HOME
HOCHSCHULE MERSEBURG FH
University of Applied Sciences

FACHBEREICH INFORMATIK UND KOMMUNIKATONS-SYSTEME

Masterarbeit

Vergleich von Cross-Plattform-Frameworks für die Appentwicklung und Validierung der Ergebnisse durch Implementierung eines XXO-Spiels

von Arne Linck

Studiengang:	Master Informations- und Kommunikationssysteme (MIKS)
Matrikelnummer:	18321
Erstbetreuer:	Prof. Dr. rer. pol. Uwe Schröter
Zweitbetreuer:	Dipl. Inf. (FH) Sebastian Kohl
Abgabedatum:	30.09.2016

Inhaltsverzeichnis

Eidesstattliche Erklärung III

1 Begriffsklärung 1

1.1 Betriebssystemunabhängigkeit 1

1.2 Architekturunabhängigkeit 2

2 Architekturansätze 2

2.1 Web 2

2.2 Hybrid 3

2.3 Interpreter 4

2.4 Cross-To-Native 5

2.5 Native 5

3 Frameworks 6

3.1 Auswahlkriterien 6

3.2 PhoneGap/Apache Cordova 7

3.2.1 Überblick 7

3.2.2 Projektstruktur 8

3.3 Sencha Touch 9

3.3.1 Überblick 9

3.3.2 Projektstruktur 11

3.4 Ionic 12

3.4.1 Überblick 12

3.4.2 Projektstruktur 13

3.5 Titanium 13

3.5.1 Überblick 13

3.5.2 Projektstruktur 14

3.6 Xamarin 16

3.6.1 Überblick 16

3.6.2 Projektstruktur 17

3.7 Eigenes Framework 18
3.8 Andere Frameworks 19
3.8.1 Unity 19
3.8.2 Unreal Engine 19
3.8.3 Windows 10 Universal App 20
3.9 Auswahl des Frameworks 20
3.9.1 Kriterien 20
3.9.2 Ausgewähltes Framework 22
4 Die App 23
4.1 Anforderungen 23
4.2 Implementierung 25
4.2.1 Buildvorgänge 25
4.2.2 Struktur 27
4.2.3 Komponenten 32
4.2.4 Algorithmen 33
4.3 Unit-Tests 35
5 Aussicht 36
Abbildungsverzeichnis IV
Listingverzeichnis IV
Tabellenverzeichnis IV

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, die vorliegende Arbeit selbstständig und ohne Zuhilfenahme unzulässiger Hilfsmittel angefertigt zu haben. Wörtliche oder dem Sinne nach übernommene Ausführungen sind gekennzeichnet, sodass Missverständnisse über die geistige Urheberschaft ausgeschlossen sind.
Diese Arbeit war bisher noch nicht Bestandteil einer Studien- oder Prüfungsleistung in gleicher oder ähnlicher Fassung.

Ort, Datum	Unterschrift

1 Begriffsklärung

Plattformunabhängigkeit ist die Eigenschaft eines Programms oder Quellcodes, auf verschiedenen Plattformen ohne Anpassungen lauffähig zu sein. Diese Unabhängigkeit von der Plattform ist nicht immer automatisch gegeben, kann aber auf verschiedene Weisen erreicht werden. Ein großer Vorteil der Plattformunabhängigkeit ist, dass das Programm ohne weiteren Entwicklungsaufwand eine größere Masse an potentiellen Nutzern erreicht, da es auf mehr Plattformen ausgeführt werden kann.

1.1 Betriebssystemunabhängigkeit

Ein Programm ist betriebssystemunabhängig, wenn es ohne Anpassungen auf verschiedenen Betriebssystemen ausführbar ist, beispielsweise auf *Windows*, *Linux* und *OS X*.

Dies kann auf verschiedene Arten erreicht werden. Zum einen können *JIT*[1]-kompilierte Sprachen eingesetzt werden, wenn für das Zielbetriebssystem eine Laufzeitumgebung zur Verfügung steht, die das Kompilieren durchführt. Prominente Beispiele für solche Sprachen sind *Java* und die Sprachen der *.NET*-Familie wie *C#* oder *F#*.

Zum anderen können interpretierte Sprachen wie *Python* und *JavaScript* verwendet werden. Sie verhalten sich ähnlich wie *JIT*-kompilierte Sprachen: Auf dem Zielbetriebssystem muss ein Interpreter zur Verfügung stehen, damit das Programm ausgeführt werden kann. Dieser interpretiert dann den Quellcode direkt, ohne ihn vorher zu kompilieren.

Eine weitere Methode der Betriebssystemunabhängigkeit ist unabhängiger Quellcode, der ohne Anpassungen für die Zielplattform kompiliert werden kann. Die Sprachen *C* und *C++* sind dafür prominente Beispiele. Wenn gewisse Regeln für plattformunabhängiges Programmieren eingehalten werden, wie zum Beispiel die Möglichkeit, dass Pointer verschiedene Größen haben können, dann kann der *C*- oder *C++*-Quellcode einfach für die Zielplattform neu kompiliert werden, denn für die meisten Betriebssysteme stehen Compiler für diese Sprachen zur Verfügung. Im Gegensatz zu *JIT*-kompilierten oder interpretierten Sprachen benötigt diese Variante aber für jede Zielplattform eine eigene Programmdatei.

[1] JIT, Just In Time: Der Bytecode des Programms wird erst zur Laufzeit in ausführbaren Code kompiliert.

1.2 Architekturunabhängigkeit

Ein Betriebssystem kann unter verschiedenen Prozessorarchitekturen lauffähig sein. Zum Beispiel gibt es für einige *Linux*-Distributionen *x86*-, *x64*- und *ARM*-Varianten. Ein kompiliertes Programm wird immer Prozessorbefehle beinhalten, die auf eine bestimmte Prozessorarchitektur zugeschnitten sind, weshalb portabler Quellcode auch für das gleiche Betriebssystem für verschiedene Prozessorarchitekturen neu kompiliert werden muss. Eine Ausnahme sind *x86*-Programme. Sofern es das Betriebssystem erlaubt, sind diese voll unter *x64* lauffähig, da *x64* voll zu *x86* abwärtskompatibel ist.

JIT-kompilierte und interpretierte Sprachen dagegen bleiben auch unter verschiedenen Prozessorarchitekturen lauffähig, wenn – wie oben beschrieben – eine entsprechende Laufzeitumgebung für die Zielarchitektur zur Verfügung steht, die das Programm entweder *JIT*-kompilieren oder interpretieren kann.

Ein Hilfsmittel, um Programme unter einer Prozessorarchitektur zum Laufen zu bringen, für die sie nicht kompiliert wurden, existiert jedoch: *Emulatoren* analysieren den Programmcode und interpretieren oder *JIT*-kompilieren ihn so, dass das Programm unter einer anderen Prozessorarchitektur ausgeführt werden kann. Ein prominentes Beispiel hierfür ist der *Dolphin*-Emulator [1], der Software auf *x86* und *x64* emuliert, die ursprünglich für den *Nintendo GameCube* und die *Nintendo Wii* gedacht waren. Emulation bringt jedoch große Geschwindigkeitseinbußen mit sich, da die Prozessorbefehle des emulierten Programms kaum auf dem emulierenden System beschleunigt werden können. Deshalb sollte Emulation, soweit es bessere Alternativen gibt, vermieden werden.

2 Architekturansätze

2.1 Web

Webseiten werden von verschiedensten Geräten aufgerufen und dabei dennoch, wenn man von mobilen Versionen und Browserhacks[1] absieht, weitestgehend identisch dargestellt. Das ist zurückzuführen auf die zunehmende Konformität der verschiedenen Browserhersteller mit dem HTML[2]-Standard des *W3C*[3] [2]. Auch die Browser, die auf Smartphones verwendet werden, profitieren von dieser Entwicklung.

[1] Browserhacks werden dazu verwendet, die Webseite auf verschiedenen Browsern anders darzustellen.
[2] HTML, Hypertext Markup Language: Sprache zur Strukturierung von Webseiten.
[3] W3C, World Wide Web Consortium: Ein Gremium zur Erstellung von Standards für Internettechnologien.

Daraus folgt, dass Smartphones Webseiten annähernd gleich gut darstellen können. Die verschiedenen Displaygrößen unterschiedlicher Geräte können mit einem dynamischen Seitenlayout ausgeglichen werden. Die App kann also komplett als Webseite konzipiert werden. Ohne Brückentechnologien oder spezielle Anpassungen an der Webseite wird so eine konsistente App geschaffen, denn Smartphones haben schon ab Werk einen Browser installiert, der Webseiten weitestgehend standardkonform darstellen kann.

Ein Vorteil dieser Methode ist, dass eine App nur ein einziges Mal entwickelt werden muss. Dies geschieht außerdem in den Webtechnologien *HTML*, *CSS*[1] und *JavaScript*, die eine sofortige Anzeige von Änderungen am Code erlauben - es ist nur ein neu Laden der Webseite vonnöten. Das bedeutet auch, dass Updates an der Webseite sofort jedem Benutzer zur Verfügung stehen. Dieser muss keine neue Version der App aus dem Store herunterladen.

Dieser Vorteil zieht aber auch einen Nachteil mit sich. Der Nutzer muss jederzeit online sein, wenn er die App benutzen will. Das kann unter Umständen das mobile Datenvolumen des Nutzers erschöpfen und Kosten verursachen, wodurch einige vielleicht auf die Benutzung der App verzichten werden, wenn sie unterwegs sind. Des Weiteren sind Webseiten im Vergleich zu auf dem Smartphone installierten Apps dahingehend limitiert, dass Webseiten keinen freien Zugriff haben auf spezielle Sensoren oder Features, die das Gerät mitbringt. Eine Positionsbestimmung ist noch möglich über die *Geolocation-API* von *HTML5* und eine Speicherung von Daten über *LocalStorage*, doch ein Zugriff auf die Kontakte oder das Notification Center gestaltet sich schon schwieriger. Für eher rechenintensive Aufgaben, solange sie mit *JavaScript* auf der Clientseite und nicht vom Server selbst berechnet werden, ist diese Methode ebenfalls nicht geeignet, da *JavaScript* als nicht kompilierte Skriptsprache langsamer ist als nativer Code.

2.2 Hybrid

Beim hybriden Ansatz wird versucht, die Nachteile einer reinen webbasierten App zu umgehen. Anstatt eine Webseite aus dem Internet abzurufen, wird sie in einem App-Container beim Installieren der App auf dem Smartphone mitgeliefert. Wird die App aufgerufen, so muss sich das Smartphone nicht mit einem Server verbinden, sondern ein angepasstes Browserfenster zeigt die in der App gespeicherte Webseite an. Die *HTML*-,

[1] CSS, Cascading Stylesheets: Sprache zum Definieren von Layout und Aussehen von z.B. *HTML*.

CSS- und *JavaScript*-Dateien werden einfach aus den Ressourcen der App aufgerufen, ohne einen einzigen Request an einen Webserver.

Eine hybride App eliminiert viele der Nachteile einer Web-App, ohne dabei ihre Vorteile zu verlieren. So muss immer noch nur eine Variante der App programmiert werden und sie wird auf allen vom Framework unterstützten Geräten weitestgehend identisch dargestellt. Gleichzeitig muss keine Internetverbindung mehr bestehen, um die App zu benutzen: Änderungen können bei Bedarf gespeichert und bei der nächsten Verbindung über AJAX[1] synchronisiert werden. Außerdem bringen die Hybridframeworks zusätzliche APIs mit, die in normalen Browsern nicht zur Verfügung stehen, wie zum Beispiel Zugriff auf die Position, aber auch die Kontakte des Benutzers, ohne dass dieser gefragt werden muss.

Einige Nachteile bleiben jedoch bestehen. So ist der hybride Ansatz, da er weiterhin *JavaScript* verwendet, nicht für rechenintensive Aufgaben geeignet.

2.3 Interpreter

Bei einer App nach dem Interpreteransatz wird eine dynamische Programmiersprache verwendet, wie zum Beispiel *JavaScript*, *Ruby* oder *Python*. Im Gegensatz zu hybriden Apps, die in einem Browserfenster laufen, werden die dynamischen Programmiersprachen direkt ausgeführt. Der dafür nötige Interpreter wird mit in der Regel mit dem Framework mitgeliefert und wird dann automatisch in die App integriert. Die UI[2] wird nicht über *HTML* spezifiziert, sondern direkt über die Programmiersprache erstellt. Interpretierte Apps laufen auf allen Geräten oder Plattformen, auf denen es einen Interpreter für die dynamische Programmiersprache gibt.

Das Wegfallen von *HTML* und *CSS* kann die App beschleunigen, da das Parsen und das Setzen des Layouts über *CSS* durch seine Flexibilität relativ komplex und damit zeitaufwendig werden kann. Außerdem hat das Framework Zugriff auf die Komponenten des Betriebssystems, wodurch die UI-Elemente genauso wie native Elemente gerendert werden können. Das erlaubt eine konsistente Benutzeroberfläche, die sich gut in das Aussehen des Betriebssystems eingliedert.

[1] AJAX, Asynchronous JavaScript and XML: Erlaubt das aufbauen von HTTP-Verbindungen zu einem Server ohne neues Laden der Webseite.
[2] UI, User Interface: Die Benutzeroberfläche.

Die Benutzung eines Interpreters führt jedoch zu einem Overhead, der den Start und die Ausführung der App wieder verlangsamen kann. Somit ist auch der Interpreteransatz nicht für sehr rechenintensive Anwendungen geeignet.

2.4 Cross-To-Native

Bei Cross-To-Native-Apps wird eine Sprache verwendet, die auf allen Plattformen lauffähig ist. Beispielsweise können *C* und *C++* dank einer breiten Auswahl an Compilern für die Zielplattform kompiliert werden [3] [4] [5] und *.NET*-Sprachen wie *C#* sind dank *Mono*[1] auf vielen Plattformen ausführbar. Allein die plattformspezifische Anbindung der Sprachen ist plattformabhängig. Dieser Teil der App kann entweder durch ein Framework implementiert oder durch den Entwickler für jede Zielplattform separat entwickelt werden. Der größte Teil der App, die Anwendungslogik und Teile der Benutzeroberflächensteuerung, werden dabei komplett plattformunabhängig ausgelegt.

Da der Code der App kompiliert (oder im Falle von *Java* und *.NET JIT*-kompiliert) wird, bietet dieser Ansatz die bestmögliche Performanz für die App, ohne dabei die Plattformunabhängigkeit aufzugeben. Somit ist Cross-To-Native auch für rechenintensive Anwendungen geeignet.

Wird ein Framework verwendet, so entsteht allerdings eine große Abhängigkeit vom Anbieter des Frameworks und die Zukunft der App wird an das Fortleben des Frameworks gekoppelt. Ohne ein Framework dagegen entsteht ein Mehraufwand für den Entwickler, da nun das Bindeglied zwischen dem plattformunabhängigen Code und dem Plattformcode für jede Plattform separat entwickelt und aktuell gehalten werden muss.

2.5 Native

Eine native App wird in der Programmiersprache und dem Framework geschrieben, die für die Plattform vorgesehen ist. Eine *Android*-App wird zum Beispiel in *Java* mit dem *Android-SDK* entwickelt. Andere Plattformen werden dabei nicht direkt unterstützt, sodass die App nicht plattformunabhängig ist.

[1] Mono: Eine Open-Source-Implementierung des *.NET-Frameworks* und der *Common Language Runtime (CLR)*.

Eine App nach dem nativen Ansatz ist sehr gut auf die Zielplattform abgestimmt, sodass plattformspezifische Besonderheiten wie Sensoren oder Clouddienste[1] ohne Probleme benutzt werden können. Außerdem laufen diese Apps direkt auf dem Betriebssystem, womit ihre Performanz gut ist. Sie sind also für rechenintensive Aufgaben geeignet.

Durch die Abhängigkeit zur Plattform allerdings muss die App fast komplett neu entwickelt oder portiert werden, wenn eine neue Plattform unterstützt werden soll. Das führt zu nicht unerheblichem Mehraufwand. Wenn die App plattformspezifische Eigenschaften wie besondere Sensoren oder Clouddienste verwendet, die auf einer anderen Plattform nicht zur Verfügung stehen, so muss zusätzlich ein Ersatz für diese Features gefunden werden und der Funktionsumfang der App könnte darunter leiden.

3 Frameworks

3.1 Auswahlkriterien

Zur Auswahl der Frameworks gab es mehrere Kriterien. So sollte für alle in Architekturansätze beschriebenen Ansätze ein Framework vorgestellt werden, das diesen implementiert. Gerade für den Ansatz Hybrid gibt es allerdings sehr viele neu entstehende Frameworks, die häufig auf *PhoneGap/Apache Cordova* aufbauen. Einige davon in dieser Arbeit nicht näher beschriebenen Frameworks sind *Mobile Angular UI* [6], *Intel XDK* [7], *Kendo UI* [8], *jQuery Mobile* [9] und *Onsen UI* [10]. Da die Technologie hinter diesen Frameworks auf dem gleichen Prinzip basiert, unterscheiden sie sich allerdings nicht signifikant voneinander, sondern haben ihre Stärken in Zusatzfeatures, Dokumentation und Entwicklerfreundlichkeit.

Des Weiteren sollte das Framework auch auf die Entwicklung von generellen Apps zugeschnitten sein, womit Spieleengines wie *Unity* entfallen, obwohl sie eine breite Auswahl an Zielplattformen unterstützen können.

Frameworks und die mobile Smartphoneumgebung entwickeln sich stetig weiter, weshalb auch nur Frameworks ausgewählt wurden, die noch aktiv weiterentwickelt werden. Für einen größtmöglichen Funktionsumfang wurden außerdem alle Frameworks ausgelassen, die sich noch in Entwicklung befinden und noch keine offizielle Releaseversion haben.

1 Google Play beispielsweise ist nur auf Android verfügbar und bietet Zugriff auf viele Google-Angebote.

3.2 PhoneGap / Apache Cordova

3.2.1 Überblick

PhoneGap wurde von *Nitobi* beim *iPhone Dev Camp* 2008 als *iPhone*-Framework entwickelt. Das Ziel der Entwickler war, Apps für das iPhone zu schreiben, ohne dass eine einzige Zeile *Objective C* geschrieben werden muss. Stattdessen sollten App-Entwickler die Möglichkeit erhalten, Apps wie Webseiten zu schreiben, mit *HTML*, *CSS* und *JavaScript*. Die App erhält dabei dennoch Zugriff auf bestimmte Geräte-APIs wie GPS, Kamera, Kontakte und andere über eine geräteübergreifende *JavaScript*-Schnittstelle [11].

Im Jahre 2011 wurde *Nitobi* von *Adobe* gekauft und das *PhoneGap*-Projekt an die *Apache Software Foundation* gespendet, wo es unter dem Namen *Apache Cordova* weitergeführt wird [12].

Die Unterschiede zwischen *PhoneGap* und *Apache Cordova* sind minimal. Abgesehen vom Namen ist *PhoneGap* identisch zu *Apache Cordova* und wird immer darauf basieren. Es besteht jedoch die Möglichkeit, dass *Adobe PhoneGap* Tools hinzufügt, die besonders gut mit anderen *Adobe*-Produkten zusammenarbeiten und durch ihre proprietäre Natur nicht für ein *Apache*-Projekt geeignet sind [13].

PhoneGap ist ein Hybrid-Framework. Eine *PhoneGap*-App besteht aus einer oder mehrerer *HTML*-Seiten, die mit *CSS* gestaltet und mit *JavaScript* mit Funktionalität gefüllt werden können. Die Webseiten werden dann in einem speziell angepassten Browserfenster auf dem Gerät dargestellt. Dieses Browserfenster stellt viele *JavaScript*-APIs zur Verfügung, mit denen auf Sensoren, Kontakte, Kamera und mehr zugegriffen werden kann, sodass im Vergleich zu nativen Apps kaum Funktionalität verloren geht. Da die Browserfenster auf älteren Smartphones einige moderne *HTML5*- und *CSS3*-Standards nicht richtig unterstützen, übernimmt das *PhoneGap*-Framework diese Funktionalität auf diesen Geräten, sodass die App selbst dort zufriedenstellend laufen kann.

Nach eigener Aussage wurde *PhoneGap* mehr als eine Millionen Mal heruntergeladen und wird von über 400.000 Entwicklern benutzt. Es befinden sich außerdem mehr als 1.000 Apps in diversen Smartphone-Stores, die mit *PhoneGap* entwickelt wurden [14]. Zudem gibt es ein offizielles *PhoneGap*-Forum, welches gut besucht ist [15]. Auf *StackOverflow*, einer beliebten Frage-Antwort-Seite für Softwareentwickler, wurden zu *PhoneGap* / *Apache Cordova* bereits mehr als 45.000 Fragen gestellt [16].

Die Dokumentation für *PhoneGap* erlaubt einen schnellen Einstieg in die Entwicklung mithilfe einiger Tutorials, die Dokumentationen der Plugins allerdings, die für die Kommunikation zu den Services des Smartphones verantwortlich sind, verlinken nur zu den Projektseiten der Plugins auf *GitHub* [17].

PhoneGap ist ein Open-Source-Framework, das heißt, der Quelltext des Frameworks ist frei einsehbar. Es basiert auf *Apache Cordova* und steht, genauso wie *Apache Cordova* selbst, unter der *Apache Lizenz, Version 2.0* [18]. Laut dieser Lizenz darf das Framework auch in kommerziellen Projekten verwendet werden, ohne dass Gebühren anfallen, und der Quelltext des Frameworks kann beliebig modifiziert werden. Allein der Text der Apache-Lizenz sowie eine „NOTICE"-Datei muss dem Projekt beigefügt werden.

3.2.2 Projektstruktur

Die generelle Projektstrukturierung bleibt bei *PhoneGap*/*Apache Cordova* dem Entwickler überlassen, nur generelle Build- und Konfigurationsdateien und -ordner werden angelegt. Eine Strukturvorgabe von beispielsweise MVC gibt es nicht. Das erlaubt dem Entwickler einige Freiheiten bei der Entwicklung, erfordert aber auch Disziplin, um das Projekt in Ordnung zu halten.

Um ein neues Projekt anzulegen, muss folgender Befehl auf einer Kommandozeile eingegeben werden. Der Name der EXE, entweder *PhoneGap* oder *Cordova*, hängt vom verwendeten Framework ab, die Parameter sind aber für beide gleich.

```
phonegap create AppName
```

Listing 1 Befehl zum Anlegen eines PhoneGap-Projekts

Der Befehl wird im aktuellen Arbeitsverzeichnis ein Verzeichnis mit dem eingegebenem Appnamen als Namen erzeugen und eine Grundstruktur für die App anlegen. Im Projektverzeichnis werden folgende Ordner und Dateien erstellt:

- `config.xml`. Diese Datei enthält wichtige Informationen über das Projekt, wie den Namen der App, die Beschreibung der App, und was die App für Erlaubnisse vom System benötigt.
- `hooks`. Kann Skripte enthalten, die vor oder nach bestimmten Aktionen ausgeführt werden, wie zum Beispiel vor dem Verpacken der App.
- `platforms`. Dieser Ordner enthält Dateien, die speziell für eine bestimmte Zielplattform gedacht sind. Das Buildsystem wird diese Dateien größtenteils selbstständig anlegen, je nachdem, welche Zielplattformen in der `config.xml`-Datei

eingetragen sind. Weitere Zielplattformen können mit Kommandozeilenbefehlen zum Projekt hinzugefügt werden.

- `plugins`. In diesem Ordner werden Plugins gespeichert, die vom Projekt benötigt werden. Plugins sind native Programmteile, die sich um die Verbindung mit dem Betriebssystem kümmern. Die Dateien in diesem Ordner werden, abhängig von den Einträgen in der `config.xml`-Datei, ebenfalls vom Buildsystem selbstständig angelegt. Weitere Plugins können über die Kommandozeile hinzugefügt werden.
- `www`. Der `www`-Ordner ist wahrscheinlich der wichtigste Ordner des Projekts, denn er enthält alle eigentlichen Daten der App. Wenn die App gestartet wird, wird als erstes die Datei `index.html` aufgerufen.

Um die App schließlich zu packen, muss folgender Befehl ausgeführt werden:

```
phonegap build
```

Listing 2 Befehl zum Bauen eines PhoneGap-Projekts

Dieser Befehl wird für alle eingetragenen Zielsysteme plattformspezifische Dateien im `platforms`-Verzeichnis erstellen. Möchte man die App nur für eine einzige Zielplattform bauen, kann der Name der Zielplattform an den Befehl angehängt werden.

PhoneGap/*Apache Cordova* erlaubt außerdem das Entwickeln der App direkt im Browser, ohne dass die App auf ein Smartphone transferiert werden muss. Dies ist möglich, da die App in *HTML*, *CSS* und *JavaScript* entwickelt wird, die welches von Browsern verstanden wird. Die Browseransicht kann mit folgendem Befehl aufgerufen werden, mit dem Wurzelverzeichnis der App als Arbeitsverzeichnis:

```
phonegap serve
```

Listing 3 Darstellen einer PhoneGap-App im Desktop-Browser

Der Browserinhalt wird sich außerdem automatisch aktualisieren, wenn die App im Editor oder der IDE bearbeitet wird. Somit werden sämtliche Änderungen an der App sofort sichtbar, was eine Beschleunigung des Entwicklungsprozesses bedeutet.

3.3 Sencha Touch

3.3.1 Überblick

Sencha Touch ist ein Framework, das auf *jQTouch* und *Raphaël* aufbaut. Es zielt darauf ab, gute Grafiken und Unterstützung für Fingergesten auf berührungsempfindlichen Bildschirmen in einem *AJAX*-Framework zu bündeln [19]. *jQTouch* (heute *jQT*) ist ein

jQuery-Plugin, mit dem die Entwicklung von Webseiten oder Web-Apps auf Smartphones oder ähnlichen Geräten erleichtert werden soll. Es bietet unter anderem Unterstützung für Fingergesten, Animationen und Themes und ist unter der *MIT*-Lizenz lizenziert [20]. *Raphaël* ist eine *JavaScript*-Bibliothek zum browserübergreifenden Rendern von Vektorgrafiken. Es ist ebenfalls unter der *MIT*-Lizenz lizenziert [21].

Sencha Touch fügt diese *JavaScript*-Bibliotheken zu einem Web-Framework zusammen. Dabei werden die Seiten der Web-App auf einem Server im Internet gehostet und über *AJAX* mit diesem kommuniziert. Das Framework erlaubt jedoch nicht die Installation einer App direkt auf dem Smartphone, sodass immer eine Internetverbindung bestehen muss.

Es besteht jedoch die Möglichkeit, dass *Sencha Touch* mit einem anderen Framework wie *Apache Cordova* kombiniert wird. Dabei wird *Sencha Touch* direkt in die App eingebettet und bietet weiter Unterstützung für Grafiken und Animationen, während *Apache Cordova* sowohl *Sencha Touch* als auch die App zu einem einzigen Paket verpackt, das auf dem Smartphone des Benutzers installiert werden kann. Diese Methode bietet außerdem den Vorteil, dass *Apache Cordova* Zugriff auf die Sensoren und andere Ressourcen des Smartphones zulässt.

Im Vergleich mit den anderen vorgestellten Frameworks scheint *Sencha Touch* weniger Aktivität aufzuweisen, mit weniger als 5.000 Fragen auf *StackOverflow*, aber einem gut besuchten Forum [22] [23]. Die Onlinedokumentation von *Sencha Touch* dokumentiert die bereitgestellten Komponenten aber gut, und es gibt Tutorials mit Beispielquelltext, die einen schnellen Einstieg in das Framework ermöglichen [24].

Sencha Touch ist unter mehreren Lizenzen verfügbar. Es gibt eine Probelizenz, die es ermöglicht, *Sencha Touch* 30 Tage lang auszuprobieren. Um die mit *Sencha Touch* entwickelten Apps zu veröffentlichen, benötigt man entweder eine kommerzielle Lizenz oder kann seine App unter die *GPLv3*-Lizenz stellen. Die *GPLv3* Open-Source-Lizenz hat unter anderem allerdings die Bedingung, dass der Quellcode der App veröffentlicht werden muss, was nicht bei jeder App sinnvoll ist. Wenn die Bedingungen der vorherigen Lizenzen für eine App zu restriktiv sind, kann auch über eine angepasste Lizenz verhandelt werden [25].

3.3.2 Projektstruktur

Sencha Touch gibt eine MVC-Strukturierung vor, damit sämtliche Projektdateien ihren korrekten Platz haben. So wird auch eine gute Trennung von Oberfläche, Businesscode und Daten erreicht.

Ein neues Projekt wird über die Kommandozeile erstellt. Das *Sencha Touch*-Kommandozeilenprogramm wird daraufhin eine leere, aber schon lauffähige App erstellen.

```
sencha –sdk /pfad/zu/sencha-touch-sdk generate app AppName
      /pfad/zu/www/appname
```

Listing 4 Befehl zum Anlegen eines Sencha Touch-Projekts

Im Projektverzeichnis werden daraufhin folgende Ordner und Dateien angelegt:

- `.sencha`. Der `.sencha`-Ordner enthält Konfigurationsdateien sowohl für die App als auch für das Kommandozeilenprogramm.
- `touch`. Dieser Ordner enthält eine Kopie des *Sencha Touch*-SDKs. Es wird dazu verwendet, die App zu bauen.
- `app`. Der Ordner `app` enthält den Quelltext der App, organisiert mit der Model-View-Controller-Struktur. Diese Struktur darf nicht verändert werden, denn das Kommandozeilenprogramm wird beim Anlegen neuer Quelltextdateien diese Struktur voraussetzen.
- `resources`. Die Ressourcen der App, wie zum Beispiel Stylesheets, Bilder und Icons werden in diesem Ordner gespeichert.
- `index.html`. Enthält die Grundstruktur des Layouts der App.
- `app.js`. Diese Quelltextdatei enthält die Initialisierungslogik der App und ist dafür verantwortlich, den Rest der App zu initialisieren.
- `app.json`. Diese Datei enthält Eigenschaften der App wie ihren Namen und wer sie entwickelt hat.
- `packager.json`. Diese Datei beinhaltet Informationen darüber, wie die App gepackt werden soll.

Neue Controller oder Modelle werden nicht manuell erstellt, sondern können mit dem Kommandozeilenprogramm angelegt werden. Das Programm wird für den Entwickler eine leere Grundstruktur in der erstellten Datei anlegen und diese in `app.js` selbstständig registrieren. Der Befehl zum Anlegen neuer Komponenten sollte mit dem Grundverzeichnis der App als Arbeitsverzeichnis ausgeführt werden.

```
sencha generate model User --fields=id:int,name,email
```

Listing 5 Befehl zum Hinzufügen von Modellen zu einem Sencha Touch-Projekt

Apps, die mit *Sencha Touch* entwickelt wurden, können sowohl als Webseite zum Anzeigen im Browser aller möglichen Geräte als auch als App für Smartphones gepackt werden. Dazu muss nur ein Parameter für den Packvorgang verändert werden, Änderungen an der App sind nicht nötig. Um die App für Smartphones zu packen, genügt folgender Befehl:

```
sencha app build native
```

Listing 6 Befehl zum Bauen eines Sencha Touch-Projekts

Die zu bauenden Versionen für verschiedene Betriebssysteme werden vom Kommandozeilenprogramm aus der `packager.json`-Datei entnommen. Während des Packvorgangs werden alle Projektabhängigkeiten zu Programmpaketen anderer Entwickler aufgelöst und nur verwendete Pakete werden auch mit der App verpackt. Außerdem werden alle Quelltexte und Stylesheets der App komprimiert, um Speicherplatz zu sparen.

3.4 Ionic

3.4.1 Überblick

Ionic wurde am 21. November 2013 als *HTML5*-Framework veröffentlicht [26]. Es soll dabei helfen, portablen *HTML5*-Apps das Nutzergefühl wie bei nativen Apps zu geben. Als Grundlage nutzt *Ionic* *AngularJS*, mit welchem Web-Apps mit einem eher aus nativer Entwicklung bekannten View-Controller-Prinzip geschrieben werden können [27]. *AngularJS* nutzt dazu Datenbindungstechniken, mit denen der *HTML*-Code und der *JavaScript*-Code gut voneinander getrennt werden können. *AngularJS* ist unter der *MIT*-Lizenz lizenziert [28]. Im Jahr 2015 hat *Ionic* 250.000 neue Entwickler hinzugewonnen [29].

Als Hybrid-Framework werden mit *Ionic* Apps in *HTML5* geschrieben, die auf dem Gerät des Anwenders installiert werden können. Als Backend dafür dient *Apache Cordova*, auf dem mit weiteren *JavaScript*-Bibliotheken aufgebaut wird [30].

Ionic besitzt ein offizielles, aktives Forum und hat mit 16.000 Fragen auf *StackOverflow* eine gute Nutzeraktivität [31] [32]. Die Dokumentation der mitgelieferten Komponenten ist ausführlich und hat eine interaktive Vorschau. Ebenfalls verfügbar sind Tutorials mit Beispielquelltext, die einen schnellen Einstieg in das Framework ermöglichen [33].

Das *Ionic*-Framework steht unter einer sehr freizügigen *MIT*-Lizenz [34].

3.4.2 Projektstruktur

Da das *Ionic* Framework auf *Apache Cordova* aufbaut, ist die grundlegende Projektstruktur die gleiche. Nur der Inhalt des `www`-Ordners wird durch die Verwendung von AngularJS stärker strukturiert, sodass die Ordnung im Projekt einfacher einzuhalten ist.

Der Befehl zum Anlegen eines neuen Projekts wird nicht nur ein neues *Cordova*-Projekt erstellen, sondern auch Dateien in das Projekt einfügen, die zu *Ionic* gehören. (Siehe auch Listing 1 Befehl zum Anlegen eines PhoneGap-Projekts).

```
ionic start AppName
```

Listing 7 Befehl zum Anlegen eines Ionic-Projekts

Da *Ionic* auf *Apache Cordova* aufbaut, erlaubt es das Entwickeln der App direkt im Browser, ohne dass die App auf ein Smartphone transferiert werden muss. Sämtliche Änderungen werden ebenfalls sofort im Browser angezeigt. Die Browseransicht kann mit folgendem Befehl aufgerufen werden, mit dem Wurzelverzeichnis der App als Arbeitsverzeichnis:

```
ionic serve
```

Listing 8 Befehl zum Anzeigen eines Ionic-Projekts im Browser

Der Quellcode der App befindet sich, wie auch bei *Apache Cordova*, im Ordner `www`. In diesem Ordner können beliebig neue Quelltextdateien angelegt werden. Empfohlen wird jedoch, dass jede neue Seite der App in ihrem eigenen Unterordner gespeichert wird, der denselben Namen wie die Seite haben sollte. Das hilft, die App zu organisieren. Um die neue Seite dann zu nutzen, muss sie danach nur in den vorhandenen Quelltexten importiert werden.

Das Packen der App wird ebenfalls über die Kommandozeile durchgeführt:

```
ionic build PlattformName
```

Listing 9 Befehl zum Packen eines Ionic-Projekts

Nachdem die App gepackt wurde, kann sie signiert und in einen App-Store der entsprechenden Plattform hochgeladen werden.

3.5 Titanium

3.5.1 Überblick

Titanium wurde im Dezember 2008 eingeführt. Es sollte als Konkurrenzprodukt zu *Adobe AIR* die einfache Cross-Plattformentwicklung zwischen verschiedenen Desktops,

Browsern und Smartphones ermöglichen. Dazu nutzte *Titanium* die Webtechnologien *HTML*, *CSS* und *JavaScript* [35]. Die Entwicklung von *Titanium* konzentrierte sich darauffolgend eher auf den Mobilbereich. 2012 wurde die Komponente für plattformübergreifende Desktops dann in ein eigenes Projekt namens *TideSDK* ausgelagert, und die Oberflächen wurden nicht länger mit *HTML* gebaut [36].

Da *Titanium* ein Interpreter-Framework ist, wird der vom Entwickler geschriebene *JavaScript*-Code auf dem Endgerät von einem *JavaScript*-Interpreter interpretiert. Im reinen Titanium werden UI-Elemente mit *JavaScript* erstellt und auf dem Bildschirm dargestellt. Mit dem MVC[1]-Framework *Titanium Alloy* ist es aber auch möglich, die Oberfläche stattdessen mit XML zu beschreiben, was sie gut vom Rest des Quellcodes trennt.

Die Aktivität von *Titanium* im Vergleich zu den anderen vorgestellten Frameworks im Internet ist eher niedrig, mit weniger als 5.000 Fragen auf *StackOverflow* [37]. Ebenso besitzt *Titanium* kein eigenes Forum, in dem Fragen gestellt werden könnten [38].

Die Dokumentationswebseite für *Titanium* ist umfangreich und beschreibt sämtliche Komponenten des Frameworks, hat aber lange Startzeiten[2] und ist auf den ersten Blick etwas unübersichtlich [39].

Sowohl *Titanium* als auch *Alloy* sind unter der *Apache*-Lizenz lizenziert und dürfen somit auch problemlos in kommerziellen Projekten verwendet werden [40].

3.5.2 Projektstruktur

Projekte in reinem *Titanium* dürfen vom Entwickler frei strukturiert werden. Dies erlaubt mehr Freiheiten, erfordert aber auch eine höhere Disziplin vom Entwickler, um die Ordnung im Projekt zu wahren. Wird *Titanium Alloy* verwendet, so ist die Struktur des Projekts hingegen vorgegeben, die Trennung der Programmteile wird also vom Framework übernommen.

Ein neues Projekt wird unter *Titanium* mit *Appcelerator Studio*, einer Entwicklungsumgebung für *Titanium*-Projekte, erstellt. Ein Wizard wird dazu die nötigen Informationen sammeln und ein neues Projekt mit folgender Struktur für den Entwickler anlegen:

- `Resources`. Dieser Ordner enthält sowohl Quelltexte der App als auch Bilder und andere Ressourcendateien.

[1] MVC, Model View Controller: Muster zur Strukturierung von Software.
[2] Die Webseite lädt etwa 3 Megabyte an JavaScript-Daten, deren Herunterladen und Ausführen das Laden der Webseite auf einem modernen PC um einige Sekunden verzögert [Aufgerufen 12. Juli 2016].

- `app.js`. Diese *JavaScript*-Datei ist der Einstiegspunkt der App. Sämtliche Programmlogik kann in dieser Datei enthalten sein, aber auch auf andere Quelltexte verteilt werden.
- `tiapp.xml`. Diese Datei enthält verschiedene Informationen über die App wie ihren Namen. Die im Wizard gesammelten Informationen sind in dieser Datei gespeichert.

Zu *Titanium* kann zusätzlich *Alloy* verwendet werden. *Alloy* ist ein *Titanium*-spezifisches MVC-Framework, mit dem Oberfläche und Programmlogik besser voneinander getrennt werden können. Wird es benutzt, werden zusätzlich folgende Ordner und Dateien innerhalb eines `app`-Ordners angelegt:

- `alloy.jmk`. Diese Datei enthält Informationen über den Packvorgang der App.
- `alloy.js`. Dieser Quelltext enthält Code zum initialisieren der *Alloy*-Komponenten.
- `config.json`. Diese Datei beinhaltet Konfigurationseinstellungen der App.
- `assets`. In diesem Ordner werden Bilder und andere Ressourcendateien gespeichert.
- `controllers`. Dieser Ordner enthält alle Controller-Quelltextdateien. Jede Datei muss eine gleichnamige View-Datei im Ordner `views` haben.
- `i18n`. Sprach- und Übersetzungsdateien werden in diesem Ordner gespeichert.
- `lib`. Falls die App weiteren Programmcode benötigt, kann er in diesem Ordner gespeichert werden.
- `migrations`. Dieser Ordner enthält *Migrations*. Das sind Quelltexte, mit denen Datenbankschemen auf die neuste Version migriert werden können, ohne Daten zu verlieren.
- `models`. Alle Modellquelltexte des MVC-Musters werden in diesem Ordner gespeichert.
- `platform`. Dieser Ordner enthält plattformspezifische Ressourcendateien.
- `specs`. Dieser Ordner verhält sich wie der `lib`-Ordner, nur dass die Dateien in ihm für ein *Production*[1]-Paket ignoriert werden. Er ist deshalb für Testdateien geeignet.

[1] Production: Eine Art, Pakete zu bauen, bei der alle Debuginformationen entfernt und das Programm mit Fokus auf Geschwindigkeit kompiliert wird. Diese Pakete sind dafür gedacht, im produktiven Einsatz zu laufen.

- `styles`. Styles für die Views werden in diesem Ordner gespeichert. Sie müssen ein *Titanium*-eigenes Format aufweisen.
- `themes`. Themes beeinflussen das komplette Aussehen der App.
- `views`. Die Views der App, also die Struktur gewisser Programmkomponenten, werden hier als XML gespeichert.
- `widgets`. *Widgets* sind Komponenten, die selbst Views, Controller und Modelle enthalten. Ihre Struktur ist ähnlich der von ganzen Apps.

Das Konfigurieren und Packen der App wird über die IDE vorgenommen. Einige speziellere Modifikationen am Projekt benötigen jedoch weiterhin eine Kommandozeile, da die IDE nicht alle Konfigurationsmöglichkeiten unterstützt.

3.6 Xamarin

3.6.1 Überblick

Xamarin wurde im Mai 2011 mit dem Ziel gestartet, *.NET*-Codeentwicklung für *iOS* und *Android* bereitzustellen. Dazu wird *Mono* verwendet, eine Open-Source-Alternative für das *.NET Framework*, wobei *Mono* selbst ebenfalls von den *Xamarin*-Entwicklern weiterentwickelt wird [41]. Im Februar 2016 wurde *Xamarin* dann von *Microsoft* übernommen. *Xamarin* wurde daraufhin mit einem Plugin in *Visual Studio* integriert, um die Cross-Plattformentwicklung mit *C#* für *iOS*, *Android* und *OS X* zu ermöglichen [42]. Im März des gleichen Jahres wurden sowohl *Mono* als auch *Xamarin* unter eine sehr offene *MIT*-Lizenz gestellt, um eine weitere Verbreitung der Technologien zu erreichen [43].

Xamarin ist ein Cross-To-Native-Framework. Der Programmcode der App wird von einem *.NET*-kompatiblen Compiler zur *IL*[1] kompiliert und erst auf dem Zielsystem, sei es Smartphone oder Desktop PC, zu Maschinencode übersetzt und ausgeführt. Die Programmiersprache ist dabei nicht auf *C#* beschränkt. Es können zum Beispiel auch *F#*, *Visual C++* oder *Visual Basic* verwendet werden, da diese ebenfalls zu *IL* kompiliert werden können.

Um nicht nur die Business-Logik von Apps plattformunabhängig schreiben zu können, sondern auch die Oberfläche, existiert die *Xamarin*-Komponente *Xamarin.Forms*. Sie erlaubt das Erstellen von Oberflächen mit XML. Die einzelnen UI-Komponenten werden dabei auf dem Zielsystem durch native Komponenten bereitgestellt, wodurch die App ein

[1] IL, Intermediate Language: Objektorientierte Assemblersprache für die *Common Language Runtime (CLR)*, ähnlich wie *Java*-Bytecode für die *Java Virtual Machine (JVM)*.

natives Benutzergefühl vermittelt. Laut *Xamarin* lassen sich so über 96% des Quelltextes, je nach Anforderungen, zwischen Plattformen wiederverwenden [44]. Wird *Forms* nicht verwendet, müssen die Oberflächen für jede Plattform einzeln geschrieben und aktualisiert werden. Dies kann unter Umständen eine Fehlerquelle sein, wenn eine Aktualisierung der App in einer der Oberflächen vergessen oder unvollständig umgesetzt wird.

Xamarin besitzt ein aktives Forum und auf *StackOverflow* hat das Framework eine gute Aktivität mit bereits 14.000 gestellten Fragen [45] [46]. Die Onlinedokumentation von *Xamarin* ist von allen vorgestellten Frameworks die umfangreichste. Es existieren neben einer kompletten Beschreibung der API zahlreiche Guides und Codebeispiele für unterschiedliche Szenarien sowie komplette Beispiel-Apps [47].

Mono, auf das *Xamarin* aufbaut, ist seit März 2016 *MIT*-lizenziert [48] und darf kostenfrei auch in kommerziellen Produkten verwendet werden. Vorher war es *LGPL*-lizenziert. Für *Xamarin* selbst existiert seit diesem Zeitpunkt ebenfalls eine kostenlose Community-Version, die *MIT*-lizenziert ist. Deren Quelltext soll in den nächsten Monaten veröffentlicht werden [43].

3.6.2 Projektstruktur

Ein *Xamarin*-Projekt wird typischerweise mit *Microsoft Visual Studio* entwickelt, es ist aber auch möglich, die *Xamarin*-eigene IDE *Xamarin Studio* zu verwenden. Wird in *Visual Studio* ein neues Projekt erstellt, so legt es je nach Auswahl der Zielplattformen eine Projektmappe mit mehreren Projektordnern an.

Der wichtigste Projektordner dabei ist der, der nach dem Namen der App benannt ist. In ihm befindet sich ein typisches *Visual Studio*-Projekt. Es enthält einen Ordner `Properties` mit einer Datei `AssemblyInfo.cs`, die Metainformationen über das Programm enthält, sowie eine Datei `App.cs`, die den Einstiegspunkt der App enthält. Dieses Projekt ist dazu gedacht, all den Quelltext zu fassen, der zwischen den Zielplattformen gemeinsam verwendet werden kann, wie den *Data Layer*[1], den *Data Access Layer*[2] und den *Business Layer*[3].

[1] Data Layer: Eine Verarbeitungsschicht, die das Speichern und Laden von Daten durchführt, sei es mithilfe einer Datenbank oder XML-Dateien.
[2] Data Access Layer: Eine Verarbeitungsschicht, die im Gegensatz zum *Data Layer* vollständige Objekte aus den geladenen Daten erzeugt und dazu in der Lage ist, ganze Objekte wieder zu speichern.
[3] Business Layer: Enthält Programmlogik, die sich mit dem Lösen der Problemstellung für das Programm befasst, anstatt technische Belange abzudecken, wie die Datenspeicherung oder grafische Oberflächen.

Für jede Zielplattform wird ein weiterer Projektordner erstellt, der jeweils mit dem Namen des Projekts anfängt und ein plattformspezifisches Suffix hat. Für *Android* beispielsweise lautet dieses Suffix *Droid*, für *iOS* lautet es *iOS*. Die Inhalte der Projekte sind dabei stark an die nativen Projekte der jeweiligen Plattform angelehnt, da mit *Xamarin Forms* die komplette Benutzeroberfläche mitsamt dem Quelltext, der diese kontrolliert, für jede Plattform einzeln entwickelt werden muss, dafür aber eine gute Leistung und Ergonomie verspricht.

Wird *Xamarin Forms* verwendet, so kann ebenso wie der *Business Layer* auch die grafische Oberfläche in das gemeinsame Projekt gespeichert werden. *Xamarin* wird dann für die Elemente der gemeinsamen *XAML*[1]-Oberfläche native Elemente generieren, die sich nicht von denen einer nativen App unterscheiden. Beispielsweise wird ein `Xamarin.Forms.Entry` ein `UITextView` auf *iOS*, ein `EditText` auf *Android* und eine `TextBox` auf *Windows Phone*.

3.7 Eigenes Framework

Wenn man sich nicht von einen Frameworkanbieter abhängig machen möchte, besteht die Möglichkeit, sein eigenes, plattformübergreifendes Framework zu schreiben. Dies ist, je nach Funktionsumfang der App, jedoch nicht trivial.

Die datenverarbeitenden Schichten der App können in einer beliebigen Sprache geschrieben werden, die entweder auf den Zielsystemen interpretiert oder für diese kompiliert werden kann. Ist die Datenverarbeitungsschicht der App sehr groß oder komplex, wird dadurch viel sonst duplizierter Code eingespart. Für jedes System, das unterstützt werden soll, muss dann sowohl eine Oberfläche inklusive Interaktionslogik als auch ein *Wrapper* um die datenverarbeitende Schicht separat geschrieben werden. Der *Wrapper* hat die Aufgabe, den datenverarbeitenden Code zu laden und ein Interface zu diesem für die Oberfläche bereitzustellen. Die Oberflächenlogik wird also niemals direkt mit der datenverarbeitenden Schicht interagieren, sondern nur mit dem *Wrapper*. Der datenverarbeitende Code der App ist in einer Bibliotheksdatei gepackt. Je nach System kann diese dann entweder statisch gegen die jeweilige Oberfläche gelinkt oder vom *Wrapper* zur Laufzeit der App dynamisch geladen werden.

Ein Vorteil, sein eigenes Framework zu schreiben, ist große Unabhängigkeit. Das Fortbestehen der App ist nicht zwangsläufig an den Erfolg eines Frameworks gebunden, das in

[1] XAML, Extensible Application Markup Language: Eine XML-basierte Sprache zur Beschreibung von grafischen Oberflächen.

Zukunft vielleicht nicht mehr weiterentwickelt wird. Außerdem erreicht man mit dieser Methode einen hohen Grad an Freiheit, denn jede Interaktion mit dem System steht unter der vollen Kontrolle der Entwickler. So werden zum Beispiel Optimierungen möglich, da die eigene App möglicherweise bestimmte Features gar nicht benötigt. Andere Frameworks können solche Optimierungen nicht durchführen, da sie nicht im Vorhinein bestimmen können, welche Features benötigt werden und welche nicht.

Diese Vorteile werden allerdings erkauft durch einen enormen Mehraufwand. So müssen pro Zielsystem eigene *Wrapper* geschrieben werden. Wenn ein Entwickler dabei auf ein Hindernis stößt, so hat er auch nicht die Möglichkeit, die Foren oder Hilfeseiten des Frameworks zu durchsuchen, sondern ist bei der Lösung seines Problems weitgehend auf sich alleine gestellt. Die Entwicklung eines eigenen Frameworks ist damit auch sehr zeitaufwendig und teuer. Es lohnt sich also nur in seltenen Fällen, diesen Weg zu gehen, wenn Plattformunabhängigkeit gewünscht ist.

3.8 Andere Frameworks

3.8.1 Unity

Unity ist eine weit verbreitete Grafikengine [49]. Mit *Unity* und den mit *Unity* mitgelieferten Werkzeugen lassen sich 2D- und 3D-Spiele für eine breite Palette an Plattformen erstellen, darunter auch *Android*, *iOS* und *Windows* [50]. Da *Unity* eine umfassende Grafikengine ist und als solche weniger ressourcensparend ist als alternative Frameworks sollte es nicht für generelle Apps verwendet werden, da sonst die begrenzte Akkukapazität mobiler Geräte schneller ausgeschöpft ist.

3.8.2 Unreal Engine

Die *Unreal Engine* von *Epic Games* ist eine hochentwickelte, flexible Grafikengine und bietet ein Paket von Programmen zur Entwicklung von Spielen an. Sowohl 2D- als auch 3D-Spiele können mithilfe der *Unreal Engine* gebaut werden [51]. *Android* und *iOS* werden unterstützt, derzeit aber noch nicht *Windows Phone* [52]. Da die *Unreal Engine* auf Spiele zugeschnitten ist, eignet sie sich nur für solche. Durch ihren Fokus auf 2D- und 3D-Grafik verbraucht sie mehr Ressourcen, die auf einem Smartphone sehr begrenzt sind, und leert somit den Akku dieser Geräte schneller. Generelle Apps für *Android* und *iOS* sollten also nicht mit ihr geschrieben werden.

3.8.3 Windows 10 Universal App

Mit *Universal Apps* stellt *Microsoft* eine API zur Verfügung, mithilfe derer plattformübergreifend Apps entwickelt werden können. Dazu stehen Entwicklern verschiedene Sprachen zur Verfügung, wie *C#*, *Visual Basic* oder *JavaScript*. Diese *Universal Apps* können dann auf Geräten unterschiedlicher Kategorien ausgeführt werden ohne angepasst werden zu müssen, denn die API ist auf allen Geräten gleich. So kann eine *Universal App* auf einem Smartphone, auf einem PC, auf einem Tablet und auf der Spielkonsole *Xbox* ausgeführt werden [53]. Da *Universal Apps* aber auf einer proprietären API von *Microsoft* aufbauen, sind sie inkompatibel zu anderen Systemen, die nicht unter *Windows 10* laufen. Dazu zählen *iOS* und *Android*, aber auch *Windows 8.1* wird nicht unterstützt [54]. Diese Beschränkung macht diese Variante der Plattformunabhängigkeit unattraktiv, wenn man Apps speziell für Smartphones entwickeln möchte.

3.9 Auswahl des Frameworks

3.9.1 Kriterien

Die Auswahl eines Frameworks für die XXO-App erfolgt an verschiedenen Kriterien.

- Nutzeranzahl des Frameworks

Eine große Anzahl an Nutzern ist in verschiedener Weise hilfreich. Mit einer größeren Community ist es wahrscheinlicher, dass Bugs im Framework eher gefunden werden. Wenn das Framework ein Open-Source-Framework ist, haben die Nutzer außerdem die Möglichkeit, diese Bugs auch selbst zu beheben. Des Weiteren führt eine größere Anzahl an Nutzern zu mehr Aktivität in Foren und auf Hilfeseiten, sodass Fragen zu bestimmten Facetten des Frameworks vielleicht bereits gestellt wurden und eine Lösung auf Probleme somit schnell gefunden ist.

Eine Nutzeranzahl lässt sich allerdings nicht immer genau bestimmen. Viele Frameworks veröffentlichen keine genauen Zahlen, oder geben sie nur stark gerundet an. Diese Zahlen lassen sich auch nicht bestätigen, denn ein Entwickler kann das Framework auch mehrfach herunterladen, oder nur einmal und es dann an andere weiterverteilen. Eine bessere Statistik bietet die Anzahl der Fragen zu einem Framework auf Hilfeseiten, beispielsweise StackOverflow.

- Wartbarkeit

Die Wartbarkeit eines Frameworks ist von großer Bedeutung, denn ein Framework, das sich nicht einfach warten lässt, wird mit der Zeit „Spaghetticode“ entwickeln. Für den

Entwickler wird es dann umso schwieriger, Fehler im alten Code zu finden und neuen Code hinzuzufügen, ohne den alten Code zu beeinträchtigen oder weitere Fehler auszulösen.

- Support

Wenn bei der Verwendung eines Frameworks ein Problem aufkommt, ist schneller und umfangreicher Support für das Framework wichtig. Dieser Support muss nicht zwangsweise von den Entwicklern des Frameworks bereitgestellt werden. Gerade bei Open-Source-Frameworks wird der Support auch von anderen Nutzern des Frameworks angeboten, die ein großes Wissen über das Framework haben oder ein ähnliches Problem bereits selbst gelöst haben.

- Dokumentation

Die Dokumentation von Frameworks erlaubt es dem Entwickler, das Framework kennenzulernen, wie es benutzt wird und wie seine APIs verwendet werden. Eine gute Dokumentation kann schnell Antworten auf häufige Fragen geben und beschleunigt somit den Entwicklungsprozess. Die vorgestellten Frameworks besitzen alle eine gute Onlinedokumentation, die wenigstens einen Startguide zum Erstellen einer App, eine Beschreibung aller vom Framework bereitgestellten Komponenten sowie diverse Tutorials mit Beispielquelltext enthalten, die die allgemeine Entwicklung abdecken.

- Lizenz

Ein Framework mit einer sehr freizügigen Lizenz eröffnet dem Entwickler mehr Möglichkeiten, wie er sein Programm entwickeln oder vermarkten kann. Lizenzen wie *MIT* oder *Apache v2* beschränken den Entwickler kaum und erlegen ihm nur minimale Pflichten auf. Sogenannte „Copyleft"-Lizenzen wie *GPLv3* verpflichten den Entwickler wiederum, seinen Quellcode unter der gleichen oder einer kompatiblen Lizenz zu veröffentlichen, sodass Interessierte den Quelltext auf Fehler oder Malware überprüfen können. Die Veröffentlichung von Quelltext ist jedoch nicht für jedes Projekt geeignet. Einige Frameworks sind auch mit kommerziellen Lizenzen erhältlich. In der Regel sind diese kostenpflichtig und schreiben vor, wie viele Entwickler gleichzeitig an einer App arbeiten dürfen, verlangen aber nicht das Veröffentlichen des Quelltextes.

Einige Frameworks sind unter mehreren Lizenzen erhältlich, oder unterschiedliche Teile des Frameworks sind unterschiedlich lizenziert. Unterstützende Programme zum Beispiel,

wie *IDE*s, Oberflächengestalter oder Codeanalysetools, die die Entwicklung erheblich beschleunigen, könnten nur unter kommerzieller Lizenz erhältlich sein, die dann kostenpflichtig ist.

Die Verwendung einer freizügigen Lizenz erlaubt dem Entwickler, ohne großen Kostenaufwand eine App zu entwickeln, muss dafür jedoch auf teils sehr nützliche Werkzeuge verzichten, die es unter diesen Lizenzen einfach nicht gibt. Kommerzielle Lizenzen dagegen erwarten einen hohen Kostenaufwand gleich zu Beginn der Appentwicklung, was nicht für jedes Projekt vertretbar ist, erlauben dafür aber den Zugriff auf nützliche Hilfsprogramme und Dienste und stellen meist auch einen professionellen Support bereit.

3.9.2 Ausgewähltes Framework

Framework	Wartbarkeit, Strukturierung	Support	Dokumentation	Lizenz
PhoneGap, Apache Cordova	Keine Struktur vorgegeben	Sehr hohe Aktivität	Tutorials	Apache v2
Sencha Touch	MVC-Struktur	Mäßige Aktivität	API, Tutorials, Komponenten	Proprietär, GPLv3
Ionic	MVC-Struktur	Hohe Aktivität	API, Tutorials, Komponenten	MIT
Titanium	Keine Struktur vorgegeben	Mäßige Aktivität	API, Tutorials, Komponenten, lange Ladezeiten	Apache v2
Titanium Alloy	MVC-Struktur			
Xamarin	MVC-Struktur, UI per Plattform	Hohe Aktivität	API, Tutorials, Komponenten, Beispiel-Apps	Proprietär, MIT
Xamarin.Forms	MVVM-Struktur			

Tabelle 1 Framework-Vergleich

Wie man an Tabelle 1 Framework-Vergleich sehen kann, erfüllt von den vorgestellten Frameworks *Ionic* die Anforderungen am besten. Eine große Nutzeraktivität mit mehr

als 320.000 Downloads pro Monat, bereits mehr als 200.000 entwickelten Apps und über 16.000 gestellten Fragen auf StackOverflow hat *Ionic* eine große Nutzerbasis, die sowohl die weitere Entwicklung des Frameworks ermöglicht, als auch im Problemfall schnelle Hilfestellung leisten kann.

Ionic setzt auf eine MVC-Strukturierung und macht es dem Entwickler somit einfacher, das Projekt ordentlich zu verwalten. Das macht die mit *Ionic* entwickelten Apps einfacher wartbar. Gut strukturierter Quellcode hat weniger Nebenwirkungen auf anderen Code und ist somit einfacher austauschbar, erweiterbar und testbar. Dies erreicht *Ionic* mit dem *AngularJS*-Framework, welches Views, Controller, Factories und Services in eigenen Komponenten kapselt. Ein Dependency-Injection-System liefert alle benötigten Ressourcen an die Komponenten, ohne dass diese stark verkoppelt werden.

Zuletzt erlaubt die freizügige *MIT*-Lizenz, den Quellcode von *Ionic* beliebig einzusehen und falls nötig zu verändern. Gleichzeitig muss der eigene Quellcode seinerseits, oder jegliche Änderungen am *Ionic*-Framework nicht veröffentlicht werden, denn er kann unter eine beliebige andere Lizenz gestellt werden, auch kommerzielle Lizenzen. Alternativ kann er gar nicht lizenziert werden, was die Nutzung des Quelltextes an sich durch Dritte komplett ausschließt. Ein wichtiger Punkt ist ebenfalls, dass durch *MIT*-lizenzierte Software keine hohen Lizenzkosten anfallen, was sich für den Umfang der XXO-App gut eignet.

4 Die App

4.1 Anforderungen

Entwickelt werden soll eine plattformunabhängige App, mit der ein XXO-Spiel (auch *Tic Tac Toe* genannt) und darauf aufbauende Spielmodi gespielt werden können. Bei XXO gibt es ein Spielfeld mit 3x3 Feldern. Der Spieler und sein Gegner dürfen in ein nicht belegtes Feld jeweils abwechselnd ihr Zeichen setzen. Ziel des Spiels ist es, drei seiner Zeichen in einer Reihe anzuordnen und den Gegner daran zu hindern, dies mit seinen Zeichen zu tun.

Ein auf das XXO-Spielkonzept aufbauender Spielmodus ist Meta-XXO: Bei diesem Spielmodus gibt es wieder ein 3x3 Spielfeld, wobei jedes Spielfeld ein eigenes untergeordnetes Spielfeld mit 3x3 Feldern beinhaltet. Um das Meta-Spiel zu gewinnen, müssen die neun kleineren Spiele gespielt werden. Diese können auch parallel gespielt werden – Spieler sind nicht gezwungen, ihren Zug im selben Feld zu machen wie ihr Gegner. Wird ein

untergeordnetes Spiel gewonnen, liegen dort also drei Spielmarken in einer Reihe, wird auf dem übergeordneten Spielfeld eine eigene Spielmarke auf dem entsprechend gerade gespielten Feld platziert. Wurde das Spiel dagegen verloren, bekommt der Gegner eine Spielmarke auf diesem Feld.

Als weiterer Spielmodus ist *Terni Lapilli*, ein dem XXO ähnliches Spiel, das schon im antiken Rom gespielt wurde. Hier stehen dem Spieler lediglich drei Spielmarken zur Verfügung. Um eine Reihe zu bilden, können die bereits gelegten Marken dann auf die benachbarten Felder verschoben werden.

Die App soll diese Spielmodi implementieren. Die Anzahl der Siege und Verluste sollen von der App gespeichert werden. Zugriff auf die anderen Spielmodi soll erst gewährt werden, wenn der Spieler bereits eine Reihe an Spielen XXO gespielt hat. Außerdem soll der Nutzer die Möglichkeit haben, entweder im Einzelspieler gegen einen Computerspieler anzutreten, oder im Multiplayermodus online gegen andere Spieler.

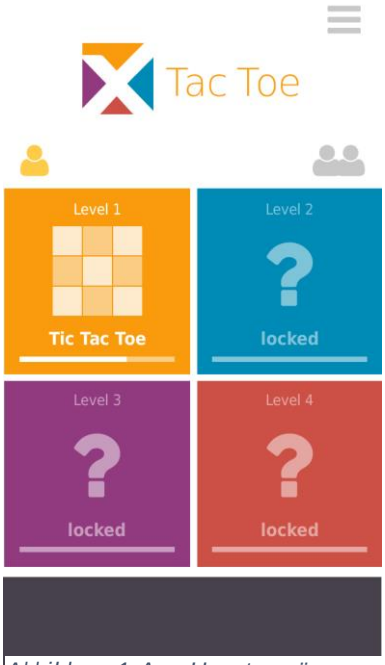

Abbildung 1 App-Hauptmenü

Wenn der Benutzer die App öffnet, soll eine Willkommensseite angezeigt werden. Auf dieser Seite wird dem Benutzer unter dem Logo der App ein Menü angezeigt, über das er den Spielmodus auswählen kann, den er als nächstes spielen möchte, solange der Spielmodus bereits freigespielt wurde (siehe Abbildung 1 App-Hauptmenü).

Nachdem der Benutzer einen der Startknöpfe mit der gewünschten Schwierigkeitsstufe gedrückt hat, soll die App auf eine zweite Seite wechseln. Diese Seite zeigt das Spielfeld an. Indem der Spieler auf ein freies Feld tippt, kann er es mit seinem Zeichen markieren. Daraufhin wird die Eingabe für den Benutzer gesperrt, sodass er keine weiteren Zeichen setzen kann.

Dann wird der Gegner ein Feld auswählen und mit seinem eigenen Zeichen markieren. Zuletzt wird das Feld wieder für den Benutzer freigegeben, damit er sein nächstes Zeichen setzen kann. Wer dabei gerade am Zug ist, wird durch eine Anzeige über dem Spielfeld dargestellt. Dort ist das Zeichen des Spielers, der gerade am Zug ist, vollfarbig hervorgehoben (siehe Abbildung 2 Setzen von Spielmarken).

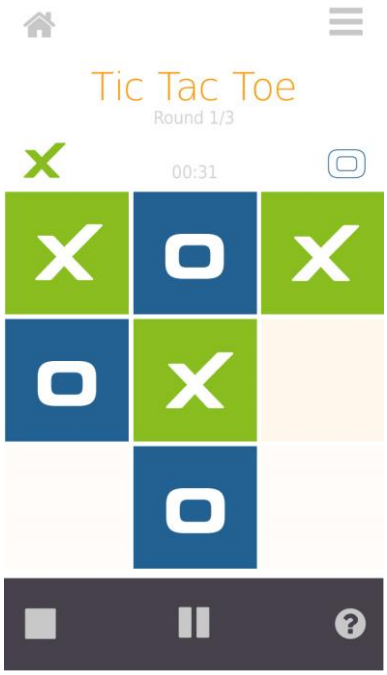

Abbildung 2 Setzen von Spielmarken

Abbildung 3 Ein Spieler gewinnt

Sollte nachdem ein Zeichen gesetzt wurde die Gewinnbedingung erfüllt sein, bleibt das Feld allerdings gesperrt und dem Benutzer wird mitgeteilt, welcher Spieler gewonnen hat. Wenn einer der Spieler keinen weiteren Zug mehr machen kann, weil alle Felder bereits belegt sind, wird dem Spieler stattdessen angezeigt, dass er ein Patt erreicht hat. Nachdem das Spiel zu Ende ist, hat der Spieler die Möglichkeit, ein neues Spiel zu starten. Alternativ kann er auch in das Hauptmenü zurückkehren, um einen anderen Spielmodus zu spielen.

4.2 Implementierung

4.2.1 Buildvorgänge

Wie in *3.4 Ionic* beschrieben, wird *Ionic* hauptsächlich über die Kommandozeile gesteuert. Die meisten modernen IDEs[1] jedoch können so konfiguriert werden, dass sie das *Ionic*-Kommandozeilenprogramm beim Buildvorgang selbstständig aufrufen. IDE und Kommandozeilenprogramm beeinflussen sich in der Regel nicht gegenseitig, also sollte eine IDE wie *WebStorm* verwendet werden, um Programmierfehler zu mindern.

[1] IDE, Integrated Development Environment: Ein Programm, um Quelltext zu erstellen. Bietet Zugriff auf Syntaxhervorhebung, Codevervollständigung, Schnellzugriff auf Dokumentation und vieles mehr, was die Produktivität von Entwicklern stark steigert und Fehlerquellen reduziert.

Zunächst muss *Ionic* installiert werden. Dies geschieht über den Paketmanager *NPM*[1] von *NodeJS*, einem asynchron arbeitenden Serverprogramm, welches Programme in *JavaScript* ausführen kann. Dieses *JavaScript* wird ohne Browser, oder „headless“ ausgeführt. Zu beachten ist, dass *Ionic* die Version 4 von *NodeJS* voraussetzt, die Version 6 ist bisher nicht kompatibel. Der Paketmanager *NPM* wird automatisch zusammen mit *NodeJS* installiert.

```
npm install -g cordova ionic
```

Listing 10 Befehl zum Installieren von Ionic

Der obige Befehl installiert sowohl *Apache Cordova*, auf das *Ionic* aufbaut, als auch *Ionic* selbst. Durch den „g“-Parameter werden die beiden Pakete global installiert anstatt in den gerade aktiven Arbeitsordner. Dies macht Sinn, denn *Apache Cordova* und *Ionic* werden unter Umständen in mehreren Projekten benötigt. Nachdem *Ionic* installiert wurde, steht das Kommandozeilenprogramm ebenfalls zur Verfügung. Um nun das XXO-Projekt anzulegen, muss der Befehl zum Erstellen eines neuen, leeren Projekts aufgerufen werden.

```
ionic start XXO blank && cd XXO
```

Listing 11 Befehl zum Anlegen des XXO-Projekts

Nachdem der Befehl vollständig abgearbeitet wurde, wurde ein Projektverzeichnis „XXO“ im aktuellen Arbeitsverzeichnis erstellt und in dieses hineingewechselt. Das Arbeitsverzeichnis der Kommandozeile sollte sich von nun an in diesem Verzeichnis befinden, wenn weitere *Ionic*-Befehle ausgeführt werden.

Die erstellte App ist leer, doch ist sie bereits eine gültige, kompilierfähige App. Da die App durch *Apache Cordova* auf *HTML*, *CSS* und *JavaScript* aufbaut, kann sie auch ohne große Anpassungen in einem normalen Webbrowser dargestellt werden. Dies ermöglicht das schnelle Entwickeln und Testen der App im Webbrowser, ohne dass die App auf einen Emulator oder ein echtes Smartphone aufgespielt werden muss, was mehr Zeit benötigt als das neu Laden einer Webseite. Mit dem folgenden Befehl wird ein Mini-Webserver in der Kommandozeile gestartet, über den die App im Webbrowser dargestellt werden kann.

```
ionic serve
```

Listing 12 Befehl zum Anzeigen der App im Webbrowser

[1] NPM: NodeJS Package Manager.

Um festzulegen, auf welchen Plattformen die App später laufen soll, müssen diese Plattformen zur Konfiguration des Projekts hinzugefügt werden. Da dies je nach Plattform unterschiedlich viel Arbeit bedeutet, da für einige Plattformen unter Umständen noch Ressourcenordner angelegt werden müssen, können die Plattformen bequem über das Kommandozeilenprogramm hinzugefügt werden. Der folgende Befehl beispielsweise fügt die Plattformen *iOS*, *Android* und *Windows* hinzu.

```
ionic platform add ios android windows
```

Listing 13 Befehl zum Hinzufügen von Zielplattformen

Um das Projekt für diese Plattformen auch erstellen zu können, müssen die SDKs der entsprechenden Plattform ebenfalls installiert worden sein.

```
01 ionic build android
02 ionic emulate android
03 ionic run android
```

Listing 14 Befehle zum Erstellen der XXO-App

Der Befehl in Listing 14 Befehle zum Erstellen der XXO-App, Zeile eins erstellt die App für eine Zielplattform, im Beispiel *Android*. Der Befehl darunter führt die erstellte App auf einem Emulator aus, während der Befehl in der dritten Zeile die App auf einem angeschlossenen Gerät ausführt. Sollte kein Gerät angeschlossen sein, so wird die App in diesem Fall stattdessen auf einem Emulator ausgeführt.

4.2.2 Struktur

Die Struktur der App baut wie in *3.4 Ionic* beschrieben auf *Apache Cordova*/*PhoneGap* auf. Speziell der Inhalt des `www`-Ordners wird durch Ionic noch weiter strukturiert. Im

Falle der *XXO*-App enthält dieser Ordner eine Struktur wie in Abbildung 4 Struktur der XXO-App in WebStorm.

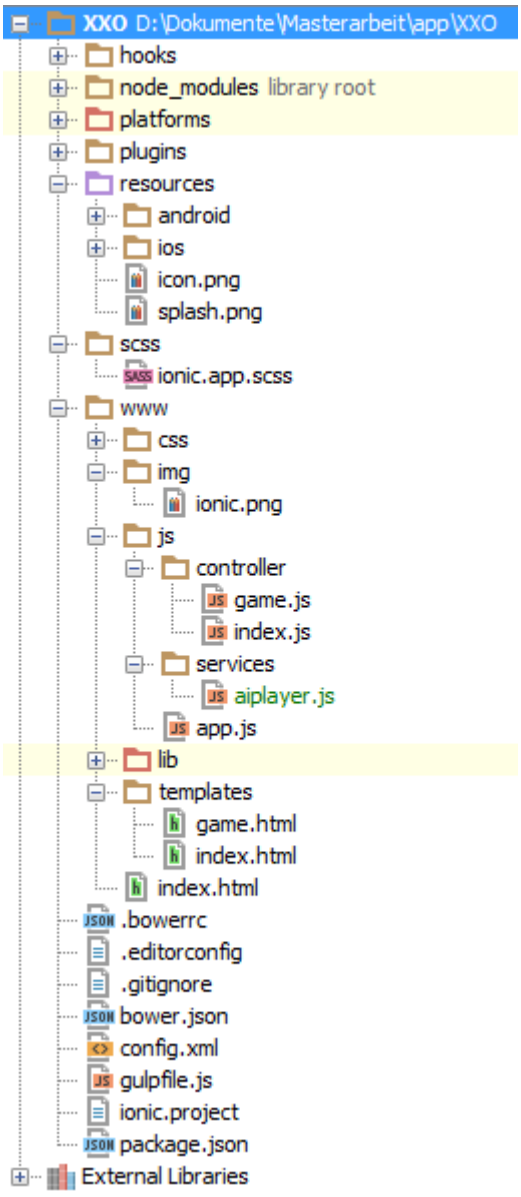

Abbildung 4 Struktur der XXO-App in WebStorm

Für die Seiten der App — die Willkommensseite und die Spielseite — existieren jeweils ein *Template* und ein *Controller*. Diese Begriffe kommen von *AngularJS*, einem verbreiteten JavaScript-Framework, das *Ionic* für seine MVC-Strukturierung verwendet.

Die *Templates* sind im Ordner `/www/templates` gespeichert. Sie sind einfache *HTML*-Dateien, die allerdings keine kompletten *HTML*-Seiten enthalten, sondern nur *HTML*-Fragmente. Diese Fragmente werden dynamisch in die Hauptdatei eingesetzt, wenn ihre Seite aufgerufen wird. Die *Templates* enthalten nur *HTML*-Code, der die Struktur der Seite deklariert, aber keine Steuerungslogik durch *Controller* oder Design durch *SASS*[1]. Das hilft, Design, Struktur und Steuerungslogik strikt voneinander zu trennen.

Die Hauptdatei wird gespeichert unter `/www/index.html` und ist der Einstiegspunkt der App. Sobald die App gestartet wird, wird zuerst diese Datei geöffnet. Die Hauptseite inkludiert dann die nötigen *CSS*- und *JavaScript*-Dateien und legt die grundlegende Struktur der App fest. Zum Beispiel kann hier ein Banner definiert werden, dass dann für alle Seiten der App gleichbleibend sein wird.

Die *Controller* der App sind für die Steuerung der Oberfläche verantwortlich. Sie werden im Ordner `/www/js/controller` gespeichert. In ihnen wird zum Beispiel auf das Drücken von Buttons reagiert oder bei gewissen Events die Oberfläche beeinflusst. In der

[1] SASS, Syntactically Awesome Stylesheets: Eine erweiterte Beschreibungssprache für *CSS*-Designdateien. Erlaubt das Definieren von Variablen, Verschachtelung, Import von weiteren *SCSS*-Dateien, Vererbung, mathematische Operationen und Anderes in Stylesheets. Diese können dann durch den *SCSS*-Prozessor zu *CSS*-Dateien umgewandelt werden, die vom Browser interpretiert werden können.

Regel gibt es einen *Controller* für jedes *Template*, doch dies ist keine strikte Regel, denn ein *Controller* kann auch für mehrere *Templates* verwendet werden.

Die Businesslogik der App wird in sogenannten *Services* gekapselt. Ein Service sollte dabei nur eine Aufgabe haben. In der XXO-App ist dies beispielsweise die Kapselung des Spielfelds und seiner Regeln (etwa, dass auf ein bereits gesetztes Feld keine weitere Markierung gesetzt werden darf) und die Steuerung des KI-Spielers. Jeder *Service* wird wieder in einer eigenen Datei im Ordner `/www/js/services` gespeichert.

Durch die Verwendung von *AngularJS* werden die *Controller* und *Services* mithilfe von *Dependency Injection* von *AngularJS* selbst initialisiert. Dies ist eine Form des Inversion of Control, oder *Steuerungsumkehr*: Das Framework *AngularJS* steuert die Initialisierung, nicht der Entwickler mit seinem Code. Die Verwendung von *Dependency Injection* hat den Vorteil, dass sich der *Controller* oder der *Service* nicht darum kümmern muss, wann er erstellt werden muss und mit welchen Konstruktorargumenten. Durch *Dependency Injection* wird dem *Controller* oder *Service* einfach eine kompatible Implementierung übergeben, je nach den Abhängigkeiten, die dieser spezifiziert hat. Daraus ergibt sich außerdem der Vorteil, dass die Implementierung von Abhängigkeiten problemlos ausgetauscht werden kann, ohne dass der *Controller* oder *Service* davon erfahren muss. Dies ist von großer Bedeutung für das *Mocking*[1] von Objekten für *Unit-Tests*.

```
01 angular.module('xxo', ['ionic', 'ngStorage'])
02   .config(function($stateProvider, $urlRouterProvider) {
03     $stateProvider
04       .state('index', {
05         url: '/',
06         templateUrl: 'templates/index.html',
07         controller: 'IndexController'
08       })
09       .state('game', {
10         url: '/game',
11         templateUrl: 'templates/game.html',
12         controller: 'GameController'
13       });
14     $urlRouterProvider.otherwise('/');
15   });
```

Listing 15 Registrierung der Pfade der App

[1] Mocking: Erstellen von Objekten für das Testen, die nach außen eine API implementieren, aber keine weitere Businesslogik beinhalten.

Die Konfiguration der Applogik wird in der Datei `app.js` im Ordner `/www/js` durchgeführt. Diese Datei muss vor den anderen vom Entwickler erstellten *JavaScript*-Dateien inkludiert werden, aber nach den Bibliotheken von *Ionic*.

Im Listing 15 Registrierung der Pfade der App in Zeile eins wird zunächst mit dem Befehl `angular.module()` ein neues Modul erstellt. Dieses Modul hält alle Klassen der App, die von *AngularJS* verwaltet werden. Bei Apps mit größerem Umfang können *Controller* und *Services* auch in Untermodule ausgelagert werden, dies ist für die XXO-App aber nicht vonnöten. Der erste Parameter, „xxo“, ist der Name des Moduls, und der zweite Parameter ist eine Liste von allen Modulen, die vom XXO-Modul benötigt werden — in diesem Fall nur „ionic“ und „ngStorage“ für die Datenpersistenz.

Der Aufruf von `config()` in Zeile zwei ist für die Registrierung aller Seiten der App zuständig. Hier kommt *Dependency Injection* das erste Mal zum Einsatz. Die Parameter `$stateProvider` und `$urlRouterProvider` sind *AngularJS* mit Namen bekannt. Es werden also Objekte vom Typ `$stateProvider` und `$urlRouterProvider` übergeben. Deshalb dürfen die Parameter auch nicht im Namen verändert werden, da sonst nicht das richtige Objekt übergeben werden kann[1].

In Zeilen vier beziehungsweise neun werden die *States*, oder Seiten definiert. Für jede Seite werden eine URL, ein *Template* und ein *Controller* definiert. Navigiert die App zu einer neuen URL, werden die hier definierten Seiten durchsucht. Passt auf die URL ein *State*, so wird der entsprechende *Controller* geladen und das *Template* in die Hauptseite eingefügt. Dies ist ein automatischer Prozess. Wird die App gestartet, ist die URL normalerweise `/`.

Sollte eine URL aufgerufen werden, die nicht registriert ist, so wird automatisch eine Weiterleitung aktiv, die die App auf eine Standardseite weiterleitet, normalerweise die Startseite. Die URL für diese Weiterleitung wird in Zeile vierzehn mit `$urlRouterProvider.otherwise()` definiert. Solange die App ohne Fehler läuft, sollte der Fall dieser Weiterleitung nicht eintreten, da der Endnutzer keinen Zugriff auf die URL der App hat.

Die Speicherung von persistenten Daten wie Konfigurationseinstellungen und die Anzahl der Siege und Niederlagen kann, da *Ionic* in einem Browserfenster ausgeführt wird, mit

[1] Es gibt auch eine spezielle Notation, mithilfe derer die Argumente umbenannt werden dürfen. Dies ist wichtig, falls der Quellcode minifiziert werden soll. In diesem Fall werden die Abhängigkeiten als Strings in einem Array notiert, mit der aufzurufenden Funktion als letztes Arrayelement.

LocalStorage erfolgen. Für *AngularJS* ist dazu ein Plugin namens *ngStorage* hilfreich. Es speichert Daten, die es übergeben bekommt, automatisch im *LocalStorage* ab und kann diese auch wieder bequem auslesen. Es müssen dazu keine `load()`- oder `save()`-Methoden aufgerufen werden und die Daten können behandelt werden, als wären sie einfache Variablen. Auch ganze Objekte werden dabei unterstützt und die Datenbindungstechniken wie auf Seite 12 beschrieben können ebenfalls verwendet werden, sodass die Datenpersistenz fast trivial wird.

Dazu wird in den `$scope` des Controllers eine entsprechende Referenz zum `$localStorage`-Objekt hinterlegt:

```
$scope.settings = $localStorage.settings;
```

Listing 16 Setzen eines $localStorage-Verweises in $scope

Das `$scope`-Objekt hat dabei eine besondere Rolle. Es ist im Modell des Controllers verfügbar. Jede Eigenschaft dieses Objekts kann also im *HTML*-View dargestellt werden. Ebenso wird das `$scope`-Objekt immer auf Änderungen überwacht. Ändert sich also eine Variable des Objekts, wird diese Änderung automatisch auch im View/der Oberfläche sichtbar. Wird andersherum auf der Oberfläche eine gebundene Variable zum Beispiel durch ein Textfeld geändert, ist der geänderte Variableninhalt sofort im Controller durch das `$scope`-Objekt verfügbar.

```
01 <ion-toggle class="item item-icon-left item-toggle"
          ng-model="settings.music">
02   <i class="icon ion-music-note"></i> Music
03 </ion-toggle>
```

Listing 17 Datenbindung in der HTML-Oberfläche

In Listing 17 Datenbindung in der HTML-Oberfläche kann die Datenbindung zum Beispiel dazu genutzt werden, zu speichern, ob Musik abgespielt werden soll oder nicht. Konkret dient dazu die Direktive `ng-model`. Ihr Inhalt bestimmt, an welchen Wert in `$scope` das Element gebunden wird, in diesem Fall „settings.music“. Nach dem Laden der App wird dieser Wert aus *LocalStorage* ausgelesen und auf der Oberfläche der *Toggle Switch* `ion-toggle` entsprechend gesetzt. Wird der *Toggle Switch* dann vom Benutzer betätigt, wird das zuerst das Modell aktualisieren. Durch die Verwendung von *ngStorage* wird der neue Wert dann automatisch und sofort auch in *LocalStorage* geschrieben, und der Wert ist direkt nach dem Betätigen App-weit verfügbar.

4.2.3 Komponenten

Ionic stellt zahlreiche Komponenten zur Verfügung, die während der Entwicklung genutzt werden können und sollten. Diese Komponenten sind häufig gebrauchte Oberflächenkomponenten, die der Zielplattform entsprechend bereits ein fertiges Aussehen/Animationen und Verhalten mitbringen. Für die App wichtige Komponenten sind unter anderem:

- `ion-nav-view` und `ion-nav-bar`

Der Tag `ion-nav-view` wird einmal im Hauptview der App benutzt, am Besten in der Datei `index.html`, welche der Startpunkt der App ist. Er wird während der Laufzeit der App automatisch mit den passenden Views der Seiten gefüllt, wie in Listing 15 Registrierung der Pfade der App festgelegt. Außerdem wird von diesem Tag jede Änderung des aktuellen Pfads verfolgt und die besuchten Pfade automatisch auf einem Stack abgelegt. Dieser Stack kann dazu benutzt werden, bequem wieder zurück zur vorherigen Seite zu navigieren. Dies kann über einen Service in einem Controller programmatisch geschehen, oder aber über Knöpfe in der `ion-nav-bar`. Diese Navigationsleiste dient dazu, Navigationsknöpfe zu halten, wie zum Beispiel den `ion-nav-back-button`. Wird dieser verwendet, ist kein weiterer Code mehr nötig, um der App eine Zurück-Navigationsmöglichkeit zu geben, wenn nötig, kann die Standardfunktionalität dieses Buttons aber auch angepasst werden.

- `ion-view` und `ion-content`

Der Tag `ion-view` dient dazu, den Inhalt von Views zu beinhalten. Die Verwendung dieses Tags erlaubt *Ionic*, Seitenwechsel in `ion-nav-view` automatisch zu verfolgen. Trivialerweise kann einfach der gesamte Inhalt eines *AngularJS-HTML*-Templates mit diesem Tag umgeben werden, um diese Funktionalität zu ermöglichen. Große Nützlichkeit hat auch der Tag `ion-content`. Alle Inhalte in diesem Tag lassen sich mit Fingergesten scrollen, sodass sie nicht auf ihrer Bildschirmposition fixiert sind und mehr Daten auf das Display passen. Ein gutes Beispiel hierfür sind Listen. Diese halten meist viele Elemente, die nicht alle auf einmal auf das Display passen. Mithilfe von `ion-content` lässt sich die Liste dann auf- und ab scrollen.

- `ion-list` und `ion-item`

Um eine Liste darzustellen, existiert die Komponenten `ion-list`. Mit dieser Liste können für das entsprechende Zielsystem formatierte Listenelemente dargestellt werden. Die einzelnen Elemente werden in der Komponente `ion-item` gehalten. `ion-item` kann nicht nur Text, sondern beliebigen Inhalt beinhalten, und hat – ebenso wie Buttons und andere Elemente – Unterstützung für Icons. Ein Beispiel für eine Liste in der XXO-App ist das Einstellungsmenü, was in Abbildung 5 Einstellungen in Form einer Liste zu sehen ist.

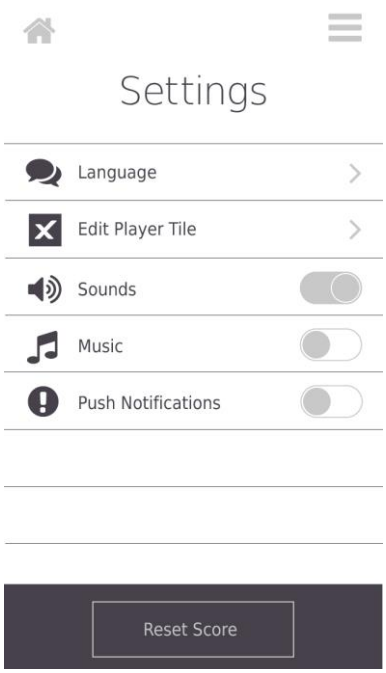

Abbildung 5 Einstellungen in Form einer Liste

- `ion-toggle`

Die `ion-toggle`-Komponente dient dazu, einen *Toggle Switch* darzustellen. Ein *Toggle Switch* funktioniert ähnlich wie eine Checkbox, lässt sich aber vor allem auf den Touchscreens von Mobilgeräten besser betätigen und zeigt seinen aktuellen Status besser an. Auch diese Komponente wird bereits von *Ionic* fertig gerendert und animiert, und passt sich an das Design der Zielplattform an.

Alle Komponenten, die nicht von Ionic bereitgestellt werden oder sehr speziell auf die App zugeschnitten sind, lassen sich außerdem mit *HTML* und *CSS* beziehungsweise *SCSS* nahtlos zwischen die mitgelieferten Komponenten eingliedern. Das modifizieren des Designs der vorhandenen Komponenten lässt sich außerdem ebenfalls sehr leicht über das Stylesheet/*CSS* der App erreichen. Eine benutzerdefinierte Komponente in der XXO-App ist beispielsweise das Spielfeld selbst. Es ist zu spezialisiert, als dass es als Standardkomponente mit *Ionic* mitgeliefert würde, konnte aber dennoch problemlos integriert werden.

4.2.4 Algorithmen

Um zu prüfen, ob das aktuelle XXO-Spiel vorüber ist, muss nach jedem Zug die Gewinnbedingung geprüft werden. Dies ist trivial, denn es muss nur nach drei gleichen Symbolen in allen drei Reihen, Spalten, und den beiden möglichen Diagonalen geprüft werden. Wenn die Gewinnbedingung nicht erfüllt ist, muss auch noch geprüft werden, ob überhaupt noch freie Felder vorhanden sind. Wenn nicht, endet das Spiel nämlich in einem Patt.

Damit der Computergegner aber entscheiden kann, in welches Feld er seine nächste Spielmarke setzen wird, ist etwas mehr als eine einfache Schleife nötig. Zur Berechnung des nächsten Zugs kann der *Minimax*-Algorithmus verwendet werden. *Minimax* ist dazu geeignet, eine optimale Spielweise zu berechnen für Spiele mit zwei Spielern, in denen alle Informationen offen liegen [55].

Minimax berechnet dazu alle möglichen zukünftigen Züge und markiert die Züge entsprechend ob sie zum Sieg oder zur Niederlage führen.

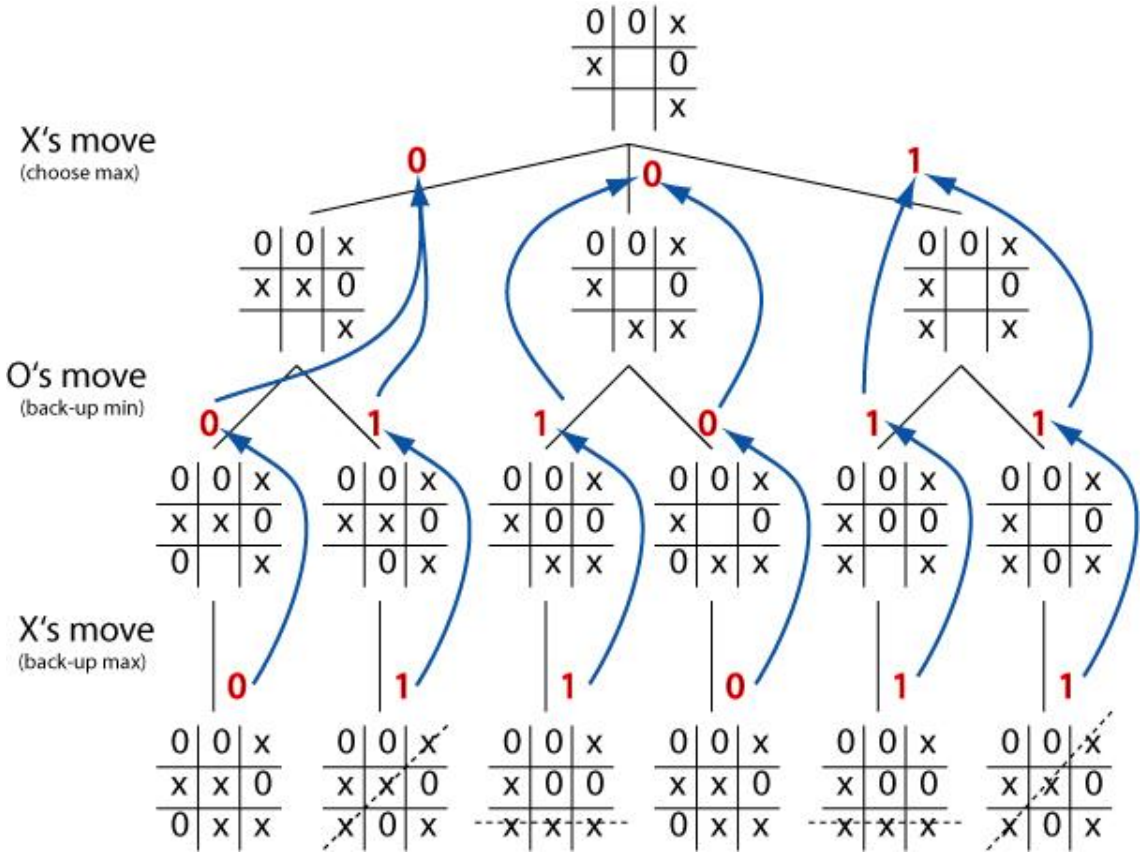

Abbildung 6 Minimax-Algorithmus

In Abbildung 6 Minimax-Algorithmus hat X drei Möglichkeiten, seinen Marker zu setzen. Für jede der drei Zugmöglichkeiten wird wiederum ein Entscheidungsbaum erstellt. Diese haben jeweils zwei Kinder, denn O hat nach dem Zug von X nur jeweils zwei Möglichkeiten, seine eigene Spielmarke zu setzen. Für den Baum links und den in der Mitte wird klar, dass O die Möglichkeit hat, den Sieg von X zu verhindern. Da davon ausgegangen werden muss, dass O ebenfalls optimal spielt, führen diese Bäume also nicht zum Sieg. Der Baum rechts dagegen führt immer zum Sieg, denn er gibt O keine Chance, noch ein Patt zu erzwingen. Dieser Baum wird also als die weitere Spielstrategie des Computerspielers genutzt.

Bei der Verwendung von *Minimax* bei einem XXO-Spiel ist jedoch zu beachten, dass eine optimale Spielweise, so wie sie von *Minimax* berechnet wird, immer zu einem Patt führen wird. Es wird für den menschlichen Nutzer also niemals möglich sein, zu gewinnen. Um den Nutzer nicht mit Niederlagen zu frustrieren, sollte also eine zufällige Komponente in die Entscheidungsfindung des Computergegners einfließen. Möglich wäre, dass der Computer mit einer gewissen, kleinen Chance den falschen Entscheidungsbaum auswählt und somit nicht immer optimal spielt.

Ebenso sollte bei Entscheidungsbäumen, die eine gleiche Bewertung haben, der nächste Entscheidungsbaum per Zufall ausgewählt werden, damit der Computergegner nicht zu berechenbar erscheint. Ansonsten wird es passieren, dass er bei gleichen Zügen immer absolut gleich antwortet, was den Spielspaß mindern kann.

Ein weiterer Randfall von *Minimax* ist ein Spiel, das nicht mehr gewonnen werden kann. Wenn eine zufällige Auswahl eines Baumes dazu führt, dass der menschliche Spieler in jedem Fall gewinnen wird, sollte der Computergegner dennoch versuchen, offensichtliche Dreierpaare zu verhindern, auch wenn dies nicht zum Sieg führen kann. Nach dem *Minimax*-Algorithmus ist jede Zugmöglichkeit gleich schlecht, doch eine der Siegmöglichkeiten des Spielers zu blocken lässt den Computergegner menschlicher erscheinen und gibt dem Benutzer somit nicht das unangenehme Gefühl, gegen einen dummen Computer zu spielen.

4.3 Unit-Tests

Unit-Tests sind ein wichtiges Mittel, um sichergehen zu können, dass der geschriebene Code auch das tut, was er soll. Außerdem helfen Unit-Tests dabei, Code nach dem Schreiben anzupassen, denn der neue Code kann sofort auf Korrektheit geprüft werden.

Unit-Tests nutzen dazu kleine Methoden, die den zu testenden Code mit verschiedenen Eingaben beliefern und prüfen, dass er die richtigen Antworten gibt. Wenn eine der Testmethoden feststellt, dass der zu prüfende Code nicht die erwartete Antwort liefert, so wurde ein Fehler gefunden – der Entwickler kann ihn sofort beheben.

Tests in *AngularJS* werden bevorzugt mit *Karma* und *Jasmine* erstellt [56]. *Karma* ist ein Kommandozeilenprogramm, welches den JavaScript-Quellcode der App laden und die darin enthaltenen Testmethoden ausführen kann. *Karma* nutzt dazu *NodeJS*, ein auf asynchronem JavaScript basiertes Serverprogramm, welches den JavaScript-Quelltext außerhalb eines Browsers ausführen kann.

Jasmine ist ein Framework für Unit-Tests. Es stellt Funktionen zur Verfügung, mit denen Testmethoden einfacher strukturiert und angesteuert werden können, sodass auf einen Blick klar ist, was die Testmethode testet. Außerdem beinhaltet *Jasmine* Methoden, um die von dem zu testenden Code zurückgegebenen Werte mit den erwarteten Werten zu vergleichen.

```
describe('sorting the list of users', function() {
  it('sorts in descending order by default', function() {
    var users = ['jack', 'igor', 'jeff'];
    var sorted = sortUsers(users);
    expect(sorted).toEqual(['jeff', 'jack', 'igor']);
  });
});
```

Listing 18 Beispiel für einen Unit-Test mit Jasmine

Da *AngularJS* starke Nutzung von *Dependency Injection* macht, ist es außerdem sehr einfach, von zu testenden Controllern oder Services benötigte Ressourcen durch Testressourcen oder Testklassen auszutauschen. Dieser Vorgang nennt sich mocken und dient dazu, gezielt Werte in den Test einzubringen, die unter der totalen Kontrolle der Testumgebung stehen. Beispielsweise kann, wenn ein Controller über den `$httpBackend` eine Verbindung zu einem entfernten Server aufbaut, der entfernte Server nicht von dem Test kontrolliert werden. Die Lösung ist ein Testobjekt, das sich exakt so wie `$httpBackend` verhält, aber vom Test fingierte Daten zurück an den Controller gibt, anstatt tatsächlich eine Verbindung zu einem Server aufzubauen. Für die in *AngularJS* mitgelieferten Services besitzt *AngularJS* das Modul *ngMock*, welches solche Testklassen zur Verfügung stellt, damit sie in Unit-Tests verwendet werden können.

5 Aussicht

Frameworks für die Cross-Plattform-Entwicklung befinden sich in einer ständigen Entwicklung. Relativ oft entstehen neue Frameworks mit interessanten, neuen Ansätzen für die Entwicklung und Plattformunabhängigkeit, und leider können durch die Masse der Frameworks nicht alle eine breite Verwendung finden.

Durch verschiedenste Anforderungen an Apps als auch durch die rasante Weiterentwicklung, die mit den technischen Innovationen von Smartphones mithalten muss, kann nie eine generelle Antwort gegeben werden, welches Framework zu nutzen ist.

Doch die rasante Entwicklung bringt viele Vorteile mit sich. So wird die Verwendung der Frameworks zunehmend einfacher. Die Schnittmenge der Funktionen und Möglichkeiten zwischen den Frameworks, aber auch den unterschiedlichen Plattformen, wird immer

größer, und neue Konzepte werden sich entwickeln. Diese Entwicklung kann also nur eine verbesserte Usability aller Apps mit sich bringen, sowie eine Vereinfachung des Entwicklungsprozesses speziell für plattformübergreifende Apps. Dadurch können größere und komplexere Apps immer einfacher entwickelt werden.

Abbildungsverzeichnis

Abbildung 1 App-Hauptmenü 24
Abbildung 2 Setzen von Spielmarken 25
Abbildung 3 Ein Spieler gewinnt 25
Abbildung 4 Struktur der XXO-App in WebStorm 28
Abbildung 5 Einstellungen in Form einer Liste 33
Abbildung 6 Minimax-Algorithmus 34

Listingverzeichnis

Listing 1 Befehl zum Anlegen eines PhoneGap-Projekts 8
Listing 2 Befehl zum Bauen eines PhoneGap-Projekts 9
Listing 3 Darstellen einer PhoneGap-App im Desktop-Browser 9
Listing 4 Befehl zum Anlegen eines Sencha Touch-Projekts 11
Listing 5 Befehl zum Hinzufügen von Modellen zu einem Sencha Touch-Projekt 12
Listing 6 Befehl zum Bauen eines Sencha Touch-Projekts 12
Listing 7 Befehl zum Anlegen eines Ionic-Projekts 13
Listing 8 Befehl zum Anzeigen eines Ionic-Projekts im Browser 13
Listing 9 Befehl zum Packen eines Ionic-Projekts 13
Listing 10 Befehl zum Installieren von Ionic 26
Listing 11 Befehl zum Anlegen des XXO-Projekts 26
Listing 12 Befehl zum Anzeigen der App im Webbrowser 26
Listing 13 Befehl zum Hinzufügen von Zielplattformen 27
Listing 14 Befehle zum Erstellen der XXO-App 27
Listing 15 Registrierung der Pfade der App 29
Listing 16 Setzen eines $localStorage-Verweises in $scope 31
Listing 17 Datenbindung in der HTML-Oberfläche 31
Listing 18 Beispiel für einen Unit-Test mit Jasmine 36

Tabellenverzeichnis

Tabelle 1 Framework-Vergleich 22

Literaturverzeichnis

[1] „Dolphin Emulator,“ [Online]. Available: https://de.dolphin-emu.org/. [Zugriff am 15 Februar 2016].

[2] @Fyrd, „Can I use ...?,“ 2016. [Online]. Available: http://caniuse.com. [Zugriff am 10 Januar 2016].

[3] Google, „Android Developers,“ [Online]. Available: http://developer.android.com/tools/sdk/ndk/index.html. [Zugriff am 6 März 2016].

[4] Apple, „Developer Connection,“ [Online]. Available: http://developer.apple.com/DOCUMENTATION/Cocoa/Conceptual/ObjectiveC/Articles/chapter_14_section_1.html via http://web.archive.org/web/20081231010709/http://developer.apple.com/DOCUMENTATION/Cocoa/Conceptual/ObjectiveC/Articles/chapter_14_section_1.html. [Zugriff am 6 März 2016].

[5] Microsoft, „Windows Dev Center,“ [Online]. Available: https://msdn.microsoft.com/de-de/library/windows/apps/jj681687(v=vs.105).aspx. [Zugriff am 6 März 2016].

[6] Mobile Angular UI, „Mobile Angular UI,“ [Online]. Available: http://mobileangularui.com/. [Zugriff am 14 Juni 2016].

[7] Intel, „Intel Developer Zone,“ [Online]. Available: https://software.intel.com/de-de/intel-xdk. [Zugriff am 14 Juni 2016].

[8] Telerik, „Kendo UI,“ [Online]. Available: http://www.telerik.com/kendo-ui. [Zugriff am 14 Juni 2016].

[9] jQuery Mobile, „jQuery Mobile,“ [Online]. Available: https://jquerymobile.com/. [Zugriff am 14 Juni 2016].

[10] Onsen UI, „Onsen UI,“ [Online]. Available: https://onsen.io/. [Zugriff am 14 Juni 2016].

[11] D. Johnson, „PhoneGap,“ 18 September 2008. [Online]. Available: http://phonegap.com/2008/09/18/phonegap-it%E2%80%99s-like-air-for-the-iphone/. [Zugriff am 10 Januar 2016].

[12] A. Charland, „PhoneGap,“ 4 Oktober 2011. [Online]. Available: http://phonegap.com/2011/10/04/nitobi-and-phonegap%E2%80%99s-new-home-at-adobe/. [Zugriff am 13 Januar 2016].

[13] B. LeRoux, „PhoneGap,“ 19 März 2012. [Online]. Available: http://phonegap.com/2012/03/19/phonegap-cordova-and-what%E2%80%99s-in-a-name/. [Zugriff am 10 Januar 2016].

[14] PhoneGap, „PhoneGap,“ [Online]. Available: http://phonegap.com/about/. [Zugriff am 13 Januar 2016].

[15] PhoneGap, „PhoneGap Forum,“ [Online]. Available: https://forums.adobe.com/community/phonegap. [Zugriff am 28 Juni 2016].

[16] Stackoverflow, „Tags: Cordova,“ [Online]. Available: http://stackoverflow.com/questions/tagged/cordova. [Zugriff am 28 Juni 2016].

[17] PhoneGap, „PhoneGap Documentation,“ [Online]. Available: http://docs.phonegap.com/. [Zugriff am 28 Juni 2016].

[18] Apache Software Foundation, „PhoneGap,“ [Online]. Available: http://phonegap.com/about/license/. [Zugriff am 15 Dezember 2015].

[19] Sencha, „Sencha Blog,“ 14 Juni 2010. [Online]. Available: http://www.sencha.com/blog/2010/06/14/ext-js-jqtouch-raphael-sencha/ via http://web.archive.org/web/20101203170022/http://www.sencha.com/blog/2010/06/14/ext-js-jqtouch-raphael-sencha/. [Zugriff am 7 März 2016].

[20] „jQT,“ [Online]. Available: http://jqtjs.com/. [Zugriff am 7 März 2016].

[21] D. Baranovskiy, „Raphaël,“ [Online]. Available: http://dmitrybaranovskiy.github.io/raphael/. [Zugriff am 7 März 2016].

[22] Stackoverflow, „Tags: Sencha-Touch,“ [Online]. Available: http://stackoverflow.com/questions/tagged/sencha-touch. [Zugriff am 28 Juni 2016].

[23] Sencha, „Sencha Forum,“ [Online]. Available: https://www.sencha.com/forum/. [Zugriff am 28 Juni 2016].

[24] Sencha, „Sencha Documentation,“ [Online]. Available: http://docs.sencha.com/. [Zugriff am 28 Juni 2016].

[25] Sencha, „Sencha Licensing,“ [Online]. Available: https://www.sencha.com/legal/. [Zugriff am 27 Januar 2016].

[26] M. Lynch, „Ionic,“ 21 November 2013. [Online]. Available: https://github.com/driftyco/ionic/releases/tag/v0.9.08-alpha. [Zugriff am 26 April 2016].

[27] M. Lynch, „Ionic,“ 8 Oktober 2013. [Online]. Available: http://blog.ionic.io/announcing-ionic/. [Zugriff am 27 Januar 2016].

[28] „AngularJS,“ [Online]. Available: https://angularjs.org/. [Zugriff am 7 März 2016].

[29] M. Lynch, „Ionic,“ 5 Januar 2016. [Online]. Available: http://blog.ionic.io/how-2015-went-for-ionic/. [Zugriff am 28 Juni 2016].

[30] M. Lynch, „Ionic,“ 6 März 2014. [Online]. Available: http://blog.ionic.io/what-is-cordova-phonegap/. [Zugriff am 7 März 2016].

[31] Ionic, „Ionic Forum,“ [Online]. Available: https://forum.ionicframework.com/. [Zugriff am 28 Juni 2016].

[32] Stackoverflow, „Tags: Ionic-Framework,“ [Online]. Available: http://stackoverflow.com/questions/tagged/ionic-framework. [Zugriff am 28 Juni 2016].

[33] Ionic, „Ionic Documentation,“ [Online]. Available: http://ionicframework.com/docs/. [Zugriff am 28 Juni 2016].

[34] Ionic, „Ionic,“ [Online]. Available: https://github.com/driftyco/ionic. [Zugriff am 27 Januar 2016].

[35] D. K. Taft, „eWeek,“ 9 Dezember 2008. [Online]. Available: http://www.eweek.com/c/a/Application-Development/Appcelerator-Takes-on-Adobe-AIR-with-Titanium. [Zugriff am 27 Januar 2016].

[36] K. Whinnery, „Appcelerator,“ 20 Januar 2012. [Online]. Available: http://www.appcelerator.com/blog/2012/01/the-future-of-titanium-desktop/. [Zugriff am 28 März 2016].

[37] Stackoverflow, „Tags: Titanium,“ [Online]. Available: http://stackoverflow.com/questions/tagged/titanium. [Zugriff am 28 Juni 2016].

[38] F. Zandbergen, „Appcelerator Blog,“ 14 Januar 2016. [Online]. Available: http://www.appcelerator.com/blog/2016/01/embracing-stack-overflow-for-appcelerator-community-support/. [Zugriff am 28 Juni 2016].

[39] Appcelerator, „Appcelerator Platform,“ [Online]. Available: http://docs.appcelerator.com/platform/latest/. [Zugriff am 28 Juni 2016].

[40] Apache Software Foundation, „Appcelerator,“ [Online]. Available: https://github.com/appcelerator/titanium_mobile/blob/master/LICENSE. [Zugriff am 27 Januar 2016].

[41] M. d. Icaza, „Tirania,“ 16 Mai 2011. [Online]. Available: http://tirania.org/blog/archive/2011/May-16.html. [Zugriff am 27 Januar 2016].

[42] F. Kalenda, „ZDNet,“ 25 Februar 2016. [Online]. Available: http://www.zdnet.de/88261158/microsoft-schluckt-cloud-entwicklerplattform-xamarin/. [Zugriff am 28 März 2016].

[43] N. Friedman, „Xamarin,“ 31 März 2016. [Online]. Available: https://blog.xamarin.com/xamarin-for-all/. [Zugriff am 5 April 2016].

[44] Xamarin, „Xamarin.Forms,“ [Online]. Available: https://www.xamarin.com/forms. [Zugriff am 28 März 2016].

[45] Xamarin, „Xamarin Forum,“ [Online]. Available: https://forums.xamarin.com/. [Zugriff am 28 Juni 2016].

[46] Stackoverflow, „Tags: Xamarin,“ [Online]. Available: http://stackoverflow.com/questions/tagged/xamarin. [Zugriff am 28 Juni 2016].

[47] Xamarin, „Xamarin Developer Center,“ [Online]. Available: https://developer.xamarin.com/. [Zugriff am 28 Juni 2016].

[48] M. d. Icaza, „Mono,“ 23 März 2016. [Online]. Available: https://github.com/mono/mono/commit/ef0ddf45c3081e799edcb4e95770186514b80cf1. [Zugriff am 5 März 2016].

[49] Gamasutra, „Gamasutra,“ 24 Mai 2012. [Online]. Available: http://www.gamasutra.com/view/news/169846/Mobile_game_developer_survey_leans_heavily_toward_iOS_Unity.php. [Zugriff am 12 April 2016].

[50] Unity, „Unity,“ [Online]. Available: https://unity3d.com/unity/multiplatform. [Zugriff am 12 April 2016].

[51] Epic Games, „Unreal Engine Features,“ [Online]. Available: https://www.unrealengine.com/unreal-engine-4. [Zugriff am 5 April 2016].

[52] Epic Games, „Mobile Game Development,“ [Online]. Available: https://docs.unrealengine.com/latest/INT/Platforms/Mobile/. [Zugriff am 5 April 2016].

[53] M. Ekuan, „Windows Dev Center,“ [Online]. Available: https://msdn.microsoft.com/de-de/library/windows/apps/dn726767.aspx. [Zugriff am 5 April 2016].

[54] T. Anderson, „The Register,“ Oktober 2015. [Online]. Available: http://www.theregister.co.uk/2015/10/14/developers_ask_microsoft_for_real_net_universal_apps_windows_mac_ios_and_android/. [Zugriff am 5 April 2016].

[55] C. Thornton, „KR-IST - Lecture 5a Game playing with Minimax and Pruning,“ [Online]. Available: http://users.sussex.ac.uk/~christ/crs/kr-ist/lec05a.html. [Zugriff am 18 September 2016].

[56] AngularJS, „Unit Testing,“ [Online]. Available: https://docs.angularjs.org/guide/unit-testing. [Zugriff am 18 September 2016].

[57] Appcelerator, „Appcelerator Platform,“ [Online]. Available: http://docs.appcelerator.com/platform/latest/\#!/guide/Alloy_Framework. [Zugriff am 28 März 2016].

[58] Appcelerator, „Appcelerator,“ [Online]. Available: http://www.appcelerator.com/pricing/. [Zugriff am 28 Juni 2016].

[59] Xamarin, „Xamarin Store,“ [Online]. Available: https://store.xamarin.com/. [Zugriff am 28 Juni 2016].

[60] Xamarin, „Xamarin Software License Agreement,“ [Online]. Available: https://www.xamarin.com/platform/license. [Zugriff am 27 Januar 2016].

[61] Xamarin, „Xamarin Customers,“ [Online]. Available: https://www.xamarin.com/customers. [Zugriff am 28 Juni 2016].

[62] J. Böhler, „Diso AG,“ 2015 Februar 2015. [Online]. Available: http://www.diso.ch/2015/02/18/plattform-uebergreifende-software-entwicklung-crossplatform-software-engineering-bern/. [Zugriff am 22 Oktober 2015].