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: 14.02.2021, Iserlohn

Inhaltsverzeichnis

1 Einführung

1.1 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 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 Beispielsweise 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 mich als hauptberuflicher Web/Backend Entwickler auch mit Smartphone Applikationen auskenne und mir einen Überblick von Flutter verschaffen kann, wird in dieser Projektarbeit Flutter evaluiert. Dafür wird eine Sudoku Applikation erstellt, um erste Eindrücke zu erlangen.

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.

1.2 Anwendungsbereiche und Zielgruppen

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

2 Anforderungsanalyse

2.1 Entwicklungsprozess

Die Sudoku Applikation soll nach dem Wasserfall Modell entwickelt werden. Es wurde sich aus folgenden 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.

2.2 Funktionale Anforderungen

/F-010/ Dem Anwender soll es möglich sein, ein Sudoku Spiel komplett zu spielen.

/F-020/ Der Anwender kann genau immer ein Feld gleichzeitig selektieren. Dabei sind die Felder ausgeschlossen, welche zum Start des Spiels vorbelegt sind.

/F-030/ Felder die beim Start des Spiels vorbelegt sind, werden durch eine dicke schwarze Schrift gekennzeichnet.

/F-040/ Das selektierte Feld wird farblich hinterlegt.

/F-050/ Im unteren Teil der Applikation, sollen Zahlen zwischen 1 und 9 hinterlegt werden.

/F-060/ Wenn ein Feld selektiert ist, kann durch drücken einer dieser Zahlen, das Feld gefüllt werden.

/F-070/ Wurde das Feld bereits gefüllt, wird der Wert mit dem neuen Wert überschrieben.

/F-080/ Eine Zahl kann aus einem Feld entfernt werden, indem länger auf das Feld gedrückt wird.

/F-090/ Ein Menü soll es geben, in dem der Anwender folgende Aktionen ausführen kann.

> **/F-092/** Generieren eines neuen Spiels, in verschiedenen Schwierigkeitsstufen (Leicht, Mittel, Schwer).
>
> **/F-094/** 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.
>
> **/F-096/** Laden des vorher gespeicherten Spielstands.

/F-100/ Wenn der Anwender ein Spiel gewinnt, wird diesem ein Pop-Up Fenster angezeigt.

/F-110/ Es soll eine Hilfe für den Anwender geben, welche aktivier-/deaktivierbar ist.

/F-120/ Wenn die Hilfe aktiviert ist, werden richtig eingefüllte Zahlen mit einem grünen Hintergrund und falsche mit einem roten Hintergrund befüllt.

/F-130/ Das Sudoku Spiel soll nur im Hochformat angezeigt werden können.

2.3 Nicht funktionale Anforderungen

/NF-010/ Das Generieren eines neuen Sudoku Spiels soll nicht länger als 5 Sekunden dauern, unabhängig von dem Schwierigkeitsgrad.

/NF-020/ Die Applikation soll auf allen Android Geräten mit einer Mindestversion von 4.1 oder höher laufen. Das entspricht dem API-Level 16.

/NF-030/ Die Applikation soll nicht für IOS Geräte veröffentlicht werden.

2.4 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 Applikation 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 eine Zahl aus einem ausgewählten Feld. Setzt voraus, dass dort eine Zahl eingefügt wurde.

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

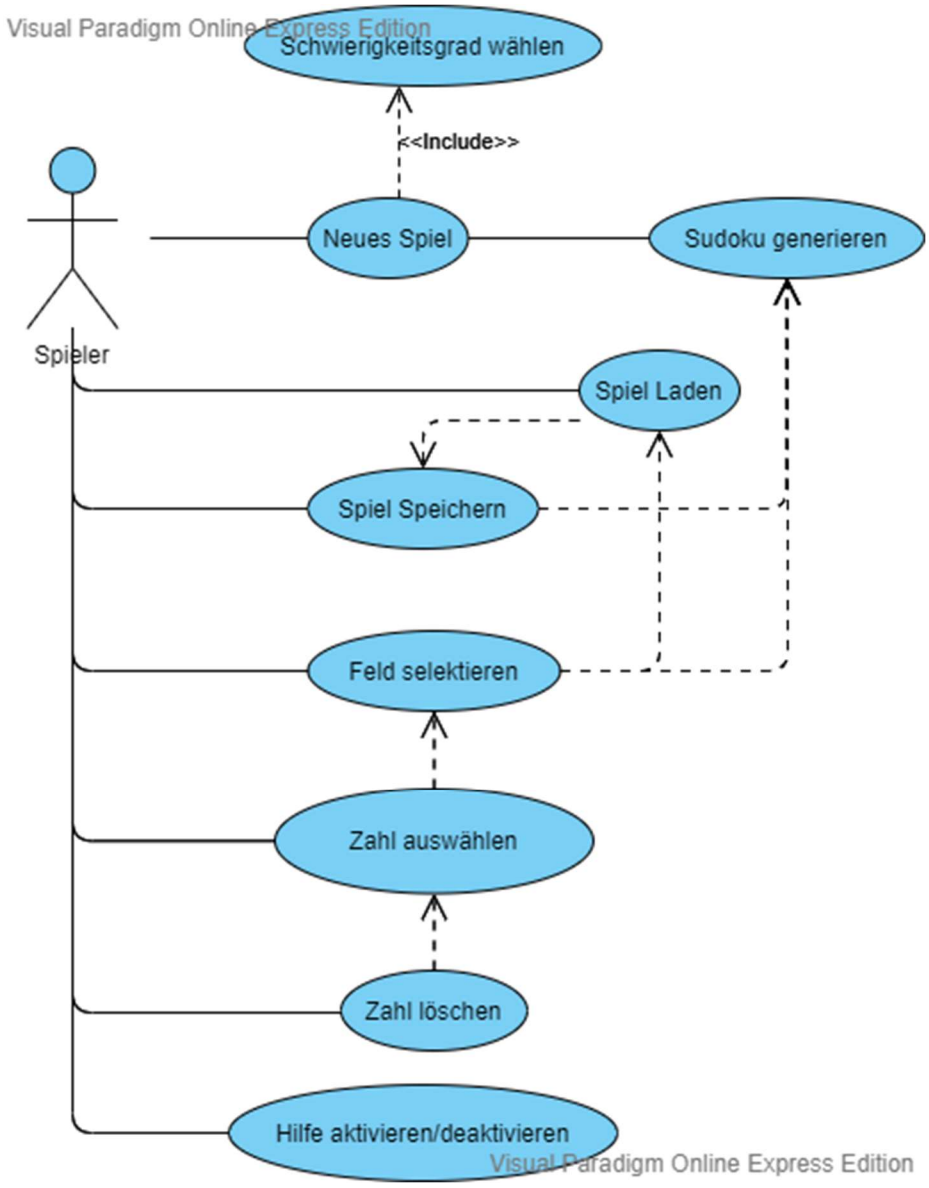

Abbildung 1 Use-Case Diagramm der Sudoku-App

2.5 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 sogenannte „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, drückt 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.

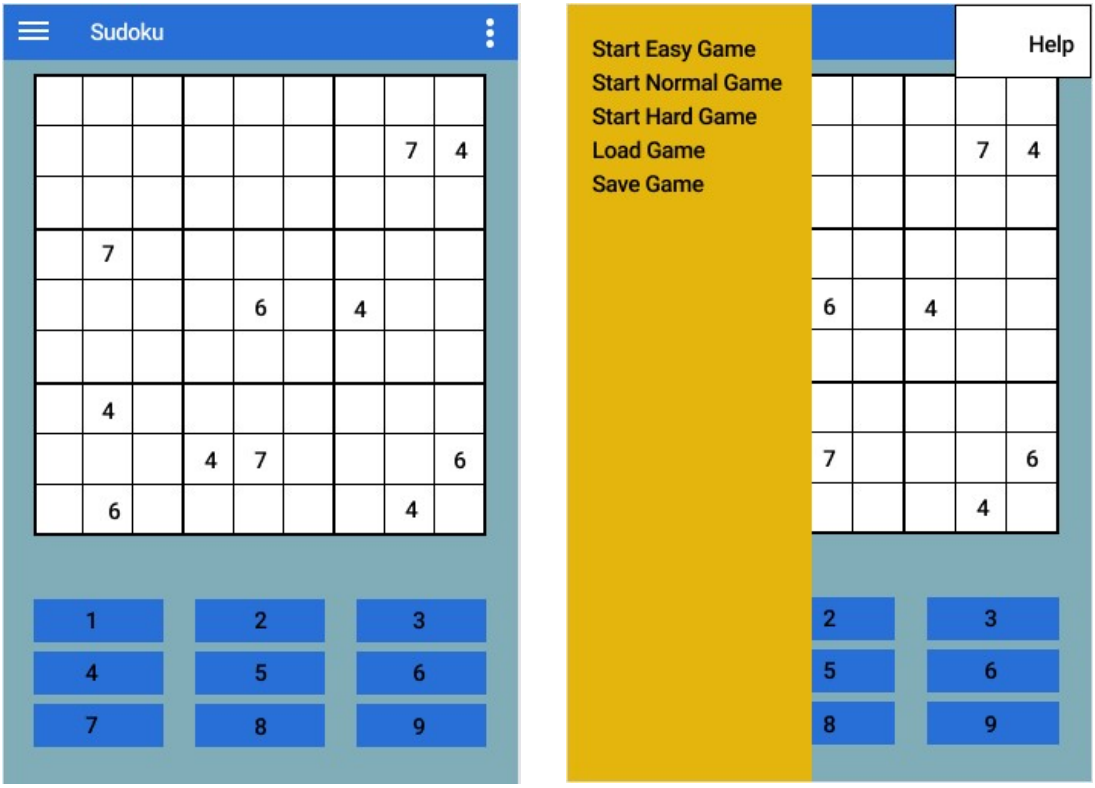

Abbildung 2 Mockups der Sudoku-App

3 Konzeption

3.1 Sudoku Generierung Algorithmus

Der Generierung Algorithmus der Sudoku Applikation basiert auf den zwei Artikeln (101 Computing, 2017) und (101 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.

3.1.1 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 1 bis 9 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.

```
// A backtracking/recursive function to check all possible combinations of
// numbers until a solution is found
bool _solvesudokuGrid(List<List<int>> sudokuGrid) {
  int row, col;

  for (int i = 0; i < 81; i++) {
    row = (i ~/ 9);
    col = i % 9;
    if (sudokuGrid[row][col] == 0) {
      for (int value = 1; value < 10; value++) {
        // Check that this value has not already be used on this row
        if (!sudokuGrid[row].contains(value)) {
          // Check that this value has not already be used on this column
          if (!transpose(sudokuGrid)[col].contains(value)) {
            // Identify which of the 9 squares we are working on
            List<int> square = SudokuArray.getSudokuSquareOfRowAndColumn(
                sudokuGrid, row, col);

            // Check that this value has not already be used on this 3x3 square
            if (!square.contains(value)) {
              sudokuGrid[row][col] = value;
              if (SudokuArray.checkSudokuGridFilled(sudokuGrid)) {
                _counter++;
                break;
              } else {
                if (_solvesudokuGrid(sudokuGrid)) {
                  return true;
                }
              }
            }
          }
        }
      }
      break;
    }
  }
  sudokuGrid[row][col] = 0;

  return false;
}
```

Abbildung 3 Codeausschnitt: Backtracking Lösungsalgorithmus

3.1.2 Backtracking Algorithmus zum Erstellen eines Sudokus

Um ein Sudoku zu erstellen, können nicht einfach zufällig die Zahlen 1 bis 9 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 1 und 9 probiert einzufügen. Hier wird jedoch nicht nach der Reihenfolge 1 bis 9 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.

```
// A backtracking/recursive function to check all possible combinations
// of numbers until a solution is found
bool _fillSudokuGrid(List<List<int>> sudokuGrid) {
  List<int> numberList = [1, 2, 3, 4, 5, 6, 7, 8, 9];
  int row, col;

  // Find next empty cell
  for (int i = 0; i < 81; i++) {
    row = (i ~/ 9); // Division with truncating
    col = i % 9;
    if (sudokuGrid[row][col] == 0) {
      _shuffle(numberList);
      for (int value in numberList) {
        // Check that this value has not already be used on this row
        if (!sudokuGrid[row].contains(value)) {
          // Check that this value has not already be used on this column
          if (!transpose(sudokuGrid)[col].contains(value)) {
            // Identify which of the 9 squares we are working on
            List<int> square = SudokuArray.getSudokuSquareOfRowAndColumn(
                sudokuGrid, row, col);

            // Check that this value has not already be used
            // on this 3x3 square
            if (!square.contains(value)) {
              sudokuGrid[row][col] = value;
              if (SudokuArray.checkSudokuGridFilled(sudokuGrid)) {
                return true;
              } else {
                if (_fillSudokuGrid(sudokuGrid)) {
                  return true;
                }
              }
            }
          }
        }
      }
      break;
    }
  }
  sudokuGrid[row][col] = 0;

  return false;
}
```

Abbildung 4 Codeausschnitt: Backtracking Algorithmus zum Erstellen eines Sudokus

3.1.3 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.

```
void _removeNumbersFromSudoku(List<List<int>> sudokuGrid) {
  // Start Removing Numbers one by one
  // A higher number of attempts will end up removing more numbers from the
  // SudokuGrid Potentially resulting in more difficiult SudokuGrids to solve!
  int attempts = _difficultyInt;
  _counter = 1;
  while (attempts > 0) {
    // Select a random cell that is not already empty
    var random = new Random();
    int row, col;
    do {
      row = random.nextInt(9);
      col = random.nextInt(9);
    } while (sudokuGrid[row][col] == 0);

    // Remember its cell value in case we need to put it back
    int backup = sudokuGrid[row][col];
    sudokuGrid[row][col] = 0;

    // Take a full copy of the sudokuGrid
    List<List<int>> copysudokuGrid =
        SudokuArray.duplicateSudokuGrid(sudokuGrid);

    // Count the number of solutions that this sudokuGrid has (using a
    // backtracking approach implemented in the _solvesudokuGrid() function)
    _counter = 0;
    _solvesudokuGrid(copysudokuGrid);
    // If the number of solution is different from 1 then we need to cancel the
    // change by putting the value we took away back in the SudokuGrid
    if (_counter != 1) {
      sudokuGrid[row][col] = backup;
      // We could stop here, but we can also have another attempt with a
      // different cell just to try to remove more numbers
      attempts -= 1;
    }
  }
}
```

Abbildung 5 Codeausschnitt: Entfernen der Zahlen aus dem vollständigen Sudoku

3.2 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 Applikationen 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 (Google, 2021) auf kurze Entwicklungszeiten und schnelle Ausführungszeiten, ohne den Verlust von „nativer User Experience“.

3.2.1 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 Exkurs: Flutter verwendet. Dart Code kann aber auch im Browser laufen, indem er mit Hilfe von Dart.js in JavaScript Code transpiliert wird. Außerdem ist Dart ECMA-standardisiert. 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#.

3.2.2 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.

3.2.2.1 Stateless Widgets

Stateless Widgets sind unveränderbar während der Laufzeit. Das heißt, ein Benutzer kann nicht mit diesem 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.

```
class SudokuApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    SystemChrome.setPreferredOrientations([
      DeviceOrientation.portraitUp,
      DeviceOrientation.portraitDown,
    ]);

    return MaterialApp(
      title: 'Sudoku',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      home: HomePage(title: 'Sudoku'),
    );
  }
}
```

Abbildung 6 Codeausschnitt: SudokuApp StatelessWidget

3.2.2.2 Stateful Widgets

Stateful Widgets können ihren State während der Laufzeit verändern und werden neu gerendert, sobald der State sich ändert. 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 erneuten Rendern 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. Ebenfalls ändern sich der Zahlenwert bei einer Eingabe vom Benutzer. Jedem Stateful Widget wird ein State übergeben, welche für das Rendern des Widgets verantwortlich ist, wie in Abbildung 7 zu sehen ist.

```
class Box extends StatefulWidget {
  // 0 means, box is not filled
  final int _value;
  // Save whether the value is already filled when the game starts.
  final bool _isInitalValue;
  final int _solutionValue;
  final Tuple2<int, int> _position;
  final Function _callback;
  final Function _deleteNumber;
  final SudokuService _sudokuService;
  final bool _isSelected;
…
…
  @override
  _BoxState createState() => _BoxState();
}

class _BoxState extends State<Box> {
  @override
  Widget build(BuildContext context) {
    return Expanded(
      child: InkWell(
…
…
    );
  }
}
```

Abbildung 7 Codeausschnitt: Box StatefulWidget

3.2.3 Stylen und Formattieren

Da Flutter ein Konzept verfolgt, indem alles ein Widget ist, wird auch das Stylen und das Formatieren mithilfe von Widgets gemacht. Dafür gibt es extra Widgets wie Beispielsweise Center, Align und Container, die lediglich für diesen Zweck in die Applikation 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 werden. 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.

```
child: Center(
      child: Text(
            widget._getValue(),
            style: TextStyle(
                  fontSize: 30.0,
                  fontWeight: widget._isInitalValue
                      ? FontWeight.bold
                      : FontWeight.normal),
            ),
```

Abbildung 8 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.

3.3 Implementierungsphase

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

3.3.1 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];*.

3.3.2 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.

3.3.3 Aufbau der Widgets

Das Klassendiagramm 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.

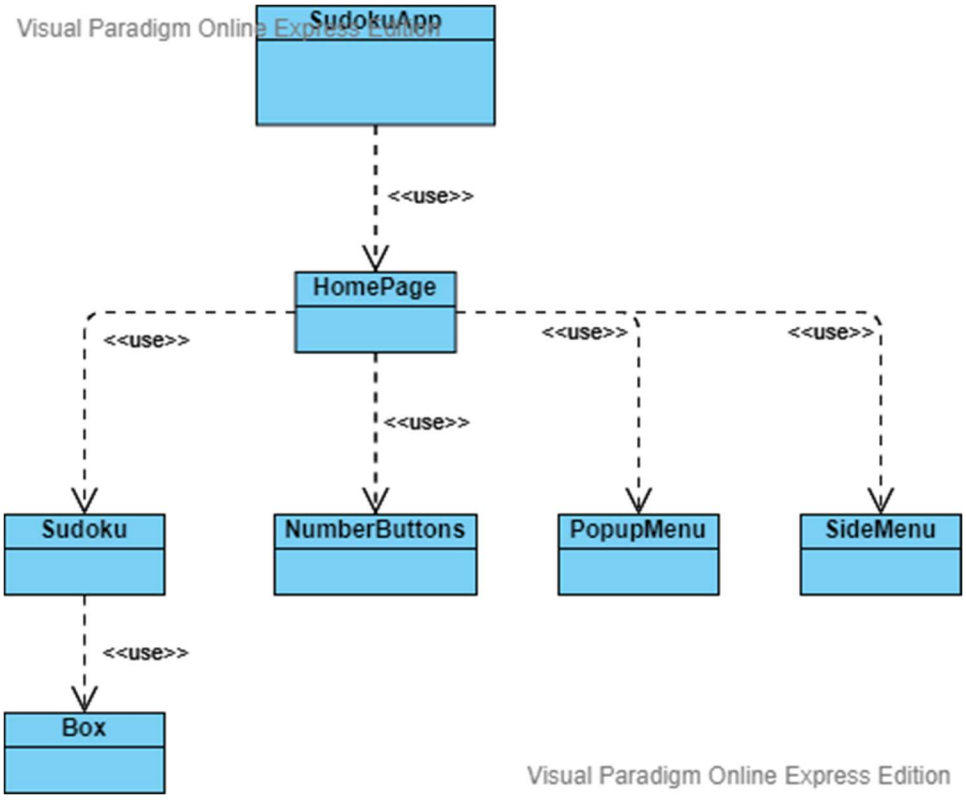

Abbildung 9 Klassendiagramm der Abhängigkeiten der Widgets

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.

3.3.4 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 umsetzt. Es benutzt intern für Android *SharedPreferences* und für IOS *NSUserDefaults*, welches auch bei MacOS zum Einsatz kommt. Mit *SharedPreferences* ist es möglich Daten lokal auf dem Handy zu speichern. Damit die Applikation aber nicht zu viel lokalen Speicher belegen kann und das für das Verwalten von mehreren Spielständen nötig wäre, gibt es immer nur genau einen Spielstand der gespeichert werden kann. Deswegen ist es nötig, den Anwender bestätigen zu lassen, dass er den alten Spielstand überschreiben wird, wenn er einen neuen Spielstand abspeichern möchte. Um *SharedPreferences* zu verwenden, wird eine

Instance benötigt und mit dieser können Key-Value Werte abgespeichert und geladen werden, siehe Abbildung 10.

```
static void saveSudoku(SudokuService sudokuService) async {
  SharedPreferences prefs = await SharedPreferences.getInstance();

  if (sudokuService.initialValues == null ||
      sudokuService.resolution == null ||
      sudokuService.acutalValues == null) return;

  prefs.setStringList(INITIALVALUES,
      SudokuArray.getSudokuAsStringList(sudokuService.initialValues));
  prefs.setStringList(RESOLUTION,
      SudokuArray.getSudokuAsStringList(sudokuService.resolution));
  prefs.setStringList(ACTUALVALUES,
      SudokuArray.getSudokuAsStringList(sudokuService.acutalValues));
}
```

Abbildung 10 Codeausschnitt: 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.

3.4 Veröffentlichung im Google Play Store

Bevor eine Applikation im Google Play Store veröffentlicht wird, sollte diese ausreichend getestet worden sein. Dafür sollten Beispielsweise Unit-Test, UI-Test und Entwicklertests ausgeführt werden. Wenn die Applikation bereit zum Veröffentlichen ist, müssen einige Bilder bereitgestellt, Konfigurationen vorgenommen und die Applikation gebaut werden.

3.4.1 Start Icon

Flutter Applikationen haben immer ein Default Icon von Flutter (Abbildung 11 Flutter Default Icon). Da dieses aber in der Regel nicht gewünscht ist, muss dies durch ein passendes Icon für die Applikation ersetzt werden. Welche Richtlinien eingehalten werden müssen, sind unter (Google, 2021) beschrieben.

Abbildung 11 Flutter Default Icon

Um ein Icon zu erstellen, kann einer der zahlreichen Editoren verwendet werden. In der Projektarbeit wurde der Editor von (Nurik, 2021) verwendet. Das erstellte Icon ist in Abbildung 12 Sudoku Icon zu sehen.

Abbildung 12 Sudoku Icon

Damit das Icon aktiv wird, muss es im Verzeichnis *<app dir>/android/app/src/main/res/* hinterlegt werden. Das hinterlegte Icon muss dann in der *AndroidManifest.xml* konfiguriert werden. Dazu wird der Eintrag *android:icon="@mipmap/ic_launcher"* angepasst. Alternativ kann das Icon auch beim initialen Konfigurieren in der Oberfläche in der (Google, 2021) hinterlegt werden.

3.4.2 Applikation signieren

Um eine Applikation zu veröffentlichen, sollte diese eine digitale Signatur bekommen.

3.4.2.1 Einen Keystore erstellen

Zuerst muss ein Keystore erstellt werden, wenn nicht bereits einer vorhanden ist. Mit dem folgenden Befehl kann ein Keystore unter Windows erstellt werden. Dabei muss ein Passwort vergeben werden.

```
keytool -genkey -v -keystore c:\Users\USER_NAME\key.jks -storetype JKS -keyalg RSA -keysize
2048 -validity 10000 -alias key
```

Abbildung 13 Windows command zum Erstellen eines Keystores

Der Befehl erstellt eine *key.jks* Datei. Die Datei darf nicht im GIT hinterlegt werden.

3.4.2.2 Keystore in der Applikation referenzieren

Im Verzeichnis *<app dir>/android/* muss eine *key.properties* Datei erstellt werden. Diese erhält folgenden Inhalt.

```
storePassword=<password from previous step>
keyPassword=<password from previous step>
keyAlias=key
storeFile=<location of the key store file, such as /Users/<user name>/key.jks>
```

Abbildung 14 key.properties Konfiguration

Auch diese Datei darf nicht im GIT hochgeladen werden.

3.4.2.3 Signieren in Gradle konfigurieren

In der *build.gradle* Datei muss zunächst die *key.properties* Datei ausgelesen werden und im *keystoreProperty* Objekt hinterlegt werden, wie in Abbildung 15 keystoreProperty auslesen zu sehen ist.

```
def keystoreProperties = new Properties()
def keystorePropertiesFile = rootProject.file('key.properties')
if (keystorePropertiesFile.exists()) {
    keystoreProperties.load(new FileInputStream(keystorePropertiesFile))
}

android {
    ...
}
```

Abbildung 15 keystoreProperty auslesen

Mit diesem Objekt können dann die *signingConfigs* für das Release eingerichtet werden, wie in Abbildung 16 signingconfig Einrichten beispielhaft beschrieben wurde. Danach wird beim Bauen, die Applikation automatisch signiert.

```
signingConfigs {
    release {
        keyAlias keystoreProperties['keyAlias']
        keyPassword keystoreProperties['keyPassword']
        storeFile keystoreProperties['storeFile'] ? file(keystoreProperties['storeFile']) : null
        storePassword keystoreProperties['storePassword']
    }
}
buildTypes {
    release {
        signingConfig signingConfigs.release
    }
}
```

Abbildung 16 signingconfig Einrichten

3.4.3 Applikationsnamen hinterlegen

In der *AndroidManifest.xml* kann der Applikationsname hinterlegt werden. Dafür muss das Attribut *android:label* angepasst werden.

Zusätzlich kann in dieser Datei Internet Zugang aktiviert werden. Dies ist aber nicht für die Sudoku Applikation notwendig. Dafür müsste das Attribut *android.permission.INTERNET* gesetzt werden.

3.4.4 Build Konfigurationen

Die Applikation brauch eine eindeutige Application ID. Diese wird in der *build.gradle* Datei bei dem Attribut *applicationId* hinterlegt.

Der *verisonCode* und der *versionName* werden in der *pubspec.yaml* Datei hinterlegt. Standardmäßig ist der *versionsCode* auf 1.0.0 gesetzt. Diese Werte, werden in der *build.gradle* Datei ausgelesen, wie in Abbildung 17 build.gradle build Konfiguration zu sehen ist.

```
defaultConfig {
    applicationId "christian.slupikowski.sudoku"
    minSdkVersion 16
    targetSdkVersion 29
    versionCode flutterVersionCode.toInteger()
    versionName flutterVersionName
}
```

Abbildung 17 build.gradle build Konfiguration

Ebenfalls müssen die *targetSdkVersion* und *minSdkVersion* gesetzt werden.

3.4.5 Applikation bauen

Es gibt zwei verschiedene Zieltypen, die beim Bauen von Flutter Apps ausgewählt werden können.

3.4.5.1 Android App Package (APK)

APKs sind bereits am längsten in Verwendung. Diese sind Dateien, die fertig zum Installieren auf einem Gerät sind. Diese Dateien haben alle benötigten Dateien, so wie den Quellcode mit im Umfang. Vergleichbar ist es mit einer .zip Datei. Dabei ist das Signieren fest im APK verankert und muss so immer neu erstellt werden und kann nicht fremd verwaltet werden.

3.4.5.2 Android App Bundle (AAB)

AAB ist das aktuellere Format für Android. 2018 wurde dieses von Google vorgestellt und ist seitdem das offizielle Format zum Veröffentlichen von Android Applikationen. In Abbildung 18 ist ersichtlich, wie eine AAB Datei aufgebaut ist. In (Android, 2021) ist genauer beschrieben, welcher Bereich wofür zuständig ist. Im Grunde ist eine APK Datei, der orange Teil einer AAB Datei. So fallen die dynamischen Bereiche weg, was einfach erklärt, wieso eine AAB Datei dynamischer ist. Dadurch ist beispielsweise möglich, dass die Signierung im Google Play Store verwaltet wird und nicht in der Datei statisch hinterlegt ist, wie es beim APK der Fall ist.

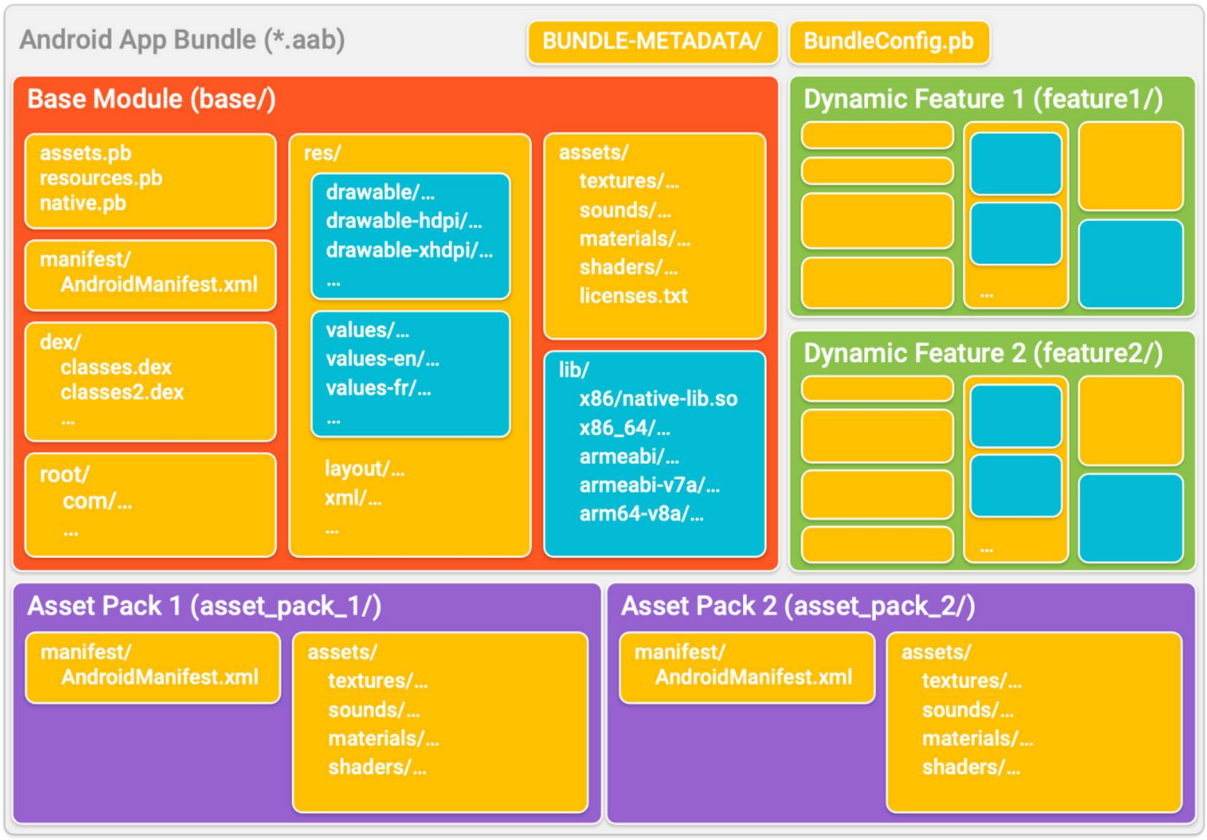

Abbildung 18 Android App Bundle Aufbau aus der (Android, 2021) Dokumentation

Da im Google Play Store empfohlen wird mit einem AAB zu veröffentlichen, wird im Weiteren auch nur dieser Prozess beschrieben.

3.4.5.3 AAB Bauen mit Flutter

Zum Bauen einer Flutter Applikation muss im Applikationsverzeichnis der Befehl

```
flutter build appbundle
```

ausgeführt werden. Dadurch wird das AAB automatisch signiert, da dies bereits konfiguriert worden ist. Im Verzeichnis *<app dir>/build/app/outputs/bundle/release/* ist die *app.aab* Datei zu finden.

3.4.6 AAB im Google Play Store veröffentlichen

Im Grunde wird auf der Google Play (Google, 2021) ein Ablauf zum Veröffentlichen einer Applikation vorgegeben. Dabei werden die bereits erstellten Dateien entsprechend hinzugefügt. Beispielweise die Screenshots und das Starticon. Ebenfalls werden Informationen abgefragt, um was für eine Applikation es sich handelt und ähnliches. Im Verlauf kann der Release Typ ausgewählt werden. Zunächst sollte ein Test-Release erstellt werden, welchem eine Benutzergruppe zugewiesen werden kann. So ist die Applikation nicht für jeden zugänglich, sondern nur den hinterlegten Testbenutzern Abbildung 19.

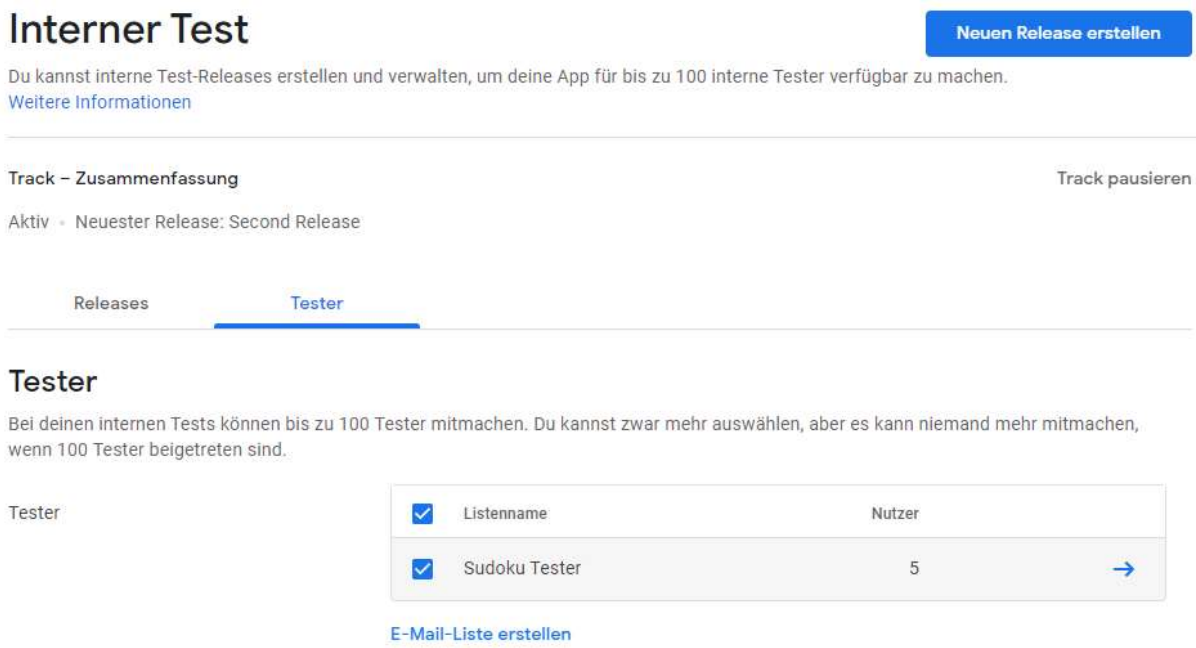

Abbildung 19 Internes Test Tester

In der Oberfläche kann die Applikation verwaltet werden. Dabei können Beispielsweise die App-Signatur eingesehen oder der Release Typ geändert werden, wenn zum Beispiel die internen Tests abgeschlossen sind.

Es werden ebenfalls automatisierte Tests von Google ausgeführt, welche Hinweise geben, auf mögliche Komplikationen auf verschiedenen Geräten.

Die Google Play (Google, 2021) ist sehr mächtig, daher wird in der Projektarbeit auf eine umfangreiche Beschreibung verzichtet. Jedoch ist die Google Play (Google, 2021) sehr intuitiv gestaltet, so dass wichtige Funktionen schnell zu finden sind.

3.4.7 Updaten der Versionsnummer

Um die Versionsnummer zu erhöhen, muss in der *pubspec.yaml* Datei die Versionsnummer angepasst werden, beispielsweise *version: 1.0.0+1*.

3.4.8 Update im Google Play Store

Beim Update der Applikation muss, nach dem Anpassungen des Quellcodes, die Versionsnummer erhöht werden. Danach kann das AAB neu gebaut werden, um es im Google Play Store in einem neuen Release zu veröffentlichen. Das Update wird dann automatisch bei den Endbenutzern angezeigt.

Ein neues Release kann in der Oberfläche bei den entsprechenden Release Status hinterlegt werden. Beispielsweise bei den internen Tests, wie in Abbildung 20 zu sehen.

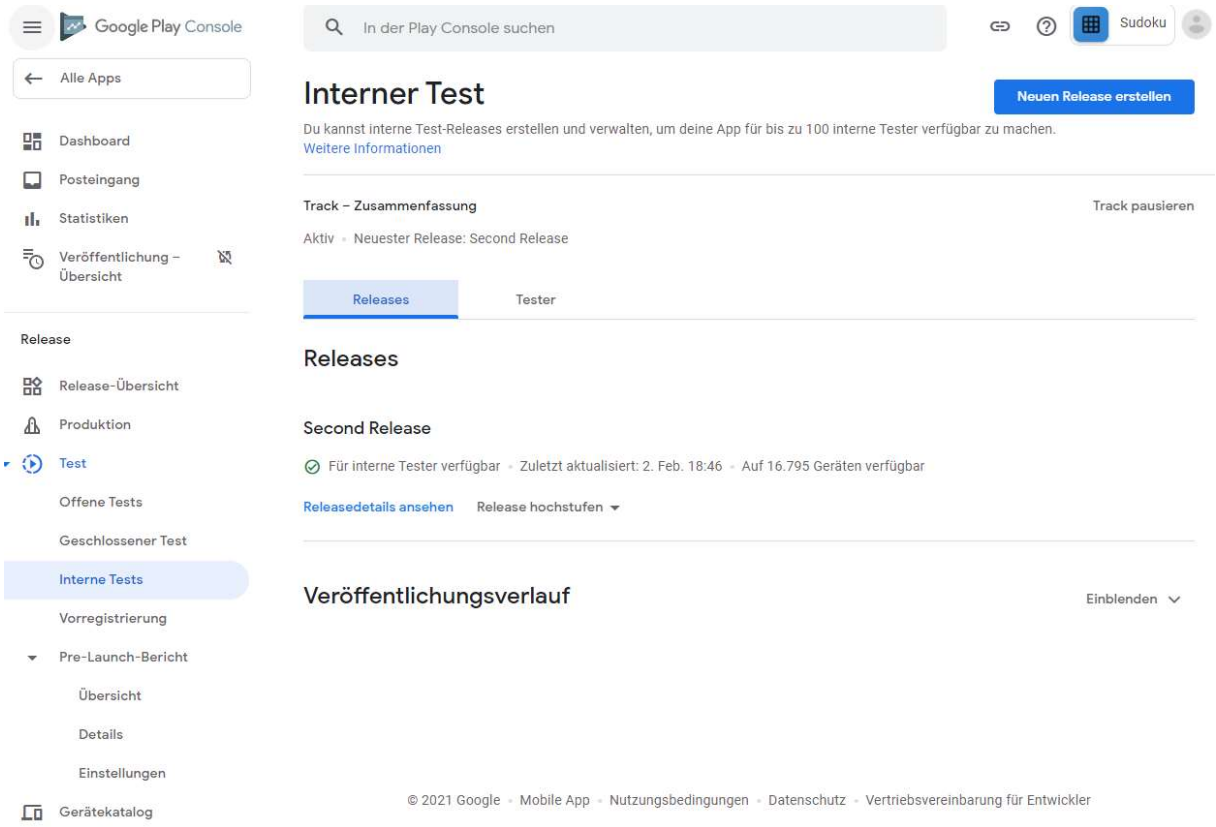

Abbildung 20 Interner Test Release

4 Fazit

Anhand der Implementierung einer Sudoku-Applikation konnte erfolgreich ein erster Eindruck von Flutter erlangt werden. Die Einarbeitung in Flutter erfolgte sehr schnell, durch die Ähnlichkeit von Widgets zu Komponenten von modernen Browserframeworks. Ebenfalls durch den Vorteil, dass die Formatierung komplett von Widgets übernommen wird, welche Ähnlichkeiten zu CSS aufweisen, konnte schnell eine Grundlage geschaffen werden. Da Dart grundsätzlich sehr ähnlich an moderne Programmiersprachen angelehnt ist, fiel auch hier eine Einarbeitung sehr gering aus. Allgemein wirkt Flutter sehr selbsterklärend, durch richtige Benennungen und einfach Konzepte. Angenehm ist auch, dass alle wichtigen Teile für die UI, in einer Datei sind. So wird der Status, die UI-Parts und die benötigten Daten, in einer Datei implementiert.

Ein großer Vorteil bei der Einarbeitung in Flutter ist, dass das Flutter Team eine Serie über verschiedene Widgets hat. Das Flutter Team postet wöchentlich ein Video auf YouTube mit dem Titel „Widget of the Week“. Dort sind die meisten Widgets bereits zu finden und kurz und einfach erklärt.

Die Statistik in Abbildung 21 Cross-Plattform Frameworks 1 Jahres Statistik zeigt, dass Flutter eines der beiden führenden Cross-Plattform-Frameworks ist und das es immer mehr ansehen bekommt.

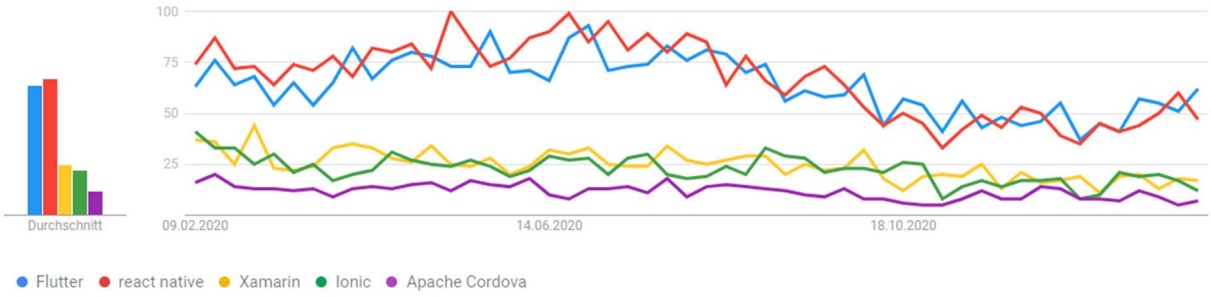

Abbildung 21 Cross-Plattform Frameworks 1 Jahres Statistik

Das Veröffentlichen im Google Play Store, kann anfangs komplizierter wirken als es ist. Das Bauen einer Flutter-Applikation ist ähnlich zu den meisten Frameworks. Ebenfalls das Signieren einer Applikation. Spezielle Werte für das Release können anhand von Anleitungen einfach angepasst werden und mit etwas Erfahrung ist dies auch ohne Anleitung möglich. Die Google Play Console, leitet einen sehr gut durch den Prozess, einer Veröffentlichung. Durch die Schritt-für-Schritt-Anleitung, ist es einfach erste Konfigurationen durchzuführen. Anfangs kann die Google Play Console jedoch unübersichtlich wirken, da diese viele Funktionen anbietet. Nach einer Einarbeitung ist jedoch schnell eine grobe Übersicht zu erlangen, was für ein erstes Test Release ausreicht. Ebenfalls ein Update für die Applikation zu veröffentlichen ist einfach möglich, da ein neues Release in der Oberfläche angelegt werden kann.

5 Mögliche Erweiterungen

5.1 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.

5.2 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.

5.3 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.

5.4 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.

5.5 Weitere Schwierigkeitsgrade

Es könnten noch weitere Schwierigkeitsgrade für das Sudoku angeboten werden. Dies ist grundsätzlich einfach zu erweitern. Schwierig wird es jedoch, wenn der Algorithmus zu lange braucht, weil die Anzahl an versuchen, eine Zahl zu entfernen, zu hoch wird. Dann sollte ein Ladebalken die Anwendung für die Zeit sperren. Wird eine noch höhere Schwierigkeit gewünscht, könnte ein besserer Algorithmus implementiert werden.

6 Abbildungsverzeichnis

7 Literaturverzeichnis

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

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

Android, 2021. *About Android App Bundles.* [Online]
Available at: https://developer.android.com/guide/app-bundle

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

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

Google, 2021. *Flutter YouTube.* [Online]
Available at: https://www.youtube.com/channel/UCwXdFgeE9KYzlDdR7TG9cMw

Google, 2021. *Google Play Console.* [Online]
Available at: https://play.google.com/console/

Google, 2021. *Google Trends.* [Online]
Available at: https://bit.ly/36KKMIi

Google, 2021. *Material Design Product icons.* [Online]
Available at: https://material.io/design/iconography/product-icons.html

Margain, E., 2020. *AAB vs APK.* [Online]
Available at: https://medium.com/better-programming/android-app-bundles-vs-apks-8b0306b38436
[Zugriff am 2021].

Nurik, R., 2021. *Android Asset Studio.* [Online]
Available at: https://romannurik.github.io/AndroidAssetStudio/index.html

Tutorialspoint, 2021. *Flutter Introduction.* [Online]
Available at: https://www.tutorialspoint.com/flutter/flutter_introduction.htm

8 Eigenständigkeitserklärung

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen Publikationen, Vorlagen und Hilfsmitteln, als die angegebenen benutzt habe. Alle Teile meiner Arbeit, die wortwörtlich oder dem Sinn nach anderen Werken entnommen sind, wurden unter Angabe der Literatur kenntlich gemacht. Gleiches gilt für von mir verwendete Internetquellen. Die Arbeit ist weder von mir noch von einem/einer Kommilitonen/in bereits in einem anderen Seminar vorgelegt worden.

Schwerte, 14.02.2021, C. Slupikowski

Ort, Datum, Unterschrift