Mobile App Entwicklung im Überblick

Es scheint fast für jeden etwas dabei zu sein. So oder ähnlich kann man die Situation beschreiben, wenn es um Technologien, Frameworks und Vorgehensweisen zur Entwicklung von Mobile-Apps geht. Wir haben versucht, den Überblick zu behalten.

Redet man über die Entwicklung von Apps für mobile Systeme, dann meint man damit die Entwicklung für Android und iOS. Doch wie kommt man zu einer solchen App? Hier können wir gleich am Anfang des Artikels festhalten, dass es für dieses Ziel eine sehr große Anzahl sehr unterschiedlicher Technologien und Vorgehensweisen gibt. Wir möchten in diesem Artikel einen Überblick über die verschiedenen Möglichkeiten geben und dabei versuchen, diese unterschiedlichen Ansätze nach bestimmten Kriterien zu systematisieren. Starten wir zunächst mit den Kriterien.

Systematik der Entwicklungsansätze

Bei der Entwicklung von Mobile-Apps kann man unterschiedliche Kriterien für die Systematisierung heranziehen. Nach den folgenden Kriterien ist eine Einteilung möglich

- nach der Art der entstehenden App, d. h. nativ, webbasiert oder hybrid
- nach der verwendeten Programmiersprache und den zum Einsatz kommenden Framework
- nach dem Umfang der Toolunterstützung, d. h. einer codebasierten Erstellung oder dem Einsatz von RAD- oder Low-Code-Tools
- nach dem Zielsystem oder ob mit Cross-Platform- bzw. Cross-Device-Programmierung mehrere Systeme aus einer gemeinsamen Quellcodebasis angesprochen werden

Weitere Kategorien sind sicherlich denkbar, die meisten relevanten Ansätze dürften sich jedoch hierunter subsumieren lassen. Die folgenden Textabschnitte stellen einige interessante Ansätze vor und versuchen sie entsprechend einzuordnen. Das wird mit Blick auf die Kriterien nicht immer überschneidungsfrei gelingen, dennoch dürfte auf diese Weise ein umfassender Überblick über die aktuellen Technologien zur Entwicklung von mobilen Apps entstehen. Starten wir mit dem naheliegendsten Kriterium, der Art der entstehenden App.

Native Apps

„Entscheidend ist, was hinten rauskommt“, meinte der ehemalige Bundeskanzler Helmut Kohl – zwar in einem vollständig anderen Zusammenhang, wir können diesen Ausspruch jedoch auch auf den Bereich der Entwicklung von Mobile-Apps übertragen. Man kann technologisch grundsätzlich drei Arten von Apps unterscheiden: native Apps, Web-Apps und hybride Apps.

Native Apps sind die Applikationen, deren Programmcode unmittelbar durch das Betriebssystem auf dem mobilen Gerät ausgeführt wird. Sie laufen in keinem Browser, sondern greifen direkt auf die Programmierschnittstellen von Android und iOS zurück. Mit dem direkten Zugriff auf die APIs der Systeme ist eine vollumfängliche Nutzung aller Geräte- und Hardwarefunktionen möglich. Beispielsweise kann man alle Sensoren ohne Einschränkungen verwenden. Auch die Ausführungsgeschwindigkeit der App ist die bestmögliche. Native Apps können offline betrieben werden und man kann sie über die App-Stores vertreiben. Wie gelangt man zu diesen nativen Apps? Als Erstes sind hier die von den Herstellern der Betriebssysteme vorgesehenen Vorgehensweisen und Tools zu nennen. Für Android ist es die Entwicklungsumgebung Android Studio [1] mit der Programmiersprache Kotlin bzw. Java (**Abb. 1**).

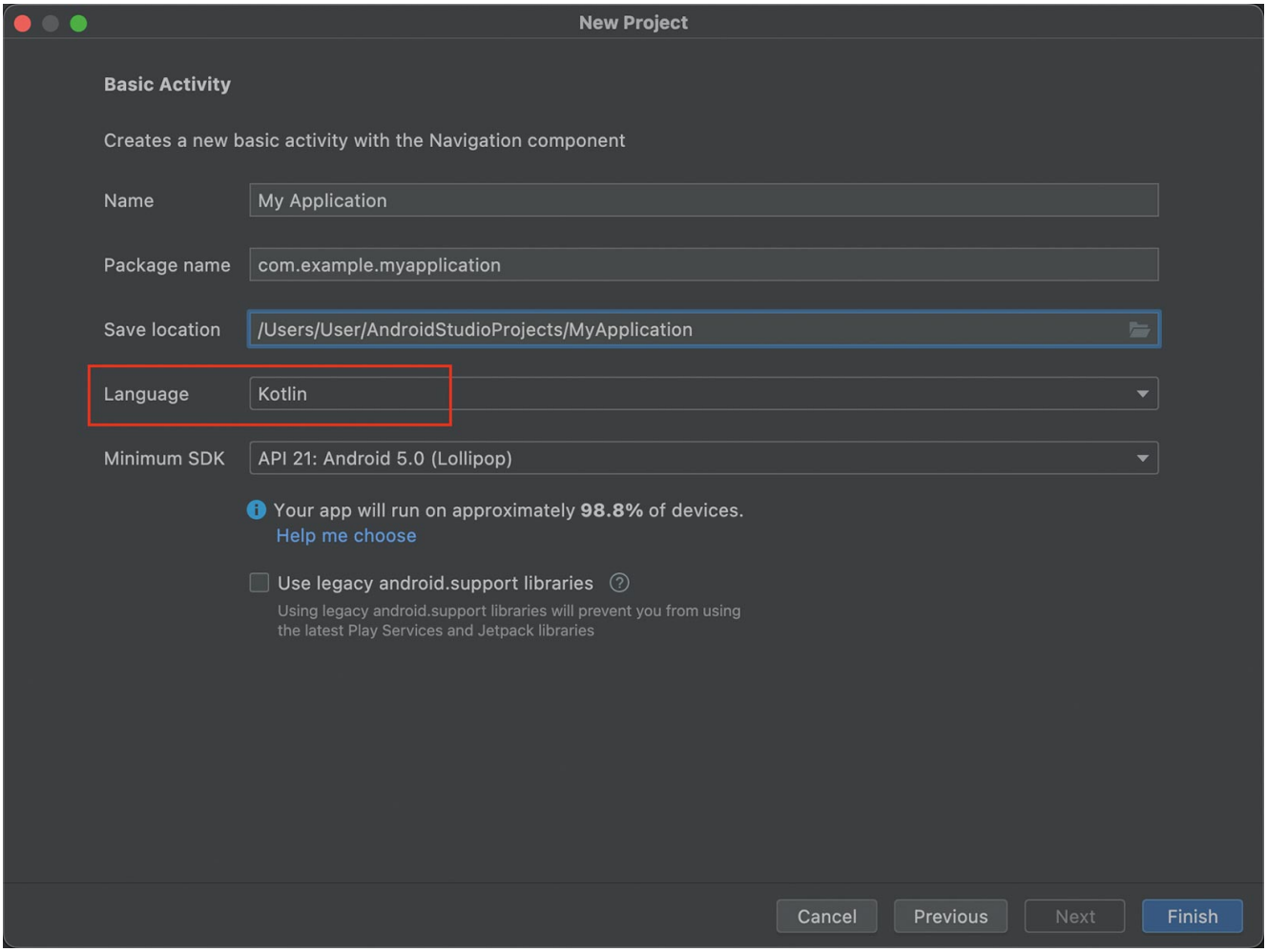

Abb. 1: Projektassistent von Android Studio für die Entwicklung von nativen Android-Apps

Die Auswahl der Programmiersprache können Sie beim Anlegen eines Projekts treffen. Beide Sprachen können jedoch auch gemischt genutzt werden. In der Entwicklungsumgebung steht u. a. ein Designer zur Verfügung, um die Oberfläche zu gestalten. Mit diesem Entwicklungsansatz können Sie auch für alle anderen Formfaktoren von Android-Devices entwickeln, beispielsweise für Android TV, Android for Car und verwandte Betriebssysteme wie Wear OS (Smartwatch) und Chrome OS. Um direkt gegen das Android SDK zu programmieren, könnte man auch eine andere Entwicklungsumgebung verwenden, die meisten Entwickler:innen werden jedoch Android Studio einsetzen.

Kommen wir zu iOS. Hier wird Xcode [2] als integrierte Entwicklungsumgebung benutzt. Diese läuft ausschließlich auf macOS und man benötigt daher zwingend einen Mac. Apple erlaubt keine andere Vorgehensweise, um das App-Package final für das

Zielsystem zu erstellen. Während der Entwicklung kann man sich ggf. mit Hilfe eines in die Cloud ausgelagerten Remote-Apple behelfen. Schauen Sie sich dabei beispielsweise die Optionen unter [3] an. Hier können Sie einen Mac mieten und mit Xcode arbeiten. Auch das Ausführen eines Simulators ist möglich, der Bildschirm wird dazu über das Netzwerk auf den eigenen Entwicklungsrechner projiziert. Dieses Vorgehen dürfte interessant sein, wenn man mit Hilfe eines anderen Ansatzes (Cross-Platform, hybrid) eine App für iOS erstellen möchte und sich dabei die Anschaffung von mehreren Macs (bei Teamarbeit) wirtschaftlich nicht lohnt. Zurück zur iOS-Entwicklung: Als Programmiersprache wird Swift verwendet. Das User Interface wird in SwiftUI deklarativ im Quellcode erstellt. Swift und SwiftUI haben Objective-C und die Nutzung von Storyboards für das Erstellen der Benutzeroberfläche weitgehend abgelöst. Die Live-Preview in Xcode zeigt bereits während des Entwurfs und der Entwicklung eine Vorschau des User Interface an und beschleunigt damit die Entwicklung maßgeblich (**Abb. 2**).

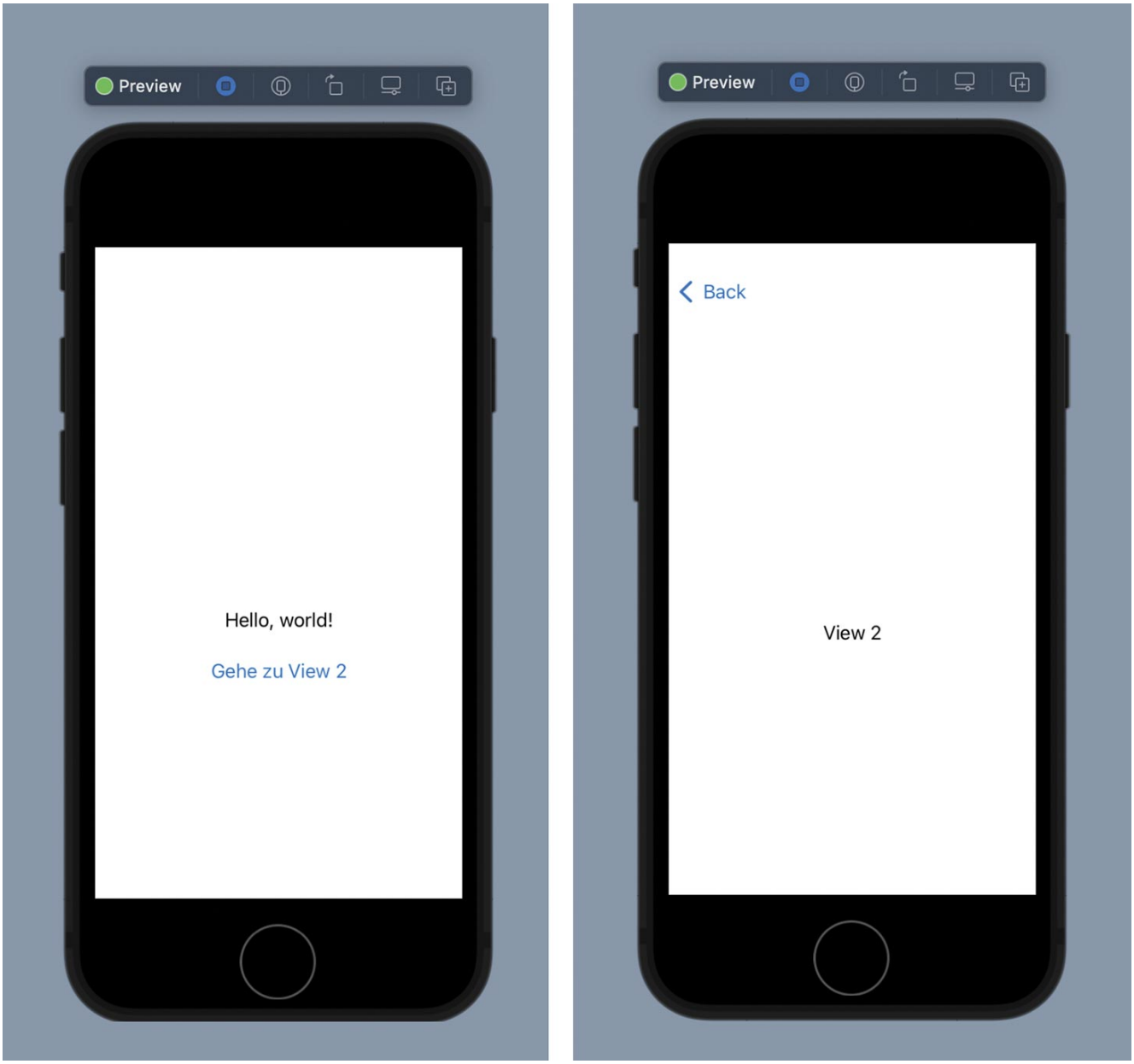

Abb. 2: Live-Preview von Xcode zur Entwicklung von nativen iOS-Apps

Es besteht kein Zweifel: Wenn Sie alle Features von Android und iOS nutzen bzw. die neuste Hardware adressieren und keinerlei Kompromisse bei der Gestaltung des User Interface machen wollen, dann bleibt Ihnen nur die Entwicklung mit diesen beiden von den Herstellern Google und Apple vorgeschlagenen Ansätzen. Will man beide Plattformen bedienen, dann muss die App zweifach programmiert werden. Es ist ein nahezu doppelter Aufwand, denn der Quellcode lässt sich zwischen den Systemen kaum teilen, wenn man von einigen übergreifenden Konzepten absieht. Hinzu kommt der Umstand, dass Android-Entwickler:innen meistens nicht über ausreichende

Kennnisse in der nativen iOS-Entwicklung verfügen und umgekehrt. Man braucht daher auch zwei Teams.

Auch wenn man sich für die Cross-Platform- oder hybride Entwicklung von Apps entscheidet, ist es hilfreich, mit den nativen Ansätzen zu experimentieren. Gelegentlich hackelt es bei der Cross-Platform-Programmierung, man muss einige plattformspezifische Einstellungen vornehmen oder man muss Plattformcode schreiben, um beispielsweise einen Sensor zu erreichen. Das geht dann nur mit Hilfe von Kenntnissen der APIs der Plattformen. Es ist auf jeden Fall nützlich, sich Wissen über iOS und Android anzueignen.

Cross-Platform- und Cross-Device-Programmierung

Es ist ein ewiger Entwicklerwunsch: „Write once, run anywhere (WORA)“. Cross-Platform-Programmierung verfolgt das Ziel, eine App zu erstellen, die auf unterschiedlichen Betriebssystemen, zum Beispiel Android und iOS, ausgeführt werden kann. Cross-Device-Programmierung geht dabei noch einen Schritt weiter. Wir adressieren hier nicht nur unterschiedliche Betriebssysteme, sondern auch verschiedene Formfaktoren der Geräte, beispielsweise Smartphone, Tablet, PC, Notebook, TV usw. Nach dieser Lesart ist die Cross-Platform-Programmierung also ein Teil der Cross-Device-Entwicklung.

Unter Cross-Platform-Programmierung verstehen wir Ansätze mit dem Ziel, Apps für Android und iOS aus einer gemeinsamen Quellcodebasis zu generieren. Die entstehenden Apps sollten dabei im Ergebnis einer nativen App möglichst nahekommen. Web-Apps bzw. hybride Apps, die wir bei einer weitergehenden Betrachtung auch als Cross-Platform bezeichnen könnten, meinen wir damit explizit nicht. Sehen wir uns einige Möglichkeiten an.

Ein neuerer Ansatz ist .NET MAUI von Microsoft [4]. Man

erreicht damit die Systeme iOS, Android, Windows und macOS und über den Herstellersupport von Samsung auch Tizen. Entwickelt wird in der Programmiersprache C# und das User Interface wird mittels XAML deklarativ erstellt. Das funktioniert für alle Zielsysteme, wobei plattformspezifische Anpassungen möglich sind. Zur Gestaltung der Oberfläche stehen visuelle Controls zur Verfügung, die während des Kompilierens auf die User-Interface-Elemente der Zielsysteme gemappt werden. Statt eines grafischen Designers wird mit dem Hot-Reload-Feature gearbeitet, d. h., Änderungen am Quellcode werden während der Entwicklung direkt in die laufende App übernommen (**Abb. 3**). .NET MAUI ist der Nachfolger des Frameworks Xamarin [5] und eine Migration für bestehende und weiterzuführende Projekte sollte geprüft werden.

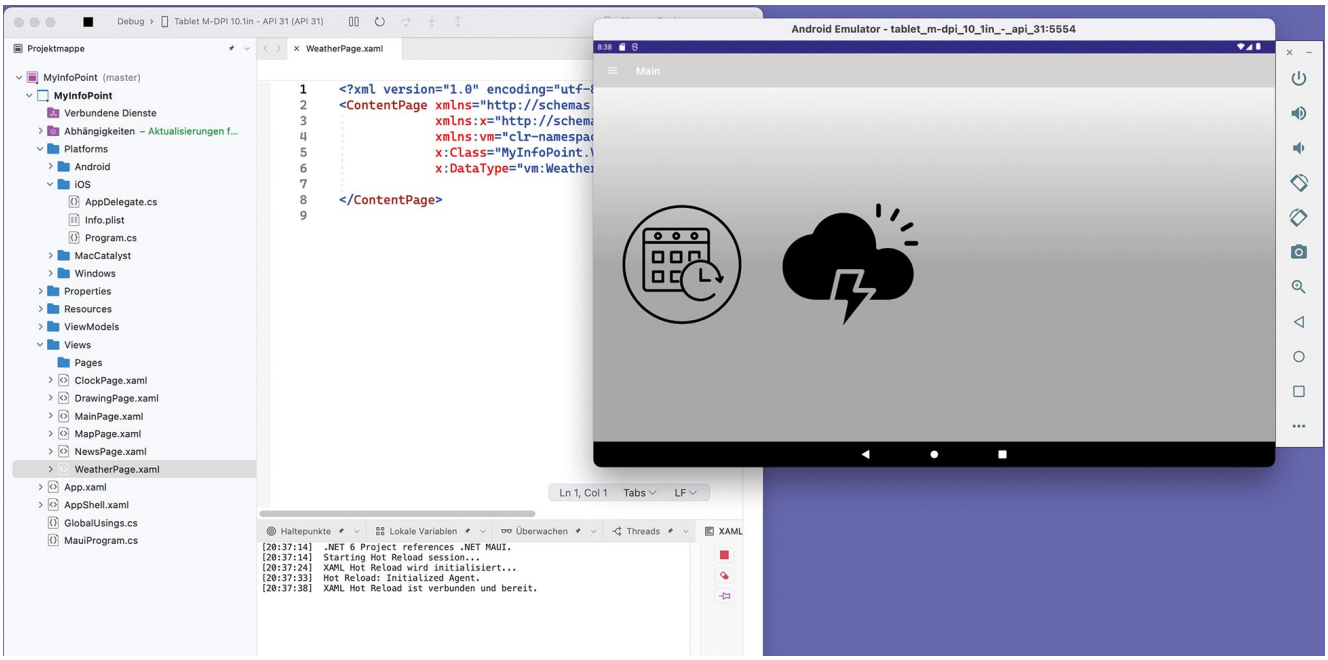

Abb. 3: Hot Reload beschleunigt die Entwicklung von MAUI-Apps (Visual Studio for macOS)

Aus dem Hause Google stammt der Ansatz Flutter [6]. Mit Flutter kann man Apps für nahezu alle relevanten Clientsysteme erstellen, d. h. für Desktop, Mobile und Web und damit auch Apps für Android und iOS. Als Programmiersprache wird Dart verwendet. Diese Sprache wurde ebenfalls von Google entwickelt. Es entstehen native Apps. Das Rendering des User Interface erfolgt über die C++-2D-Rendering-Engine Skia, die

u. a. auch in Google Chrome, Mozilla Firefox und anderen Programmen verwendet wird. Flutter zeichnet die User-Interface-Controls eigenständig und stellt dazu vielfältige Widgets zur Implementierung der Benutzeroberfläche zur Verfügung. Die Entwicklung mit dem Flutter SDK kann auf unterschiedlichen Systemen durchgeführt werden (Windows, macOS, Linux, Chrome OS). Auch bei der Wahl des Editors bzw. der Entwicklungsumgebung ist man flexibel, beispielsweise Android Studio (bevorzugt) oder Visual Studio Code.

Mit RAD Studio [7] gibt es einen weiteren Ansatz, um eine App für mehrere Plattformen zu erstellen. Programmiert wird in der Sprache Delphi (Object Pascal) und man kann Anwendungen für die Systeme Windows, macOS, Linux, iOS und Android erstellen. Die Entwicklungsumgebung bietet einen grafischen Designer, in der die Benutzeroberfläche der App vollständig aus visuellen Controls zusammengesetzt wird (**Abb. 4**).

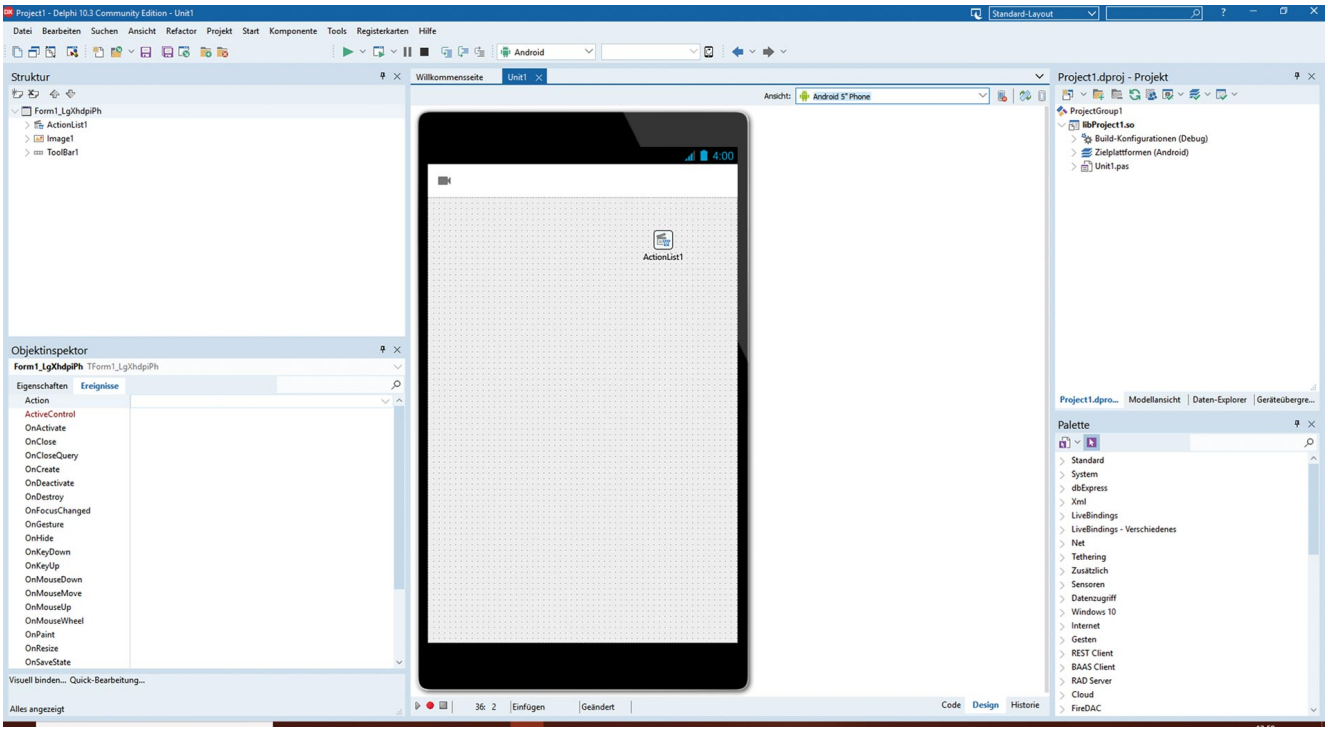

Abb. 4: Grafischer Designer in RAD Studio zur UI-Erstellung mit FireMonkey

Nichtvisuelle Controls stehen beispielsweise für den Zugriff auf Gerätefunktionen (z. B. Dateisystem) oder Sensoren (z. B. Bluetooth) zur Verfügung. Es handelt sich um native Apps, die

direkt auf den Zielsystemen ausgeführt und in die App Stores eingestellt werden können. RAD Studio nutzt für das Rendering des UI das geräteübergreifende Grafikframework Fire Monkey, das die systemeigenen Grafikbibliotheken der Zielsysteme verwendet. Die besprochenen und weitere Ansätze zur geräte- und plattformübergreifenden Entwicklung haben wir für Sie in Tabelle 1 zusammengefasst.

Ansatz/IDE	Zielsystem	Programmier-sprache/Framework	Betriebssystem	Hersteller	Hinweise	Link
Android Studio	Android	Kotlin Java	Windows Linux macOS	Google	der direkte und offizielle Weg zu einer nativen App für Android Zugriff auf alle APIs der Plattform keine Einschränkungen bei den Features und den Kompatibilitäten	https://developer.android.com/studio
Xcode	iOS	Swift SwiftUI	macOS	Apple	der direkte und offizielle Weg zu einer nativen App für iOS Zugriff auf alle APIs der Plattform keine Einschränkungen bei den Features und den Kompatibilitäten	https://developer.apple.com/xcode/
.NET MAUI/ Visual Studio	iOS Android Windows (WinUI 3) macOS (Tizen)	C# XAML .NET-Plattform	Windows macOS	Microsoft	native Apps für die genannten Zielsysteme aus einer gemeinsamen Quellcodebasis UI wird deklarativ mittels XAML erstellt Plattformanpassungen und plattformspezifischer Code sind möglich Einsatz von Controls für das UI Nachfolger von Xamarin.	https://dotnet.microsoft.com/en-us/apps/maui
Xamarin	iOS Android Windows (UWP) (Tizen macOS 10.13 GTK# WPF)	C# XAML .NET-Plattform	Windows macOS	Microsoft	gemeint ist hier die Entwicklung mit Xamarin.Forms technologischer Vorgänger von .NET MAUI, wird aktuell noch gepflegt eine Migration nach .NET MAUI ist zu prüfen	https://dotnet.microsoft.com/en-us/apps/xamarin
RAD Studio	iOS Android Windows macOS Linux	Delphi	Windows	Embarcadero	RAD-Tool zur Entwicklung von App für unterschiedliche Zielsysteme, u. a. Mobile-Apps für iOS und Android grafischer Designer, in dem mittels Definition von Eigenschaften die Benutzeroberfläche erstellt wird	https://www.embarcadero.com/products/rad-studio
Flutter	iOS Android Windows macOS Linux Web	Dart	Windows macOS Linux Chrome OS	Google	geräteübergreifende Programmierung von Apps für diverse Clientbetriebssysteme und das Web programmiert wird in der Sprache Dart User Interface wird mittels Widgets erstellt und eigenständig gezeichnet	https://flutter.dev/

Ansatz/IDE	Zielsystem	Programmier-sprache/Framework	Betriebssystem	Hersteller	Hinweise	Link
Native-Script	iOS Android	JavaScript TypeScript	Windows macOS Linux	ursprünglich Telerik; aktuell Community	Erstellung nativer Apps für Android und iOS es werden die APIs der Zielsysteme verwendet Programmierung in JavaScript (TypeScript) User Interface kann mit Hilfe von Komponenten erstellt werden (XML)	https://nativescript.org
React-Native	iOS Android	JavaScript TypeScript React	Windows macOS Linux	Metaplattform	Ermöglicht das Erstellen von mobilen Apps für Android und iOS mit Hilfe des React-Frameworks	https://reactnative.dev

Tabelle 1: Übersicht über Ansätze zur geräte- bzw. plattformübergreifenden Entwicklung

Web-Apps

Nicht immer ist es notwendig oder sinnvoll, eine native App zu erstellen. Der Anteil der Nutzer:innen mit mobilen Endgeräten nimmt stetig zu. Gerade in der jüngeren Generation wird heute bereits ein Großteil der Aufgaben im Internet über das Smartphone erledigt. Wird eine Webseite bzw. -applikation für die Nutzung auf einem Smartphone optimiert (Mobile First), dann handelt es sich um eine Web-App. Eine solche Web-App läuft direkt im Browser und wird nicht installiert. Nach der Eingabe der Internetadresse ist sie sofort nutzbar. Da die Applikation auf dem Server läuft, ist eine ständige Datenverbindung notwendig. Dadurch, dass keine Installation auf dem Zielgerät notwendig ist, können diese Apps auch nicht in den Store eingestellt werden. Auch wollen Nutzer:innen diese oftmals ad hoc verwenden, zum Beispiel, um einmalig ein Ticket zu buchen, und eine Installation einer App aus dem Store ist daher nicht zumutbar.

Technisch betrachtet sind Web-Apps meist clientseitige Single-Page Applications. Für eine gute Reaktionsfähigkeit und eine bestmögliche User Experience werden diese typischerweise mit den bekannten Webframeworks Angular, Vue oder React entwickelt. Im Kern basieren sie technisch auf HTML (Struktur), CSS (Layout, Design) und JavaScript (Interaktion). Durch die Verwendung von auf die mobile Nutzung ausgerichteten UI-Bibliotheken kann man das Design der Web-Apps weitgehend

identisch zu einer nativen App gestalten. Ein Beispiel für eine solche UI-Bibliothek ist OnsenUI [8]. OnsenUI bietet einen umfassenden Satz von grafisch reichhaltigen UI-Komponenten, die speziell für Mobile-Apps entwickelt wurden. Diese orientieren sich an den Design-Guidelines von Android und iOS. Das Framework steht unter einer Open-Source-Lizenz (Apache v2) zur Verfügung und kann mit Vue, Angular und React eingesetzt werden.

Beispiele zeigen, dass man mit Web-Apps heute das Feeling (Design, Handling) von nativen Apps nahezu erreicht. **Abbildung 5** zeigt die Ansicht einer Wetter-App, die mit Onsen UI gestaltet wurde [9].

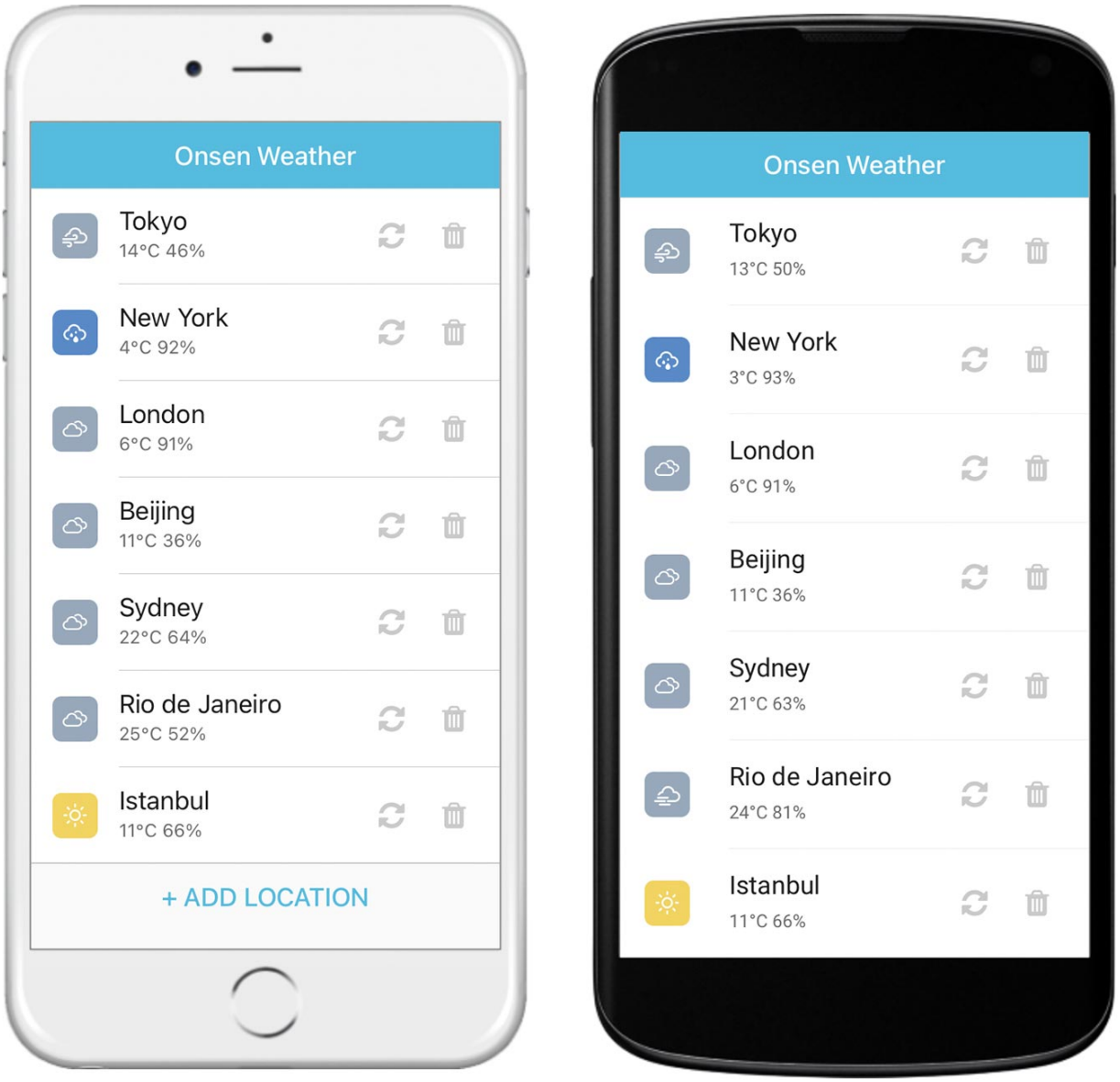

Abb. 5: Wetter-App mit Onsen UI

Da es sich um eine Webapplikation handelt, kann man diese direkt im Browser eines jeden beliebigen Systems aufrufen und starten. Interessant ist dabei auch, dass man das Design an iOS bzw. Android anpassen kann. Wie man sieht, kommt man dem nativen App-Erlebnis hier sehr nahe. Eine Alternative für die Optimierung des UI für die mobile Nutzung bietet auch das Framework Ionic [10], das mit React, Vue und Angular kombiniert werden kann. Ein Beispiel für eine mobile App mit einem Kalender-Control, implementiert mit Ionic und Angular, zeigt **Abbildung 6**.

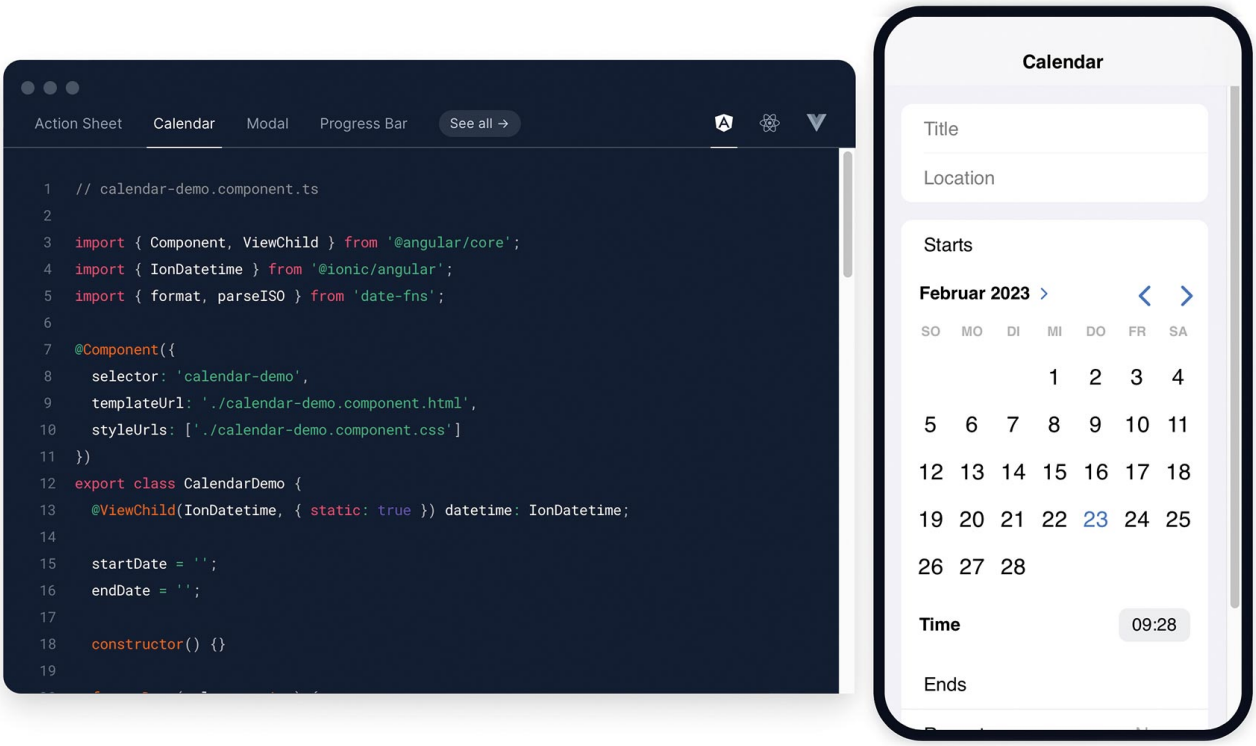

Abb. 6: App mit Kalender-Control mit dem Ionic-Framework

Diese UI-Frameworks bieten Lösungen für die Gestaltung der mobilen App. Da diese direkt im Browser des Systems läuft, sind Geräte- und Systemzugriffe zunächst auf das beschränkt, was über den Browser unmittelbar möglich ist. Das JavaScript API zur Ortung kann beispielsweise ohne Einschränkungen genutzt werden. Dagegen ist ein Zugriff auf die Sensoren eines Smartphones eingeschränkt bzw. nahezu ausgeschlossen.

Hier kommen wir jedoch zu zwei weiteren wichtigen Erweiterungsmöglichkeiten. Eine Web-App, die auf einen mobilen Device ausgeführt wird, kann mit Hilfe einer Progressive Web App (PWA) und mittels eines nativen App-Containers (hybride App) erweitert werden. Mit einer PWA ist es möglich, eine Web-App auch mit Offlinefunktionalität auszustatten. Ebenso kann man für eine solche PWA auch ein Icon auf den Home Screen des mobilen Geräts ablegen. Bei einer hybriden App wird die Web-App im internen Browser einer nativen App verpackt und erhält dann über diesen Zugriff auf die Geräte- und Hardwarefeatures des Smartphones oder Tablets. Wir werden im kommenden Abschnitt noch genauer auf diese Technologie eingehen.

Üblicherweise entwickelt man Web-Apps mit Webtechnologien und -frameworks. Es gibt jedoch auch zu dieser Vorgehensweise Alternativen. An Entwickler:innen, die in der Java-Welt zu Hause sind, richtet sich Vaadin [11]. Mit Hilfe dieses Frameworks kann man eine Web-App mit Java-Quellcode generieren, ohne direkt HTML, CSS und JavaScript schreiben zu müssen. Es steht auch hier eine Reihe von UI-Komponenten zur Verfügung, um die Benutzeroberfläche zu gestalten.

Vaadin kann man auch innerhalb der integrierten Entwicklungsumgebung RapidClipse [12] nutzen. Rapid-Clipse ist eine IDE, die auf Eclipse aufsetzt und u. a. einen grafischen Designer für Vaadin bietet, mit dem man die Benutzeroberflächen intuitiv gestalten kann. Für die Entwicklung von Mobile-Apps (Mobile First) gibt es ein nützliches Feature: Durch einen Button im GUI Builder wird die erstellte Seite lokal gehostet. Für diese Seite wird dann ein QR-Code generiert, den man mit dem Smartphone einscannen kann (**Abb. 7**). Die Seite wird anschließend direkt auf dem mobilen Gerät angezeigt. Voraussetzung ist, dass sich der Entwicklungsrechner mit dem lokalen Server und das Smartphone im gleichen lokalen Netzwerk befinden.

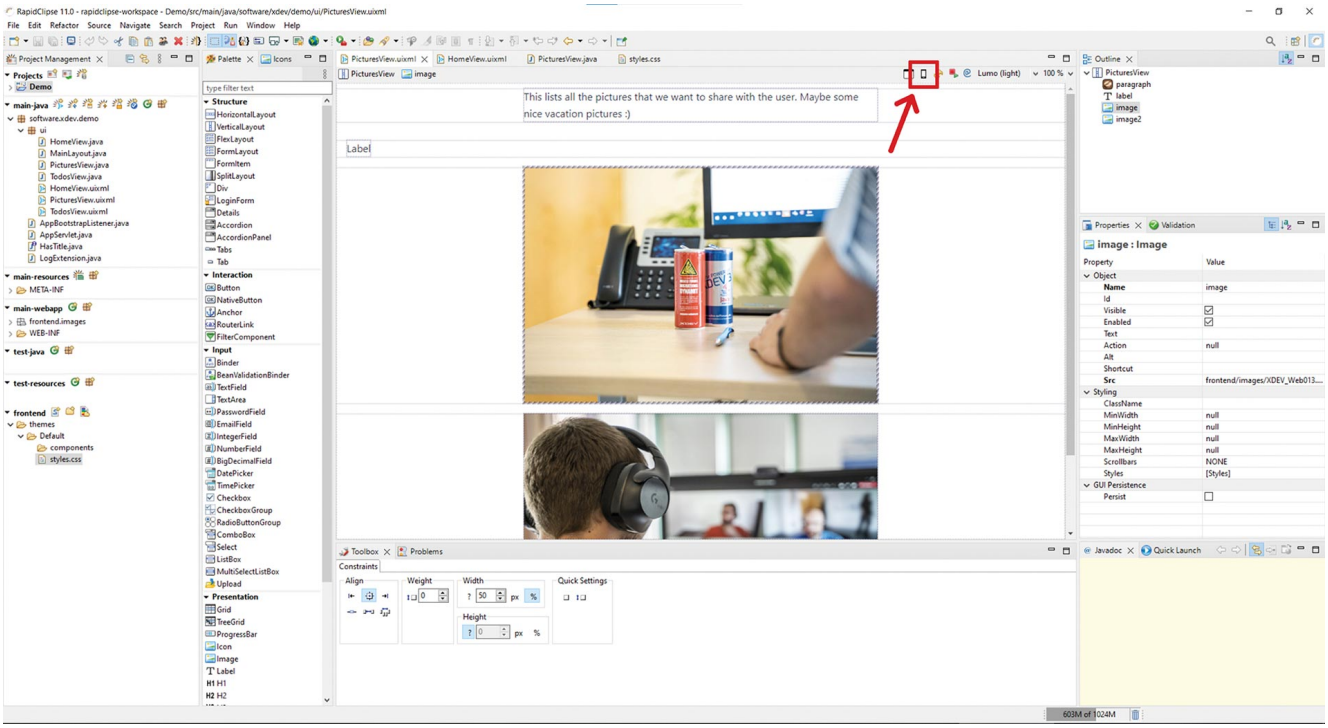

Abb. 7: Web-App mit RapidClipse und Vaadin erstellen und für mobile Geräte testen

Hybride Apps

Das Prinzip ist einfach: Wir haben einen nativen App-Container, der auf das jeweilige Zielsystem angepasst ist. Innerhalb dieses Containers läuft ein Browser (ohne Möglichkeit der Änderung des URL), in dem die Web-App ausgeführt wird. Für das Erstellen der App steht das gesamte Spektrum der bisher behandelten Technologien zur Verfügung. Der Container fungiert als Schnittstelle zwischen den APIs des betreffenden Betriebssystems und der App und ermöglicht dieser einen Zugriff auf die Gerätefunktionen und Sensoren des mobilen Geräts. Das bekannteste Framework ist Cordova [13]. Die Architektur ist in **Abbildung 8** zu sehen.

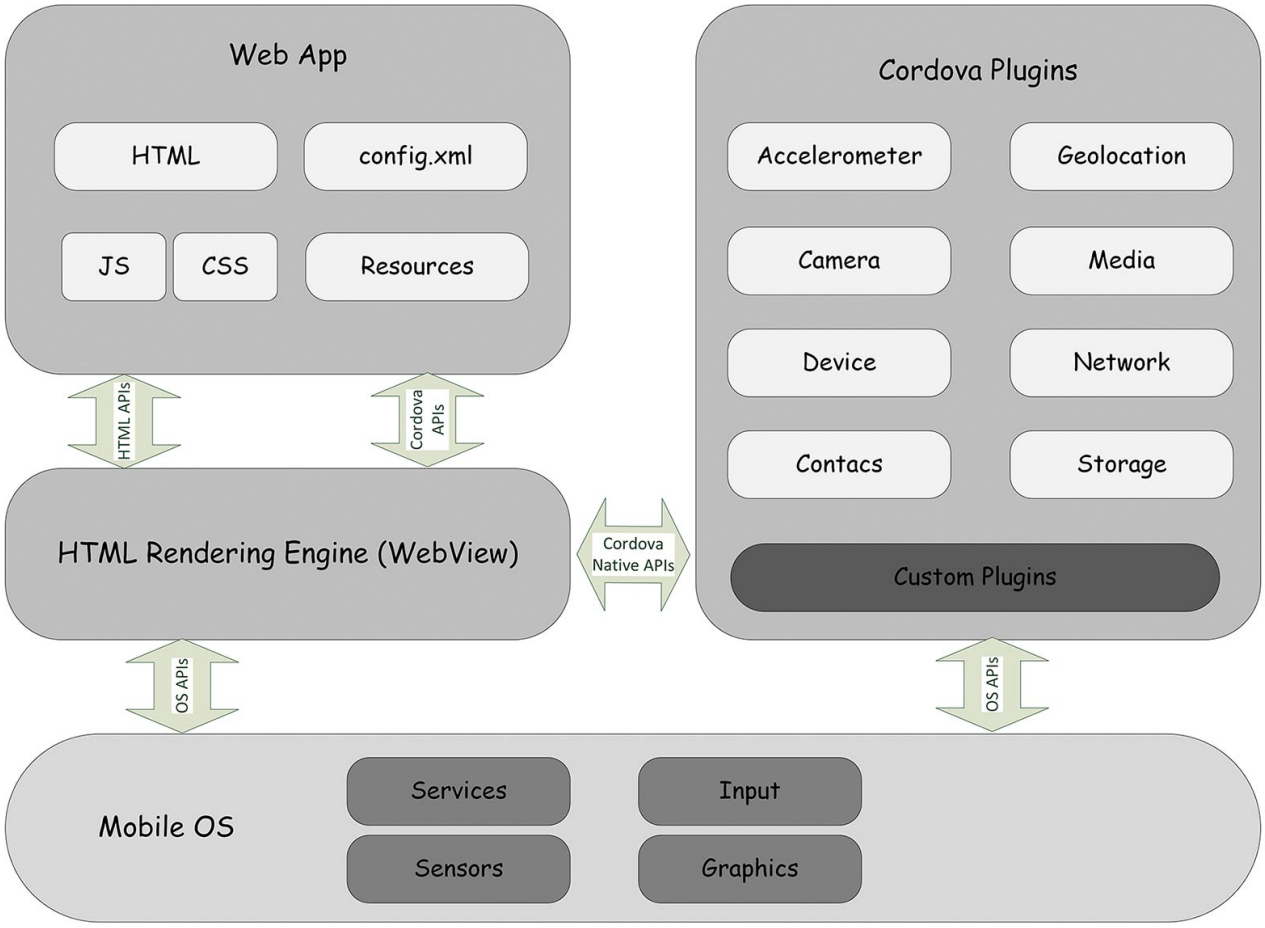

Abb. 8: Die Architektur von Cordova für hybride Apps

Über Plug-ins wird der Zugriff auf die Systemfunktionen sichergestellt. Hybride Apps erscheinen für das System wie eine native Anwendung – sie können offline genutzt werden und man kann sie in den App Stores einstellen. Das Prinzip der hybriden App ist also eine interne Web-View, die eine Webapplikation anzeigt.

Auch hier gibt es weitere Technologien zur Umsetzung. Beispielsweise geht das Framework Wisej.NET einen anderen Weg: Hier wird die Webapplikation auf Basis von .NET erstellt und auf einem Webserver gehostet. Als Entwicklungsumgebung kommt Visual Studio zum Einsatz und man arbeitet mit visuellen Komponenten für die Gestaltung der Oberfläche. Das Framework übersetzt die .NET-Applikation, die auf dem Server läuft, in eine Single-Page App. Diese kann wiederum in einer Web-View auf den mobilen Geräten angezeigt werden. Dazu bietet der Hersteller vorbereitete native Apps, die man individuell an

die Web-App anpassen kann. Gleichwohl sorgt diese Web-View mittels Wisej Mobile dafür, dass man aus der Web-App Zugriff auf das API des mobilen Geräts hat. Die Besonderheit dieses Ansatzes liegt in der Erstellung von Web-Apps. Dazu kann, vergleichbar mit Windows Forms, in Visual Studio per Drag and Drop die Oberfläche mit einem grafischen Designer konfiguriert werden. Die Businesslogik wird in C# programmiert, wodurch es nicht notwendig ist, HTML-, CSS- und Java-Script-Code zu schreiben. Informationen zum Erstellen von Mobile-Apps mit Wisej.NET finden Sie unter [14].

Weitere Ansätze

Haben wir jetzt alle Vorgehensweisen zum Erstellen einer Mobile-App beisammen? Zumindest die grundsätzlichen Ansätze zur Entwicklung von nativen, webbasierten und hybriden Apps, einschließlich der plattformübergreifenden Entwicklung haben wir aufgeführt. Dennoch gibt es weitere Technologien. Daher folgen in diesem Abschnitt einige weitere Tools, mit deren Hilfe man ebenfalls zu einer Mobile-App kommt:

- *Qt:* Das plattformübergreifende GUI-Toolkit kann auch dazu genutzt werden, Mobile-Apps für Android und iOS zu erstellen. Die Programmlogik wird in C++ erstellt und das User Interface mittels QML deklariert. Ein grafischer Designer (Qt-Creator) erleichtert die Gestaltung. Informationen für den Einstieg findet man in der Dokumentation [15].
- *Xojo:* Hierbei handelt es sich um ein RAD-Tool zur Anwendungsentwicklung für sehr unterschiedliche Zielsysteme. Der Schwerpunkt liegt auf einer beschleunigten Entwicklung mit Hilfe eines grafischen Designers und vorgefertigten Komponenten. Die Businesslogik wird mit Hilfe eines Basic-Dialektes erstellt. Gemäß der Dokumentation kann man mit Hilfe von Xojo auch Apps für iOS erstellen. Android-Apps stehen

auf der Roadmap für künftige Releases der Entwicklungsumgebung. Informationen zur App-Entwicklung für iOS findet man in der Onlinedokumentation unter [16].

- *NativeScript* [17]: Ein Open-Source-Framework zum Entwickeln von Apps für iOS und Android. Als Programmiersprachen werden JavaScript und TypeScript eingesetzt. Der Einsatz von Frameworks wie Vue oder Angular wird unterstützt. Es entstehen native Apps, die die APIs von Android und iOS verwenden. Zur Gestaltung des User Interface gibt es entsprechende Komponenten. Die Oberfläche kann in XML oder in JavaScript bzw. TypeScript kodiert werden.
- *React Native* [18]: React Native setzt auf React auf und erlaubt, native Apps mit JavaScript für Android und iOS zu realisieren.
- *Kotlin Multiplatform* [19]: Dieses Framework geht einen anderen Weg. Im Fokus der Cross-Platform-Entwicklung steht die Entwicklung der Geschäftslogik mit Hilfe einer gemeinsamen Codebasis. Und das UI? Das wird weiterhin nativ mit den Technologien von Android und iOS erstellt. Als Programmiersprache kommt Kotlin zum Einsatz, als Entwicklungsumgebung Android Studio.

Wenn Sie im Netz suchen, werden Sie noch weitere, durchaus interessante Ansätze finden, um eine mobile App für Android oder iOS zu erstellen. Nicht erwähnt haben wir hier Low-Code-Tools. Auch mit diesen Entwicklungstools kommt man bei speziellen Anforderungen ans Ziel. Für alle webbasierten Technologien gilt: Packt man die Web-App in einen nativen Container, kann man sie über die hybride Technologie dem nativen App-Feeling deutlich näherbringen.

Fazit

Lassen wir diesen Artikel mit einem allgemein bekannten Spruch

ausklingen: „Wer die Wahl hat, hat die Qual“. Das gilt in der Tat für die Auswahl der Entwicklungsansätze für das Mobile-App-Development. Doch diese Qual hat einen großen Vorteil: Entwickler:innen können sich den Technologiestack heraussuchen, der den eigenen Kenntnissen in Sachen Programmiersprache, Framework und bisherigen Erfahrungen am nächsten kommt. An erster Stelle der Auswahl wird i. d. R. der geforderte Funktionsumfang der App stehen. Ebenso ist es wichtig, wie oft man eine App für die Mobile-Systeme erstellen möchte. Ist es nur ein Nebenprodukt, dann tut es vielleicht eine webbasierte App. Regelmäßige App-Entwickler werden dagegen meist zu den Cross-Platform-Ansätzen greifen, um keine oder nur wenige Einschränkungen auf den Zielsystemen in Kauf nehmen zu müssen.

Muss die bestmögliche Plattformanpassung erreicht werden, dann bleibt nur die Entwicklung mit den Werkzeugen der Hersteller, in diesem Fall separat für iOS und Android. Für die meisten Zwecke dürfte man jedoch mit der Cross-Platform- bzw. hybriden Entwicklung gut bedient sein.

Dr. Veikko Krypczyk ist Softwareentwickler, Trainer und Fachautor und u.a. auf die Themen WinUI 3 und .NET MAUI spezialisiert. Sein Wissen gibt er über Fachartikel, Seminare und Workshops gern an Interessierte weiter und steht mit seiner Expertise auch für eine individuelle Unterstützung in Projekten zur Verfügung.

Elena Bochkor arbeitet am Entwurf und Design mobiler Anwendungen und Webseiten. Beginnend bei einer systematischen Nutzerforschung, über visuelle Prototypen bis hin zu einem barrierefreien Design gestaltet sie die User Experience moderner digitaler Produkte.

Trainings und Workshops zu diesen Themen können Sie unter https://wp.larinet.com anfragen und die Agenda vorab einsehen.

Links & Literatur

[1] https://developer.android.com/studio

[2] https://developer.apple.com/xcode/

[3] https://www.macincloud.com

[4] https://dotnet.microsoft.com/en-us/apps/maui

[5] https://dotnet.microsoft.com/en-us/apps/xamarin

[6] https://flutter.dev

[7] https://www.embarcadero.com/products/rad-studio

[8] https://onsen.io

[9] http://argelius.github.io/react-onsenui-redux-weather/demo.html

[10] https://ionicframework.com

[11] https://vaadin.com

[12] https://rapidclipse.com

[13] https://cordova.apache.org

[14] https://docs.wisej.com/mobile

[15] https://www.qt.io/product/mobile-app-development/

[16] https://documentation.xojo.com/getting_started/quickstarts/ios_quickstart.html

[17] https://nativescript.org

[18] https://reactnative.dev

[19] https://kotlinlang.org/lp/mobile/