteller der Anwendung kann somit dem Enduser verschiedenste Möglichkeiten geben sich von einer Seite zur nächsten Hand zu haben. In vielen der großen bzw. erfolgreichen Apps ist es üblich dem User über eine Tab-Bar, sprich eine Leiste am unteren Rand des Bildschirms mit unterschiedlichen Tabs, navigieren zu lassen. Außerdem für die Navigation geeignete

[9] Adalo, „Adalo - Build Your Own No Code App“.

Elemente, sind Buttons. Mit Hilfe von Buttons kann der User genau zur gewünschten Stelle geleitet werden.

2.3.2. **Eingaben**

Schon beim Login einer App, muss der User sehr oft seine E-Mail-Adresse und das Passwort eingeben. Diese schreibbaren Textfelder sind nur ein Beispiel für Input-Elemente. Durch das Platzieren solcher Bausteine kann man das Eingeben von Daten oder eine Auswahl, die der User treffen soll, ermöglichen. Neben der klassischen Text-Eingabe gibt es unter anderem die Möglichkeit Bausteine wie Checkboxen, Datumsfelder, oder auch ein Bild, bzw. Dateien Upload als Eingabe zu verwenden. Zu dieser Kategorie kann man aber auch den Button dazuzählen, da durch das Klicken eines Buttons eine Auswahl getroffen werden kann und dadurch der User auch neue Daten erzeugen kann.

2.3.3. **Datendarstellung**

Um die eigengeben, oder bestehenden Informationen darzustellen, benötigt es Komponenten, die diese Aufgabe lösen können. Zur Visualisierung können Elemente wie Listen, Tabellen oder einfach nur Text verwendet werden. Bilder oder Videos sind vor allem in den Social Media Apps eine sehr verbreitete Art und Weise einer Datendarstellung.

Neben den Komponenten ist ein weiteres Grundkonzept einer jeden NoCode oder LowCode Plattform ist das Verwenden von „Triggers und Actions“. Triggers oder auch in manchen Anwendungen Events genannt, sind dazu da, um eine vom Benutzer ausgeführte Handlung zu erkennen und die richtige Aktion auszuführen. Auch hier gibt es wieder unterschiedlicher Events, die ein User auslösen kann. Am häufigsten benötigt werden Trigger, die von User durch das Interagieren mit einem Element im User Interface erkannt werden. Ohne diese Art von Events und den darauf resultierenden Aktionen würde auch keine Navigation in einer App funktionieren. Zum Beispiel wird der Marker einer Karte angeklickt oder ein Button betätigt, um auf eine weitere Seite zu springen. Für jedes definierte Event muss zuvor festgelegt werden, auf welches Element, oder welche Komponente es sich bezieht. Eine andere, etwas komplexere Art von Triggern können die sein, die sich auf den Kontext und die hinterlegenden Parameter beziehen. Diese Events sind nur indirekt mit den Eingaben und Handlungen des Users verbunden. Ein Beispiel in diesem Zusammenhang wäre ein erfolgreiches Login in eine Applikation. Nach dem Überprüfen der Login-daten auf ihre Richtigkeit, wird beispielsweise ein Parameter oder Status geändert und genau diese Statusänderung kann als Trigger genutzt werden, um eine oder mehrere Aktionen auszuführen. Anders gesagt, sind diese Art von Events auf eine Kondition bezogen, welche überprüft und erfüllt werden muss. [10]

Ausgelöste Events führen zur Ausführung einer oder mehrerer zugeordneter Funktionen. Bei der Erstellung einer App kann man somit definieren was nach dem Ausführen eines Triggers passieren soll. Eine solche Aktion kann beispielhaft so festgelegt werden, dass sich mit einem Klick vom User auf Button ‚A‘ Seite ‚B‘ öffnen soll. Klarerweise können auch hier neben den klassischen Navigationsaktionen auch viele andere Typen und Arten von ‚Actions‘ ausgeführt werden, die zum Beispiel im Hintergrund ablaufen, wie eine Datenabfrage an eine API, oder einen Zahlungsprozess starten. Durch das aneinanderreihen und definieren von Events und den

[10] Bubble.io, „Bubble Docs - Building Workflows“.

dazugehörigen Aktionen können Entwickler mit Hilfe von NoCode Tools Workflows, Prozesse oder Abläufe für ihre Anwendungen entwerfen und ohne Code zu schreiben in die Applikation implementieren.

Ein weiteres Hilfsmittel das praktisch jedes NoCode oder LowCode Tool zur Verfügung stellt sind sogenannte ‚Data-Connections', oder auch einfache Datenbanken an sich, wo die Daten der Applikation gespeichert und verarbeitet werden können. Data-Connections sind dagegen nicht für das Aufbewahren von Daten gedacht, sondern für den Transport von Daten und die Anbindung, bzw. Kommunikation mit externen Quellen, wie zum Beispiel einer API-Schnittstelle.

Eine Datenbank im Hintergrund zu haben ist in fast allen Fällen notwendig, wenn man eine neue Applikation baut, ob nun mit NoCode bzw. LowCode Tools erstellt oder programmiert. Informationen, wie zum Beispiel, Login Daten von den Usern, müssen schlussendlich irgendwo geordnet und von der App zugänglich gespeichert werden. Datenbanken von NoCode Plattformen sind aber einfacher aufgebaut und leichter zu bedienen als herkömmliche Datenbank-Anwendungen, da auch meistens ein niedriger Komplexitätsgrad notwendig ist und sich nur die Daten der einen App in ihr befinden.

In der nachfolgenden Abbildung sieht man wie in der NoCode Anwendung Adalo die Entität „Users" verwaltet werden kann und welche Attribute zu diesem Objekt in der Datenbank vorhanden sind, die befüllt und abgefragt werden können:

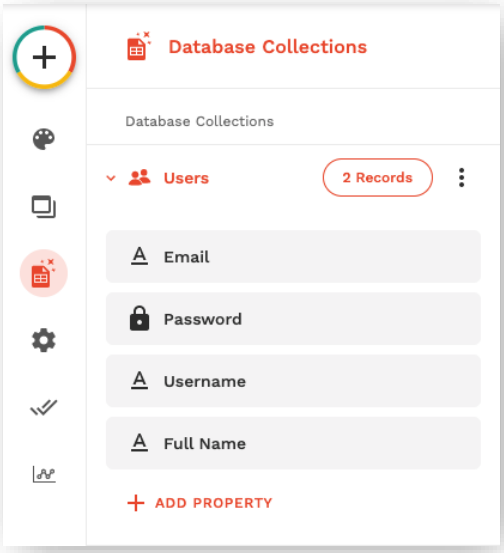


Abbildung 2: Users Entität - Adalo Datenbank[11]

Im Zuge der Bearbeitung dieser integrierten Datenbanken spielt nicht nur die Anlage neuer Entitäten oder die Attribute von diesen Objekten eine Rolle, sondern auch die Verbindungen, bzw. Relationships zwischen den einzelnen Entitäten. Die Modellierung der Daten und wie diese miteinander korrelieren ist bei, oder besser gesagt vor der Erstellung einer Anwendung

[11] Adalo, „Adalo - Build Your Own No Code App".

über NoCode nicht zu vernachlässigen und kann einer nicht technischen Person schon einiges an Kopfzerbrechen bereiten. Um so wichtiger, sich vorab ein Datenmodell passend zu den Möglichkeiten des gewählten NoCode Tools zu erstellen, wobei die dazugehörige Dokumentation hilfreich sein kann und genutzt werden sollte.

Da man in seiner NoCode Applikation häufiger eine komplexe Datenverarbeitung verwenden und auf eine externe Schnittstelle zugreifen möchte stellen fast alle Plattformen eine Möglichkeit zur Verfügung, die es erlaubt sogenannte Data-Connections anzulegen. Die Begrifflichkeiten unterscheiden sich von Tool zu Tool aber die Funktionalität, mit externe Datenquellen oder APIs zu kommunizieren, ist immer die gleiche. Diese Connections, sprich Verbindungen, werden einmalig konfiguriert und angelegt, damit sie danach wiederverwendet werden können und beispielsweise in bestimmten Aktionen integriert. Im Zuge dieser Konfiguration wird auch die notwendige Authentifizierung einmalig hinterlegt, damit eine Verbindung gewährleistet werden kann.

Durch den Einsatz und die Funktionalität von Data Connections wird es dem User möglich gemacht auch verschiedenste NoCode Tools miteinander zu Verbinden. Um genau das effizient und relativ einfach umzusetzen, gibt es eigene NoCode bzw. LowCode Tools, die genau für diese Systemübergreifenden Zusammenhänge und Workflows entwickelt wurden. Ein Beispielhaftes Produkt wäre „Airtable“, mit dem es möglich ist Anwendungen zu verbinden, ohne programmieren zu müssen. [12]

2.4. **Mobile App Entwicklung mit NoCode**

Da mobile Anwendungen in der heutigen Welt wichtiger denn je sind und wir uns im speziell mit der Entwicklung einer iOS App beschäftigen, soll in diesem Abschnitt auf die verschiedenen Arten von Mobilen App, sprich auf Unterschiede zwischen diesen Entwicklungsvarianten, eingehen. In der allgemeinen Mobile App Entwicklung kann zwischen „Native App“, „Web App“ und einer „Hybrid App“ Unterschieden werden, auf die im Folgenden eingegangen wird:

2.4.1. **Native App**

Bei einer Native App spricht man von einer Applikation die nur für ein spezifisches mobiles Betriebssystem, z.B. iOS oder Android, designt und entwickelt wurde. Der Zugang zur App, sowie der Zugang zu Updates oder neue Versionen, ist nur über den dazugehörigen App Store möglich, von dem die Applikation heruntergeladen und lokal am Endgerät installiert wird. Durch die lokale Installation und der Möglichkeit sich mit dem Internet zu verbinden, kann eine Native App online als auch offline genutzt werden. Möchte man die App verschiedene Betriebssysteme zur Verfügung stellen so muss für jede unterstützende Plattform eine eigene native App entwickelt werden. Aufgrund der Plattformabhängigkeit können die Entwicklungskosten und -zeiten recht hoch sein, da die Entwicklung einer App für jede Plattform Zeit in Anspruch nimmt und für jede Plattform unterschiedliche Fähigkeiten erforderlich sind.[13]

[12] Airtable, „Airtable Website“.
[13] Randleff, *Native App vs Web App*.

Ein besonderer Vorteil von Native Apps ist die direkte Integration und Kommunikation zu anderen installieren Apps und den Hardwarefunktionalitäten, wie zum Beispiel, das Verwenden der Kamera oder Zugriff zu GPS. Möglich gemacht wird das durch die Verwendung der Betriebssystem eigenen API (Application Programming Interface), welche von nativen Apps genutzt werden kann. [14]

Im nachfolgenden Kapitel 2.5 wird genauer darauf eingegangen wie so einen Native-App im Kontext des iOS Betriebssystems entwickelt wird und welche Tools und Programmiersprachen dabei zur Anwendung kommen.

2.4.2. **Web-App**

Alle modernen Smartphones besitzen mittlerweile einen leistungsfähigen Internetbrowser, der verschiedenste Webanwendungen auf das mobile Endgerät bringt. Dazu zählen auch Web-Apps. Web-Apps werden nicht lokal auf dem Gerät installiert, sondern über eine URL im Webbrowser (z. B. Google Chrome und Safari) ausgeführt. Diese Gegebenheit vereinfacht die Aktualisierung der Web-App, da die aktuelle Version direkt an alle Nutzer verteilt werden kann. Web-Apps werden grundsätzlich mit dem Programmiersprachen HTML, CSS und JavaScript entwickelt. So gesehen ist eine Web-App nichts anderen wie eine klassische Website, die im PC-Browser aufgerufen wird, nur eben für die mobile Bedienung optimiert. Da eine Web-App für einen mobilen Webbrowser entwickelt wird, ist sie nicht davon abhängig, auf welchem mobilen Betriebssystem sie läuft. Sie sind daher plattformunabhängig, was bedeutet, dass es ausreicht, nur eine Anwendung für alle Plattformen zu entwickeln. Die Kombination aus der Tatsache, dass nur eine App entwickelt werden muss, und der Einfachheit der Aktualisierung von Inhalten macht die Web-App-Entwicklung in der Regel zu einer schnellen und effizienten Alternative. Durch die vielen Entwicklungsplattformen, welche für die Webentwicklung zur Verfügung stehen, ist es auch einfach die Applikation zu debuggen und direkt im Browser zu testen. [15]

Viele NoCode Tools wie zum Beispiel *bubble.io* oder *Glide*[16] verflogenen diesen Ansatz, in dem sie den Usern die Möglichkeit eine Webanwendung für alle Plattformen, unteranderem auch für den Standard PC-Browser, zu erstellen. Dadurch haben sie die Flexibilität alle Umgebungen abzudecken, müssen aber in Bezug auf die Smartphone-Variante auf Funktionalitäten verzichten.

2.4.3. **Hybrid-App**

Ein weiterer spannender Ansatz ist der von Hybrid-Apps, denn durch diese Variante können die Vorteile der nativen Entwicklung und der Webentwicklung in gewisser Weise kombiniert werden. Bei diesem Vorgehen schreiben die Entwickler wesentliche Teile ihrer Anwendung in plattformübergreifenden Webtechnologien, während sie bei Bedarf weiterhin direkten Zugriff auf native APIs haben. Dieser native Teil der Anwendung nutzt die APIs des Betriebssystems, um eine eingebettete HTML-Rendering-Engine zu erstellen, die als Schnittstelle zwischen dem

[14] IBM Corporation Software Group., „Native, Web or Hybrid Mobile-App Development“.
[15] IBM Corporation Software Group.
[16] Glide.

Browser und den Geräte-APIs dient. Dank dieser Schnittstelle kann die hybride Anwendung alle Standard Funktionen moderner Smartphones voll ausschöpfen. Der native Teil der Anwendung kann unabhängig entwickelt werden, aber einige Lösungen auf dem Markt bieten diese Art von nativem Container als Teil ihres Produkts an, wodurch der Entwickler in die Lage versetzt wird, eine fortschrittliche Anwendung zu erstellen, die alle Funktionen des Geräts nutzt und dabei nichts anderes als Websprachen verwendet. Der Webteil der Hybrid-App kann entweder eine Webseite sein, die sich auf einem Server befindet, oder in den Anwendungscode gepackt und lokal auf dem Gerät gespeichert werden. Beide Ansätze haben Vorteile und Einschränkungen. HTML-Code, der auf einem Server gehostet wird, ermöglicht den Entwicklern kleinere Aktualisierungen an der Anwendung vorzunehmen, ohne den in einigen App-Stores erforderlichen Einreichungs- und Genehmigungsprozess zu durchlaufen. Leider ist bei diesem Ansatz die Offline-Verfügbarkeit nicht gegeben, da der Inhalt nicht zugänglich ist, wenn das Gerät nicht mit dem Internet verbunden ist. Andererseits kann die Integration des Webcodes in die Anwendung selbst die Leistung und Zugänglichkeit verbessern, lässt aber keine Fernaktualisierungen zu.

NoCode Tools die den Ansatz der Hybrid-App wählen, bieten dem Entwickler die Option den Interface teil für mehrere Betriebssysteme gleichermaßen zu erstellen und binden diesen dann beim Ausrollen der App in den nativen Container ein, welcher dann am Endgerät installiert wird. Die schon zuvor erwähnte Plattform Adalo stellt genau diese Möglichkeit zur Verfügung, Apps in Form von Hybrid-Apps im App Store zu veröffentlichen und eine uneingeschränkte User Experiance zu ermöglichen.

Vergleich der Unterschiedlichen Ansätze:
Um die verschiedenen Methoden zusammen zu fassen und diesen Vergleich auch visuell zu verdeutlichen, wird in **Error! Reference source not found.** der technische Aufbau dieser Varianten grafisch dargestellt.
s

Abbildung 3: Aufbau Native-, Web-, Hybrid App[17]

Wie zuvor beschrieben verwendet die Hybrid-App einen nativen Container, welcher wie die Native-App selbst, mit den Geräte spezifischen Funktionalitäten kommunizieren kann. Die

[17] IBM Corporation Software Group., „Native, Web or Hybrid Mobile-App Development“.

Web-App hingegen ist auf die Limitierungen des Browsers beschränkt. Außerdem in der Hybrid-App gut ersichtlich ist die Webanwendung, die vom nativen Container aufgerufen wird und plattformübergreifend verwendet werden kann.

Zusätzlich zeigt auch die nachfolgende Tabelle, dass der hybride Ansatz einen Mittelweg bietet, der in vielen Situationen das Beste aus beiden Welten, vor allem wenn die App auf mehreren Betriebssystemen verwendet werden soll, vereint. Werden NoCode Tools für die Erstellung verwendet so ist der erste Punkt „Entwicklungsprache“ irrelevant, da die Verwendung von Code nicht notwendig ist.

Tabelle 1: Vergleich Native-, Hybrid- und Web App[18]

Feature	Native app	Hybrid app	Web app
Development language	Native only	Native and web or web only	Web only
Code portability and optimization	None	High	High
Access device-specific features	High	Medium	Low
Leverage existing knowledge	Low	High	High
Advanced graphics	High	Medium	Medium
Upgrade flexibility	Low (Always by way of app stores)	Medium (Usually by way of app stores)	High
Installation experience	High (From app store)	High (From app store)	Medium (By way of mobile browser)

2.5. **Aktuelle Standards in der iOS App Entwicklung**

Da im zweiten Teil dieser Arbeit Prototyping als methodischer Ansatz verwendet wird und hierbei eine iOS App mit der klassischen codebasierten Entwicklung zum Vergleich mit der NoCode Variante erstellt wird, soll hier in diesem Kapitel auf die Grundlagen und aktuellen Standards der iOS Entwicklung eingegangen werden. Wie im Kapitel zuvor beschrieben, handelt es sich hierbei um eine Native-App.

Falls noch nicht bekannt, handelt es sich bei dem Begriff „iOS“ um das vom Unternehmen Apple entwickelte mobile Betriebssystem für das erstmalig 2007 veröffentlichte iPhone. Die darauf laufenden Applikationen, kurz Apps, können von Apple selbst zur Verfügung gestellt werden, oder durch Dritte über den App Store, die Vertriebsplattform für iPhone Applikation, zum Download angeboten werden. Die aktuelle Version des Betriebssystems iOS ist iOS 15, welches im Jahr 2021 veröffentlicht wurde.

Um diese Apps zu bauen, benötigt es wie bei jeder Softwareentwicklung eine Entwicklungsumgebung und eine Programmiersprache. Will man Produkte für Apple Produkte wie das iPhone, iPad, Apple Watch oder den Mac-Geräten erstellen, so muss man fast immer auf die von Apple selbst zur Verfügung gestellte Plattform Xcode zurückgreifen. Auch von Apple bereitgestellt wird die Programmiersprache, bzw. Programmiersprachen, welche benötigt werden, um eine iOS App zu erstellen.

[18] IBM Corporation Software Group.

2.5.1. **Entwicklungsplattform Xcode:**

Wollen wir hier nun genauer auf die Entwicklungsplattform Xcode blicken. Ein wesentlicher Punkt von Xcode ist, dass man es nur auf Apple Computern, wie zum Beispiel einen MacBook verwenden kann, und somit in gewisser Weise gezwungen ist, einen Mac zu verwenden, um Softwareprodukte für Apple Geräte zu entwickeln.

Abbildung 4: Xcode Symbol

Das erste Mal erschienen und von Apple veröffentlich wurde Xcode 2003.[19] Damals klarerweise nur für die Entwicklung von Software für die Mac-Endgeräte und das dazugehörige MacOS Betriebssystem, da das erste iPhone erst 2008 vorgestellt wurde. Bei Xcode spricht man auch von einer integrierten Entwicklungsumgebung, bzw. im englischen IDE (Integrated Development Environment), welche es dem Benutzer ermöglicht alle seine Aufgaben in einer Anwendung durchzuführen. Die wesentlichsten Komponenten die Xcode für die Softwareentwicklung zur Verfügung stellt sind die folgenden:

- Ein Quellcodeeditor mit dazugehörigen SDKs mit allen notwendigen Frameworks und Bibliotheken, inklusive der Funktionalität zum kompilieren und debuggen.
- iPhone Simulator: Mit dieser Komponente kann ein iPhone simuliert werden, um Tests ohne viel Aufwand an einem Endgerät durchzuführen.
- Instruments wird verwendet, um Anwendungen zu profilieren und zu analysieren, die Leistung zu verbessern und Speicherprobleme zu finden. Alle diese Informationen werden auch von Instruments aufbereitet und visualisiert.[20]

Für das Prototyping, sprich für die Entwicklung der App mit Xcode, in Phase zwei dieser Arbeit wird auf die am 20.09.2021 veröffentlichte Xcode Version 13 zurückgegriffen.[21]

2.5.2. **Programmiersprachen iOS:**

Um nun Software für iOS Applikationen zu schreiben, benötigt es natürlich auch eine Programmiersprache mit der in Xcode entwickelt werden soll. Dazu werden von Apple zwei Sprachen zur Verfügung gestellt. Zum einen Objective-C, die auf C basierte sowie ältere Sprache, und zum anderen die etwas neuere und von Apple selbst entwickelte Programmiersprache Swift. Wobei hier Apple immer mehr auf die die mittlerweile ausgereifte Sprache Swift setzt. Zum ersten Mal veröffentlicht wurde Swift 2014 bei der World Wide Developer Conference, welche von Apple veranstaltet wurde. Zum aktuellen Stand, 21.11.2021, wurden 5. Major Releases veröffentlicht.[22]

Abbildung 5: Swift Symbol

[19] Apple, „Technologies | Apple Developer Documentation“.
[20] Apple, „Xcode | Apple Developer Documentation“.
[21] Apple, „Xcode Release Notes | Apple Developer Documentation“.
[22] Apple, „Swift - Apple Developer“.

Auch in der praktischen Ausführung in Phase zwei, wird Swift in der 5. Version zur Anwendung herbeigezogen, da diese Programmiersprache einfacher zu lernen und die Syntax modern und intuitiv aufgebaut ist.

Möchte man generell eine App Erstellen gibt ein paar Grundschritte jedes Entwicklungsprozesses die durchlaufen werden sollten, um eine erfolgreiche App zu entwickeln, egal ob man als Team oder allein daran arbeitet. Dieser Grundansatz gilt nicht nur für iOS App sondern, soll eine generelle Basis für jeden Development Prozess bilden. Auch wenn zwischen einzelnen Development Modellen und Prozessen unterschieden werden kann, so findet man immer wieder übergreifende Schritte, die in fast jedem Modell abgehandelt werden. Dazu zählen zum einen Konzeption bzw. verarbeiten der Anforderungen, Design, Entwicklung bzw. Implementierung des Designs, Durchführung von Tests, Veröffentlichung und Instandhaltung. Diese Schritte werden nun im Folgenden zusammengefasst im Detail beschrieben[23]:

2.5.3. **Konzeption**

Bevor man mit der Entwicklung der App beginnt, sollte ein durchgängiges Konzept vorhanden sein. Offene Fragen sollten vor dem Start beantwortet sein und der Sinn bzw. Nutzen der App klar definiert. Um ein abgerundetes Konzept zu erstellen, muss auch die Zielgruppe erkennbar sein, sowie ein Plan und die nächsten Schritte zur Entwicklung der App. Hierbei sollte man sich die Frage stellen, für welches Betriebssystem, Android oder iOS, möchte ich meine App zur Verfügung stellen und welchen Ansatz, bzw. welche Tools, Plattformen oder Frameworks kann ich dabei verwenden. In diesem Fall sprechen wir klarerweise über iOS und die Verwendung von Xcode als Entwicklungsplattform und Swift als Framework.[24]

2.5.4. **Design**

In diesem Schritt kommen die Ideen wie die App ausschauen soll erstmalig auf Papier, oder eher auf den Bildschirm. Diesen visuellen Entwurf nennt man auch „Wireframe“. Mit Hilfe eines Wireframes ist es schnell und einfach möglich die wichtigsten Elemente und Funktionalitäten der App demonstrativ darzustellen. Dazu zählen Punkte wie, das optische Erscheinungsbild, die grobe Navigation und eine einfache Beschreibung der primären Features. Durch Stift und Papier beispielsweise kann schon ganz einfach ein Wireframe entstehen. Der Nachteil daran ist, dass Anpassungen im Nachhinein nur sehr schwer und aufwendig durchzuführen sind. Außerdem wird auch das Teilen des Ergebnisses über digitale Medien nicht ganz einfach, bzw. wird es nicht die gewünschte Qualität haben. Besser geeignet sind speziell dafür gemachte Tools, die für das Erstellen eines Wireframes ausgelegt sind.

Ein durchgängiges Design ist wesentlich für eine gute App, da es das Erste ist, was der User sieht und diese Design Idee sollte sich auch auf allen anderen Seiten wiederfinden, um ein geschlossenes Bild abzugeben. Auch dazu gehört ein schlüssiges Benutzer-, oder Anwendererlebnis, damit der User intuitiv und einfach durch die Applikation navigieren kann.

[23] Jabangwe, Edison, und Duc, „Software Engineering Process Models for Mobile App Development“.
[24] Merenych, „What Is the Mobile App Development Process Steps?“

Diese beiden Punkte sind häufiger unter den Begriffen UI und UX zu hören, was für User Interface bzw. User Experience steht.[25]

Im Ansatz mit NoCode, wäre dieser Design Schritt und der Entwicklungsschritt zusammengefasst, was bedeutet das mit dem Erstellen des Designs im NoCode Tool auch schon das funktionale Produkt im Hintergrund entstanden ist und sofort anwendbar eingesetzt werden kann.

2.5.5. **Entwicklung & Test**

Hier startet die Programmierung der App anhand des zuvor Erstellten Wireframes, was in gewisser Weise als Basis dienen soll. Welche Tools und Anwendungen dabei verwendet werden, sollte aus dem Konzept ersichtlich sein. Flexibilität zum Ändern des Konzepts bei unvorhergesehenen Problemen oder Hindernissen, sollte aber auch gegeben sein. Eine große Rolle spielt es auch, ob man allein an der App entwickelt oder als Team. Davon abhängig können sich auch Methoden oder Vorgehensmodelle unterscheiden, mit denen die Entwicklung stattfindet.[26] Das am häufigsten verwendeten Modell im Mobile-App-Development ist das Scrum Framework, welches sich auf die agile Arbeitsweise im Softwareentwicklungsbereich spezialisiert hat. [27]

Auch wichtig in diesem Schritt ist es, immer wieder Tests des aktuellen Entwicklungstandes durchzuführen, um sicherzustellen, ob man noch auf dem richtigen Weg ist und dass sich die programmierten Funktionen auch wie gewünscht verhalten.

2.5.6. **Launch & Support**

Der wichtigste Schritt ist das Veröffentlichen der App, denn ohne den Launch wären alle Schritte und der Aufwand hinfällig. Häufig in einer inkrementellen Arbeitsweise ist es so, dass relativ früh, sprich auch schon während der Entwicklungsphase eine Art MVP Live geht, heißt ein „Minimum Viable Product“ (MVP) wird veröffentlicht und dieses wird dann Stück für Stück verbessert und mit den geplanten Features und Funktionalitäten erweitert. Da wir in diesem Fall von einer iOS App sprechen, muss diese auch über den Apple App Store veröffentlicht werden, was auch nicht ganz einfach ist.

Um seine App für den App Store einzureichen, muss man sich zuallererst für das „Apple Developer Program“ registrieren, wo auch eine jährliche Gebühr fällig wird. Vor dem Hochladen der App, sollte man sichergestellt haben, dass die entwickelte App alle von Apple vorgegebenen Guidelines erfüllt, die sich auf technische, inhaltliche und grafische Kriterien bezieht. Nach Eingabe von Daten und Informationen wie Name, Icons, Schlüsselwörter, oder Beschreibung kann die App über Xcode hochgeladen werden und für die Überprüfung von Apple freigeben werden[28]

[25] Merenych.
[26] Merenych.
[27] Jabangwe, Edison, und Duc, „Software Engineering Process Models for Mobile App Development“.
[28] Apple, „Submit your iOS and iPadOS apps to the App Store - Apple Developer“.

Nach dem Herausgeben der App spielt auch der Support eine relevante Rolle. Durch das Feedback der Nutzer können Fehler erkannt werden, oder es wird sichtbar welches Feature oder welche Funktionalität am dringendsten Benötigt werden, um die App durch Updates weiter zu verbessern. Der Lebenszyklus geht somit in einen weiteren Status und dieser sollte nicht vernachlässigt werden.[29]

[29] Merenych, „What Is the Mobile App Development Process Steps?“

3. NoCode Tools in der Anwendung

In Kapitel 3 wird auf die Anwendung von NoCode/LowCode Tools in der Praxis eingegangen und welche Anwendungsfälle bzw. Plattformen es wirklich gibt. Im speziellen, wird zu Beginn genauer auf Auswertungen zur Zufriedenheit und Akzeptanz von NoCode Usern geschaut, sowie auf Herausforderungen, die bei der Verwendung von NoCode oder LowCode Tools im Allgemeinen vorherrschen, eingegangen. Danach werden häufige Anwendungsfälle betrachtet und gezeigt welche realen Tools und Plattformen in der Praxis zum Einsatz kommen. Zu guter Letzt werden die Unterschiede verschiedener Umsetzungen von Mobilen Apps besprochen, als auch das Tool, welches im Prototyping Teil dieser Arbeit ihren Einsatz findet, genauer vorgestellt.

3.1. **Akzeptanz und Adaption**

Da es mittlerweile schon eine große Anzahl an NoCode oder LowCode Nutzer gibt die täglich auf diesen Plattformen arbeiten, wäre es sinnvoll einen Blick auf die Akzeptanz zu werfen und zu schauen welche Auswirkungen die Verwendung solcher Tools in den Entwicklungsprozess haben. Dazu Aufschluss gibt uns zum einen eine im März 2019 weltweit durchgeführte Umfrage, an der über 3.000 Personen aus dem IT-Bereich teilnahmen. Die Umfrage und Auswertung wurde von OutSystems durchgeführt. Die behandelten Themen waren hierbei aktuelle Trends in der Applikationsentwicklung und welche Rolle NoCode/LowCode Tools in diesem Prozess spielen bzw. welche Auswirkungen sie haben. [30]

Die erste Grafik in Abbildung 6 zeigt den Unterschied in Bezug auf den wohl wesentlichsten Vorteil einer LowCode Anwendung, die Entwicklungszeit für neue Applikationen bzw. Erweiterungen und Features. Der verkürzte Entwicklungszeitraum wird somit auch grafisch deutlich. Konkret kann man sagen, dass 64% aller Low-Code erstellten Anwendungen nicht länger als 4 Monate zur Fertigstellung benötigt haben. Was in Gegenüberstellung zur klassischen Entwicklung einen 15-prozentigen Vorteil ergibt. Der Anteil an Applikationen mit einer langwierigen Entwicklungszeit von über 12 Monaten ist im Vergleich auch deutlich weniger. Als Ergebnis kann nun somit schneller und effektiver auf Probleme oder dringliche Anforderungen reagiert werden. Weiters lässt sich daraus schließen, dass Applikationen, die einen geplanten geringen Umfang haben sollen, eher mit LowCode oder NoCode Tools umgesetzt werden.[31]

[30] OutSystems, „The State of Application Development“.
[31] OutSystems.

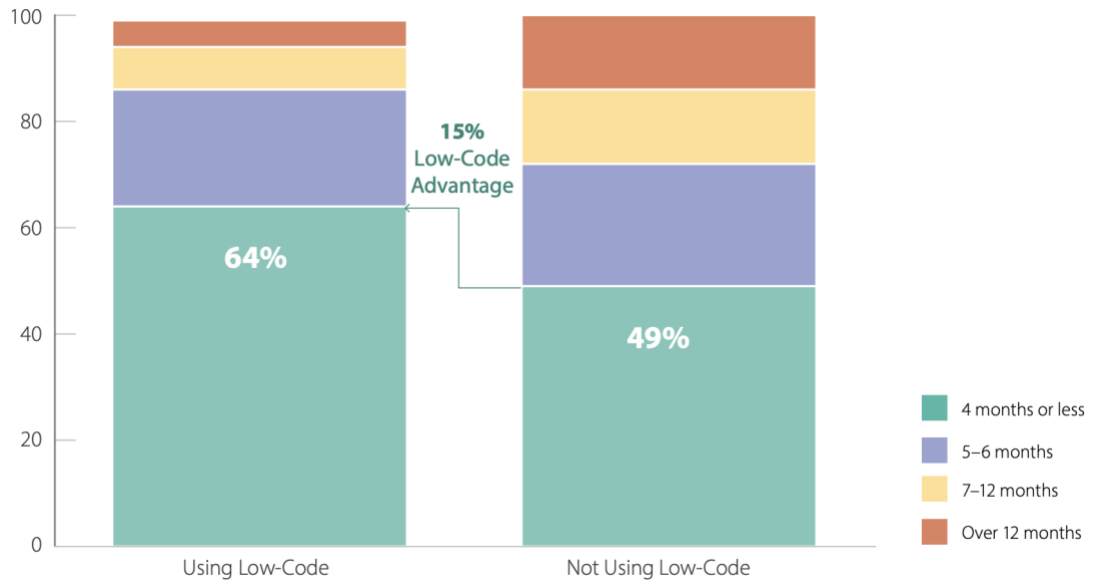


Abbildung 6: Mobile App – Entwicklungszeit[32]

Betrachtet man nun die in Abbildung 7 behandelte Unternehmenszufriedenheit des Release Zyklus im Zusammenhang mit der entwickelten Applikation, so zeigt sich auch hier, dass durch die Anwendung von NoCode/LowCode Anwendung die Zufriedenheit deutlich gesteigert wurde. 37 Prozent der befragten LowCode-Anwender beschreiben ihren Output als „satisfied“ in Bezug auf die Häufigkeit bzw. Frequenz der Softwareveröffentlichungen. Verglichen mit nur 26 % derjenigen, die Low-Code nicht verwenden. Wobei mehr als 50% der Befragten einen Release Zyklus von einem Monat oder noch kurzer haben. Im Gesamten bringen diese Ergebnisse zum Ausdruck das die Agilität im Ganzen gesteigert werden kann. Das zeigt auch die Auswertung des „Organizational Agility Score“, welcher bei der Befragung von OutSystems erhoben wurde. Durch die Verwendung von NoCode und LowCode Tools könnte die Agilität somit um 8% gesteigert werden. Außerdem wurden bei diesen Unternehmen viel schneller agile Praktiken und Methoden implementiert, die den gesamten Entwicklungsprozess verbessern sollen.[33]

Abbildung 7: Unternehmenszufriedenheit mit dem Release Zyklus[34]

[32] OutSystems.
[33] OutSystems.
[34] OutSystems.

Die hohe Akzeptanz ist aber auch den praktischen Vorteilen zu verdankten die NoCode und LowCode Anwendungen mit sich bringen. Im Wesentlichen sprechen wir hier neben den schon zuvor angesprochenen Themen, auch beispielsweise über die einfachen Einstiegsmöglichkeiten und was sonst noch zur Akzeptanz und Adaption beiträgt.

Wenig überraschend ist es daher, dass die schnelle Entwicklung und dadurch verbunden kurze Auslieferungsdauer der Applikation als wichtigster Vorteil gesehen. Knapp gefolgt von der schnellen Lernkurve und relativ einfachen Anwendbarkeit von NoCode und LowCode Plattformen, was neben der kurzen Entwicklungszeit zum grundlegendsten Vorteil zählt. Des Weiteren sind im Falle eines Prototypes oder PoC die Produktionskosten einer Applikation, die über NoCode erstellt, deutlich niedriger sind, da auch einfach und schnell Verschiedene Plattformen evaluiert werden können. Ausgestattet mit einer Einsteigerfreundliche Benutzeroberfläche können auch nicht-technische Benutzer, Software auf einfache und vertraute Weise erstellen. Dazugehörig wird somit auch die User Experiance von NoCode/LowCode Tools als wesentlicher Vorteil gesehen.[35]

Interessanterweise gibt es aber noch immer viele Personen, auch in technischen Rollen, die skeptisch und noch unsicher gegenüber NoCode und LowCode Tools stehen und sich lieber für eine Alternative Entwicklungsmethodik entscheiden.[36] Warum einige noch nicht ganz von diesen Plattformen überzeugt sind und welche Herausforderungen, bzw. Problematiken auf Benutzer zukommen, wird im folgenden Kapitel ausführlicher behandelt.

3.2. **Herausforderungen von NoCode Plattformen**

Trotz der schon hohen Akzeptanz und Verwendung gibt es noch immer einige Herausforderungen und Kritikpunkte die NoCode/LowCode betreffen und dadurch ein Einsatz für manche aktuell noch keine Option ist. Deshalb ist es auch wichtig auf diese Challenges hinzuweisen und nicht darüber hinwegzuschauen.
Wie schon bekannt ist es mit Low/No-Code-Plattformen viel einfacher Applikationen zu erstellen als herkömmliche Programmiersysteme. Es ist jedoch falsch anzunehmen, dass ein Citizen Developer auch ganz schnell und einfach komplexe Anwendungen erstellen kann nach nur ein oder zwei Trainingseinheiten. Es darf nicht vernachlässigt werden, dass auch Businessanwender ohne Programmierkenntnisse zuerst lernen und üben müssen, wie die verschiedenen Teile der Low-/No-Code-Plattform zusammenpassen. Darüber hinaus müssen die Benutzer Zeit investieren, um die Plattformfunktionen zu nutzen bzw. effizient einsetzten zu können und robuste und skalierbare Anwendungen zu erstellen.[37] Zusätzlich ist es fast immer der Fall, dass sich bei einem Wechsel von einem NoCode Tool zum anderen, die Funktionalität, wie zum Beispiel das Testen der Anwendung, oder auch Gestaltungsmöglichkeit, sprich die Grafische Oberfläche zum Erstellen der Anwendung grundlegend ändern.[38]

[35] Luo u. a., „Characteristics and Challenges of Low-Code Development“.
[36] Beranicˇ, Rek, und Hericˇko, „Adoption and Usability of Low-Code/No-Code Development Tools“.
[37] Hiren Amin und Saurabh Kapoor, „KPMG - Opportunities and Challenges in a Low/No-Code World“.
[38] Khorram, Mottu, und Sunyé, „Challenges & Opportunities in Low-Code Testing“.

Dazu anschließend hat man bei der Verwendung von NoCode oder LowCode Tools eben die Herausforderung, in gewisser Weise von einer Plattform abhängig zu sein. Es ist zwar leicht, sich in eine NoCode Anwendung einzuarbeiten und ein Produkt zu erstellen, aber schwer, diese Plattform zu verlassen bzw. zu wechseln. Die Problematik, die sich dabei ergibt, ist es keinen wiederverwendbaren Anteil der Applikation extrahieren zu können, der dann als Grundlagen in einer anderen NoCode Plattformen dienen könnte. Es muss so zu sagen immer von Grund auf neu gebaut werden, da die erstellten Applikationen von einer Entwicklungsperspektive eine Art Black-Box darstellen. Zwar kann es Plattformen geben, bei denen es möglich ist Teile des erzeugten Codes lokal abzuspeichern, nur stellt sich dann die Frage, was man damit tun soll.

Viele vor allem neue Low-/No-Code-Plattformen befinden sich noch in der Entwicklungsphase und sind noch nicht komplett ausgereifte Produkte. Die Unternehmen haben zwar Versionen auf den Markt gebracht, mit denen sich Anwendungen relativ einfach erstellen lässt, aber die Benutzer sind manchmal frustriert, weil sie bestimmte Fehler nicht verstehen und nicht beheben können. Das liegt häufig am Debugging, was für Citizen Developer oft nicht sehr einfach nachzuvollziehen ist. Bei jungen NoCode Anwendung kann es vorkommen, dass noch keine solide und fundierte Testfunktionalität mit zusätzlichen Debugging-Features vorhanden ist.[39] Auch wenn es darum geht Applikationen zu bauen die für kritische oder sehr sensible Anforderungen bzw. Anwendungsfälle gedacht wären, sind viele Benutzer noch nicht der Meinung dabei auf NoCode oder LowCode Tools zu setzen, da vielleicht das Risiko, nicht das gewünschte Ergebnis zu erreichen, zu hoch ist.[40]

Ein weiterer Punkt, der nicht vernachlässigt werden soll, und vor allem bei Enterprise Benutzern relevant werden könnte, sind die Kosten für manche Anwendungen bei einem gewerblichen Gebrauch der Endanwendung, insbesondere wenn eine größere Anzahl an Mitarbeitern auf die NoCode/LowCode Plattform Zugriff haben soll.[41]

3.3. **Use & Business Cases**

Wofür werden NoCode und LowCode Plattformen benutzt, bzw. wofür finden sie in der Praxis eine Anwendung? In diesem Kapitel soll kurz behandelt werden, wo und für was diese Tools eingesetzt werden.

Dank der zeitlichen Komponenten werden NoCode Tools in erste Linie gerne von Startups, oder Personen genutzt, die eine Proof of Concept oder eine Prototyp Lösung in relativ kurzer Zeit erstellen möchten, ohne dabei größere Investitionen tätigen zu müssen. Größere Unternehmen sehen den Use Case hinter den NoCode und LowCode Tools in der Einbindung von Citizen Developern in den Entwicklungsprozess, wodurch Personen mit Fachwissen über diese Plattformen Produkte und Applikationen erstellen können.

In Bezug auf Business Cases bzw. praktische Anwendungsfälle zeigen Auswertungen, dass einige Branchen häufiger auf NoCode und LowCode Produkte zurückgreifen und diese

[39] Hiren Amin und Saurabh Kapoor, „KPMG - Opportunities and Challenges in a Low/No-Code World“.
[40] Beranicˇ, Rek, und Hericˇko, „Adoption and Usability of Low-Code/No-Code Development Tools“.
[41] Luo u. a., „Characteristics and Challenges of Low-Code Development“.

Plattformen nutzen. So sieht man das Wirtschaftszweige wie E-Commerce, Social Media oder Customer-Relationship-Management, im Gegensatz zu konservativeren Branchen, wenig überraschend auf diese Innovativen Tools zurückgreifen. Ein Grund dafür ist sicherlich, dass man mit potenziellen und bestehenden Kunden rein über das Internet kommuniziert und alle Kontaktarten virtuell abwickelt. Im Falle einer Social Media Anwendung, kann es sich zum Beispiel um eine Messaging-, Dating-, oder Blogging-App handeln, die mit NoCode oder LowCode Tools erstellt wurde. Es können nicht nur spezifische Anwendungsfälle mit NoCode gelöst werden, sondern auch spezifische Teile einer größeren, komplexeren Gesamt-Softwarelösung. Nicht unwahrscheinlich ist es daher, den Frontend Teil der Applikation über ein NoCode Tool zu erstellen, wobei der Backend Teil wiederum auf klassischer Weise entwickelt wird. Auch möglich wäre, dass nur die Integration, sprich die Schnittstelle zwischen Front-und Backend über NoCode/LowCode Plattformen abgewickelt wird. Die am häufigsten von NoCode Tools benötigten Teile sind die Frontend, Webflow, Integrationsparts. Aber auch Daten Visualisierung kann beispielsweise als Anwendungsfall auftreten. Viele Produkte decken aber auch alle diese Applikationsparts gesamtheitlich ab, wodurch man sich die Kommunikation der verschiedenen Produkte spart. Darunter zählen Lösungen wie „Bubble.io“, “Webflow“, oder “Adalo“.[42]

In der nachfolgenden Abbildung werden NoCode Tools zusammengefasst, welche heutzutage hohe Anwendung finden und technologisch auf neusten Stand sind:

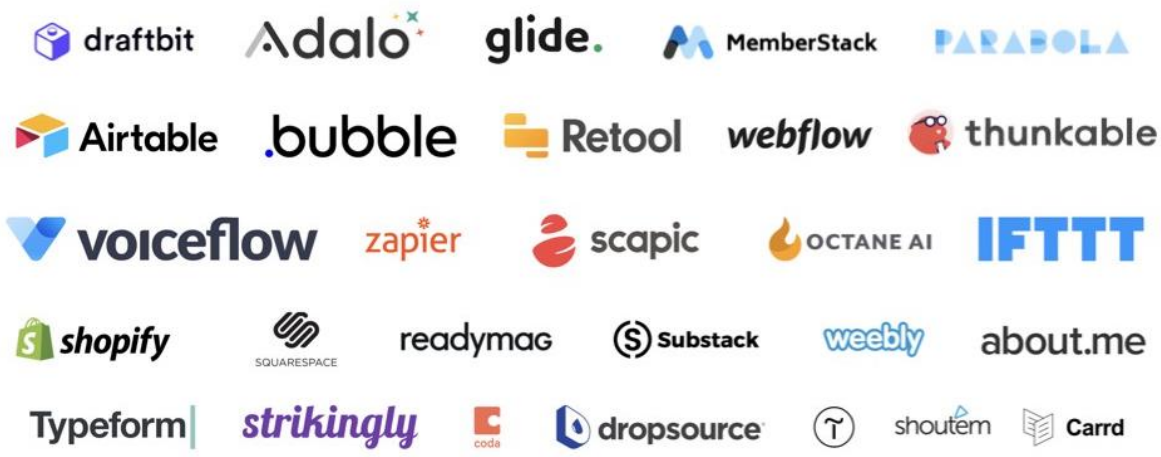


Abbildung 8: Überblick über aktuelle NoCode Tools (eigene Darstellung)

3.4. **Auswahl des NoCode Tools**

Um das bestmögliche Ergebnis zu erzielen und eine aussagekräftige Auswertung zu machen, soll nicht ein zufällig Gewähltes NoCode Tool zu Anwendung hergezogen werden. Dazu wurden unterschiedliche Tools verglichen und schlussendlich eine Wahl aufgrund bestimmter Merkmale und Kriterien festgelegt. Die infrage kommenden Plattformen, welche eine App für

[42] Luo u. a.

Smartphones herstellen können, wären bubble, Appgyver, Glide und Adalo. Zum einen stehen diese Plattformen in der engeren Auswahl, da sie den häufigsten Gebrauch finden. Deutlich wird diese Annahme, durch die meisten dazugehörigen Posts und Beiträge in Foren wie zum Beispiel Stack Overflow haben, was von einer großen Community zeugt.[43] Dank einer großen Community können Probleme, oder Schwierigkeiten beim Erstellen der Applikation schneller behoben werden, weil sie mit großer Wahrscheinlichkeit schon mal eine andere Person hatte und Möglichweise über Foren einen Lösungsweg teilen kann.

Um festzustellen welches Tool am besten für die Umsetzung geeignet ist, wurde mit jeden Einzelnen der vier Tools eine kleine und kurz Demo Applikation erstellt. Mit Hilfe dieser Demoapplikation war es gut zu erkennen, wie intuitiv und schnell man sich auf die Benutzeroberfläche einlassen kann und zurechtfindet. Im Zuge dieser Demos wurden hauptsächlich die einzelnen Features in einer leere App mit mehreren Seiten gebaut und getestet, um eine Gefühl zu bekommen, wie gut die Komponenten einzubinden und zu integrieren sind. Zusätzlich war es auch nach kurzer Zeit schon grob ersichtlich wie die Hauptbestandteile aussehen und worauf der Fokus der Plattform gesetzt ist. Beispielsweise war der Fokus von bubble.io auf Workflows ausgelegt, welche komplex und eher aufwendiger zu gestalten sind. Glide wiederum legt den Fokus auf die Datenverarbeitung, bietet aber wie bubble.io keine Native App oder Hybrid-App an, die über den Apple oder Google Store veröffentlicht werden kann. Stattdessen baut man eine Web-App, die über den Browser am PC aufgerufen werden kann, sowie am Smartphone über einen Mobilen Browser. Um aber einen guten Vergleich zur klassischen iOS Entwicklung festzustellen, sollte die NoCode Anwendung im besten Fall eine App über den jeweiligen Store zur Verfügung stellen, damit der End User die gleiche Experiance hat. Appgyver, eine der vier Tools bietet neben Adalo die möglich an die erstelle Applikation über die App Stores zu veröffentlichen. Im Gegensatz zu Appgyver, hat aber Adalo die ausgereifteren Komponenten und kann auch mit der Anpassbarkeit, sprich detaillierteren Konfiguration dieser Features punkten.

Aus diesen Gründen, welche aus meinen kurzen Demos ersichtlich waren, habe ich mich schlussendlich dazu entschieden, Adalo als NoCode Tool der Wahl für das Prototyping im zweiten Teil der Arbeit zu benutzen.

3.5. **Adalo**

Eines der ausgereiftesten und bekanntesten NoCode Tools für das Erstellen einer Mobilen Applikation ist Adalo. Das im Jahr 2019 von David Adkin und Ben Haefele gegründete Startup hat sich schnell als Anlaufstelle der Mobilen NoCode App Entwicklung herausgestellt und verbessert seine Plattform stätig weiter.

Abbildung 9: Adalo Logo

Auf der Website von Adalo wird folgende Vision dargestellt, die zeigen soll, wo sich das NoCode Tool hin entwickeln möchte:

[43] Luo u. a.

„ Ultimately our vision is for Adalo to be the go-to platform for innovation — for startups, for enterprise, and for everyone who had an idea they didn't know how to turn into reality. "[44]

Wie im Kapitel zuvor angesprochen, wird das Endprodukt, welches mit Adalo entwickelt wird, als Hybrid-App bezeichnet und kann sowohl als iOS und Android Version erstellt und zur Verfügung gestellt werden. Durch eine große und aktive Community, sowie einer umfangreichen und einsteigerfreundlichen Dokumentation sowie Produktbeschreibung ist es auch für nicht technische Personen schnell möglich mit Adalo eine funktionstüchtige App für Smartphones zu erstellen. Eine weitere Besonderheit ist die Option, dass Endprodukt direkt aus Adalo im Apple App Store und/oder im Google Play Store veröffentlichen zu können, ohne Umwege und aufwendige Exporte zu durchlaufen. Die Verwendung von Adalo ist für eine einzelne Person kostenfrei, nur kann es sein, dass man bei bestimmten Funktionalitäten auf Limitierungen stößt, welche gegen eine Monatliche Gebühr ausgeweitet werden können. Auch das schlussendliche Freigeben für den Apple und Google Store ist nur bei einer bezahlten Lizenz möglich, das Testen ist jedoch für alle Benutzer möglich.

Aufgrund dieser Gegebenheiten wird auch einer der zwei Prototypen, welche im zweiten Teil dieser Arbeit behandelt werden mit Adalo entwickelt.

[44] Adalo, „About Our Mission, Vision and Team".

4. Zielsetzung

Für die Zielsetzung der praktischen Umsetzungen der beiden Applikationen wird eine gemeinsame Aufgabenstellung, bzw. eine gemeinsame Anforderungsliste erstellt, welche für die NoCode App als auch für die Xcode App gilt. Außerdem muss für eine korrekte Auswertung eine Definition von messbaren Parametern vorhanden sein. Diese Parameter müssen auch die Möglichkeit bieten, dass beide Varianten damit miteinander bzw. gegeneinander verglichen werden können, da sonst keine Auswertung stattfinden kann.

4.1. **Aufgabenstellung**

Die Aufgabenstellung ergibt sich wie zuvor erwähnt aus der Anforderungsliste, welche Punkte zu Funktionalität, Design und Datenhandling umfasst. Auch Grundanforderungen an die jeweiligen Entwicklungsumgebungen sind hier erfasst. Zu jedem dieser Punkte wurden folgende Spalten definiert:

- **Bereich** – Funktionalität, Design oder Datenhandling
- **Titel** – damit eine grobe Zuordnung möglich ist
- **Beschreibung** – genaue Definition der Anforderung
- **Zeitschätzung NoCode (in Stunden)** – Voraussichtliche Zeit, die benötigt wird, um diese Anforderung zu lösen
- **Zeitschätzung Xcode (in Stunden)** – dito
- **Anmerkung** – Zusätzliche Information zur jeweiligen Anforderung

Die konkrete Aufgabenstellung in dieser Arbeit ist es, eine **Wetter App** als Proof of Concept für iOS Geräte, jeweils mit NoCode und Xcode, zu entwickeln. Diese App soll aus einer Start- sowie Detailseite bestehen und das aktuelle Wetter, wie auch eine Vorschau zum Wetter, zu einem ausgewählten Ort anzeigen. Orte sollen frei wählbar sein und in einer Liste auf der Startseite angezeigt werden. Die Daten dazu, sollen aus einer externen Wetter API dynamisch in die App geladen werden.

Die dafür ausgewählte Wetter API, ist jene von OpenWeatherMap. Die API bietet verschiedenste Aufrufe an, über die Wetterdaten abgefragt werden können. Es wurde diese API gewählt, weil sie eine der am häufigsten benutzten ist, was für eine hohe Qualität spricht, und auch weil die Daten-Ergebnisse genau den Anforderungen entsprechen. Die Gratisversion der API erlaubt es je nach Abfrage bis zu 60 Aufrufe pro Minute zu tätigen. Um die Schnittstelle anwenden zu können und um die Menge an erlaubten Aufrufen zu überprüfen, wird ein API-Key zur Verfügung gestellt, welcher bei der Verwendung der API zwingend notwendig ist.[45]

Insgesamt wurden dazu 16 Punkte mit der jeweiligen Zeitschätzung für NoCode sowie für Xcode definiert, welche im Anhang ersichtlich sind. In der folgenden Abbildung ist Punkt 8 beispielhaft zu sehen:

[45] OpenWeather, „Weather API - OpenWeatherMap“.

	Bereich	Titel	Beschreibung	Zeitschätzung NoCode (h)	Zeitschätzung Xcode iOS (h)	Anmerkung
8	Design / Funktionalität	Detailseite - Unterer Teil	Im unteren Bereich der Detailansicht soll eine 5 Tagesvorschau ersichtlich sein, welche die minimale und maximale Temperatur des Tages anzeigt.	6	14	Anbindung an die Wetter API und synchrones befüllen der Daten

Abbildung 10: Punkt 8 - Anforderungsliste

Die erste Spalte ist der Bereich, welcher hier mit Design und Funktionalität zu sehen ist. Da sich eine Anforderung oft mit mehreren Bereichen beschäftigt, sprich die Bereiche oft schneiden und schwer zu trennen sind, wird diese Spalte oft zwei Bereiche beinhalten. Danach wurde als Titel „Detailseite – Unterer Teil“ gewählt, gefolgt von der Beschreibung, welche mittig platziert ist. Die Beiden Zeitschätzungen mit 6h für NoCode und 14h für Xcode. Hierbei ist noch zu erwähnen, dass es sich bei dieser Schätzung rein um das Umsetzen dieser Anforderung handelt und zusätzliche Aufwände, die nicht direkt mit diesem Punkt zu tun haben, hier nicht zu messen sind. Ganz rechts ist noch eine Anmerkung zu sehen.

4.2. Definition der Parameter

Die Parameter, die für die Aufgabenstellung definiert werden, lassen sich im Wesentlichen in zwei verschiedene Bereiche teilen. Zum einen kann der gesamte zeitlichen Aufwand betrachtet werden und zum anderen die Erfüllung der Anforderungen.

Man darf bei diesen Parametern natürlich nicht vernachlässigen, dass es ein wesentlicher Bestandteil ist, wie erfahren und talentiert das Entwicklungsteam, sprich der Umsetzer dieser Apps ist. Insbesondere beim zeitlichen Aufwand können hier enorme Unterschiede entstehen. Aber auch wenn es um die Thematik, den Scope, bzw. Umfang, oder bestimmte Funktionalitäten der zu entwickelten Applikation geht, kann die Dauer der Entwicklung deutlich variieren.

4.2.1. Zeitlicher Aufwand

Der Zeitliche Aufwand ist sehr wahrscheinlich auch der wichtigste und relevanteste Parameter, der bei der Entwicklung einer neuen Software gemessen und bewertet werden kann. Vor allem da mit dem Faktor Zeit nicht nur die Dauer an sich abgedeckt werden kann, sondern noch viel mehr dahintersteckt. Betrachtet man den Parameter Zeit in einem wirtschaftlichen Blickwinkel, so ist bei Softwareprojekten die Zeit der wesentlichste Punkt in der Budgetierung eines solchen Projektes, denn die Arbeitsstunden des Entwicklers, sprich des Projektteams, die Hauptkosten ausmachen.

In dieser Arbeit werden in Bezug auf den zeitlichen Aufwand, zwei Dinge betrachtet. Die **Einarbeitungsdauer**, die es braucht, um mit einem gewissen Tool arbeiten zu können. Darunter zu verstehen ist auch die Menge und Qualität der angebotenen Hilfsmittel und Dokumentation, die einem diese Einarbeitungsdauer verringern sollen.

Am wichtigsten aber ist der zeitliche **Umsetzungsaufwand,** der bei der Abarbeitung und der Anforderungsliste entsteht. Hiermit soll gemessen werden, wie lange wirklich an der Umsetzung gearbeitet wurde. Die Planung der NoCode und Xcode App wird in diesem Umsetzungsaufwand erfasst, damit auch jeder Aufwand, der bei der Erstellung der beiden Apps entsteht, mit einbezogen werden kann.

Mit Hilfe des zeitlichen Parameters, soll also sichtbar werden, ob der Faktor Umsetzungsdauer ein mögliches, bzw. valides Kriterium für die Beantwortung der Forschungsfrage ist und ob man wie angenommen bei der NoCode Entwicklung einen deutlichen Vorteil gegenüber der Xcode Variante hat. Eine gewisse Referenz soll dazu auch die Zeitschätzung geben, welche in der Anforderungsliste eingetragen wurde. Laut dieser sollte die NoCode App weniger als halb so lang dauern als die programmierte Variante.

4.2.2. **Erfüllung der Anforderungen**

Es ist aber nicht nur die Dauer zu bewerten, sondern auch das Ergebnis der Umsetzung und somit die Erfüllung der Anforderungen. Bewertet werden soll nicht nur die Erfüllung der Anforderung an sich, sondern viel eher wie man zu diesem Ergebnis gekommen ist. Die zwei Sichtweisen, die hier bewertet werden können, sind zum einen, ob die Erfüllung einer spezifischen Anforderung, oder Funktionalität, mit Standard, Out-of-the-Box, -mittel möglich ist, oder ob Workarounds und Alternativen notwendig sind. Vor Allem im Hinblick auf die NoCode Entwicklung stellt sich die Frage, ob man sich mit einem separatem Tool weiterhelfen muss. Außerdem kann noch betrachtet werden, wie hoch die Komplexität für die Umsetzung der Anforderung für die jeweilige App ist.

Bei der Fragestellung Standard oder Workaround, ist es klar als Vorteil zu bewerten, wenn die gewünschte Funktionalität mit den Standardmitteln, welche auch in Dokumentationen erfasst und beschrieben sind, umgesetzt werden kann. Der große Nachteil bei Workarounds ist oft, dass die Mittel, mit denen die Umsetzung des Workarounds erfolgt, weitere Abhängigkeiten bilden und vor allem bei NoCode Tools ständige Updates der Software oft zu Problemen führen können. Zusätzlich kann die alternative Einbindung von weiteren NoCode Tools erhebliche Änderungen bei den Lizenzkosten mit sich führen.

Schaut man auf den direkten Vergleich der beiden Applikationen, so wird es interessant zu sehen, wie komplex die jeweilige Anforderung für jede der beiden Apps ist. Die Komplexität der Umsetzung für eine spezifische Anforderung kann beispielsweise bei der Xcode App sehr hoch sein, aber möglicherweise gibt es bei der NoCode App ein fertiges Modul, oder einen Baustein, der verwendet werden kann, wodurch die Komplexität klar geringer ist.

Natürlich muss auch hier erwähnt werden, dass die Komplexität und die Bewertung der Erfüllung eigentlich auch zeitliche Auswirkungen hat und dass die beiden Parameter sehr stark miteinander verknüpft sind.

5. Proof-of-Concept - No Code

In diesem Kapitel wird die Planung und Erstellung des PoC (Proof-of-Concept) für die NoCode Variante beschrieben und dargestellt. Insbesondere sollen unvorhergesehene Probleme und notwendige Workarounds hervorgehoben werden, die so nicht geplant waren und einen extra Aufwand im Sinne der Umsetzungsdauer gekostet haben, aber auch die Komplexität der Umsetzung für die spezifische Anforderung erhöht haben. Diese Problemstellungen werden in den folgenden Unterpunkten beschrieben.

5.1. **Planung**

Wie in Kapitel 3 beschrieben, soll Adalo als NoCode Tool für die Erstellung der iOS App verwendet und eingesetzt werden. Um loszulegen, muss man sich nur auf der Adalo Website anmelden und einen Account erstellen. Dann ist es auch schon möglich mit der Gratisversion eine erste App zu bauen. Es gibt auch eine Premiumversion, die es ermöglicht eine externe Datenbank anzubinden, welche aber kostenpflichtig ist und in dieser Umsetzung nicht Verwendung finden soll.

Der erste Teil dieser App, wird die Startseite sein, welche die Orte in einer Listenform anzeigen soll. Wie gerade erwähnt, soll dazu keine externe Datenbank angebunden werden, auf der die Orte abgespeichert werden. Vielmehr ist angedacht, die internen Lösungen zu verwenden, die es einem ermöglichen, Daten in einer definierbaren Form abzulegen und abzufragen. Dazu soll ein „Location"-Objekt definiert werden. Wie auch in den Anforderungen beschrieben, sollen auch neue Orte über ein Plus Symbol hinzugefügt werden können. Dieses Plus-Symbol leitet auf eine weitere Seite, mit der ein neuer „Location" - Eintrag erstellt werden kann. Auf dieser neu angelegten Location wird zunächst nur der eingegebene Name übernommen.

Die Weiterleitung auf die Detailseite und Darstellung der Werte ist an sich kein Problem. Die Aufgabe, die es hier aber zu lösen gibt ist, wie die Wetterdaten, sprich aktuelle Temperaturen und Wettervorhersagen in die NoCode App kommen und aktualisiert werden. Dazu soll die zuvor erwähnte API von OpenWeatherMap verwendet werden. Da Adalo keine passende Möglichkeit anbietet manuelle, custom API-Abfragen zu machen muss dazu nach einem weiteren NoCode Tool gesucht werden, welches diese Aufgaben übernehmen kann. Dieses Tool muss die Datenintegration und Kommunikation mit der API lösen und diese Informationen wieder an Adalo zurückschicken. Diese Wetterdaten werden dann auf dem jeweiligen Location-Objekt abgespeichert und wie in der App definiert am Frontend angezeigt. Der genaue Ablauf dieser Integration ist im nächsten Kapitel 5.2 beschrieben.

Bei der Planung und Umsetzung der NoCode App sind schnell gewisse Grenzen zu sehen. So ist für wenige Anforderungen, schon von Beginn an, eine Umsetzung ausgeschlossen, da sich die technischen Möglichkeiten begrenzen. So ist beispielsweise die dynamische Änderung des Hintergrundbildes, wie es in Anforderung 11 beschrieben ist, nicht möglich, weil ein Hintergrund nur fix gesetzt werden kann. Es wird dazu in Kapitel 7, Auswertung der beiden Varianten, näher auf diese Einschränkungen und Schwierigkeiten eingegangen.

5.2. **Erstellung**

Bei der Erstellung einer neuen Adalo App wählt man im folgenden Auswahlbildschirm, Abbildung 11, die Native Mobile App aus, damit diese dann am Ende der Umsetzung auch auf iOS Geräte ausgerollt werden könnte. Nach dem optionalen Auswählen eines Templates und setzten eines Namens, sowie die Wahl der Primär- und Sekundärfarbe der App, kann direkt losgelegt werden.

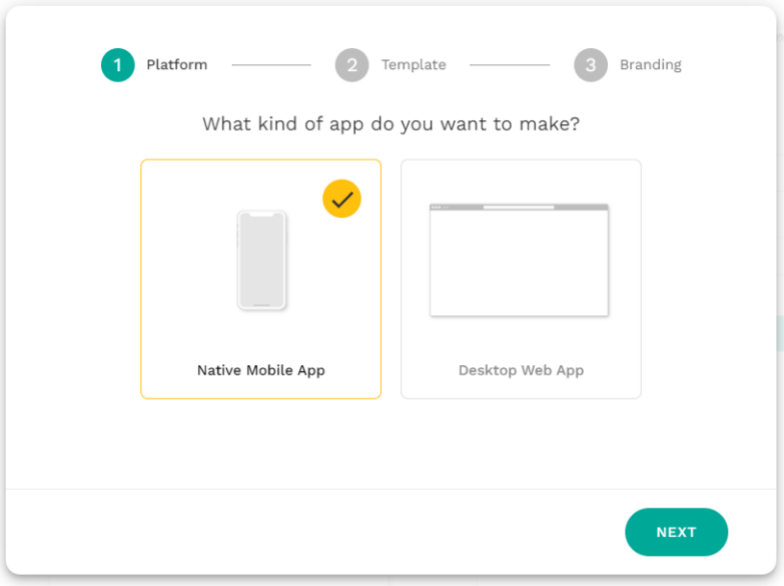


Abbildung 11: Erstellung einer neuen Adalo App

Zu Beginn wird einem nur der Startbildschirm allein im Editor angezeigt, welcher noch ungefüllt ist. Im ersten Schritt wird wie geplant eine Liste auf die Startseite gezogen, in der nachher die Locations angezeigt werden. Im folgenden Screenshot ist gut zu sehen, wie sich die Komponenten auf dem Bildschirm verhalten:

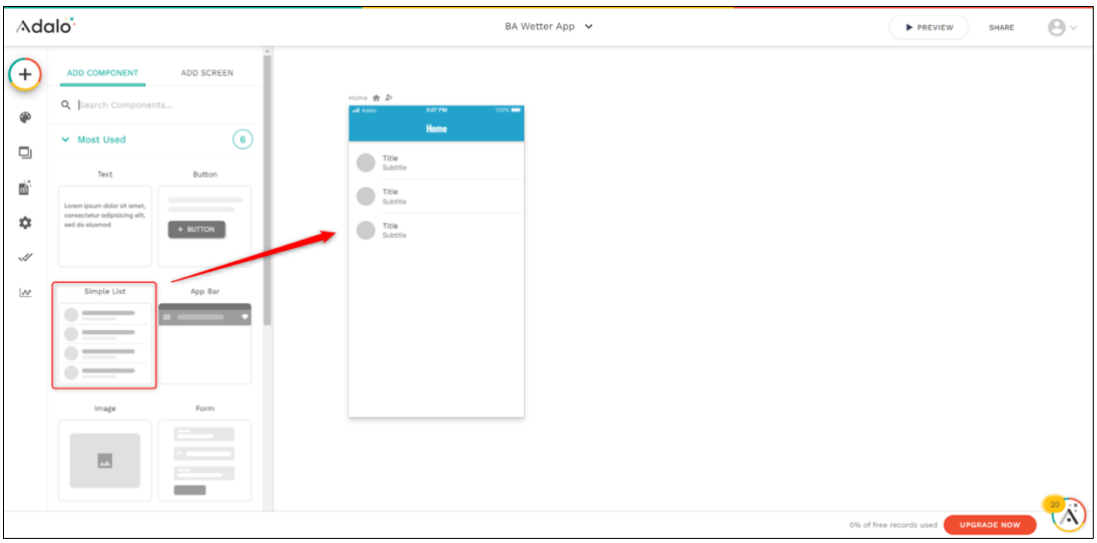


Abbildung 12: Adalo - Liste hinzufügen

Bevor aber Daten in der Liste angezeigt werden können, muss das Location Objekt definiert werden. Um möglichst viel Komplexität von dem Datenmapping der API-Abfrage zu nehmen, werden alle Felder auf einer Ebene definiert und es werden keine Relationen zu weiteren Objekten hergestellt. Jedes Feld, auch Temperaturwerte, werden als „String"-Felder definiert, da die Daten, welche über die Wetter Integration in die App kommen, schon korrekt und passend formatiert werden. Dadurch können die Parameter eines jeden Ortes einfach in das Frontend eingebunden und verwendet werden, da keine Umwandelungen und Formatierungen mehr notwendig sind. Im folgenden Screenshot sind die Definitionen der ersten Felder des Location-Objekts zu sehen:

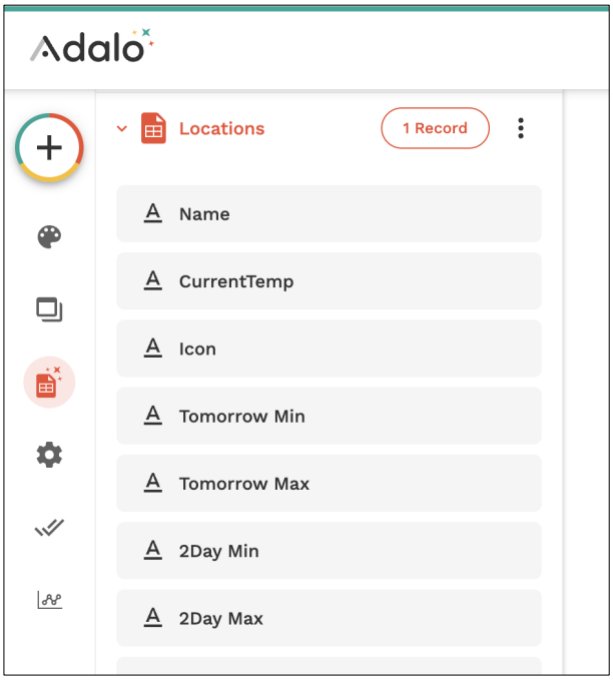


Abbildung 13: Adalo - Location Objekt

Das Hinzufügen eines neuen Ortes über die App wurde auch recht simpel gehalten. Über ein Plus-Symbol, wie in Abbildung 14 zu sehen, wird auf eine extra Seite weitergeleitet, die nur ein einfaches Texteingabe-Feld hat. Durch eine Bestätigung wird dann der neue Datensatz angelegt und der Parameter Name gesetzt. Alle anderen Werte müssen nun, wie schon angesprochen, über eine API-Integration befüllt werden.

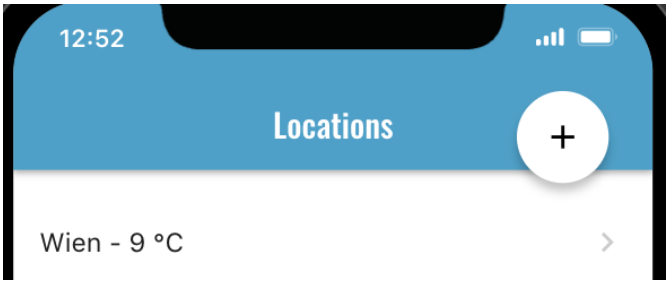


Abbildung 14: Adalo - Plus-Symbol

Wie schon in der Planung angesprochen, kann Adalo keine Custom Abfragen an eine API durchführen, weshalb ein weiteres Tool zur Unterstützung herzugezogen werden muss. Für die API-Abfrage und Speicherung der Daten wurde die NoCode Applikation „make.com“, früher auch Integromat genannt, verwendet. Es ist das am weitesten verbreitete NoCode Integrationstool, welches Datenflüsse über mehrere Plattformen hinweg ermöglicht. Mit Drag&Drop werden Bausteine, oder in diesem Fall Module, aneinandergereiht und die definierten Aktionen der Reihe nach ausgeführt.

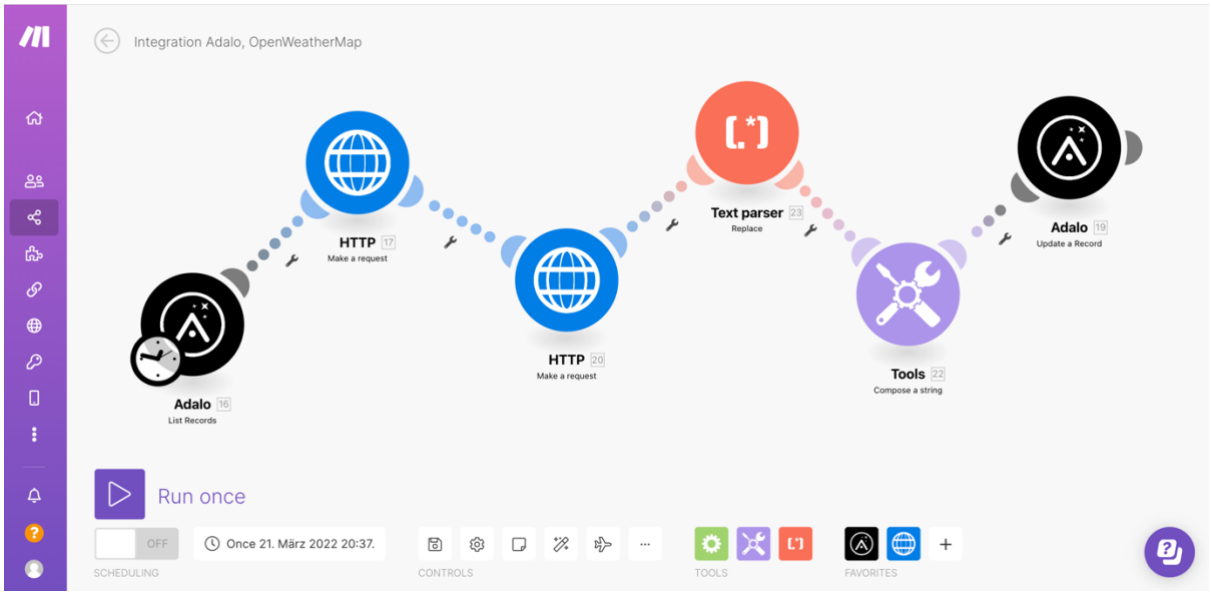


Abbildung 15: make.com - Datenfluss

In Abbildung 15 ist der Datenfluss zu sehen, welcher für die Anreicherungen der Wetterdaten für unsere Adalo App verantwortlich ist. In make gibt es für fast jedes Tool ein vordefiniertes Modul, welches mit fertigen Aktionen eingebunden und für den speziellen Anwendungsfall konfiguriert werden kann. So ist das erste Modul eine vordefinierte Adalo-Aktion, welche Daten aus der App holen kann. Im zweiten Schritt wird ein custom http-Request abgeschickt, welcher uns die initialen Wetterdaten holt, sprich hier wird die API von OpenWeatherMap aufgerufen. Danach wird ein weiterer Aufruf gestartet, welcher die Wetterprognosen und die stündliche, sowie tägliche Vorschau abfragt, welche für die Umsetzung notwendig sind. Auch hier wird wie zuvor ein custom Request an OpenWeatherMap geschickt. Hat man alle notwendigen Daten bekommen, werden Temperaturwerte und Detailinformationen zum Wetter so umformatiert, dass sie in der Adalo App ohne Problem und direkt verwendet werden können. Dazu sind die Module vier und fünf verantwortlich. Am Ende werden dann, im letzten Modul dieses Datenflusses, alle Werte auf den spezifischen Ort, auf die spezifische Location gespeichert.

Betrachtet man die Performance dieses Datenflusses und alle durchzuführenden Aktionen, so liegt die durchschnittliche Durchlaufzeit bei 1-2 Sekunden, was sehr beeindruckend ist, da hier Daten mehrfach abgefragt und wieder abgespeichert werden.

Da dieser Datenfluss nicht selbst aus der App gestartet werden kann, muss in „make“ ein Intervall konfiguriert werden, durch den dieser Datenfluss regelmäßig gestartet wird und die aktuellen Daten aus der App ausliest und die neuen Wetterdaten wieder aktualisiert. Im folgenden Screenshot ist zu sehen, wie dieses Intervall in „make“ gesetzt wird. Man sieht aber auch, dass es ein Limit für 15 Minuten gibt. Heißt, es braucht mindestens 15 Minuten bis die Wetterdaten wieder aktualisiert werden können.

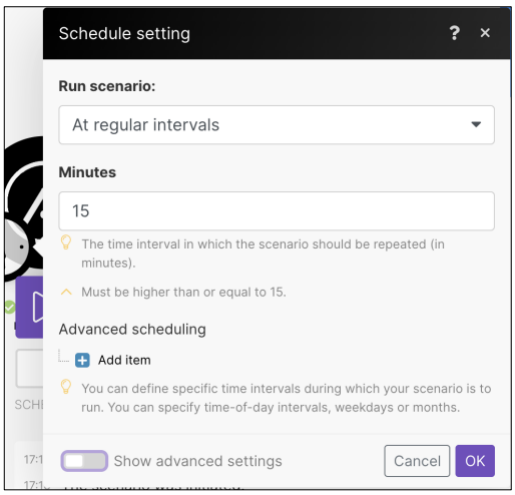


Abbildung 16: Datenfluss Intervall

Zu guter Letzt muss noch die Detailseite erstellt werden, auf der dann alle Wetterdaten und Prognosen angezeigt werden. Diese Seite kann spielerisch über Drag&Drop zusammengebaut werden, indem man die gewünschten Komponenten in den Bildschirm zieht und dann die notwendigen Werte als Platzhalter definiert. Jeder Text und jede Komponente kann ganz einfach verschoben und platziert werden. In Abbildung 17 ist genau das zu sehen. Hier wird in der Detailansicht ein Feld von der ausgewählten Location, genauer gesagt die minimale Temperatur für den nächsten Tag am Bildschirm platziert und definiert, welches Feld dieses Objektes in den Platzhalter geschrieben werden soll.

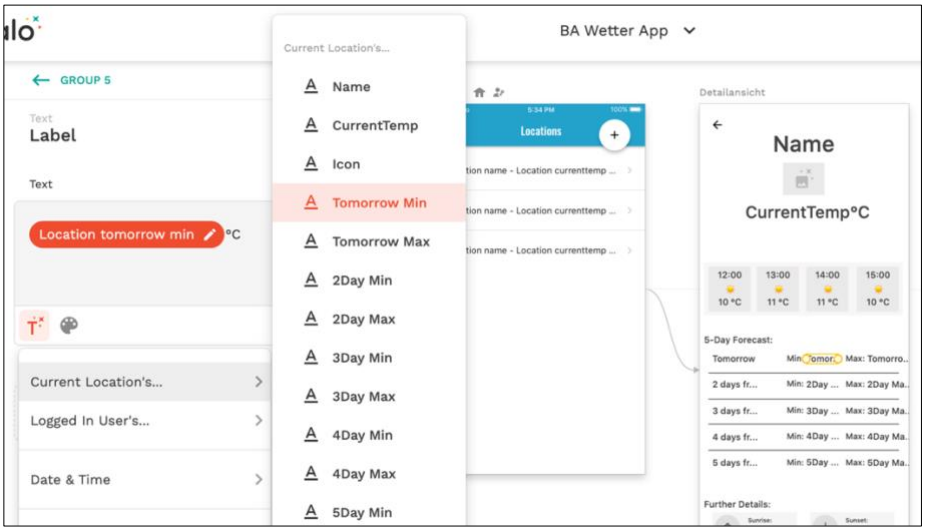


Abbildung 17: Adalo - Wetterdaten platzieren

Eine Limitierung, die auf der Detailseite auftritt, ist die Verwendung und Definition von einzelnen, horizontalen scroll-baren Bereichen. Es würde sich der ganze Bildschirm horizontal bewegen und nicht nur ein ausgewählter Bereich. Was damit genau gemeint ist, ist die Anzeige der stündlichen Prognose. Hier soll es laut Anforderungen möglich sein, die 24-Stunden Vorschau anzuzeigen, welche mit einem scrollen nach rechts eingesehen werden kann. Durch diese Limitierung macht es maximal Sinn vier bis fünf einzelne Stunden anzuzeigen, da es sonst nicht mehr lesbar wäre.

Das Endergebnis der Detailansicht, welche im Emulator des Browsers geöffnet wurde, sieht wie folgt aus:

Abbildung 18: Adalo – Detailseite (eigene Darstellung)

6. Proof-of-Concept – Xcode

In Kapitel 6 wird parallel zur NoCode Entwicklung auch die Umsetzung der Wetter App mit Xcode dokumentiert. Somit wird auch hier zuerst eine grobe Planung beschrieben, gefolgt von der tatsächlichen Umsetzung. Davor soll aber noch erwähnt sein, dass ich wie bei der NoCode App keine großen Erfahrungen damit habe, sprich noch keine vollständige Applikation mit Xcode oder der Sprache Swift entwickelt habe. Daher kann man sehr gut vergleichen, wie gut sich die Dokumentationen und Online-Hilfsmittel zur Erstellung dieser Apps nutzen lassen und ob es große Unterschiede gibt.

6.1. Planung

Der Grundansatz soll bei der Xcode Variante sein, allein mit den Mittel die Xcode standardmäßig zur Verfügung stellt die Wetter App zu erstellen, ohne dabei weitere Anwendungen oder Tools anzubinden.

Der erste Schritt ist die Startseite, welche die Liste der Orte anzeigen soll, sowie die aktuellen Temperaturen. Wie in Anforderung 1 beschrieben, soll durch einen Klick auf die Zeile, zur Detailansicht weitergeleitet werden. Der Inhalt der Zeile ist in Anforderung 4 „Startseite - Zeilen“ beschrieben. Da sich die Anforderungen und gewünschten Funktionalitäten zur NoCode App nicht unterscheiden, werde ich in diesem Unterkapitel 6.1, in Bezug auf die konkreten Anforderungen, nur die Unterschiede zur NoCode Planung anführen. Die gesamte Anforderungsliste ist wie erwähnt im Anhang einsehbar und spiegelt in gewisser Weise die Planung wider.

Zurück zu Schritt eins, hier stellt sich natürlich die große Frage, wie und wo man die Orte speichern und wieder in die App laden soll. Da keine externe Anwendungen wie eine Datenbank verwendet werden soll, bieten sich nur die internen Möglichkeiten Daten zu speichern.

„CoreData“ und „UserDefaults“ sind die beiden Optionen, die sich hier ergeben. Kurz erklärt ist CoreData ein integriertes Framework, welches wie eine interne Datenbank funktioniert und auch große Datenmengen verarbeiten kann. UserDefaults hingegen ist für kleine Datenmengen besser geeignet und wird meist für das Speichern von User Settings genutzt. Die Daten sind am Mobilgerät selbst gespeichert und beim nächsten Öffnen der App können die Informationen genau mit demselben Stand wieder abgeholt werden. Technisch gesehen, werden somit Keys definiert und mit Values (=Daten) befüllt. [46] Auf Grund der geringeren Komplexität und einfacheren Bedienung habe ich mich für die UserDefaults zur Speicherung der Orte entschieden. Gespeichert werden soll Name und Koordinaten.

Um einen Ort mit Koordinaten überhaupt speichern zu können muss eine Suchfunktion gegeben sein, welche die benötigen Werte, sprich die Koordinaten, zurückliefert. Dazu bietet Xcode die Möglichkeit „MapKit“ einzubinden. Ein Standard-Framework, das einem ermöglicht Orte und Sehenswürdigkeiten zu suchen. Zum Beispiel wird für die Suchen und Anzeigen der Standard iOS Karten-App auch dieses Framework verwendet, welches einem hier frei zur Verfügung steht.

[46] Apple, „UserDefaults | Apple Developer Documentation“.

Da nun Orte mit korrekten Koordinaten zur Verfügung stehen sollten, wäre der nächstgeplante Schritt, die Anbindung der OpenWeatherApi. Zum einen benötigt man die Wetterdaten auf der Startseite für das Anzeigen der Information auf der Liste und zum anderen werden die Wetterdaten auf der Detailseite selbst benötigt.

Sind die Wetterdaten in der App verfügbar, müssen diese in einem letzten Schritt richtig verarbeitet und angezeigt werden. Neben dem grundlegenden Design, sprich die Anordnung und korrekte Platzierung der abgefragten Werte, ist noch die Funktionalität der Wettersymbole und des Hintergrundbildes der Detailseite offen. Hierbei können spezielle Werte aus der API-Abfrage als Referenz hergenommen werden, mit denen ein Mapping auf Vordefinierte Symbole und Hintergrundbilder durchgeführt werden kann.

6.2. **Erstellung**

Wollen wir uns in diesem Kapitel nun der konkreten Umsetzung und Lösung spezieller Anforderungen widmen. Um loszulegen, muss zuerst ein Xcode Projekt angelegt werden, welches für die iOS Entwicklung definiert ist. Nach dem Auswählen eines iOS Templates, in diesem Fall, eine Standard-App, wird auch eine Basis File Struktur mit default Dateien erstellt, mit denen gestartet werden kann.

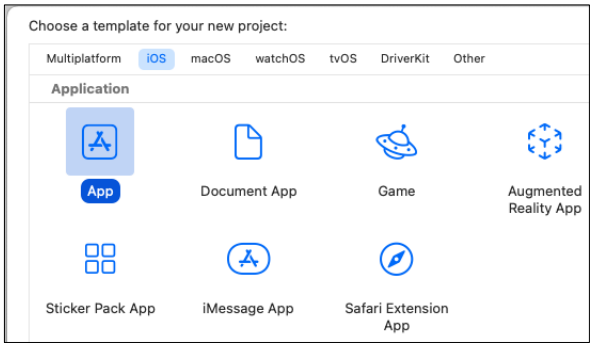


Abbildung 20: Xcode Template

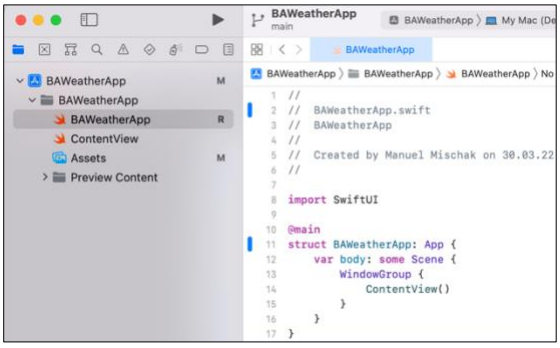


Abbildung 19: Xcode Basis File Struktur

Wie schon zuvor angemerkt, werden die Orte mittels UserDefaults abgespeichert und wieder in die App reingeladen. Dafür muss ein DTO, Data Tranfer Object, definiert werden in dem die Informationen des Ortes enthalten sind. Damit aber nicht für jeden Ort ein eigener Key in den UserDefaults angelegt werden muss, wird ein Array „locationList“ verwenden, in dem jeder Ort enthalten ist.

Das DTO für einen einzelnen Ort wäre einfach aufgebaut und würde wie folgt ausschauen:

- Id: Unique Identifier
- Name: String
- Latitude: Double
- Longitude: Double

Wird vom Benutzer der App ein neuer Ort der List hinzugefügt oder von der Liste entfernt, so wird auch das Array „locationList“ dementsprechend aktualisiert und in den UserDefaults des iOS Gerätes als JSON String wieder abgespeichert.

Um Orte mit vollständigen Informationen hinzuzufügen, müssen diese zuerst gesucht und gefunden werden. Wie schon in Kapitel 6.1 erklärt, wird dazu das Standard Framework MapKit verwendet. Über ein einfaches Input Feld wird der zu suchende Text in einem Request an die Standard Suche weitergegeben. Theoretisch kann man mit dem MapKit auch auf Features wie GeoLocating zugreifen, sprich man könnte den aktuellen Standort des Gerätes finden und als Standard-Ort für die Wetterabfrage hernehmen. Aufgebaut ist dieser Request wie folgt, wobei „searchText“ den eingegebene Ort widerspiegelt:

```
private func request(searchText: String) {
    let request = MKLocalSearch.Request()
    request.naturalLanguageQuery = searchText
    request.pointOfInterestFilter = .includingAll
    request.resultTypes = .address
    request.region = MKCoordinateRegion(center: center,
                                        latitudinalMeters: radius,
                                        longitudinalMeters: radius)
    let search = MKLocalSearch(request: request)
```

Abbildung 21: Ort suchen – Request (eigene Darstellung)

In der App selbst ist unter der Texteingabe, der vorgeschlagene Ort angezeigt, welcher mit einem Klick der Liste hinzugefügt werden kann. Durch das Klicken auf den, von MapKit zurückgegebenen Ort, wird die locationList sofort aktualisiert und beim Zurückspringen auf die Startansicht ist der Ort direkt sichtbar und eine Abfrage der Wetterdaten ist möglich. Im Nachfolgenden Screenshot ist die Umsetzung in der App zu sehen:

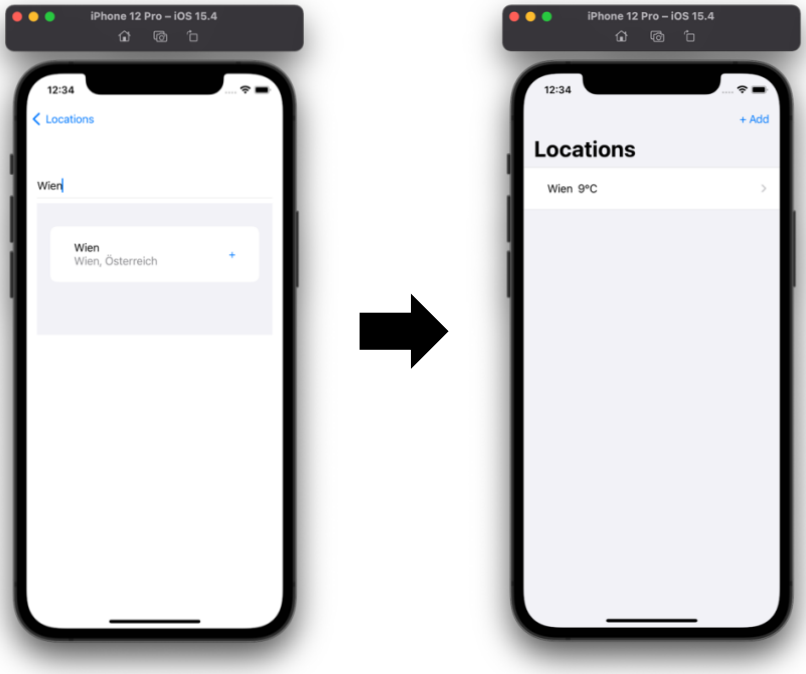


Abbildung 22: Xcode - Ort hinzufügen (eigene Darstellung)

Nun da neue Ort der Liste hinzugefügt und gespeichert werden können, ist der wichtigste Teil der Applikation an der Reihe, die Abfrage der aktuellen Wetterdaten. Wie schon im Kapitel zuvor erwähnt, wird dazu die API von OpenWeatherMap verwendet. Im konkreten wurden zwei Abfragen ausgewählt, die zu den Anforderungen passen. Für die Temperaturanzeige auf der Startliste wurde die „CurrentWeather"-Abfrage ausgewählt, da diese nur wenige Daten enthält und schnell bzw. einfach zu laden ist. Für die Detailseite hingegen wird die „OneCall" – Abfrage verwendet. Wie der Name schon sagt, hat dieser Aufruf den großen Vorteil, dass mit nur einem Call sehr viele Informationen zu einem Ort geladen werden können, dazu zählen eine Stündliche Vorschau, Tagesvorschauen, oder auch weitere Detaildaten zum aktuellen Wetter. Als Request-Parameter nimmt dieser Aufruf die Longitude und Latidute Werte des gewünschten Ortes entgegen.

Um die Antwort der API lesen zu können, muss der Response deserialisiert werden. Sprich, es wird versucht den gelieferten JSON String mit einem definierten DTO in ein Objekt umzuwandeln, mit dem dann weitergearbeitet werden kann. Dieses DTO muss genau dem Ergebnis der API entsprechen, sonst würde das Auslesen und Einbinden der Wetterdaten nicht funktionieren. OpenWeatherMap bietet auf der Website zu jedem Aufruf eine genaue Beschreibung des Ergebnisses an, und beschreibt die jeweiligen Felder mit den dazugehörigen Feldtypen.

Im folgenden Ausschnitt ist ein Teil des „OneCall"-Response DTO zu sehen und wie es in Xcode angelegt wurde. Auf jedes dieser Felder kann dann einfach zugegriffen werden, wie zum Beispiel „wind_speed". Dieses Feld wird problemlos in den weiteren Details der App angezeigt.

```
struct OneCallWeatherDto: Decodable, Identifiable{
    var current: CurrentResponse
    var hourly: [HourlyResponse]
    var daily: [DailyResponse]
    var id: UUID?

    struct CurrentResponse: Decodable {
        var sunrise: Int
        var sunset: Int
        var temp: Double
        var feels_like: Double
        var pressure: Int
        var humidity: Int
        var wind_speed: Double
        var rain: RainDto?
        var weather: [WeatherResponse]
    }
```

Abbildung 23: Ausschnitt API-Response DTO (eigene Darstellung)

Eine weitere komplexere Aufgabe, war es den Hintergrund der Detailseite an das aktuelle Wetter anzupassen und das Bild somit dynamisch anhand den jetzigen Bedingungen zu setzen. Durch den OneCall Aufruf der API, ist über ein Feld die spezifische Wetterlage bekannt, mit dem das Mapping auf das jeweilige Bild stattfindet. Beispielsweise wird bei den zurückgegeben Wert „Clouds", ein Hintergrundbild eingeblendet, welches ein bewölktes Wetter widerspiegelt. Insgesamt kann zwischen fünf verschiedenen Bildern unterschieden werden, die folgende Wetterlagen darstellen sollen:

- Klarer Himmel, Sonnenschein
- Bewölkt, bedeckter Himmel
- Regen
- Gewitter
- Schneefall

Beispielhaft werden in den folgenden Screenshot drei Szenarien dargestellt, wo man die Anpassung des Hintergrunds beobachten kann. Im ersten Fall herrscht Sonnenschein, deshalb ist auch ein klarer Himmel als Hintergrund gesetzt. Im zweiten Bild ist ein bewölkter Himmel zu sehen und im dritten Fall regnet es gerade an dem gesuchten Ort. Der korrekte Hintergrund wird im Moment des öffnens der Detailansicht geladen, so kann man auch zwischen Orten wechseln und es könnte überall ein anderer Hintergrund angezeigt werden, je nach Wetterlage natürlich:

Abbildung 24:Dynamischer Hintergrund (eigene Darstellung)

In den Screenshots zuvor sieht man auch sehr gut, wie sich die Simulation bei der Erstellung der App verhält. Es wird Live auf einem komplett simuliertem iPhone die Applikation geladen und gestartet. Man kann jede Geste und Bewegung ausführen, sowie es auch bei einem echtem iPhone der Fall wäre. Diese Emulationsmöglichkeit ist in Xcode standardmäßig verfügbar und kann während der Entwicklung nebenbei verwendet werden, da bei Code-Änderungen sofort eine Aktualisierung durchgeführt wird. Es wäre auch möglich, die Emulation auf einem echten verbunden iPhone laufen zu lassen und sich dort den aktuellen Stand der Entwicklung anzuschauen.

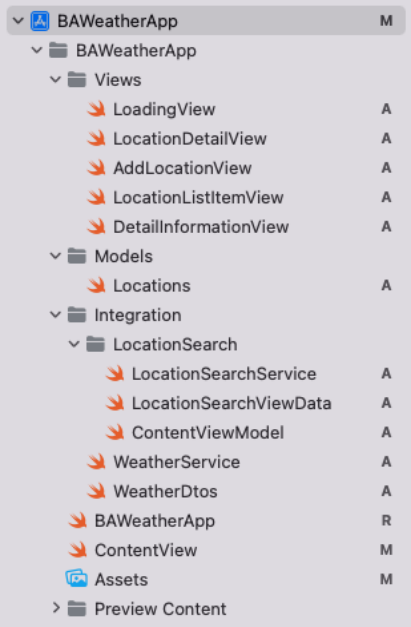


Abbildung 25: Xcode Endgültige File-Struktur (eigene Darstellung)

Da wir zu Beginn kurz die Basis File Struktur gesehen haben, möchte ich hier nochmal die endgültige Strukturierung und die gesamten Dateien nochmal darstellen. Von oben beginnend, die Views, welche die optischen und Designelemente beinhalten, man kann hier in gewisser Weise vom Frontendteil der App sprechen. Im zweiten Ordner, „Models“, sind nur die Locations mit dem Definieren DTO und der Kommunikation zu den UserDefaults. Im Teil Integration, gibt es zum einen den Unterordner „LocationSearch“, in dem die Suche nach den Orten, bzw. deren Koordinaten zusammengefasst wurde. Außdem ist auch der Wetterservice, welcher mit der API interagiert, sowie die Wetter DTOs im Ordner Integration enthalten. Im Punkt „Assets“ sind alle vordefinierten Hintergrundbilder als .jpeg hinterlegt, und er gilt als genereller Ablageort für Bilddateien, die in der App verwenden werden.

7. Auswertung der beiden Varianten

In Kapitel 7 werden nun die Ergebnisse, sprich die iOS Apps der beiden Varianten ausgewertet und verglichen. Zum einen wollen wir die tatsächliche Umsetzungsdauer im Vergleich zur Schätzung und generellen Annahme betrachten, sowie die Erfüllung der Anforderungen und Limitierungen, die bei der Entwicklung der beiden Apps aufgekommen sind. Alle Anforderungen sind in Listenform als Anhang am Ende der Arbeit zu finden.

7.1. Xcode

Wollen wir zunächst auf die Umsetzung der Xcode App schauen. Im Großen und Ganzen war ich wirklich beeindruckt, wie gut die Xcode Entwicklung funktioniert hat. Vor allem, da ich selbst zuvor noch nicht allzu viel Erfahrung in der iOS Entwicklung hatte. Begonnen von den umfangreichen frei zugänglichen Dokumentationen und den Informationen, welche über Foren oder Blogs von der Community zur Verfügung gestellt wird, bis hin zur Programmierung an sich und dem Testen über den integrierten Simulator. Die eigens für Xcode entwickelte Programmiersprache Swift war, auch mit Hilfe der zuvor genannten Dokumentationen, intuitiv und einfach anzuwenden.

Mit der Xcode Variante wurden somit alle Anforderungen umgesetzt und erfüllt. Von der Wetterdaten Integration und Abfrage bis zu komplexeren UI Aufgaben, wie der angepasste Hintergrund, oder die Anzeige des stündlichen Forecasts, konnte alles problemlos umgesetzt werden. Klarerweise ist man zwischendurch auf kleine Schwierigkeiten gestoßen, die aber alle samt nach einer kurzen Recherche in den Dokumentationen von Xcode, oder einer Recherche im Internet, schnell gelöst werden konnten. Das Interessanteste dabei ist aber die Dauer der Umsetzung, welche um einiges kürzer ausgefallen ist als zu Beginn angenommen. Themen wie Design und Frontend Erstellung wurden unter anderem durch die Verwendung von Swift sehr vereinfacht und deshalb viel schneller umgesetzt als in den Schätzungen gedacht. Der Hauptaufwand lag größtenteils in der Datenanreicherung, -verarbeitung und -aufbereitung, sowie dem generellen strukturellen Aufbau der Applikation.

Um das Ganze zu konkretisieren: Die Gesamtzeitschätzung aus der Anforderungsliste betrug 122 Stunden für die Xcode Variante. Effektiv für die Entwicklung aufgebracht wurden dann aber nur circa 60 Stunden, was weniger als die Hälfte der angenommenen Zeit ist.

7.2. NoCode

Betrachten wir nun das Ergebnis der NoCode Entwicklung. Hier fällt als Erstes auf, dass nicht alle Anforderungen umgesetzt werden konnten, bzw. einige Workarounds und Alternativlösungen für speziellere Aufgaben benötigt wurden. Sehr positiv zu sehen, ist die einfache Anmeldung und initiale Erstellung der App. Mit nur wenigen Klicks hat man seine neue App auf dem Bildschirm und kann direkt loslegen die App zu simulieren und kann sie theoretisch auch schon veröffentlichen, alles nur in ein paar Minuten.

Durch die benötigten Workarounds und Recherchen zur Lösung der einen oder anderen Limitierung, erhöhte sich auch die Dauer der Umsetzung, was eigentlich der Hauptvorteil der

NoCode Variante sein sollte. Wenn wir auch hier die Zeitschätzung der Anforderungsliste als Referenz nehmen, welche 50 Stunden betrug, und mit der tatsächlichen Dauer der Gesamtumsetzung vergleicht, so liegen wir, wie bei der Xcode Variante, unter der Zeitschätzung, aber bei weitem nicht so deutlich. Insgesamt flossen also circa 48 Stunden in die Umsetzung der NoCode App, was fast der Zeitschätzung entspricht.

Dazu muss aber noch gesagt werden, dass die 50 Stunden Schätzung als Zielsetzung hatte, alle Anforderungen umzusetzen, was am Ende aber nicht möglich war. Betrachtet man die wirklich angefallene Dauer der Entwicklung mit der Tatsache, dass nicht alle Anforderungen umsetzbar waren, muss der zeitliche Faktor negativer bewertet werden als zu Beginn angenommen. Natürlich unter der Annahme, dass man nicht schon viel Erfahrung mit dem ausgewählten NoCode Tool hat, da man sich dann viele Überlegungen und Suchen nach Alternativen sparen kann.

Wenn wir die Limitierungen und Grenzen der NoCode App zusammenfassen, sind im Wesentlichen zwei Punkte hervorzuheben. Zum einen das dynamische Ändern und Anpassen des Hintergrundbildes auf einer einzelnen Seite, sowie das Scroll-Verhalten ausgewählter Bereiche, sprich gewisse Freiheiten in der Gestaltung auf der Frontendseite. Zum anderen noch die nicht vorhandene Möglichkeit, direkt aus Adalo Requests an externe Schnittstellen abschicken zu können. Aufgrund des letzteren Punktes musste in diesem Fall sogar eine weitere NoCode Anwendung als Workaround angebunden werden, was die Wartbarkeit der App erschwert und die Gesamtkomplexität erhört.

7.3. **Vergleich**

In Kapitel 4.2 wurden zu Beginn zwei Bereiche definiert, welche nun nach der Umsetzung der beiden Apps ausgewertet werden können. Betrachtet wir den ersten Parameter, den zeitlichen Aufwand, welcher unter anderem Einarbeitungszeit und Umsetzungsdauer beinhaltet. Auf der einen Seite kann die Dauer mit der initialen Schätzung verglichen werden, sowie die Gesamtdauer der beiden Apps untereinander.

NoCode:
- Schätzung: 50 Stunden
- Gesamtaufwand: 48 Stunden

Xcode:
- Schätzung: 122 Stunden
- Gesamtaufwand: 60 Stunden

Betrachtet man die Schätzung sowie den tatsächlichen Aufwand, so kann man gut erkennen, dass vor allem die Xcode Variante um einiges schneller umgesetzt wurde als zuerst angenommen. Das Ganze hängt natürlich stark vom Detailgrad ab, denn man im Frontend am Ende erreichen möchte. Im Gegensatz dazu hat die Dauer der NoCode Entwicklung circa dem entsprochen, was angenommen wurde. Untereinander verglichen, hat die Xcode Entwicklung 1,5 Tage länger gedauert als die NoCode Entwicklung, was in anbetracht des Ergebnisses kein wesentlicher Unterschied ist.

Wenn man nun auch den zweiten Parameter, die Erfüllung der Anforderungen und den Umfang der Funktionalität, brachtet, kann man einen größeren Unterschied erkennen. Dort wo bei der Xcode Variante alle Anforderungen und Funktionalitäten mit den Standardmitteln von Xcode implementiert wurden, gab es bei der NoCode Variante die zu zuvor angesprochenen Limitierungen und Problemstellungen, die teilweise über Workarounds gelöst wurden, oder gar überhaupt nicht umgesetzt werden konnten. Nicht umgesetzt, oder nur teilweise umgesetzt wurden somit Anforderung 11 und 7. Da zwei Punkte, sprich Anforderungen nicht umgesetzt werden konnten, sind die 1,5 Tage die für Xcode länger benötigt werden kein Nachteil im Vergleich, weil alle Anforderungen möglich waren.

Das Verwenden der API und des zusätzlich benötigten NoCode Tools verursachten im Proof-Of-Concept Aufbau keine Kosten, genauso wie bei der Xcode Variante. Möchte man aber die Applikation vollständig veröffentlichen, so müssten die Lizenzkosten erweitert werden und je nach Benutzerzahlen aufgestockt werden. Hier kann aber grundsätzlich, auch ohne konkrete Nutzerzahlen, davon ausgegangen werden, dass die NoCode Variante einen höheren Kostenaufwand als die programmierte Variante verursacht. Allerdings sind die Kosten für die Ressourcen, welche die iOS App entwickeln oft deutlich höher.

Im Endeffekt kann als Grundregel angenommen werden, dass man nur einen wesentlichen zeitlichen Vorteil bei kleinen Anwendungsfällen, welche keine externe Anbindung an eine API oder ähnliches benötigt, hat. Dann wird der Gesamtzyklus von Erstellung der Applikation bis hin zur Veröffentlichung der App in den meisten Fällen schneller sein als eine programmierte Variante.

8. Reflexion der Ergebnisse

Im letzten Kapitel werden die wesentlichen Aussagen noch mal übersichtlich zusammengefasst, die Forschungsfrage beantwortet und ein kurzer Ausblick gegeben.

8.1. Beantwortung der Forschungsfrage

Folgende Forschungsfrage gilt es zu beantworten: „Unter welchen Kriterien ist die Entwicklung einer iOS App mit NoCode Tools besser geeignet als die Entwicklung mit einer code-basierten Herangehensweise über Xcode?“

Die dazu aufgestellte Hypothese war folgende: „Bei der Entwicklung einer iOS App über NoCode ergibt sich im Wesentlichen ein zeitlicher Vorteil. Bei einem zeitlichen Kriterium ist somit die NoCode Variante besser geeignet. Gegenüber der programmierten Variante ergibt sich aber der Nachteil, dass man vor allem in der visuellen Gestaltung eingeschränkter ist. So ist bei dem Kriterium, welches genaue Designvorgaben beinhaltet, die programmierte Variante besser geeignet.“

Nach der Umsetzung und Auswertung der beiden App lässt sich die Forschungsfrage wie folgt beantworten: Die Entwicklung einer iOS App ist mit NoCode Tools besser geeignet, wenn man einen schnellen Gesamtprozess, sprich schnell von der Idee bis hin zur Veröffentlichung der App kommen möchte, dabei aber keine externen Anbindungen und Schnittstellen benötigt. Die wesentlichen Kriterien, die also für die NoCode Entwicklung sprechen sind, wenn es zeitlichen schnell gehen muss und, oder, wenn man keine Ressourcen mit Programmiererfahrung zur Verfügung hat. Das zeitliche Kriterium kann aber vernachlässigt werden, wenn komplexere Aufgaben oder Schnittstellen zur Anforderungsliste zählen, weil dann die Xcode Entwicklung in den meisten Fällen auch von der Umsetzungsdauer gleich ist, sofern man etwas mit der Programmierung von Software vertraut ist und sich schnell in das Thema Xcode und Swift einarbeiten kann. Ist das wichtigste Kriterium die visuelle Gestaltung und die Designanforderung, so sollte auf die programmierte Variante und Xcode gesetzt werden, da hier keinerlei Limitierungen und Grenzen vorhanden sind, welche die iOS App Entwicklung betrifft.

8.2. Ausblick

Abschließend möchte ich noch einen kleinen Ausblick zu NoCode Tools und deren Anwendbarkeit im Vergleich zu programmierten Varianten geben. Diese noch sehr junge Technologie wird sich in den nächsten Jahren sehr stark verbessern und viele der hier angesprochenen Limitierungen und Grenzen werden vielleicht keine Probleme mehr darstellen. Die Lücke zwischen der programmierten Xcode Variante und der NoCode Variante wird mit jeder Iteration geringer, denn die Funktionalitäten und der Umfang dieser Tools wird stetig besser werden.

Die heut zu tage noch etwas geringere Akzeptanz wird mit Sicherheit in der breiter Masse steigen und NoCode wird immer mehr als relevante Option bei der Methodenwahl zur Entwicklung neuer Software wahrgenommen.

Dennoch wird es aber so sein, dass man nicht vollkommen ohne Programmierungen auskommen wird. Vor allem bei komplexeren und umfangreicheren Aufgabenstellungen muss man auf die klassische Entwicklung zurückgreifen, da man nur so alle Möglichkeiten und Freiheiten hat.

Literaturverzeichnis

Adalo. „About Our Mission, Vision and Team“. Zugegriffen 5. Dezember 2021. https://www.adalo.com/our-story.

———. „Adalo - Build Your Own No Code App“. Zugegriffen 15. Jänner 2022. https://www.adalo.com/.

Airtable. „Airtable Website“. Zugegriffen 5. Dezember 2021. https://www.airtable.com/.

Apple. „Submit your iOS and iPadOS apps to the App Store - Apple Developer“. Zugegriffen 5. Dezember 2021. https://developer.apple.com/ios/submit/.

———. „Swift - Apple Developer“. Zugegriffen 5. Dezember 2021. https://developer.apple.com/swift/.

———. „Technologies | Apple Developer Documentation“. Zugegriffen 5. Dezember 2021. https://developer.apple.com/documentation/technologies.

———. „UserDefaults | Apple Developer Documentation“. Zugegriffen 24. April 2022. https://developer.apple.com/documentation/foundation/userdefaults.

———. „Xcode | Apple Developer Documentation“. Zugegriffen 5. Dezember 2021. https://developer.apple.com/documentation/xcode.

———. „Xcode Release Notes | Apple Developer Documentation“. Zugegriffen 5. Dezember 2021. https://developer.apple.com/documentation/xcode-release-notes.

Beranič, Tina, Patrik Rek, und Marjan Heričko. „Adoption and Usability of Low-Code/No-Code Development Tools“, 2020, 7.

Bubble.io. „Bubble Docs - Building Workflows“. Zugegriffen 5. Dezember 2021. https://manual.bubble.io/help-guides/building-workflows/the-basics.

Glide. Glideapps Homepage. Zugegriffen 12. Oktober 2021. https://www.glideapps.com/.

Harvard Business Review. „Empower Your Team with No-Code/Low-Code Software Applications“, April 2021. https://hbr.org/sponsored/2021/04/empower-your-team-with-no-code-low-code-software-applications.

Hiren Amin und Saurabh Kapoor. „KPMG - Opportunities and Challenges in a Low/No-Code World“. KPMG, 2. Juni 2021. https://home.kpmg/ae/en/blogs/home/posts/2021/06/opportunities-and-challenges-in-a-low-no-code-world.html.

IBM Corporation Software Group. „Native, Web or Hybrid Mobile-App Development“, April 2012, 12.

Jabangwe, Ronald, Henry Edison, und Anh Nguyen Duc. „Software Engineering Process Models for Mobile App Development: A Systematic Literature Review“. *Journal of Systems and Software* 145 (November 2018): 98–111. https://doi.org/10.1016/j.jss.2018.08.028.

Khorram, Faezeh, Jean-Marie Mottu, und Gerson Sunyé. „Challenges & Opportunities in Low-Code Testing“. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 1–10. Virtual Event Canada: ACM, 2020. https://doi.org/10.1145/3417990.3420204.

Luo, Yajing, Peng Liang, Chong Wang, Mojtaba Shahin, und Jing Zhan. „Characteristics and Challenges of Low-Code Development: The Practitioners' Perspective“. In *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 1–11. Bari Italy: ACM, 2021. https://doi.org/10.1145/3475716.3475782.

Mary K., Pratt. „What Are Low-Code and No-Code Development Platforms?“ SearchSoftwareQuality, März 2021. https://searchsoftwarequality.techtarget.com/definition/low-code-no-code-development-platform.
Merenych, Sofiya. „What Is the Mobile App Development Process Steps? 7 Main Stages of App Development - Clockwise Software“, 27. August 2021. https://clockwise.software/blog/mobile-app-development-process/.
OpenWeather. „Weather API - OpenWeatherMap“. Zugegriffen 24. April 2022. https://openweathermap.org/api.
OutSystems. „The State of Application Development“. London, United Kingdom, 2020 2019. https://www.outsystems.com/local-gov/-/media/053D5BCC32364C2993C8D0BAFA880DB1.ashx.
Randleff, Veronica. *Native App vs Web App: Multi-Criteria Decision-Making for Optimised Mobile Solution*, 2018. http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-232001.

Anhang

Anforderungen an Funktionalität, Verhalten und Design - Wetter App

	Bereich	Titel	Beschreibung	Zeitschätzung NoCode (h)	Zeitschätzung Xcode iOS (h)	Anmerkung
1	Design / Funktionalität	Startseite	Auf der Startseite soll sich eine Liste an gespeicherten Orten befinden, welche auf eine Detailseite weiterleiten sollen.	1	4	
2	Design / Funktionalität	Startseite - neuen Ort hinzufügen	Ein Plus-Symbol soll angezeigt werden, mit dem ein neuer Ort der Liste hinzugefügt werden kann. Durch das auswählen des gesuchten Ergebnisses wird dieser Ort direkt der Liste hinzugefügt.	2	6	
3	Design / Funktionalität	Startseite - Ort entfernen	Hinzugefügte Orte zur Startseite sollen auch wieder entfernt werden können, was über ein entsprechenden Minus Symbol, welches Rot hinterlegt ist durchgeführt werden kann.	1	4	
4	Design	Startseite - Zeilen	Die Inhalte der einzelnen Zeilen sollen von links nach rechts: den Namen des Ortes, das aktuelle Wetter in Form eines Symbols und die aktuelle Temperatur in Grad Celsius.	1	6	
5	Design	Detailseite	Die Detailseite ist grob in drei Teile unterteilt die den aktuellen Wetterstand, eine 24h Prognose und eine 5-tages Vorschau des ausgewählten Ortes anzeigt. Diese drei Bereiche sollen ohne zu scrollen immer Sichtbar sein, und sich von der Größe her so anpassen, dass sie bei gängigen Bildschirmgrößen alles ausfüllen. Es soll weiters die Möglichkeit geben dem Benutzer durch ein scrollen nach unten, noch weitere Details anzeigen zu lassen.	-	-	
6	Design	Detailseite - Oberer Teil	Im oberen Bereich des Bildschirms soll ganz oben der Name des Ortes und die Aktuelle Temperatur zentriert angezeigt werden. Sodass im ersten Blick ersichtlich ist, wie viel Grad es hat und ob man den richten Ort ausgewählt hat.	1	2	
7	Design / Funktionalität	Detailseite - Mittlere Teil	Im mittleren Bereich der Detailansicht soll ein stündlicher Forecast des ausgewählten Ortes angezeigt werden, welcher für die nächsten 24h in die Zukunft die vorausgesagte Temperatur und das Wetter in Form eines Symbols anzeigen soll. Die einzelnen Stunden sollen von links nach rechts anzeigt werden und mit einem Scrollen nach rechts sollen kann die Prognose betrachtet werden.	6	14	Anbindung an die Wetter API und synchrones befüllen der Daten
8	Design / Funktionalität	Detailseite - Unterer Teil	Im unteren Bereich der Detailansicht soll eine 5 Tagesvorschau ersichtlich sein, welche die minimale und maximale Temperatur des Tages anzeigt.	6	14	Anbindung an die Wetter API und synchrones befüllen der Daten
9	Design / Funktionalität	Detailseite - weitere Details	Der durch das scrollen nach unten erreichbare Bereich soll in zwei Spalten weitere Details zum Wetter des ausgewählten Ortes anzeigen. Im Konkreten geht es um: Regenmenge, Windstärke, Luftfeuchtigkeit, Sonnenauf- und untergang, sowie die gefühlte Temperatur.	4	10	Anbindung an die Wetter API und synchrones befüllen der Daten
10	Funktionalität	Detailseite - Zurückspringen	Um von der Detailseite wieder auf die Startseite zurückspringen zu können, soll links oben ein Pfeil angezeigt werden, welcher beim Anklicken auf die Startseite zurückleitet.	1	2	

11	Design / Funktionalität	Detailseite - Hintergrund	Der Hintergrund der Detailseite soll sich anhand des aktuellen Wetters anpassen. Bei zum Beispiel wolkenlosem Wetter, soll auch der Hintergrund dementsprechend mit vordefinierten Bildern befüllt werden.	8	16	
12	Daten	Ort	Orte sollen mit den Parametern (Name, Latitude, Longitude) lokal auf dem Mobilen Endgerät gespeichert werden und beim nächsten öffnen der Applikation wieder zur Verfügung stehen.	4	16	Aufbau der Entität und der Datenstruktur
13	Daten / Funktionalität	Wetter	Die Wetterdaten und Informationen sollen immer von einer API abgefragt werden und direkt befüllt werden, damit bei jeder Aktion die aktuellsten Daten zur Verfügung stehen und verwendet werden.	10	16	Initialisierung und Konfiguration der API Connection - hierzu soll die API von https://openweathermap.org verwendet werden
14	Design	Wetter - Symbol	Alle Symbole die für das aktuelle Wetter oder für Wettervorhersagen eingeblendet werden, sollen je Wetterlage entsprechend angezeigt werden. So dass bei klaren Himmel und Sonnenschein auch eine Sonne als Symbol, und bei Regen, eine Regenwolke als Symbol verwendet werden. Insgesamt werden von der Wetter API neun verschiedene Wetter Codes verwendet, was wiederum die Anzahl der möglichen Symbole weiderspiegelt.	5	12	Symbole werden als Bilder in der Applikation hinterlegt, und in einem vordefinierten Platzhalter eingefügt.
15	Technisch	Testbarkeit	Die Entwicklungsplattform, soll eine Möglichkeit bieten, den aktuellen Stand der Entwicklung schnell und einfach in Form einer Simulation der App zu testen.	-	-	
16	Technisch	Responsive	Die Applikation soll wie zu erwarten über ein Responsives Design verfügen, ohne dazu Anpassungen im Zuge der Implementierung durchführen zu müssen.	-	-	
			Gesamt Zeitschätzung	50	122	