Projektarbeit

Implementierung eines Sudoku-Spiels zur Evaluierung der technischen Konzepte von Flutter

Autor: Christian Martin Slupikowski

Matrikel-Nr.: 10049494

E-Mail: slupikowski.christianmartin@fh-swf.de

Prüfer: Prof. Dr. Christian Gawron

Abgabe: 17.02.2021, Iserlohn

# Inhaltsverzeichnis

[Inhaltsverzeichnis 2](#_Toc62912199)

[1 Einführung 4](#_Toc62912200)

[1.1 Motivation und Zielsetzung 4](#_Toc62912201)

[1.2 Anwendungsbereiche und Zielgruppen 4](#_Toc62912202)

[2 Anforderungsanalyse 5](#_Toc62912203)

[2.1 Entwicklungsprozess 5](#_Toc62912204)

[2.2 Funktionale Anforderungen 5](#_Toc62912205)

[2.3 Nicht Funktionale Anforderungen 5](#_Toc62912206)

[2.4 Use Cases 6](#_Toc62912207)

[2.5 Entwurf 7](#_Toc62912208)

[3 Konzeption 9](#_Toc62912209)

[3.1 Sudoku Generierung Algorithmus 9](#_Toc62912210)

[3.1.1 Backtracking Lösungsalgorithmus 9](#_Toc62912211)

[3.1.2 Backtracking Algorithmus zum Erstellen eines Sudokus 10](#_Toc62912212)

[3.1.3 Entfernen der Zahlen aus dem vollständigen Sudoku 12](#_Toc62912213)

[3.2 Exkurs: Flutter 14](#_Toc62912214)

[3.2.1 Dart 14](#_Toc62912215)

[3.2.2 Widgets 14](#_Toc62912216)

[3.2.3 Stylen und Formattieren 16](#_Toc62912217)

[3.3 Implementierungsphase 17](#_Toc62912218)

[3.3.1 Datenstruktur der Sudoku Werte 17](#_Toc62912219)

[3.3.2 Fehlendes new Keyword in Dart 17](#_Toc62912220)

[3.3.3 Aufbau der Widgets 17](#_Toc62912221)

[3.3.4 Speichern und Laden eines Spielstandes 18](#_Toc62912222)

[3.4 Veröffentlichung im Google Play Store 19](#_Toc62912223)

[4 Fazit 19](#_Toc62912224)

[5 Mögliche Erweiterungen 19](#_Toc62912225)

[5.1 Veröffentlichung im Apple Store 19](#_Toc62912226)

[5.2 Offiziell im Google Play Store veröffentlichen 19](#_Toc62912227)

[5.3 Flutter auf Betaversion erhöhen und Web Kompilierung Testen 20](#_Toc62912228)

[5.4 Automatisches Speichern 20](#_Toc62912229)

[6 Abbildungsverzeichnis 21](#_Toc62912230)

[7 Literaturverzeichnis 22](#_Toc62912231)

# Einführung

## Motivation und Zielsetzung

Smartphones sind schon seit einiger Zeit der tägliche Begleiter von vielen Menschen. Dabei werden sie in sehr vielen Bereichen eingesetzt, ob es berufliche Zwecke oder private Zwecke sind. Daher werden die Ansprüche an entwickelter Software immer höher und neue Technologien werden entwickelt, um das Erstellen von neuen Applikationen zu vereinfachen. Dabei ist ein häufiges Problem, dass Applikationen für jedes Betriebssystem, wie Beispielweise Android und IOS, entwickelt werden müssen. Dadurch sind dies unabhängig voneinander arbeitende Applikationen, welche ihren eigenen Quellcode zugrunde liegen. Dabei sollen die Applikationen genau dasselbe tun und greifen meistens sogar auf dasselbe Backend zu. Um die Entwicklung, sowie den Wartungsprozess von Applikationen zu beschleunigen, ist eine Applikation für beide Betriebssysteme deutlich einfacher. Für diesen Fall wurde Flutter entwickelt. Flutter kompiliert mit Dart geschrieben Quellcode in jeweilige Applikationen für Android und IOS. Eventuelle Anpassungen für Betriebssystem spezifische Funktionen, die nicht von Flutter für beide Betriebssysteme verallgemeinert werden können, können dennoch Betriebssystemspezifisch implementiert werden.

Um als Softwareentwickler auf den neusten Stand zu bleiben ist es wichtig in allen Bereichen einen Überblick zu behalten, welche Techniken aktuell sind und welche verwendet werden. Damit ich als Hauptberuflicher Web/Backend Entwickler mich auch in Smartphone Applikationen auskenne und einen Überblick von Flutter verschaffen kann, wird in dieser Projektarbeit eine Sudoku-App erstellt.

Sudoku ist ein aus den 80er Jahren stammendes, mittlerweile sehr populäres, Logikrätsel, für das es eine ganze Reihe von Lösungsalgorithmen gibt. Dabei geht es bei Sudoku darum, ein 9x9 gefülltes Raster mit Zahlen von 1-9 zu füllen. Dabei dürfen die Zahlen 1-9 jeweils nur einmal Waagerecht, Senkrecht und in einem 3x3 Kästchen vorkommen.

## Anwendungsbereiche und Zielgruppen

Die Applikation ist für alle Rätselbegeisterte gedacht, die gerne auf Ihrem Smartphone, Tablett oder anderen Android Geräten Sudoku spielen möchten. Da verschiedene Schwierigkeitsgrade zur Auswahl stehen sollen, ist die Applikation sowohl für Anfänger als auch für fortgeschrittene Spieler geeignet.

# Anforderungsanalyse

## Entwicklungsprozess

Die Sudoku-App soll nach dem Wasserfall Modell entwickelt werden. Es wurde sich ausfolgenden Gründen gegen einen agilen Entwicklungsprozess entschieden. Da die Projektarbeit in Einzelarbeit entwickelt wird, stehen keine konkreten Tester zur Verfügung, sodass die Software, neben eigenen Tests bei der Entwicklung, zusätzlich nur durch eine ausgewählte Gruppe zum Abnahmetest getestet wird. Ebenfalls ist nicht klar wie viele Kapazitäten pro Zeiteinheit zur Verfügung stehen, daher ist es einfacher in Paketen zu planen, als in Zeitlichen Sprints. Außerdem wird am Ende des Projektes ein fertiges Produkt entstanden sein, welches in der Regel nicht mehr weiterentwickelt wird. Wenn allerdings trotzdem weitere Features für das Projekt in Frage kommen sollten, dann könnten diese mit einem agilen Entwicklungsprozess entwickelt werden.

## Funktionale Anforderungen

1. Dem Anwender soll es möglich sein, ein Sudoku Spiel komplett zu spielen.
2. Der Anwender kann genau immer ein Feld gleichzeitig selektieren. Dabei sind die Felder ausgeschlossen, welche zum Start des Spiels vorbelegt sind.
3. Felder die beim Start des Spiels vorbelegt sind, werden durch eine dicke schwarze Schrift gekennzeichnet.
4. Das selektierte Feld wird farblich hinterlegt.
5. Im unteren Teil der Applikation sollen Zahlen zwischen 1 und 9 hinterlegt werden.
6. Wenn ein Feld selektiert ist, kann durch drücken einer dieser Zahlen, das Feld gefüllt werden.
7. Wurde das Feld bereits gefüllt, wird der Wert mit dem neuen Wert überschrieben.
8. Eine Zahl kann aus einem Feld entfernt werden, indem länger auf das Feld gedrückt wird.
9. Ein Menü soll es geben in dem der Anwender folgende Aktionen ausführen kann
   1. Generieren eines neuen Spiels, in verschiedenen Schwierigkeitsstufen (Leicht, Mittel, Schwer).
   2. Speichern des aktuellen Spielstands. Es soll immer nur genau ein Spielstand gespeichert werden können. Dabei muss sichergestellt werden, dass der Anwender nochmal gefragt wird, ob er wirklich den aktuell gespeicherten Spielstand überschreiben möchte.
   3. Laden des vorher gespeicherten Spielstands.
10. Wenn der Anwender ein Spiel gewinnt, wird diesem ein Pop-Up Fenster angezeigt.
11. Es soll eine Hilfe für den Anwender geben, welche aktivier-/deaktivierbar ist.
12. Wenn die Hilfe aktiviert ist, werden richtig eingefüllte Zahlen mit einem grünen Hintergrund und falsche mit einem roten Hintergrund befüllt.
13. Das Sudoku-Spiel soll nur im Hochformat angezeigt werden können.

## Nicht Funktionale Anforderungen

1. Das generieren eines neuen Sudoku Spiels soll nicht länger als 5 Sekunden dauern, unabhängig von dem Schwierigkeitsgrad.
2. Die Applikation soll auf allen Android Geräten mit einer Mindestversion von 1.1.1 laufen.
3. Die Applikation soll nicht für IOS Geräte veröffentlicht werden.

## Use Cases

Um das Verständnis des Use-Case Diagramms zu verbessern, wird zunächst auf jeden Use-Case anhand des Use-Case-Diagramms eingegangen.

**Spieler:** Der Spieler ist der Anwender, welcher die App bedient und ein Sudoku spielen möchte.

**Neues Spiel:** Startet ein Neues Spiel und löst somit das Generieren eines Sudokus aus.

**Schwierigkeitsgrad wählen:** Beim Starten eines neuen Spiels, kann der Schwierigkeitsgrad ausgewählt werden.

**Sudoku Generieren:** Generiert ein Neues Spiel, mit neuen Zahlwerten. Hinterlegt die initialen Werte in der Sudoku-Tabelle. Intern wird die Lösung gespeichert.

**Spiel Laden:** Lädt ein Sudoku Spielstand. Setzt voraus, dass bereits ein Spielstand gespeichert wurde.

**Spiel Speichern:** Speichert den aktuellen Spielstand. Setzt voraus, dass ein Sudoku Spiel generiert worden ist.

**Feld selektieren:** Selektiert ein Feld, welches kein initiales Feld sein darf. Setzt voraus, dass ein Sudoku Spiel generiert worden ist.

**Zahl auswählen:** Eine Zahl zwischen 1 und 9 kann ausgewählt werden. Setzt voraus, dass ein Feld selektiert wurde.

**Zahl löschen:** Löscht die eine Zahl aus einem ausgewählten Feld. Setzt voraus, dass dort eine Zahl ausgewählt wurde.

**Hilfe aktivieren/deaktivieren:** Aktiviert oder Deaktiviert den Hilfemodus für den Spieler.

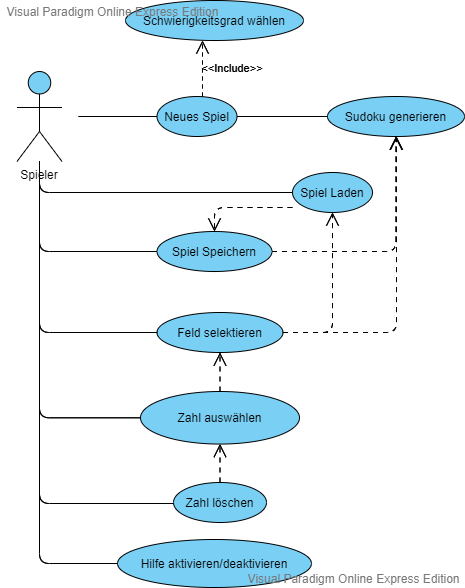


Abbildung Use-Case Diagramm der Sudoku-App

## Entwurf

Im Folgenden wird auf die wichtigsten Komponenten in der Sudoku Applikation eingegangen und beschrieben welche Aktionen diese ausführen.

In der Sudoku Applikation sollen zwei Menüs zu Auswahl stehen. In der linken oberen Ecke ist das so genannte „Burger-Menü“. Wird dieses geöffnet, erscheint ein Menü, in dem der Anwender folgende Optionen hat: Neues Spiel Starten mit den jeweiligen Schwierigkeitsgrad Leicht, Mittel, Schwer, ein Spiel Laden und ein Spiel Speichern. Welche Funktionen diese Optionen haben, wurde bereits in dem Kapitel 2.4 Use Cases erläutert. In der rechten oberen Ecke befindet sich das Kontextmenü, welches durch drei Punkte gekennzeichnet ist. Dort kann der Anwender die Hilfe aktivieren oder deaktivieren. Hat der Anwender die Hilfe aktiviert, wird ein Hacken vor dem Wort Hilfe angezeigt, um dies so ersichtlich zu machen. Wenn die Hilfe aktiv ist, werden in der Sudoku Tabelle die entsprechend richtig eingetragenen Zahlen grün hinterlegt und die falsch eingetragenen Zahlen rot hinterlegt.

Möchte der Anwender ein Feld selektieren, drück dieser auf das gewünschte Feld. Dies führt dazu, dass das Feld als markiert erkannt wird und blau hinterlegt wird. Dabei können Zahlen die von dem Algorithmus initial gesetzt worden sind, nicht selektiert werden. Die initial hinterlegten Zahlen werden durch eine dickere Schrift dargestellt.

Hat der Anwender ein Feld selektiert, kann er mit den unten angezeigten Tasten eine Zahl zwischen 1 und 9 auswählen. Ist bereits eine Zahl in dem selektierten Feld, wird diese Überschrieben. Möchte der Anwender eine Zahl komplett entfernen, kann er dies durch gedrückt halten eines Feldes erreichen.

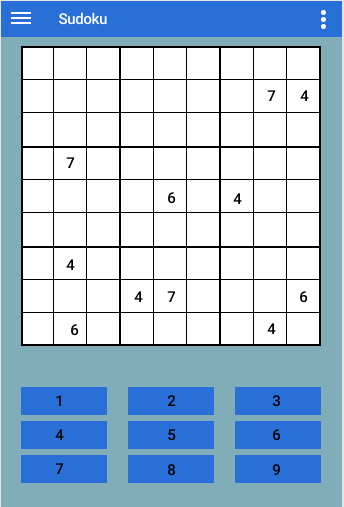
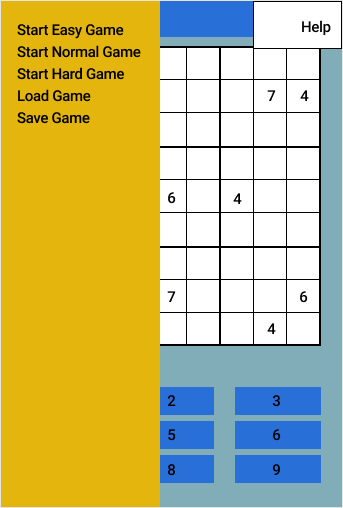
 

Abbildung Mockups der Sudoku-App

# Konzeption

## Sudoku Generierung Algorithmus

Der Generierung Algorithmus der Sudoku App basiert auf den zwei Artikeln (Computing, 2017) und (Computing, 2019). Das Ziel dieses Algorithmus ist es ein Sudoku Spiel zu generieren, welches exakt eine mögliche Lösung bietet. Dabei gibt es entsprechend zwei Ausgaben. Einmal ein komplettes Sudoku mit allen gefüllten Zahlen, welches die Lösung widerspiegelt und ein Sudoku in dem Zahlen aus der Lösung entfernt wurden, wobei geprüft werden muss, dass es dabei eindeutig Lösbar bleibt. Die Anzahl der entfernten Zahlen bestimmt den Schwierigkeitsgrad, wobei darauf in diesem Kapitel noch detaillierter eingegangen wird.

Daraus lässt sich also ableiten, dass der Algorithmus in drei Teile und dementsprechend eigentlich aus drei Algorithmen besteht. Zum einen das Generieren eines kompletten Sudokus, zum anderen das Entfernen der Zahlen und zuletzt dem Prüfen der eindeutigen Lösbarkeit des Sudokus.

Das Entfernen von Zahlen sollte eindeutig simpel sein. Um dabei aber zu prüfen, ob das Sudoku noch eindeutig lösbar ist, nachdem eine Zahl entfernt wurde, wird ein Backtracking Algorithmus eingesetzt, welcher alle Möglichen Lösungen herausfindet.

### Backtracking Lösungsalgorithmus

Der Backtracking Algorithmus ist ein Lösungsalgorithmus für ein Sudoku. Das Sudoku wurde also bereits entsprechend bearbeitet, dass dort Felder entfernt wurden. Für dieses Sudoku soll der Algorithmus prüfen, ob es Lösbar ist und ob es eindeutig Lösbar ist.

Der Algorithmus iteriert zunächst über alle Felder des Sudokus, betrachtet dabei aber lediglich die leeren Felder. Hat der Algorithmus ein leeres Feld gefunden, wird für die Zahlen eins bis neun geprüft, ob diese bereits in der Reihe, Spalte und dem 3x3 Quadrat vorgekommen ist. Findet der Algorithmus eine Zahl für die dies nicht zutrifft, gibt es eine mögliche Lösung und wird dem Sudoku hinzugefügt. Wenn jetzt das Sudoku komplett gefüllt ist, kann die Anzahl an möglichen Lösungen hochgezählt und mit der nächsten Zahl fortgefahren werden. Ist das Sudoku noch nicht komplett gefüllt, wird der Algorithmus erneut aufgerufen, mit dem neuen Sudoku wo die aktuelle Zahl hinzugefügt worden ist. Findet der Algorithmus keine Zahl die hinzugefügt werden kann, ist das Sudoku nicht lösbar. Findet der Algorithmus mehrere Lösungen nachdem das Sudoku komplett gefüllt ist, ist es nicht eindeutig lösbar.



Abbildung Codeausschnitt Backtracking Lösungsalgorithmus

### Backtracking Algorithmus zum Erstellen eines Sudokus

Um ein Sudoku zu erstellen, können nicht einfach zufällig die Zahlen eins bis neun in eine 9x9 Tabelle geschrieben werden. Sie müssen dabei die Sudoku Regeln einhalten und dürfen nur einmal in jeder Reihe, Spalte und in jedem 3x3 Quadrat vorkommen.

Dafür wird über jedes Feld des Sudokus iteriert und für leere Felder die Zahlen zwischen eins und neun probiert einzufügen. Hier wird jedoch nicht nach der Reihenfolge eins bis neun eingefügt, sondern die Reihenfolge wird randomisiert, so dass nicht jedes Mal dasselbe Sudoku entsteht. Wird eine Zahl gefunden, welche noch nicht in der Reihe, Spalte und dem 3x3 Quadrat vorgekommen ist, wird diese dem Sudoku hinzugefügt. Ist das Sudoku nun komplett gefüllt, ist der Algorithmus fertig. Gibt es noch weitere leere Felder, wird der Algorithmus mit dem neuen Sudoku rekursiv aufgerufen. Dies geschieht so lange, bis keine freien Felder mehr vorhanden sind.



Abbildung Codeausschnitt Backtracking Algorithmus zum Erstellen eines Sudokus

### Entfernen der Zahlen aus dem vollständigen Sudoku

Zunächst wird zufällig ein Feld selektiert, welchen eine Zahl entfernt werden soll. Dabei darf dieses nicht bereits leer sein. Haben wir eine Zahl entfernt, wird anhand des Backtracking Lösungsalgorithmus geprüft, ob es eine Lösung gibt und ob diese eindeutig ist. Ist dies der Fall, wird die nächste Zufallszahl entfernt. Ist es aber nicht der Fall, wird die Zahl zurück in das Sudoku gepackt und mit einer anderen Zufallszahl fortgefahren. Dabei wird sich gemerkt wie oft eine Zahl zurück in das Sudoku gepackt wurde. Anhand dieser Zahl lässt sich so eine Schwierigkeit ermitteln. Wird der Algorithmus entsprechend erst bei einer hohen Zahl gestoppt, ist die Schwierigkeit entsprechend hoch. Erfahrungswerte haben hier gezeigt, dass ein Wiederholen von 5 Fehlversuchen eine einfache Schwierigkeit, 10 Fehlversuche eine mittlere Schwierigkeit und 15 Fehlversuche eine schwere Schwierigkeit ergibt. Beobachtungen haben entsprechend auch gezeigt, dass je höher die Fehlversuche sind, desto weniger Zahlen sich im Sudoku befinden, was auch ein Indikator für die Schwierigkeit sein kann.



Abbildung Codeausschnitt Entfernen der Zahlen aus dem vollständigen Sudoku

## Exkurs: Flutter

Flutter ist ein UI-Entwicklungs-Kit von Google. Flutter ist Open-Source und mit C++ implementiert worden. Mit Flutter lassen sich Cross-Plattform Apps entwickeln, dabei wird die Programmiersprache 3.2.1 Dart verwendet. Zielplattformen von Flutter sind hauptsächlich Android und IOS Geräte. Der Dart Code wird entsprechend für die jeweilige Zielplattform kompiliert und in einer Dart-VM ausgeführt. Ebenfalls ist bereits die Unterstützung für Web-Apps in der Umsetzung. Dort wird der Code in JavaScript umgewandelt und ist so in den modernen Browsern lauffähig. Dabei liegt der Fokus laut (Flutter, 2021) auf kurze Entwicklungszeiten und schnelle Ausführungszeiten, ohne den Verlust von „nativer User Experience“.

### Dart

Dart wurde hauptsächlich von Google entwickelt und gilt als eine moderne alternative zu JavaScript. Dabei dient Dart aber auch als Vielzweck-Programmiersprache und ist entsprechend vielseitig einzusetzen. So wird Dart beispielsweise in 3.2 Flutter verwendet. Dart Code kann aber auch im Browser laufen, indem er mit Hilfe von Dart.js in JavaScript Code transpiliert werden kann. Außerdem ist Dart ECMA-standarisiert. Gerade als Webentwickler ist man sich der schwächen von JavaScript bewusst und genau diese möchte Google mit Ihrer Programmiersprache Dart entgegenwirken. Dabei ist Google davon überzeugt, dass dies nicht mit aufsetzen auf JavaScript möglich ist, sondern eine neue Programmiersprache nötig war. Die Entwicklung begann 2010 und das erste Mal wurde Dart im Jahre 2011 vorgestellt. Die Version 1.0 wurde Ende 2013 veröffentlicht. Dart arbeitet wie die meisten Programmiersprachen Objektorientiert und ähnelt sehr den gängigen Programmiersprachen wie Beispielsweise Java oder C#.

### Widgets

Widgets ähneln Komponenten wie sie aus Angular 2+ oder Vue.js bekannt sind. Widgets sind einzelne Teile einer Anwendung in Flutter. Widgets können beliebig viele Abhängigkeiten zu anderen Widgets haben. Eine Flutter-App besteht also aus einem Widget, in dem beliebig viele weitere verschachtelte Widgets sind. Das hat den großen Vorteil, dass Widgets voneinander losgelöst entwickelt werden können und beliebig oft wiederverwendbar sind. Flutter verfolgt jedoch nicht das Prinzip von Vererbung. Stattdessen bietet Flutter eine Vielzahl von Widgets an, die zu eigenen Custom-Widgets zusammengesetzt werden können. Im Rahmen dieser Projektarbeit konnte kein fehlendes Widget ermittelt werden. Ebenfalls das Stylen wird mit Widgets umgesetzt, so gibt es Beispielsweise ein Widget „Center“ welches Widgets zentrieren kann. Damit die Anwendung möglichst Performant läuft, wird zwischen zwei Widgets unterschieden „Stateful“ und „Stateless“ Widgets.

#### Stateless Widgets

Stateless Widgets sind unveränderbar während der Laufzeit. Das heißt, ein Benutzer kann nicht mit diesen Interagieren. Das hat den Vorteil, dass Flutter diese nicht beobachten muss. Beispielsweise bieten sich Stateless Widgets für Icons, Menüs, Buttons oder Text an. Ein Stateless Widget wird durch eine Unterklasse der Klasse StatelessWidget implementiert. In der Sudoku Applikation wird ein Stateless Widget für das oberste Widget verwendet, wie in Abbildung 6 zu sehen.



Abbildung Codeausschnitt SudokuApp StatelessWidget

#### Stateful Widgets

Stateful Widgets können ihren State während der Laufzeit verändern und werden, sobald der State sich ändert, neu gerendert. Ein Stateful Widget wird implementiert durch zwei Klassen. Eine Klasse als Unterklasse von StatefulWidget und eine als Unterklasse von State<>. Wenn der State der Klasse verändert wird, löst das State Objekt die Methode setState() aus, welches zum rerendern führt. Beispiele für Stateful Widgets sind Checkboxen, Slider, Textfelder. In der Sudoku App ist ein Beispiel für ein Stateful Widget die Box, welche die einzelnen Felder des Sudokus widerspiegeln. Diese ändern Ihre Farbe, je nachdem ob sie selektiert sind oder die Hilfe angestellt ist und der Wert ändert sich. Jedem Stateful Widget wird ein State übergeben, welche für das Rendern des Widgets verantwortlich ist, wie in Abbildung 7 zu sehen ist.



Abbildung Codeausschnitt Box StatefulWidget

### Stylen und Formattieren

Da Flutter ein Konzept verfolgt, indem alles ein Widget ist, wird auch das Stylen und das Formatieren mit Hilfe von Widgets gemacht. Dafür gibt es extra Widgets wie Beispielsweise Center, Align, Container, die lediglich für diesen Zweck eingebaut werden. Das macht das Stylen extrem einfach, allerdings kann es auch in Spezialfällen beschränken. Es ist daher nicht ähnlich dem Stylen von HTML-Seiten, welche mit extra CSS-Klassen gestylt wird. Jedoch sind die Namen der Widgets sehr angelehnt an den Style-Attributen aus CSS.

Je nachdem, um welches Widget es sich dreht, können bestimmte Attribute übergeben werden. Beispielsweise kann einem Container eine Höhe, Breite und Farbe übergeben werden.



Abbildung Codeausschnitt Box styling

In Abbildung 8 ist das Zentrieren des Textes in den einzelnen Feldern zu sehen. Dafür wird der Text mit einem *Center-Widget* zentriert und dem Text-Widget ein *TextStyle* übergeben. Mit dem *TextStyle* wird die Schriftgröße gesetzt. Außerdem wird hier für die initial gesetzten Werte die Schrift Fettgedruckt angezeigt.

## Implementierungsphase

In diesem Kapitel wird auf die Besonderheiten bei der Entwicklung eingegangen und auf spezielle Entscheidungen, die bei der Entwicklung getroffen wurden.

### Datenstruktur der Sudoku Werte

Als Datenstruktur für die Werte des Sudokus wurde ein List-Objekt verwendet, welches wiederum List-Objekte enthält. Die Liste sieht folgendermaßen aus *List<List<int>>*. In vielen Programmiersprachen, wie beispielsweise C#, Java und TypeScript, ist der Typ der für ein Array verwendet wird, gekennzeichnet durch *datentyp[]*. Dies ist mit Dart nicht möglich und der zu wählende Typ ist entsprechend *List<datentyp>*. Die Initialisierung kann aber nach der gewohnten Syntax erfolgen. *List<int> array = [1,2,3];*.

### Fehlendes new Keyword in Dart

Im Quellcode der Sudoku-Applikation fällt auf, dass viele neue Klasse und Widgets mit Konstruktoren instanziiert werden, jedoch meistens das new Keyword vor dem Konstruktor Aufruf fehlt. Dies ist eine Vereinfachung von Dart, da auf dieses Keyword verzichtet werden kann.

Es ist also dem Programmierer überlassen, ob er *var container = Container(…);* oder *var container = new Container(…);* schreiben möchte.

### Aufbau der Widgets

Das Klassediagramm in Abbildung 9 soll verdeutlichen, wie die Abhängigkeiten der Widgets untereinander Aufgebaut sind. Dabei sind lediglich die Custom-Widgets betrachtet und nicht die Flutter eigenen.

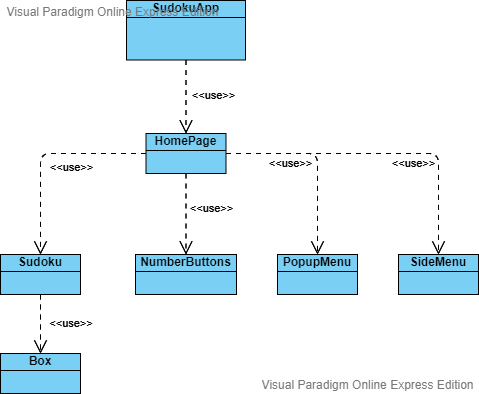


Abbildung Klassendiagramm der Abhängigkeiten der Widgets

Das oberste Widget ist die *SudokuApp*, in diesem Widget wird unter anderem ein Theme für die Applikation gesetzt und die Einstellung unternommen, dass nur die Anzeige im horizontalen Modus erlaubt. Das *HomePage* Widget stellt die Home Ansicht dar, welche die einzelnen Bereiche in Form von Widgets verwendet und positioniert. Das *Sudoku* Widget stellt den Bereich für das Sudoku Gitter und deren Felder dar, welche durch ein *Box* Widget implementiert sind. Jeder der 81 *Box* Widgets spiegelt ein Feld im *Sudoku* wider. Diese sind interagierbar und ermöglichen so dem Anwender das Auswählen eines Feldes und das Löschen von Zahlen. Die Zahlen die zum Auswählen da sind, werden durch das *NumberButtons* Widget dargestellt, in dem die 9 *Buttons* gerendert werden. Das *PopupMenu* Widget wird in dem *HomePage* Widget verwendet und ist ebenfalls interagierbar. Außerdem wird dem *HomePage* Widget noch das *SideMenu* hinzugefügt, welches ein Hintergrundbild anzeigt und die Möglichkeit gibt, über dem *Sudoku\_Service* das aktuelle Spiel zu verändern.

### Speichern und Laden eines Spielstandes

Beim Speichern eines Spielstandes, wurde sich dafür entschieden dies ohne Benutzerverwaltungssystem umzusetzen. Das Speichern wurde mit *SharedPreferences* umgesetzt, welches das Speichern Plattformunabhängig macht. Es benutzt intern für Android *SharedPreferences* und für IOS *NSUserDefaults*, welches auch bei MacOS zum Einsatz kommt. Mit *SharedPreferences* istesmöglich Daten lokal auf dem Handy zu speichern. Damit die Applikation aber nicht zu viel lokalen Speicher belegen kann und das verwalten von Spielständen nötig wäre, gibt es immer nur genau einen Spielstand der gespeichert werden kann. Deswegen ist es nötig, den Anwender bestätigen zu lassen, das er den alten Spielstand überschreiben möchte, wenn er einen neuen Spielstand abspeichern möchte. Um *SharedPreferences* zu verwenden, wir eine Instance benötigt und mit dieser können Key-Value Werte abgespeichert und geladen werden, siehe Abbildung 10.



Abbildung SudokuPersister Beispiel zum Speichern eines Spielstandes

Zum Speichern sind folgende Datentypen zugelassen, Bool, Double, Int, String und StringList. Da das Sudoku aber in einem List<List<int>> Objekt arbeitet, müssen diese beim Speichern und Laden jeweils geparst werden.

## Veröffentlichung im Google Play Store

* Bauen der App für Android
* Keys etc.
* Klicken der Oberfläche
* Beta Veröffentlichung
* Teste hinzufügen

# Fazit

* Fazit aus Sicht des Webentwicklers
  + Code ist alles in einer Datei und nichts getrennt.
* YouTube Widget of the Week

# Mögliche Erweiterungen

## Veröffentlichung im Apple Store

Nachdem in dieser Projektarbeit die Sudoku Applikation erfolgreich im Google Play Store Veröffentlichung und getestet worden ist, könnte die App auch für den Apple Store vorbereitet und Veröffentlicht werden.

## Offiziell im Google Play Store veröffentlichen

Zurzeit befindet sich die Applikation im Google Play Store noch beschränkt auf ausgewählte Tester und ist somit nicht für jeden zugänglich. Ein nächster Schritt könnte sein, offiziell die Testphase zu beenden und die Applikation für alle zugreifbar zu machen.

## Flutter auf Betaversion erhöhen und Web Kompilierung Testen

Da Flutter in der Beta Version bereits über die Möglichkeit verfügt, die Applikation für den Browser erstellen zu lassen, kann im weiteren Verlauf die Flutter Version in dem Projekt gepatcht werden und somit die Applikation für den Browser verwendet werden. Da dies allerdings zurzeit noch in der Beta ist, empfiehlt es sich nicht damit Produktiv zu gehen.

## Automatisches Speichern

Ein weiteres praktisches Feature wäre es, ein automatisches Speichern einzubauen, welches der Anwender aktivieren kann. Dies könnte Beispielsweise nach jeder Zahleingabe oder Zahllöschung zwischenspeichern. So müsste der Anwender nicht mehr manuell Speichern. Das Ganze sollte jedoch manuell aktiviert werden müssen, da sonst alle Spielstände die eventuell noch zu Ende gespielt werden möchten, überschrieben werden.

Alternativ könnte auch ein Mix aus beiden implementiert werden, dass sowohl ein Spielstand des aktuellen Spiels automatisch gespeichert wird und zusätzlich der Anwender ein Spielstand manuell speichern kann. So wäre zwei Spielstände lokal auf dem Gerät gespeichert, was akzeptabel ist.

# Abbildungsverzeichnis

[Abbildung 1 Use-Case Diagramm der Sudoku-App 7](#_Toc62912232)

[Abbildung 2 Mockups der Sudoku-App 8](#_Toc62912233)

[Abbildung 3 Codeausschnitt Backtracking Lösungsalgorithmus 10](#_Toc62912234)

[Abbildung 4 Codeausschnitt Backtracking Algorithmus zum Erstellen eines Sudokus 12](#_Toc62912235)

[Abbildung 5 Codeausschnitt Entfernen der Zahlen aus dem vollständigen Sudoku 13](#_Toc62912236)

[Abbildung 6 Codeausschnitt SudokuApp StatelessWidget 15](#_Toc62912237)

[Abbildung 7 Codeausschnitt Box StatefulWidget 16](#_Toc62912238)

[Abbildung 8 Codeausschnitt Box styling 17](#_Toc62912239)

[Abbildung 9 Klassendiagramm der Abhängigkeiten der Widgets 18](#_Toc62912240)

[Abbildung 10 SudokuPersister Beispiel zum Speichern eines Spielstandes 19](#_Toc62912241)

# Literaturverzeichnis

Computing, B. A., 2017. *Backtracking Algorithm – Sudoku Solver.* [Online]   
Available at: https://www.101computing.net/backtracking-algorithm-sudoku-solver/

Computing, S. G., 2019. *Sudoku Generator Algorithm.* [Online]   
Available at: https://www.101computing.net/sudoku-generator-algorithm/

Flutter, G., 2021. *Flutter.* [Online]   
Available at: https://flutter.dev/

Google, D., 2021. *Dart.* [Online]   
Available at: https://dart.dev/

Tutorialspoint, 2021. *Flutter introduction.* [Online]   
Available at: https://www.tutorialspoint.com/flutter/flutter\_introduction.htm