

Eingereicht von

**Bernhard Auinger**

Angefertigt am

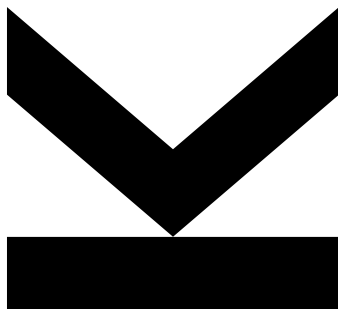
**JKU**  
Institut für  
Wirtschaftsinformatik -  
Information  
Engineering

Betreuer

**Ass.Prof. Dipl.-Ing. Dr.**  
**Rainer Weinreich**

April 2022

# **KONZEPTE DER REAKTIVEN PROGRAMMIERUNG AM BEISPIEL EINER FEUERWEHREINSATZ ANWENDUNG**



Bachelorarbeit

zur Erlangung des akademischen Grades

Bachelor of Science (BSc)

im Bachelorstudium

Wirtschaftsinformatik

## EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Bachelorarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die vorliegende Bachelorarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Ort, Datum



St. Pantaleon, 28.07.2022

Unterschrift

# INHALTSVERZEICHNIS

1. Einleitung.....	8
2. Konzepte .....	10
2.1. Reaktive Systeme .....	10
2.2. Reaktive Programmierung .....	10
2.2.1. Propagation of Change & Continuous time-varying Values.....	11
2.3. Taxonomie .....	13
2.3.1. Grundlegende Abstraktionen (Basic Abstraction).....	13
2.3.2. Auswertungsmodell (Evaluation Model).....	13
Pull-basiert .....	13
Push-basiert .....	14
2.3.3. Vermeidung von Inkonsistenzen (Glitch Avoidance) .....	14
2.3.4. Hebung (Lifting).....	14
Implizite Hebung.....	14
Explizite Hebung.....	14
Manuelle Hebung .....	15
2.3.5. Multidirektionale Ausbreitung von Änderungen (Multidirectionality) .....	15
2.3.6. Verteilungsunterstützung (Support of Distribution).....	15
2.4. Klassifikation .....	15
2.4.1. Funktionale reaktive Programmierung .....	15
2.4.2. Verwandte der Reaktiven Programmierung .....	16
2.4.3. Synchrone Programmierung, Datenflussprogrammierung und synchrone Datenflussprogrammierung .....	16
Synchrone Programmierung.....	16
Datenflussprogrammierung .....	16
Synchrone Datenflussprogrammierung.....	16
2.5. Reaktive Architekturen .....	17
Antwortbereit .....	17
Widerstandsfähig.....	17
Elastisch .....	18
Nachrichtenorientiert .....	18
2.6. Vor- und Nachteile Reaktiver Programmierung.....	19
Codekomplexität und Codesbarkeit .....	19
Wartbarkeit.....	19
Exaktere Fehlerbehandlung.....	19

Debugging.....	19
3. Ansätze .....	20
3.1. Literaturrecherche .....	20
3.2. Ansätze der funktionalen reaktiven Programmierung.....	20
3.2.1. Fran.....	20
3.2.2. Yampa.....	21
3.2.3. Elm .....	22
3.2.4. Flapjax.....	23
3.2.5. Sodium .....	24
3.3. Ansätze Verwandter der reaktiven Programmierung.....	25
3.3.1. ReactiveX.....	25
3.3.2. Project Reactor.....	26
3.3.3. Svelte .....	27
4. Analyse und Ergebnis.....	28
4.1. Fallstudie.....	28
4.2. Analyse der bestehenden Feuerwehreinsatzanwendung.....	29
4.3. Umsetzung der neuen Anwendung.....	39
4.4. Vergleich der Anwendungen.....	45
4.5. Evaluierung und Ergebnis.....	49
Codekomplexität.....	49
Codelesbarkeit .....	49
Wartbarkeit .....	49
Exaktere Fehlerbehandlung.....	49
Debugging .....	50
5. Zusammenfassung und Ausblick .....	51
6. Literaturverzeichnis.....	52
7. Tabellenverzeichnis .....	56
8. Abbildungsverzeichnis .....	56
9. Anhang.....	58
9.1. Auflistung der politischen Bezirke und deren Zugehörigkeit zu den Abschnitts- und Bereichs- bzw. Bezirksalarmzentralen .....	58
9.2. Übersicht Einsatzarten .....	59
9.3. Klassendiagramm der Neuentwicklung.....	60
9.4. Tabelle mit den Messwerten der zyklomatischen Komplexität .....	60
9.5. Wartbarkeitsmessung.....	61

## Kurzzusammenfassung

Das Ziel dieser Arbeit ist es die Vor- und Nachteile reaktiver Programmierung zu bestimmen. Dazu wurde die folgende Forschungsfrage gestellt: „Welche Vor- bzw. Nachteile ergeben sich durch den Einsatz der Konzepte reaktiver Programmierung für Webanwendungen?“. Außerdem wurden folgende Teilfragen gestellt: „Welche Konzepte und Ansätze für die reaktive Programmierung gibt es im Kontext von Webanwendungen?“, „Wie kann eine existierende Anwendung im Bereich Feuerwehreinsatzdarstellung reaktiv umgesetzt werden?“ und „Welche Vor- und Nachteile ergeben sich durch die reaktive Umsetzung der untersuchten Fallstudie?“.

Um diese Fragen zu beantworten, wird eine Literaturrecherche und eine Fallstudie durchgeführt. In der Fallstudie wird eine nicht-reaktive Feuerwehreinsatzanwendung untersucht und mittels reaktiver Ansätze neu implementiert. Weiters werden die beiden Anwendungen verglichen, um mögliche Vor- und Nachteile zu identifizieren. Abschließend werden die Ergebnisse der durch die Literatur recherchierten Vor- und Nachteile reaktiver Implementierungen mit den Ergebnissen des Vergleichs der beiden Anwendungen abgeglichen.

Das Abgleichen der Ergebnisse der Literaturrecherche und des Vergleichs der beiden Anwendungen ergibt, dass reaktive Anwendungen zwar eine geringere Codekomplexität und eine bessere Wartbarkeit aufweisen, aber beim Debugging Nachteile gegenüber imperativ programmierten Anwendungen haben.

Diese Arbeit zeigt, dass reaktive Programmierung Vorteile gegenüber konventionellen Programmierparadigmen hat und insbesondere bei größeren Anwendungen, bei denen die Wartbarkeit und Codekomplexität eine große Rolle spielen, als Mittel der Wahl in Betracht gezogen werden sollte.

## Abstract

The main goal of this thesis is to determine the advantages and disadvantages of reactive programming. The following research question was asked: "What are the advantages and disadvantages of using the concepts of reactive programming for web applications?". In addition, the following sub-questions were asked: "What concepts and approaches for reactive programming are there in the context of web applications?", "How can an existing application in the area of visualising fire service incidents be implemented reactively?" and "What advantages and disadvantages result from the reactive implementation of the examined case study?".

To answer these questions, a literature review and a case study are conducted. In the case study, a non-reactive fire service application is examined and re-implemented using reactive approaches. Furthermore, the two applications are compared to identify possible advantages and disadvantages. Finally, the advantages and disadvantages of reactive applications that are identified from a literature research are compared with the results of the comparison of the two applications.

The comparison of the results of the literature research and the comparison of the two applications shows that reactive applications have a lower code complexity and better maintainability, but have disadvantages in debugging compared to imperatively programmed applications.

This work shows that reactive programming has advantages over conventional programming paradigms and should be considered as the method of choice, especially for larger applications where maintainability and code complexity play a major role.

## Abkürzungsverzeichnis

RP .....	Reaktive Programmierung
FRP .....	Funktionale reaktive Programmierung
WASTL .....	Warn- und Alarmstufenliste
NÖ .....	Niederösterreich
BAZ .....	Bereichs- bzw. Bezirksalarmzentrale
AAZ .....	Abschnittsalarmzentrale
LWZ .....	Landeswarnzentrale
CLI .....	Command Line Interface
CYC .....	Cyclomatic Complexity
LOC .....	Lines of Code
IDE .....	Integrated Development Environment

# 1. Einleitung

Ob Angular mit „Reactive Forms“ oder Svelte mit „Truly reactive“: viele Webframeworks und Programmiersprachen behaupten heutzutage „reaktiv“ zu sein (Google, 2022; Svelte, 2022). Aber was bedeutet „reaktiv“ im Kontext der Webentwicklung? Welche *Konzepte* und *Ansätze* gibt es im Bereich der reaktiven Programmierung für Webanwendungen? Im Rahmen dieser Arbeit soll genau diesen Fragen nachgegangen werden. Dabei umschreibt *Konzept* die zugrundeliegende Theorie der reaktiven Programmierung (RP) und unter *Ansätzen* verstehen wir Frameworks und Programmiersprachen, die die reaktive Programmierung unterstützen. Darüber hinaus soll herausgefunden werden, wie eine bereits existierende (nicht-reaktive) Feuerwehreinsatzanwendung mit reaktiven *Konzepten* umgesetzt werden kann und welche Vor- und Nachteile sich aus dieser Umsetzung gegenüber der ursprünglichen Anwendung ergeben.

In dieser Fallstudie wird zunächst die bestehende Anwendung zur Anzeige von aktuellen Feuerwehreinsätzen in NÖ analysiert. Die Ergebnisse dieser Analyse sollen Probleme der Anwendung identifizieren und Aufschluss über die Struktur der bestehenden Anwendung geben. Anschließend werden Möglichkeiten einer reaktiven Neuimplementierungen geprüft und eine dieser Möglichkeiten zur Umsetzung ausgewählt. Nach deren praktischer Umsetzung wird auch diese analysiert und in der Folge gegenübergestellt, um mögliche Vor- und Nachteile der reaktiven Implementierung zu identifizieren. Im letzten Schritt werden die Ergebnisse der durch die Literatur recherchierten Vor- und Nachteile reaktiver Implementierungen mit den Ergebnissen des Vergleichs der beiden Anwendungen abgeglichen und präsentiert.

Ziel dieser Arbeit ist es die *Konzepte* und *Ansätze* von reaktiver Programmierung abzubilden und durch Umsetzung und vergleichende Analyse dieser webbasierten Feuerwehreinsatzanwendung feststellen zu können, welche Vor- und Nachteile sich daraus ergeben.

Nach dem Einleitungskapitel beschäftigt sich der Abschnitt *Konzepte* mit den theoretischen Grundlagen der reaktiven Programmierung. Im zweiten Abschnitt werden die *Ansätze* der reaktiven Programmierung dargestellt. Der darauffolgende Teil „Analyse und Ergebnis“ analysiert die bereits bestehenden Feuerwehreinsatzanwendung, eruiert die Rahmenbedingungen für die Neuentwicklung der Anwendung mittels reaktiver *Konzepte* und analysiert auch diese. Weiters werden in diesem Kapitel die Anwendungen miteinander verglichen. Im letzten Teil des Kapitels werden die Forschungsergebnisse interpretiert und dargestellt. Am Ende, wird noch ein Überblick über mögliche weiterführende Forschung gegeben.



Um die Forschungsfrage bzw. die Teilfragen der Forschungsarbeit den Kapiteln zuweisen zu können wurde die nachfolgende Tabelle 1 erstellt. Dabei wird auch auf die verwendete Methodik eingegangen.

Fragestellung	Kapitel	Methodik
<b>Welche Vor- bzw. Nachteile ergeben sich durch den Einsatz der Konzepte reaktiver Programmierung für Webanwendungen?</b>	4 (4.5)	Fallstudie bzw. vergleichende Analyse und Literaturrecherche
<b>Teilfrage 1:</b> Welche Konzepte und Ansätze für die reaktive Programmierung gibt es im Kontext von Webanwendungen?	2 und 3	Literaturrecherche
<b>Teilfrage 2:</b> Wie kann eine existierende Anwendung im Bereich Feuerwehreinsatzdarstellung reaktiv umgesetzt werden?	4 (4.3)	Literaturrecherche und Analyse (Inhaltsanalyse) der Anwendung
<b>Teilfrage 3:</b> Welche Vor- und Nachteile ergeben sich durch die reaktive Umsetzung der untersuchten Fallstudie?	4 (4.4)	Fallstudie und vergleichende Analyse

Tabelle 1: Forschungsfragen mit Kapitel und Methodik

## 2. Konzepte

In diesem Abschnitt der Bachelorarbeit, werden die *Konzepte* der reaktiven Programmierung bzw. die theoretischen Grundlagen der Arbeit dargelegt.

### 2.1. Reaktive Systeme

Grundsätzlich wird als reaktives System ein ständig mit der Umgebung interagierendes System bezeichnet. Dieses System reagiert, mittels Ausgaben, kontinuierlich auf die Eingaben (*Events*) der Umgebung. Ein reaktives System ist somit seiner Umgebung untergeordnet (Jourdan et al., 1994, S. 211). Das Verhalten eines solchen Systems wird durch eine Abfolge von mehreren Sequenzen von Aktionen beschrieben. In dieser Abfolge von Aktionen wird überprüft welche Eingaben wann, und ob überhaupt, relevant sind und wie das System auf ein Signal reagiert, zum Beispiel eine bestimmte Ausgabe erzeugt (Jourdan et al., 1994, S. 211).

Laut dieser Definition zeichnet sich ein reaktives System insbesondere durch folgende drei Eigenschaften aus:

- Interaktivität,
- Kontinuität und
- Sequenzen von Aktionen.

### 2.2. Reaktive Programmierung

Es ist schwierig, ein reaktives System mittels sequenzieller Programmieransätze umzusetzen, da es nicht möglich ist, die Abfolge der eingehenden Eingaben vorherzusagen. Außerdem müssen bei jeder Änderung alle Abhängigkeiten und Daten im System aktualisiert werden. Das kann zu einer sehr komplexen und fehleranfälligen Aufgabe werden. (Bainomugisha et al., 2013, S. 3).

Hier soll die reaktive Programmierung Abhilfe schaffen, da sie für die Entwicklung von ereignisgesteuerten Anwendungen sehr gut geeignet ist (Bainomugisha et al., 2013, S. 3).

Folgende Eigenschaften unterstützen die reaktive Programmierung dabei:

- Programme werden als Reaktion auf externe Ereignisse ausgedrückt und
- Zeitfluss, Daten- und Berechnungsabhängigkeiten werden automatisch verwaltet, so muss sich der Programmierer nicht um die Reihenfolge der Ereignisse kümmern

(Bainomugisha et al., 2013, S. 2).

Diese Eigenschaften erleichtern die Entwicklung von deklarativ geprägten Anwendungen. Der Entwickler kann sich auf das was zu tun ist konzentrieren und muss sich nicht um das wann kümmern (Bainomugisha et al., 2013, S. 3; Shibanaï & Watanabe, 2018, S. 13)

Das reaktive Programmierparadigma wurde durch die synchronen Datenflussprogrammierung geprägt und hat folgende Eigenschaften: Programme werden als Reaktion auf externe Ereignisse ausgedrückt. Zeitfluss, Daten- und Berechnungsabhängigkeiten werden automatisch verwaltet. Und die sogenannte Dynamik des Datenflusses beschreibt, dass dieser sich zur Laufzeit ändern kann. (Bainomugisha et al., 2013, S. 2).

### 2.2.1. Propagation of Change & Continuous time-varying Values

Eines der Grundprinzipien von reaktiver Programmierung ist, was mit „Propagation of Change“ bezeichnet wird und frei übersetzt „die Ausbreitung von Änderungen“ bedeutet. Diese freie Übersetzung, bringt das Prinzip sehr gut auf den Punkt: Zustandsänderungen werden durch das zugrundeliegende Ausführungsmodell zu allen von diesen Änderungen betroffenen Berechnungen weitergegeben (Bainomugisha et al., 2013, S. 3).

```
propagation of change

int var1 = 1;
int var2 = 3;

int varResult = var1 + var2;

print(varResult);
//Ausgabe: 4

var1 = 5;
print(varResult);
//Ausgabe: 8
```

Abbildung 1: Beispiel „propagation of change“  
In Anlehnung an: (Bainomugisha et al., 2013, S. 3)

Um das verständlicher zu machen wird das Prinzip folgend in einem Beispiel erläutert.

In diesem Beispiel wird die Summe von den beiden Variablen var1 und var2 berechnet. Dazu werden den beiden Variablen verschiedene Werte zugewiesen, anschließend addiert und varResult zugewiesen. In einer imperativen und sequenziell ausgeführten Sprache, würde in der varResult der Zahlenwert gespeichert werden. In der reaktiven Programmierung bleibt der Wert in der varResult jedoch immer aktuell. Der Wert wird automatisch neu berechnet, wenn es Änderungen in den Variablen var1 und var2 gibt. Der Wert von varResult ist somit abhängig von den zwei anderen Variablen. Diese können sich über die Zeit ändern.

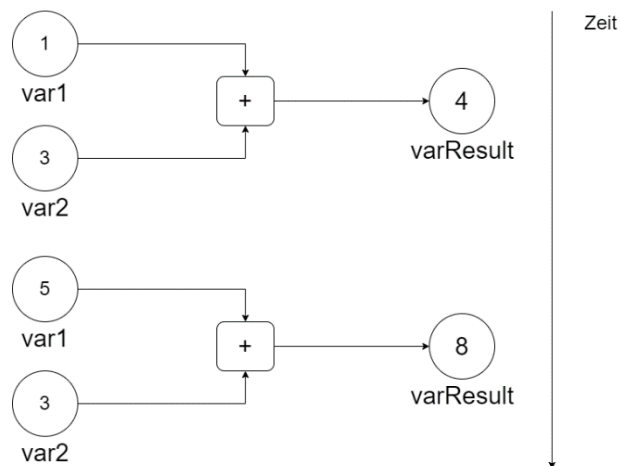


Abbildung 2: Zustandsgraph  
In Anlehnung an: (Bainomugisha et al., 2013, S. 3)

In Abbildung 2 werden die Zustände im Laufe der Zeit dargestellt. Anhand der Pfeile und des Operators werden die Abhängigkeiten der jeweiligen Variablen gezeigt.

Um die oben beschriebenen Grundprinzipien umzusetzen, werden bei reaktiven Programmieransätzen typischerweise zwei verschiedene Arten von Werten verwendet. Einerseits *Behaviors* und andererseits *Events* (Bainomugisha et al., 2013, S. 4).

Der Begriff des *Behaviors* wird bei Werten verwendet, die sich kontinuierlich im Lauf der Zeit verändern. Ein Beispiel hierfür wäre die Y-Koordinate einer Maus, die sich über die Zeit hinweg ändert, aber immer einen Wert zugewiesen hat (Bainomugisha et al., 2013, S. 4).

Die zweite Art von Werten, die in reaktiven *Konzepten* Anwendung finden, sind die *Events*. Ein Event stellt einen diskreten Wert im System dar. Diese können hin und wieder auftreten, haben aber nicht zu jedem Zeitpunkt einen Wert. Ein Beispiel für ein Event wäre ein Mausklick (Bainomugisha et al., 2013, S. 4).

## 2.3. Taxonomie

In diesem Abschnitt werden sechs Eigenschaften vorgestellt, anhand derer reaktive Programmiersprachen als solche klassifiziert werden können. Dieses Klassifizierungsschema wird als Taxonomie bezeichnet.

### 2.3.1. Grundlegende Abstraktionen (Basic Abstraction)

Wie es in imperativen Programmiersprachen primitive Datentypen (z.B.: Integer) bzw. Operatoren (z.B.: +) als grundlegende Abstraktion (engl: Basic Abstraction) gibt, so gibt es auch in der reaktiven Programmierung grundlegende Abstraktion, um die Programmierung zu erleichtern. Viele der reaktiven Programmiersprachen bieten als grundlegende Abstraktion *Behaviors* und *Events* an. Die Abstraktionen können oft auch zusammensetzbar verwendet werden, um ein Programm zu schreiben (Bainomugisha et al., 2013, S. 4).

### 2.3.2. Auswertungsmodell (Evaluation Model)

Das Auswertungsmodell beschäftigt sich mit der Frage wie Änderungen über einen Abhängigkeitsgraphen von Werten und Berechnungen weitergegeben werden und so die Propagation of Change umgesetzt werden kann. Dieser Aktualisierung aller abhängigen Werte passiert aus Sicht des Programmierers automatisch (Bainomugisha et al., 2013, S. 4).

Auf Sprachebene gibt es zwei verschiedene Bewertungsmodelle zur Umsetzung der automatischen Aktualisierung. Dabei wird unterschieden wer die Weitergabe von Änderungen initiiert, wie in Abbildung 3 dargestellt (Bainomugisha et al., 2013, S. 4).

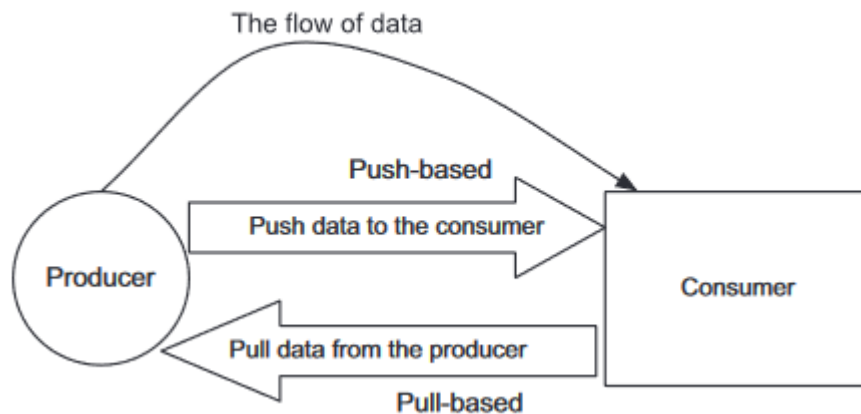


Abbildung 3: Evaluierungsmodelle  
Quelle: (Bainomugisha et al., 2013, S. 6)

#### Pull-basiert

Beim Pull-basierten Modell werden die abhängigen Daten von der Quelle, dem Produzenten, geholt. Das bedeutet, die Ausbreitung wird durch die Nachfrage nach neuen Daten vorangetrieben, ist also bedarfsgesteuert (demand driven) (C. M. Elliott, 2009, S. 25). Ein Beispiel für eine Pull-basierte Umsetzung ist Fran (C. Elliott & Hudak, 1997). Ein Vorteil des Pull-Evaluierungsmodells ist die Flexibilität, dass der Konsument Werte wirklich nur dann abrufen, wenn er diese auch wirklich benötigt. Der Nachteil bei diesem Modell ist, dass es zu einer hohen Latenz, zwischen dem Auftreten eines Ereignisses und dem Verarbeiten des Ereignisses, kommen kann (Bainomugisha et al., 2013, S. 4f).

## **Push-basiert**

Im Gegensatz zum Pull basierten- werden beim Push basierten-Modell die Daten von der Quelle zum Konsumenten geschoben. Wenn der Produzent neue Daten hat, werden diese Daten an abhängige Berechnungen übertragen. Daher ist der Push basierte Ansatz nach Verfügbarkeit der Daten orientiert (data driven) (Bainomugisha et al., 2013, S. 6; C. M. Elliott, 2009). Ein Beispiel für eine auf der Push Evaluierungsmethode basierenden Sprache ist Flapjax. Ein Nachteil von diesem Ansatz ist, dass es sehr oft zu Neuberechnungen kommen kann, wenn sich die Eingabe ändert, was sehr verschwenderisch sein kann (C. M. Elliott, 2009, S. 25). Aber auch zu Inkonsistenzen, die in 2.3.3 behandelt werden, kann es kommen. Der Vorteil von Push ist es, dass dadurch die Reaktion so schnell wie möglich erfolgt (Bainomugisha et al., 2013, S. 6).

Es gibt auch Sprachen, die beide Modelle kombinieren, um die jeweiligen Schwächen auszugleichen wie beispielsweise das Lula-System (Sperber, 2001).

### **2.3.3. Vermeidung von Inkonsistenzen (Glitch Avoidance)**

Glitch Avoidance ist eine Eigenschaft, die bei Einsatz des Push basierten Evaluierungsmodell eine Rolle spielt. Glichtes sind Aktualisierungsinkonsistenzen, die während der Weitergabe von Daten über den Abhängigkeitsgraphen entstehen (Shibanai & Watanabe, 2018, S. 14).

Inkonsistenzen treten dann auf, wenn eine Berechnung ausgeführt wird, bevor alle abhängigen Ausdrücke ausgewertet sind. Das kann dazu führen, dass alte Werte mit neuen Werten kombiniert werden. Es gibt verschiedene Ansätze, um Aktualisierungsinkonsistenzen zu vermeiden. Die meisten Sprachen verwenden einen topologisch sortierten Graphen, dadurch kann sichergestellt werden, dass einen Ausdruck immer ausgewertet wird wenn bereits alle abhängigen Quellen ausgewertet wurden (Bainomugisha et al., 2013, S. 7).

### **2.3.4. Hebung (Lifting)**

Die reaktive Programmierung wird oft mittels Frameworks oder Bibliotheken in sog. Hostsprachen eingebettet. Um gewöhnliche Operatoren, Funktionen und Methoden der Hostsprache zu modifizieren, so dass sie auf *Behaviors* (Wert, der sich über die Zeit hinweg ändert) reagieren können, wird das Lifting verwendet (Bainomugisha et al., 2013, S. 8).

Lifting verändert einerseits die Typsignatur, zum Beispiel einer Funktion (Argumente und Rückgabewert), andererseits führt das Lifting die Registrierung eines Abhängigkeitsgraphen im Datenflussdiagramm der Anwendung durch (Bainomugisha et al., 2013, S. 8).

Es gibt drei verschiedene Arten von Lifting, die je nach Art der Typisierung der Hostsprache zur Anwendung kommen.

#### **Implizite Hebung**

Bei der impliziten Hebung werden gewöhnliche Sprachoperatoren automatisch „gelifted“, wenn dieser auf ein *Behaviors* angewendet wird, das macht das Programmieren sehr transparent. Dieser Ansatz wird von dynamisch typisierten Hostsprachen verfolgt (Bainomugisha et al., 2013, S. 9).

#### **Explizite Hebung**

Die explizite Hebung stellt meistens mehrere Kombinationen zu Verfügung, um gewöhnliche Operatoren zu liften. Explizites Lifting wird meistens bei statisch typisierten Sprachen verwendet (Bainomugisha et al., 2013, S. 9).

## Manuelle Hebung

Manuelles liften stellt keine Lifting-Operatoren zur Verfügung, sondern es wird der aktuelle Wert eines *Behaviors* mit dem gewöhnlichen Operator der Hostsprache verwendet. Diese Art des Liftings wird in Hostsprachen verwendet, die kein *Behaviors* anbieten (Bainomugisha et al., 2013, S. 9).

### 2.3.5. Multidirektionale Ausbreitung von Änderungen (Multidirectionality)

Eine weitere Eigenschaft reaktiver Programmiersprachen ist die Multidirectionality, dabei geht es darum ob Änderungen nur in eine Richtung oder in beide Richtungen ausgebreitet werden (Propagation of Change) (Bainomugisha et al., 2013, S. 10). Mit Multidirectionality werden Änderungen auch zu Werten zurückgegeben, von denen sie abgeleitet wurden. Als Beispiel wird hier auf die Berechnung des Umfangs eines Quadrats verwiesen  $U = 4 \cdot a$ . Bei einer unidirektionalen Ausbreitung (in eine Richtung) wäre nur  $U$  von  $a$  abhängig und  $a$  würde sich bei einer Veränderung von  $U$  nicht aktualisieren. Ist die Ausbreitung, aber multidirektional würde sich der Wert von  $a$  verändern, wenn der Wert von  $U$  verändert wird.

### 2.3.6. Verteilungsunterstützung (Support of Distribution)

Die Eigenschaft der Verteilung unterstützt das Schreiben von verteilten reaktiven Programmen. Es ermöglicht Abhängigkeit zwischen Berechnungen oder Daten über mehrere Knoten zu verteilen. Der Bedarf für die Unterstützung von verteilten Systemen resultiert aus der zunehmenden Verteilung von Programmen insbesondere im Webbereich (Bainomugisha et al., 2013, S. 10).

## 2.4. Klassifikation

Wie Bainomugisha et al. (2013, S. 12) darlegen, lassen sich durch die in Kapitel 2.3 definierten Eigenschaften Reaktive Programmiersprachen in drei Kategorien einteilen:

- Funktionale reaktive Programmierung
- Verwandte der Reaktiven Programmierung
- Synchrone Programmierung, Datenflussprogrammierung und synchrone Datenflussprogrammierung

### 2.4.1. Funktionale reaktive Programmierung

Fran oder ausgeschrieben Functional Reactive Animation, ist ein in die funktionale Programmiersprache Haskell eingebettete Bibliothek (Wan & Hudak, 2000), mit deren Hilfe interaktive Multimedia Animationen erstellt werden können. Fran ist die Grundlage für die funktionale reaktive Programmierung und führt die Idee von *Behaviors* (kontinuierliche Werte) und *Events* (diskrete Werte) ein (*Functional reactive animation | Proceedings of the second ACM SIGPLAN international conference on Functional programming*, o. J.). Funktionale reaktive Programmiersprachen bzw. Bibliotheken müssen diese grundlegenden Abstraktionen (*Events* und *Behaviors*) enthalten um als solche zu gelten (Bainomugisha et al., 2013, S. 12). In FRP Sprachen, verändern sich Argumente in Funktionen im Laufe der Zeit und lösen so eine automatische Neu-Anwendung der Funktion aus. FRP stellt meistens Kombinatoren zum Zusammensetzen von *Events*, Unterstützen eines Higher Order Datenflusses und zur dynamischen Rekonfiguration des Datenflusses zur Verfügung (Bainomugisha et al., 2013, S. 12). Diese Art der Programmierung gibt dem Nutzer die Möglichkeit, mit einem deklarativen Stil zu arbeiten (Wan & Hudak, 2000, S. 243).

## 2.4.2. Verwandte der Reaktiven Programmierung

Bibliotheken und Programmiersprachen, die nicht die gleichen grundlegenden Abstraktionen für die Darstellung von zeitveränderlichen Werten haben und sich auf die automatische Verwaltung von Zustandsänderungen konzentrieren, werden als Verwandte der Reaktiven Programmierung bezeichnet. Die grundlegenden Abstraktionen dieser Bibliotheken bzw. Sprachen dienen dazu, die Abhängigkeit zwischen Wertproduzent und Wertkonsument zu schaffen. Beispiele sind ReactiveX-Implementierungen und Cells (Bainomugisha et al., 2013, S. 22).

## 2.4.3. Synchrone Programmierung, Datenflussprogrammierung und synchrone Datenflussprogrammierung

Synchrone Programmierung, Datenflussprogrammierung und synchrone Datenflussprogrammierung werden verwendet, um reaktive Echtzeitsysteme zu modellieren (Bainomugisha et al., 2013, S. 28).

### Synchrone Programmierung

Die synchrone Programmierung ist das früheste Programmierparadigma zur Entwicklung von reaktiven Echtzeitsystemen, es basiert auf der synchronen Hypothese (engl. Synchronous Hypothesis, Berry & Gonthier, 1992), die besagt, dass Reaktionen keine Zeit beanspruchen und atomar sind. Ein Beispiel für eine synchrone Programmiersprache ist Esterel (Bainomugisha et al., 2013, S. 29).

### Datenflussprogrammierung

Die Datenflussprogrammierung wurde ursprünglich entwickelt, um die parallele Programmierung zu vereinfachen. Programme werden als gerichteter Graph mit Knoten (Operationen) und Bögen (Datenabhängigkeiten) dargestellt. Eine Datenflusssprache wäre beispielsweise LabVIEW (Bainomugisha et al., 2013, S. 29).

### Synchrone Datenflussprogrammierung

In diesem Paradigma werden die synchrone und die Datenflussprogrammierung kombiniert. Das bedeutet, dass bei der synchronen Datenflussprogrammierung die Struktur des Graphen zur Kompilierzeit bekannt ist und in ein sequenzielles Programm umgewandelt werden kann. Ein Beispiel für eine synchrone Datenflussprogrammiersprache ist Lustre (Bainomugisha et al., 2013, S. 29)..

In Kapitel 3 werden keine *Ansätze* aus dem im Unterkapitel 2.4.3. ausgearbeiteten Programmierparadigmen weiter behandelt, da die Hauptdomäne dieser Sprachen in der Mess-, Regel- und Steuerungstechnik (Echtzeitsysteme) liegt und sie für die Webentwicklung nicht relevant sind.



## 2.5. Reaktive Architekturen

Nicht nur auf Ebene der Programmiersprachen hat sich der Begriff der Reaktivität durchgesetzt, auch im Bereich der Softwarearchitektur findet es Anwendung. Die Thematik der reaktiven Architekturen soll in dieser Ausarbeitung maßgeblich durch die *Konzepte* des Reaktive Manifest dargestellt werden.

Das reaktive Manifest ist ein von Jonas Bonér, Dave Farley Martin Thompson und Roland Kuhn erstelltes Dokument, in dem auf die Frage eingegangen wird, wie ein reaktives Softwaresystem aufgebaut sein muss.

Dabei sollen insbesondere die

- Skalierbarkeit,
- Anpassungsfähigkeit,
- Wartbarkeit,
- Zuverlässigkeit und
- Fehlertoleranz

verbessert werden (Bonér et al., 2014).

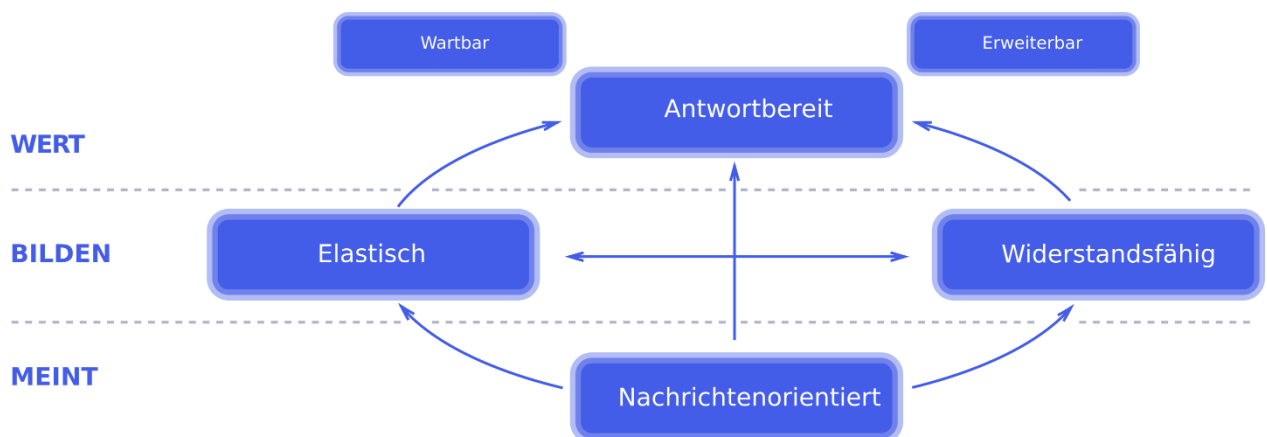


Abbildung 4: Merkmale von reaktiven Systemen  
Quelle: (Das Reaktive Manifest, o. J.)

Laut dem reaktiven Manifest müssen vier Qualitätseigenschaften erfüllt sein um ein Softwaresystem als reaktives System bezeichnen zu können. (*Das Reaktive Manifest*, o. J.)

### Antwortbereit

Um laut reaktivem Manifest einem System das Merkmal Antwortbereitschaft zusprechen zu können, muss dieses, wenn möglich, innerhalb einer festgelegten Zeit antworten. So kann von anderen Systemen, mit denen kommuniziert wird, schnell erkannt werden, ob ein System beispielsweise ausgefallen ist. Dieses Verhalten hilft die Benutzbarkeit des Systems zu steigern, in dem entsprechend auf Fehler reagiert wird und so ein nachvollziehbares Verhalten für alle beteiligten Parteien widerspiegelt werden kann (Bonér et al., 2014).

### Widerstandsfähig

Mit der Widerstandsfähigkeit bezieht sich das reaktive Manifest auf die Ausfallsicherheit einer Softwaresystems. Um die Funktion eines Systems zu gewährleisten, muss garantiert werden

können, dass bei Ausfällen von Hard- und Software die Systeme *antwortbereit* bleiben. Widerstandsfähigkeit lässt sich durch folgende Punkte erreichen:

- Replikation der Funktionalität
- Eindämmung von Fehlern
- Isolation von Komponenten
- Delegation von Verantwortung

Durch diese Maßnahmen können mögliche Ausfälle auf einen Teil des Systems begrenzt werden und andere Teilsysteme geschützt werden. Nach dem Ausfall eines Teilsystems, kann durch Delegation einer übergeordneten Komponente eine Replikation der untergeordneten Komponente zu Verfügung gestellt werden (Bonér et al., 2014). In Bezug auf die Isolation von Komponenten werden Mikroservice-Architekturen verwendet (Bonér, 2016).

### **Elastisch**

Diese Eigenschaft soll sicherstellen, dass ein System bei sich verändernden Lastbedingungen die Antwortbereitschaft erhalten kann. Auch in diesem Punkt spielt die Replikation von Funktionalität, um auf Veränderungen reagieren zu können die Hauptrolle. Dazu sollte ein Aufgabengebiet und unabhängige Teile zerlegt und auf viele Ressourcen verteilt werden. Die Architektur des Systems bzw. das System selbst sollte von vornherein keine Engpässe aufweisen. Reaktive Systeme sind in der Lage ihre Auslastung zur Laufzeit zu erfassen, um auf mögliche Veränderungen in der Auslastung reagieren zu können (Bonér et al., 2014).

### **Nachrichtenorientiert**

Die Nachrichtenübermittlung in dem System sollte asynchron zwischen den verschiedenen Komponenten ablaufen, um die Isolation sicherzustellen. Außerdem zur Fehlerübermittlung an übergeordnete Komponenten. Überwachung von Nachrichtenpuffern ermöglicht eine Priorisierung der Nachrichtenflüsse. Nachrichtenübermittlung führt zu Ortsunabhängigkeit, also dass es keinen Unterschied macht, ob ein Programm auf einem verteilten Netzwerk oder auf einem Computer ausgeführt wird. Außerdem sollten reaktive Systeme nicht-blockierend sein, um Ressourcen zu sparen (Bonér et al., 2014).

Alle vier in diesem Kapitel vorgestellten Eigenschaften sollten auf jeder Ebene der Systemarchitektur berücksichtigt werden, um laut reaktiven Manifest ein robustes und anpassungsfähiges System zu erstellen (Bonér et al., 2014).

## 2.6. Vor- und Nachteile Reaktiver Programmierung

In diesem Abschnitt werden die Vor- und Nachteile der reaktiven Programmierung aus der Literatur dargestellt.

### **Codekomplexität und Codelesbarkeit**

Durch den Einsatz von statischen Analysetools wurde in einer Studie von Holst & Gillberg die Unterschiede zwischen einem objektorientierten Programm (Java) und einem reaktiven Programm (RxJava) in Hinsicht auf die Codelesbarkeit und die Codekomplexität untersucht (2020, S. 7). Bei der statischen Analyse wurde jede Methode der Anwendung einzeln untersucht und der Durchschnitt aller Methoden einer Anwendung wurde ausgewertet (Holst & Gillberg, 2020, S. 7). Bei der Untersuchung konnte festgestellt werden, dass die Codekomplexität bei der reaktiven Anwendung deutlich geringer war als bei der imperativen Version. Bei der Lesbarkeitsmetrik wurde die reaktive Anwendung schlechter bewertet als die imperative Version. Wobei darauf hingewiesen wird, dass die verwendeten Lesbarkeitsmetriken nicht für die reaktive Programmierung geeignet sind (Holst & Gillberg, 2020, S. 11).

### **Wartbarkeit**

Die Studie die von Toczé et al. durchgeführt wurde zeigt dass bei der Messung der Wartbarkeit die reaktive Anwendung besser abschneidet als die imperative Anwendung (2016, S. 3). Zum Vergleich der imperativen und der reaktiven Anwendung wurde der Code in Hinblick auf die Größe untersucht und es wurde bei der imperativen Version eine 1,5 mal größere Codebasis festgestellt als bei der reaktiven Version (Toczé et al., 2016, S. 3).

### **Exaktere Fehlerbehandlung**

Bei der Entwicklung mit reaktiven Streams stehen sehr viele Operatoren zur Verfügung, insbesondere zur Fehlerbehandlung, wie zum Beispiel der `timeout()` Operator der einen Fehler wirft wenn in einer bestimmten Zeit kein Ergebnis zurückkommt (Dakowitz, 2018, S. 71). Dakowitz behauptet das die Streams eine leichtere Fehlerbehandlung und eine präzisere Kontrolle des Datenflusses zulassen (2018, S. 72).

### **Debugging**

Debugging stellt mit den klassischen Debuggern in IDEs und den Entwicklertools in Browsern nach wie vor ein Problem für reaktive Programmierung dar (Alabor & Stolze, 2020). Auch wenn es bereits Ansätze für reaktive Debugging-Tools gibt und eine große Anzahl von Entwickler diese kennen, werden noch die klassischen, für die imperative Programmierung ausgelegten Tools, verwendet, weil die Integration der reaktiven Debugger in vielen IDEs noch nicht gegeben ist. Außerdem ist der Umfang der zur Verfügung gestellten reaktiven Debuggern nicht vergleichbar (Alabor & Stolze, 2020).

## 3. Ansätze

In diesem Kapitel sollen Programmiersprachen bzw. -bibliotheken der Kategorien funktionale reaktive Programmierung und Verwandte der reaktiven Programmierung vorgestellt werden. Um die Implementierung der Sprachen zu zeigen, wird das gleiche Beispiel in jeder Sprache implementiert. Das Beispiel zeigt die Umwandlung der Geschwindigkeit von km/h in Knoten (knots). Zudem wird im ersten Unterkapitel die Vorgehensweise bei der Literaturrecherche beschrieben.

### 3.1. Literaturrecherche

Die *Konzepte* und *Ansätze* reaktiver Programmierung werden anhand der Ergebnisse einer strukturierten Literaturrecherche dargestellt. Unter anderem wird dadurch eine fundierte Basis für die Beantwortung der Forschungsfrage geschaffen.

Die Literaturrecherche umfasst mehrere Quellen, um mögliche Lücken in den entsprechenden Forschungsfeldern zu schließen (C. McCabe & Timmins, 2005, S. 42 ff.). Sie ist anhand der Vorgaben von McCabe & Timmins erstellt und strukturiert (2005).

### 3.2. Ansätze der funktionalen reaktiven Programmierung

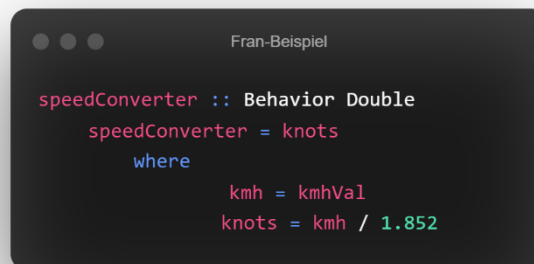
In diesem Unterkapitel werden die *Ansätze* der funktionalen reaktiven Programmierung besprochen. Mittlerweile gibt es unzählige FRP, doch um den Umfang dieser Arbeit nicht zu sprengen wurden nur jene, die speziell für den Webbereich interessant sind, ausgewählt. Weitere FRP, die in dieser Arbeit nicht behandelt werden, sind:

- Frappé
- FrTime
- NewFran
- Scala.React
- AmbientTalk/R
- etc.

#### 3.2.1. Fran

Fran oder auch Functional Reactive Animation ist eine Sammlung von Datentypen und Funktionen zum Erstellen interaktiver Animationen. Wie bereits im Kapitel 2.4.1. beschrieben, ist Fran der Ursprung für die Funktionale Reaktive Programmierung. Fran ist eine reaktive Programmierbibliothek, die in Haskell eingebettet ist (Bainomugisha et al., 2013, S. 13). Die Sprache wurde auf einem sehr hohen Abstraktionslevel entworfen, so muss der Programmierer nur ausdrücken, was gemacht werden soll und nicht wie. Fran bietet als grundlegende Abstraktion *Behaviors* und *Events* an, die über Kombinatoren zusammengestellt werden können. *Behaviors* werden aus anderen *Behaviors* und *Events* mittels Kombinatoren aufgebaut. Da Haskell eine statisch typisierte Sprache ist, muss das Lifting explizit durchgeführt werden. Die neuste Fran Version kombiniert das Push- und das Pull-basierte Auswertungsmodell (Bainomugisha et al., 2013, S. 13).

Der Geschwindigkeitsumrechner kann in Fran, wie in Abb. 5 dargestellt, implementiert werden.



```

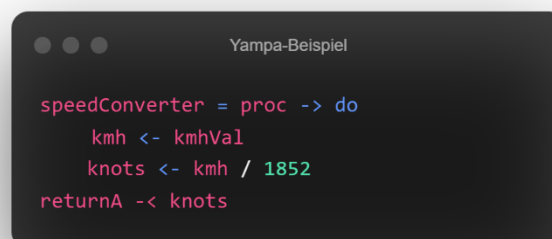
speedConverter :: Behavior Double
speedConverter = knots
  where
    kmh = kmhVal
    knots = kmh / 1.852
  
```

Abbildung 5: Fran-Beispiel

Die Funktion `speedConverter` gibt als Rückgabewert ein `Behavior` zurück, welches den zeitveränderlichen Wert der Knoten hält. Weiters wird der als gegeben angenommene Wert `kmhVal` `kmh` zugewiesen und mittels der `kmh` Variable die Knoten berechnet.

### 3.2.2. Yampa

Auch Yampa ist eine in Haskell eingebettete Sprache, die auf die Ideen von Fran zurückgeht und von der Yale Haskell Group an der Yale University entwickelt wurde. Yampa wurde für die Entwicklung von leistungskritischen Anwendungen in Haskell eingebettet. Die grundlegenden Abstraktionen in Yampa sind `Signal Functions` und `Events` (Bainomugisha et al., 2013, S. 14; *Yampa*, o. J.). `Signal Functions` sind im Gegensatz zu `Behaviors` Funktionen, die die zeitveränderlichen Werte kapseln, trotzdem ist eine `Signal Function` eine erstklassige Abstraktion. `Events` werden in Yampa über `Signal Functions`, als Event-Stream dargestellt, die immer ein Ereignis zu einem bestimmten Zeitpunkt liefert. Wie Fran bietet auch Yampa Kombinatoren zum Kombinieren von Ereignissen beziehungsweise `Signal Functions`. Yampa basiert auf dem Pull-Evaluierungsmodell und verwendet explizites Lifting. Einer der Hauptunterschiede zwischen Fran und Yampa ist, dass in Yampa der Eingabetyp einer `Signal Function` explizit angegeben werden muss, Fran bestimmt diesen Eingabetyp implizit (Bainomugisha et al., 2013, S. 14; *Yampa*, o. J.).



```

speedConverter = proc -> do
  kmh <- kmhVal
  knots <- kmh / 1852
  returnA <- knots
  
```

Abbildung 6: Yampa-Beispiel

Hier ist `SpeedConverter` eine `Signal Function`, welche mittels des Keywords „`proc`“ definiert wird. Weiters wird davon ausgegangen, dass `kmhVal` eine `Signal Function` ist, die immer den aktuellen Wert in km/h hält. Dieser wird dann mittels des Pfeiloperators zugewiesen beziehungsweise zuvor berechnet und dann zugewiesen. Im Anschluss wird der Wert von `knots` zurückgegeben.

### 3.2.3. Elm

Elm ist eine junge FRP-Sprache, welche 2012 von Evan Czaplicki als Teil seiner Masterarbeit entwickelt wurde. Anschließend wurde Elm im Jahr 2013 durch die Firma Prezi unterstützt, indem Evan Czaplicki für die Weiterentwicklung der Sprache angestellt wurde (Czaplicki, 2016). 2016 wurde die Elm Software Foundation gegründet, um die Bekanntheit der Sprache zu erweitern, die Sprache selbst weiterzuentwickeln beziehungsweise zu schützen und das Wachstum der Community zu fördern. Heute wird Elm bereits in vielen Unternehmen professionell verwendet (Czaplicki, 2016; Czaplicki et al., 2016).

Elm ist eine deklarative Sprache zur Erstellung von grafischen Benutzeroberflächen für das Web, wobei sich der Entwickler auf das konzentriert, was er anzeigen möchte und nicht wie (Czaplicki, 2012). Die Programmiersprache Elm kompiliert zu JavaScript, HTML und CSS. Elm kombiniert funktionale Programmierkonzepte mit reaktiven primitiven Abstraktionen, um Flexibilität und Ausdrucksstärke zu bieten (Czaplicki, 2012). Dabei wird in Elm die Datenpräsentation explizit von den Daten getrennt. Die Programmiersprache bietet zwei Arten von grafischen Grundelementen: Elements und Forms (Czaplicki, 2012).

Elements sind die Rechtecke mit einer gewissen Höhe und Breite, die Texte, Bilder oder Videos enthalten können. Diese Elemente können auch kombiniert werden, um schnell zusammengesetzte Elemente zu erstellen (Czaplicki, 2012).

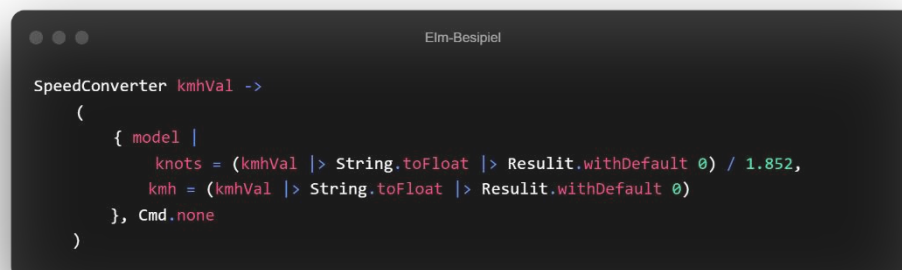
Das zweite Grundelement sind die Forms, welches die Erstellung von unregelmäßigen Formen und Texten bietet. Mit Forms können beispielsweise Polygone und Linien erstellt, skaliert, verschoben und gedreht werden. Forms können auch überlappen und bieten die Möglichkeit, komplexere Layouts zu erstellen als mit Elements (Czaplicki, 2012).

Die Sprache Elm bietet als Web-User-Interface Sprache folgende Vorteile:

- Breite Geräteunterstützung (alle Browser mit modernen Webstandards),
- statische Typisierung ohne Typdeklaration,
- kleine JavaScript-Bibliothek und
- angenehme Abstraktionen von HTML und CSS, um sonst schwierige Aufgaben umzusetzen.

(Czaplicki, 2012)

Das Beispiel der Geschwindigkeitsumwandlung kann in Elm, wie in Abbildung 7 gezeigt, umgesetzt werden.



```

Elm-Beispiel

SpeedConverter kmhVal ->
(
  { model |
    knots = (kmhVal |> String.toFloat |> Result.withDefault 0) / 1.852,
    kmh = (kmhVal |> String.toFloat |> Result.withDefault 0)
  }, Cmd.none
)
  
```

Abbildung 7: Elm-Beispiel

Der Typ `SpeedConverter` nimmt einen Parameter `kmhVal` entgegen. Anschließend werden die Knoten (*knots*) vom `kmhVal` berechnet, welcher zuvor von String zu Float konvertiert wird. Weiters wird der in `kmhVal` eingegebene Wert, wieder nach vorhergegangener Konvertierung, in `kmh` gespeichert.

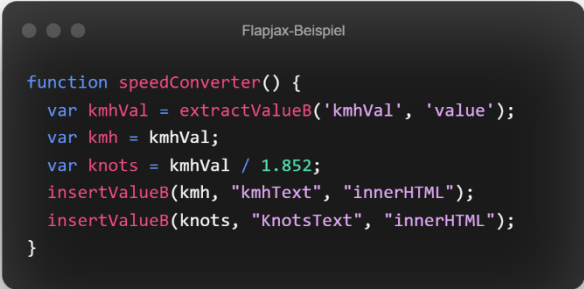
### 3.2.4. Flapjax

Flapjax ist eine Programmiersprache, die auf JavaScript aufgesetzt ist und die für die Erstellung von modernen Webanwendungen dient (Meyerovich et al., 2009). Sie wurde 2006 von Leo A. Meyerovich, Arjun Guha, Jacob Baskin, Gregory H. Cooper, Michael Greenberg, Aleks Bromfield und Shriram Krishnamurthi unter einer BSD Lizenz veröffentlicht (Krill, 2006). Die zwei Schlüsselfunktionen von Flapjax sollen das Schreiben von Webanwendungen erleichtern. Die erste Schlüsselfunktion ist die einheitliche Abstraktion (*Events-Streams*) für die Kommunikation innerhalb des Programmes sowie mit externen Webdiensten (Meyerovich et al., 2009). Die Zweite ist die Reaktivität der Sprache, Abhängigkeiten werden automatisch verfolgt und entlang dieser aktualisiert. Dies bietet einen deklarativen Programmierstil. Flapjax kann als Programmiersprache verwendet werden, die mittels eines Haskell-Compilers zu JavaScript kompiliert wird. Es kann aber auch als JavaScript-Bibliothek verwendet werden (Meyerovich et al., 2009). Flapjax unterstützt implizites und explizites Lifting von Funktionen zu *Behaviors* und das Evaluierungsmodell von Flapjax ist Push-basiert (Bainomugisha et al., 2013, S. 17). In Flapjax findet die AJAX Bibliothek für die Kommunikation mit Servern mittels einer eigens entwickelten Abstraktion Eingang (Bainomugisha et al., 2013, S. 17).

Folgende Vorteile hat Flapjax:

- Interoperabilität mit bereits vorhandenen JavaScript-Code,
- keine inkonsistenten Mutationen und
- Support in allen modernen Browser

(Meyerovich et al., 2009).



```
function speedConverter() {
  var kmhVal = extractValueB('kmhVal', 'value');
  var kmh = kmhVal;
  var knots = kmhVal / 1.852;
  insertValueB(kmh, "kmhText", "innerHTML");
  insertValueB(knots, "KnotsText", "innerHTML");
}
```

Abbildung 8: Flapjax-Beispiel

In Flapjax wird die Geschwindigkeitskonvertierung mittels dieser `SpeedConverter`-Funktion implementiert. Diese Funktion wird `onLoad()` aufgerufen und liest den Wert von `kmhVal` dauerhaft aus. Von diesem Wert ausgehend werden die Berechnungen durchgeführt und in HTML-Tags mittels der Flapjax-Funktion `extractValueB()` eingefügt.

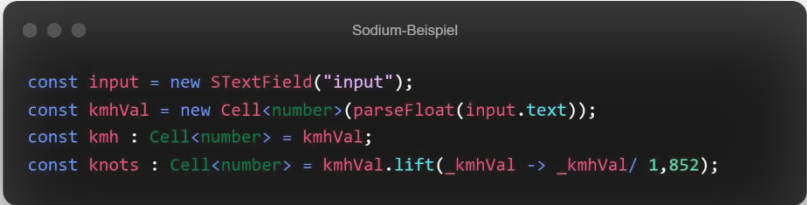
### 3.2.5. Sodium

Sodium ist eine FRP Bibliothek für mehrere Programmiersprachen, die unter der BSD-Lizenz veröffentlicht wurde. Sodium basiert auf Flapjax, Yampa, Scala.React und weiteren funktionalen reaktiven Programmieransätzen. Sodium FRP wurde von Stephen Blackheath gegründet und wird auf GitHub von zahlreichen Mitwirkenden unterstützt und weiterentwickelt. Sodium verwendet ein Push-basiertes Evaluierungsmodell (Blackheath, 2016, S. 19). Zeitveränderliche Werte werden in Sodium Cells genannt und eine Reihe von *Events* wird Stream genannt.

Sodium ist für die Programmiersprachen:

- C++,
- C#,
- F#,
- Java,
- Kotlin,
- Scala,
- TypeScript/JavaScript,
- Haskell (veraltet) und
- Rust

verfügbar (*SodiumFRP/sodium: Sodium - Functional Reactive Programming (FRP) Library for multiple languages*, o. J.). Das Codebeispiel würde in Sodium folgendermaßen aussehen:



```
const input = new STextField("input");
const kmhVal = new Cell<number>(parseFloat(input.text));
const kmh : Cell<number> = kmhVal;
const knots : Cell<number> = kmhVal.lift(_kmhVal -> _kmhVal / 1,852);
```

Abbildung 9: Sodium-Beispiel

In der ersten Zeile des Codes wird mittels des Typs „STextField“ der Wert des Textfeldes mit der Id „input“ eingelesen. Der Typ „STextField“ ist ein selbstdefinierter Typ, der mittels Event-Listener bei Veränderungen den neuen Wert liefert. Weiters wird der Wert des Textfelds aus der Variable Input in eine Zahl umgewandelt und in der Variable *kmhVal* von Typ Cell, die einen zeitveränderlichen Wert darstellt, gespeichert. In der dritten Zeile wird der Wert der *kmhVal* Variable der *kmh* Variable zugewiesen. In der letzten Zeile wird mittels des Lift-Operators eine Umrechnung durchgeführt und dann der Variable *knots* zugewiesen.



### 3.3. Ansätze Verwandter der reaktiven Programmierung

In diesem Abschnitt werden *Ansätze* von Verwandten reaktiver Programmierung vorgestellt und anhand eines Beispiels gezeigt. Auch hier kann nur ein Auszug der Frameworks beziehungsweise Programmiersprachen behandelt werden.

#### 3.3.1. ReactiveX

ReactiveX ist eine Bibliothek zum Erstellen von asynchronen eventbasierten Programmen unter Verwendung von beobachtbaren Sequenzen.

Fälschlicherweise wird ReactiveX manchmal der funktionalen reaktiven Programmierung zugewiesen, die Einleitung auf der Webseite zu ReactiveX sagt dazu: „ReactiveX may be functional, and it may be reactive, but “functional reactive programming” is a different animal. One main point of difference is that functional reactive programming operates on values that change continuously over time, while ReactiveX operates on discrete values that are emitted over time.“ (*ReactiveX*, o. J.).

ReactiveX verwendet das Observable Pattern, um Streams von asynchronen Ereignissen mit ähnlichen Operatoren, wie sie auf Collections von Datenelementen angewendet werden können, (verkettbare Operatoren) zu verarbeiten und zu kombinieren (*ReactiveX*, o. J.). Die Observables werden von den sogenannten Observers subscribed. Danach reagieren die Observer auf die Elemente, die das Observable sendet. Ein Vorteil dieses Observers ist, dass mehrere Aufgaben gleichzeitig starten können und sofern sie nicht voneinander abhängen, sich nicht blockieren. ReactiveX verwendet ein Push-basiertes Evaluierungsmodell (*ReactiveX*, o. J.).

ReactiveX ist auf zwanzig Programmiersprachen beziehungsweise -plattformen verfügbar, die wichtigsten für die Webentwicklung sind:

- RxJava (Java)
- RxJS (JavaScript)
- Rx.NET (C#)
- RxScala (Scala)
- RxPHP (PHP)
- RxKotlin (Kotlin)
- Rx.rb (Ruby)
- RxGo (Go)

(*ReactiveX*, o. J.)

Das Beispiel wurde mit RxJS umgesetzt.



```

RxJS-Beispiel

speed$.subscribe( kmhVal =>
{
    kmh = kmhVal;
    knots = (kmh * 1.8) + 32;
}
)

```

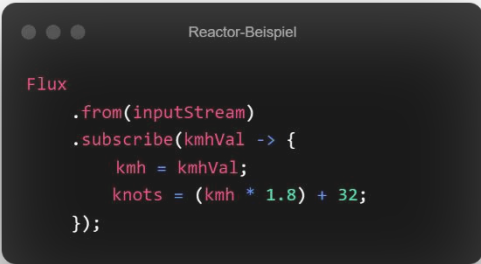
Abbildung 10: RxJS-Beispiel

Das Observable *speed\$* liefert immer den neusten *kmhVal*, dieser wird dann die Berechnung durchgeführt und zugewiesen.

### 3.3.2. Project Reactor

Project Reactor ist eine reaktive Programmierbibliothek, welche ab Java 8 verfügbar ist. Reactor basiert auf der Reactive Streams Spezifikation, einer Initiative, die einen Standard für die asynchrone, nicht blockierende Stream Verarbeitung bietet. Wie in ReactiveX, werden auch in dieser Bibliothek asynchrone Streams mit *Events* verarbeitet (*Project Reactor*, o. J.).

Reactor bietet zwei Datentypen Mono und Flux, beide stellen einen Stream mit *Events* dar und implementieren das Publisher-Interface der Reactive Streams API. Ein Mono ist ein Stream mit 0...1 Elementen und ein Flux kann 0...N Elemente enthalten. Der Datentyp Mono wird angeboten, da einige Operationen nur auf ein Element sinnvoll angewendet werden können (*Project Reactor*, o. J.).



```

Reactor-Beispiel

Flux
    .from(inputStream)
    .subscribe(kmhVal -> {
        kmh = kmhVal;
        knots = (kmh * 1.8) + 32;
    });

```

Abbildung 11: Reactor-Beispiel

In diesem Beispiel wird ein Flux aus einem *inputStream* erstellt. Daraufhin wird dieser subscribed und den *kmh* bzw. nach der Umrechnung den Knoten zugewiesen.

### 3.3.3. Svelte

Svelte ist ein Open Source Framework zur Erstellung von Benutzeroberflächen mit JavaScript oder TypeScript. Das SPA-Framework Svelte wurde in TypeScript geschrieben. Im Gegensatz zu vielen anderen Webframeworks wird Svelte-Code zu JavaScript-Code ohne Abhängigkeit von externen Programmierbibliotheken kompiliert (Svelte, 2022). In der folgenden Abbildung wird die Umrechnung der Geschwindigkeit mit Svelte gezeigt.

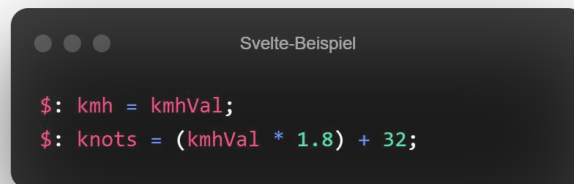


Abbildung 12: Svelte-Beispiel

Mit „\$:“ können reaktive Variablen und Statements deklariert werden. Die so deklarierten Variablen beziehungsweise Statements werden aktualisiert, wenn sich einer der ihnen zugewiesenen Werte ändert (Svelte, 2022).

## 4. Analyse und Ergebnis

In diesem Abschnitt wird die Vorgehensweise bei der Durchführung der Fallstudie beschrieben, die bestehende Feuerwehreinsatzanwendung analysiert, die Umsetzung der neuen Anwendung diskutiert und nach einem Vergleich der Analysen das Ergebnis präsentiert.

### 4.1. Fallstudie

Die in Kapitle 3.1 beschriebene Literaturrecherche ist die Basis für die erstellte Fallstudie.

Die Fallstudie ist eine empirische Forschungsmethode, welche die Untersuchungsobjekte in einer natürlichen Umgebung betrachtet und diese mittels qualitativer und quantitativer Datenquellen untersucht (Yin, 2017, S. 45 f.).

Es wird zwischen erforschendem, beschreibendem, erklärendem und verbesserndem Zweck für Fallstudien unterschieden, wobei ein einzelner Fall oder mehrere Fälle untersucht werden können (Runeson & Höst, 2008, S. 135 f.). Die Fallstudie, die hier umgesetzt wird, soll einen verbessernden Zweck haben. Ziel der Fallstudie ist es, durch Identifikation der Probleme der alten Anwendung eine verbesserte reaktive Neuimplementierung zur Verfügung zu stellen.

Die Fallstudie sollte nicht als einzelne Erhebungs- oder Auswertungstechnik verstanden werden, sondern als Prozess, mit dessen Hilfe eine spezifische Situation möglichst realitätsnah erfasst und beschrieben werden kann. So werden Sekundäranalysen, wie zum Beispiel eine Dokumentenanalyse, als Informationsquellen für die eigentliche Fallstudie benötigt (Schögel & Tomczak, 2009, S. 81 f.).

Die in dieser Bachelorarbeit durchgeführte Fallstudie wird anhand einer Inhaltsanalyse beziehungsweise vergleichenden Analyse durchgeführt. Dabei wird im ersten Schritt der Fallstudie die alte Anwendung analysiert und die Probleme der alten Anwendung identifiziert. Die Analyse gibt auch Aufschluss über die Struktur der alten Anwendung. Daraus leiten sich Anforderungen an die neue Implementierung ab. Weiters werden die Umsetzungsmöglichkeiten abgewogen und eine Implementierungsmöglichkeit gewählt. Dies erlaubt es, im nächsten Schritt die beiden Anwendungen zu vergleichen, um mögliche Vor- und Nachteile der reaktiven Implementierung zu identifizieren. Im finalen Schritt werden die Ergebnisse der durch die Literatur recherchierten Vor- und Nachteile reaktiver Implementierungen mit den Ergebnissen des Vergleichs der beiden Anwendungen abgeglichen und präsentiert.

## 4.2. Analyse der bestehenden Feuerwehreinsatzanwendung

Die Feuerwehreinsatzanwendung Grisu NÖ ist eine Webapplikation, welche von Christoph Scheidl, Philipp Kolmann und Alex Gassner, Mitglieder der Freiwilligen Feuerwehr Wolfsgraben, in Zusammenarbeit mit der Freiwilligen Feuerwehr Krems entwickelt und unter der MIT Lizenz veröffentlicht wurde (Gassner et al., 2016; *NÖ Landesfeuerwehrverband - WASTL / GRISU - System*, o. J.). Die Hauptfunktion der Applikation ist es, Echtzeitdaten über laufende Feuerwehreinsätze in Niederösterreich anzuzeigen (Gassner et al., 2016).

Vor der Veröffentlichung von Grisu NÖ gab es für die Anzeige von laufenden Einsätzen bereits eine Webapp namens WASTL, was für „Warn- und Alarmstufenliste“ steht. Diese Applikation wurde in Zusammenarbeit mit dem NÖ Landesfeuerwehrverband, der NÖ Landeregierung, dem Bundrechenzentrum, dem Bezirksfeuerwehrkommandos Krems, dem Magistrat der Stadt Krems sowie der Freiwilligen Feuerwehr Krems entwickelt (*NÖ Landesfeuerwehrverband - WASTL / GRISU - System*, o. J.).

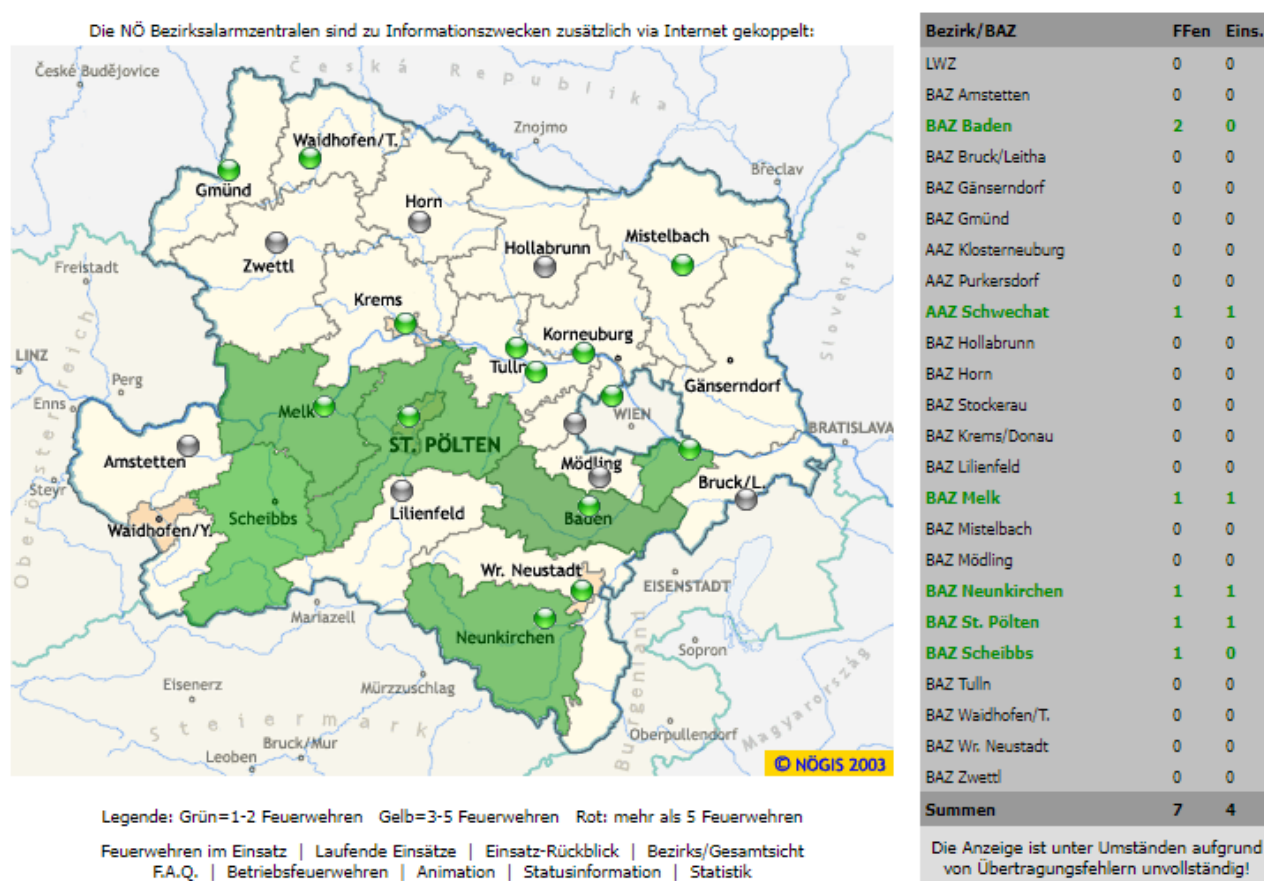


Abbildung 13: Screenshots aus der für einen PC-Webbrowser optimierten WASTL-App  
Quelle: <https://www.feuerwehr-krems.at/CodePages/Wastl/wastlmain/ShowOverview.asp>

Da die WASTL-App zunächst für die Darstellung auf PC-Webbrowsern optimiert wurde, war das Ziel der GrisuNÖ-Webapplikation, die Inhalte der WASTL-App optimiert auch für mobile Endgeräte zur Verfügung zu stellen. Aus diesem Grund wurde GrisuNÖ mittels des Hybrid-Webframeworks Ionic erstellt (Gassner et al., 2016). Das Ionic Framework bietet zusätzlich zur Möglichkeit der

Veröffentlichung von Apps im Web auch die Veröffentlichung im Android PlayStore und im iOS AppStore als hybride App an. Dabei wird die Webapp mittels Einbettung eines WebView-Containers in eine native Android beziehungsweise iOS App, angezeigt. Zudem bietet Ionic für mobile Endgeräte optimierte Komponenten an (*Introduction to Ionic | Ionic Documentation*, o. J.).

Die GrisuNÖ WebApp teilt sich in vier Hauptmenüpunkte: Übersicht, Bezirke, Statistik und Wasser. In dieser Analyse werden nur drei Menüpunkte betrachtet, da der Menüpunkt „Wasser“ nichts mit der Hauptfunktionalität der App, dem Anzeigen von Echtzeitdaten über laufende Einsätze in NÖ, zu tun hat, sondern eigens für die NÖ Feuerwehren geschaffen wurde, um Wasserentnahmestellen in der Nähe anzuzeigen.

Der Einstiegspunkt der Webapp ist die Übersicht. Auf dieser Seite werden:

- die Summe der aktuellen Einsätze in Niederösterreich,
- die Summe der aktuell ausgerückten Feuerwehren in Niederösterreich,
- die Summe der aktiven Bezirke in Niederösterreich und
- eine interaktive Karte von Niederösterreich die den Warnstatus einer Bereichs-, Bezirks- oder Abschnittsalarmzentrale mittels verschiedener Farben anzeigt, dargestellt.

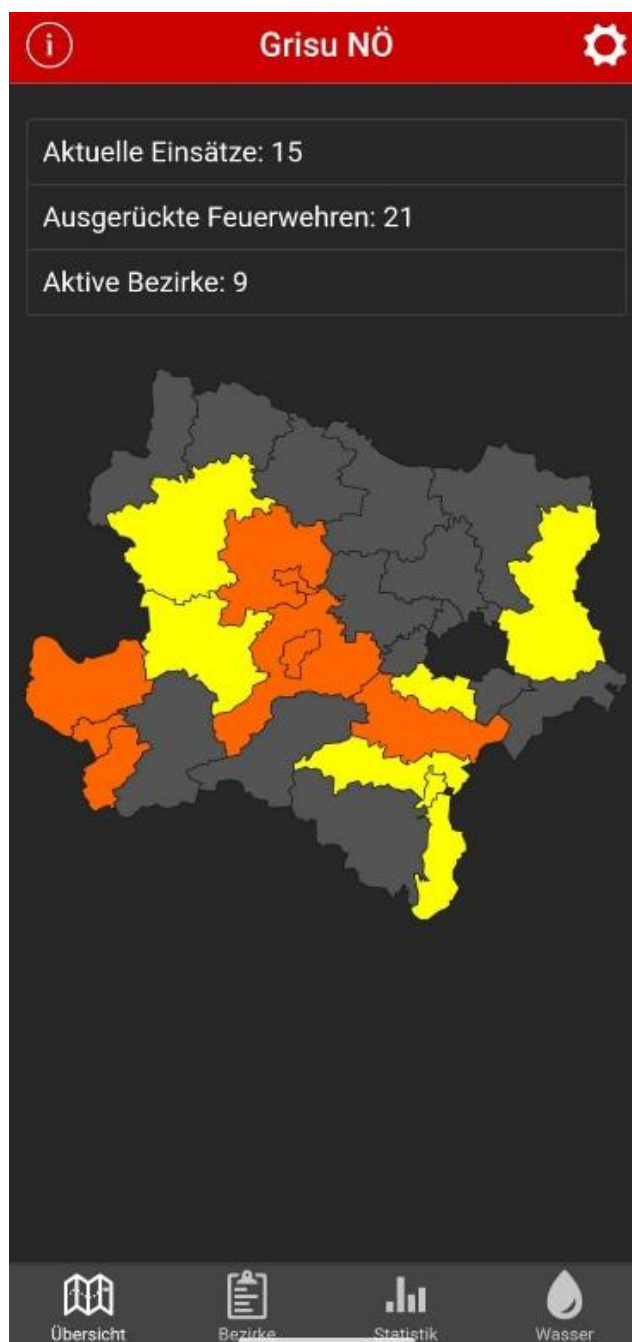


Abbildung 14: Übersichtsseite Einstiegsseite Grisu NÖ-Webapp mit Einsatzkarte und der aktuellen Einsatzstatistik  
Quelle: Grisu NÖ App

In der App gibt es vier verschiedene Warnstufen: Grau steht für keine Warnung, Gelb für eine niedrige Warnstufe (1-2 Feuerwehren ausgerückt), Orange für eine mittlere Warnstufe (3-5 Feuerwehren ausgerückt) und Rot für eine hohe Warnstufe (mehr als 5 Feuerwehren ausgerückt) in der jeweiligen Alarmzentrale (*FF Krems | Warn- und Alarmstufenliste*, o. J.).

Um eine Zuordnung zu erleichtern, wurden im Anhang unter Punkt 9.1 alle politischen Bezirke und ihre Zugehörigkeiten zu den jeweiligen Alarmzentralen aufgelistet. Mithilfe der in Abbildung 14 dargestellten Karte kann mittels Klick auf einen Bezirk auf der Karte zur Einsatzübersicht der Alarmzentrale, wie in Abbildung 15 dargestellt, navigiert werden

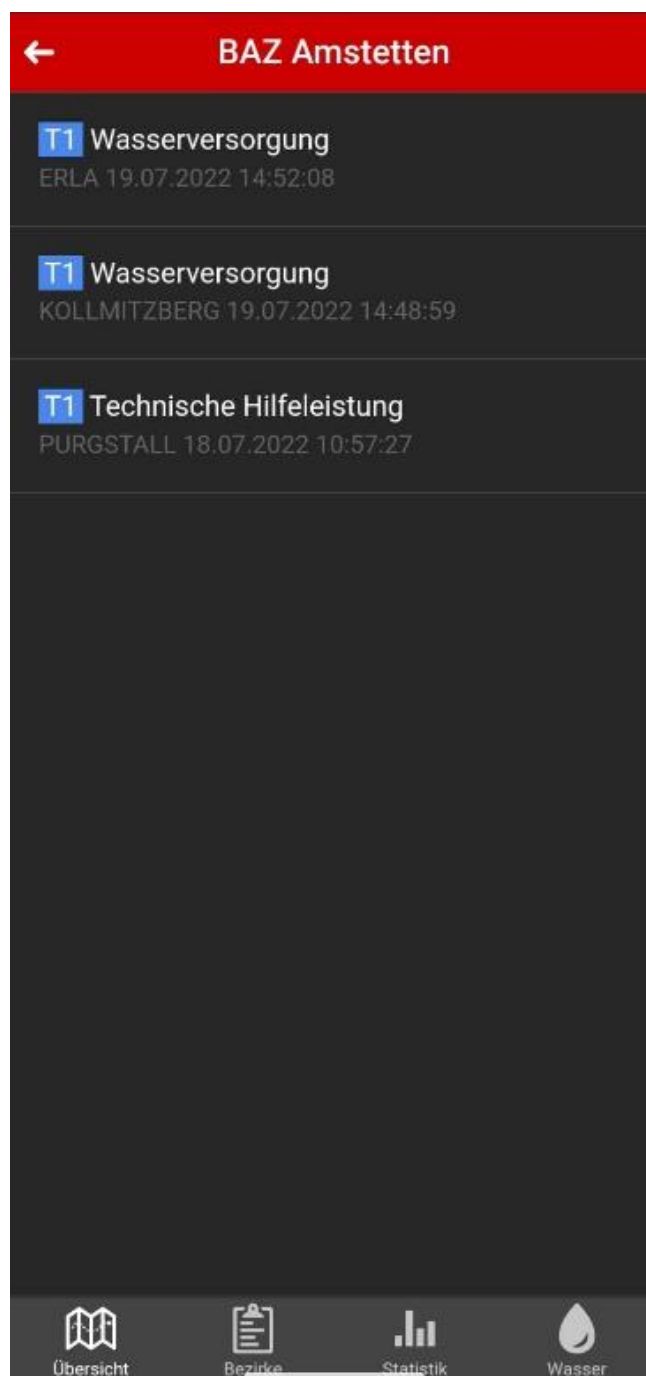


Abbildung 15: Auflistung der Einsätze der  
Bereichsalarmszentrale Amstetten

Quelle: Grisu NÖ App

Wie in Abbildung 15 ersichtlich ist, wird zum jeweiligen Einsatz, in der Auflistung einer Alarmzentrale:

- die Alarmstufe,
- der Einsatztext,
- der Einsatzort und
- die Einsatzzeit angezeigt.



Durch Klick auf einen der Einsätze kann zu der jeweilige Detailseite des Einsatzes navigiert werden, wie in Abbildung 16 dargestellt.

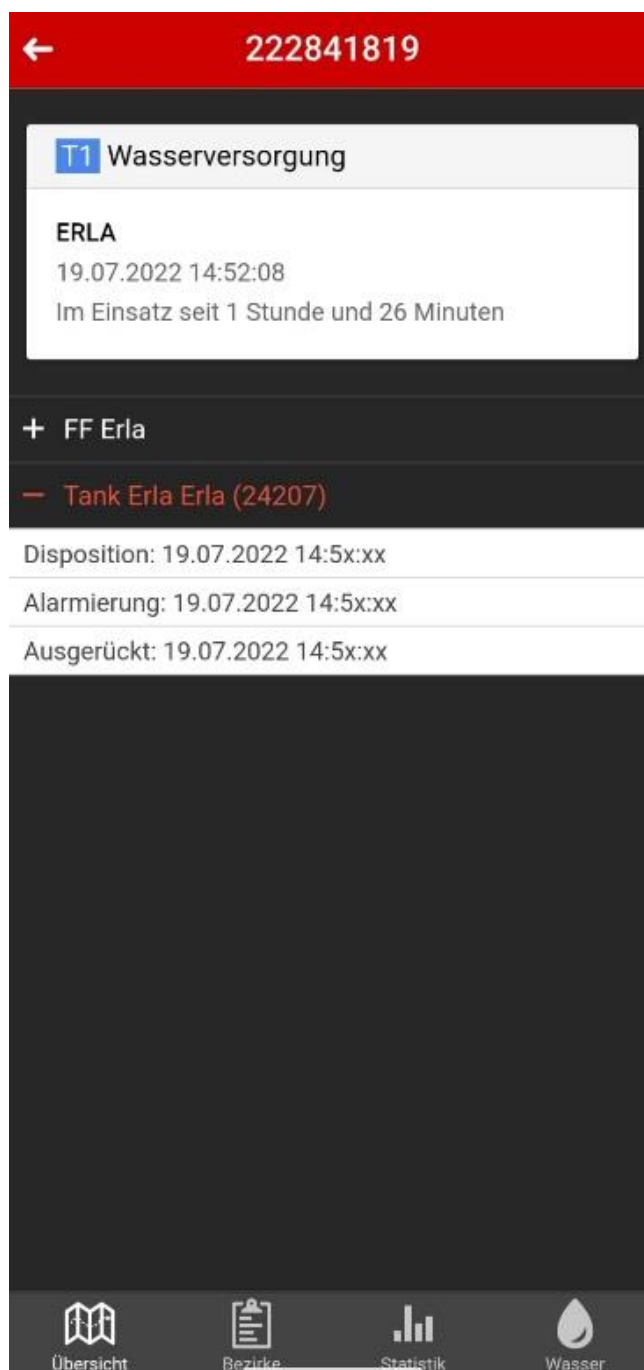


Abbildung 16: Detailseite eines Einsatzes

Quelle: Grisu NÖ App

Auf der Detailseite in Abbildung 16 werden zusätzlich zu den bereits in der Einsatzauflistung angezeigten Informationen die ausgerückte/n Feuerwehr/en mit Dispositions-, Alarmierungs-, Ausrücke- und Einrückzeit angezeigt.

Im Gegensatz zur Karte auf der Übersichtsseite, zeigt der nächste Menüpunkt namens „Bezirke“ die Bereichs-, Bezirks- und Abschnittsalarmszentralen in Listenform an. Der nächste Menüpunkt heißt „Bezirke“ und zeigt die Bereichs-, Bezirks- und Abschnittsalarmszentralen im Gegensatz zur Karte auf der Übersichtsseite in Listenform an.

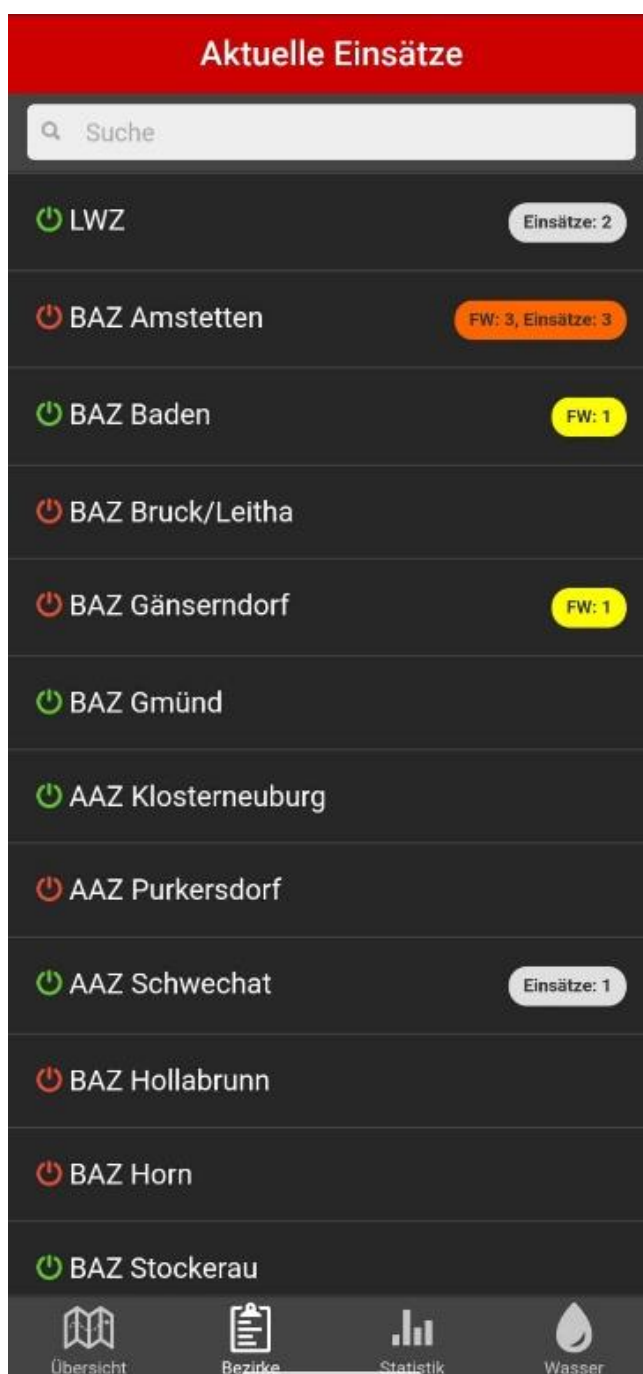


Abbildung 17: Auflistung aller Bereichs-, Bezirks- und Abschnittsalarmszentralen

Quelle: Grisu NÖ App

Wie in Abbildung 17 gezeigt, ist die Liste der Alarmzentralen durchsuchbar. Zudem wird durch ein rotes, für inaktiv stehendes Symbol oder grünes, für aktiv stehendes Symbol am Zeilenanfang der Status der Alarmzentrale visualisiert. Weiters wird in der jeweiligen Zeile mit farblich markierten Badges (farbliche Markierung der Warnstufen) die Anzahl der laufenden Einsätze beziehungsweise der ausgerückten Feuerwehren angezeigt. Zusätzlich zu den Alarmzentralen wird in dieser Ansicht die Landeswarnzentrale kurz LWZ angezeigt. Mittels Klicks auf den Eintrag einer der Alarmzentralen beziehungsweise der LWZ, wird der gleiche Dialog wie in Abbildung 15 mit der jeweiligen Alarmzentrale beziehungsweise mit der LWZ geöffnet.

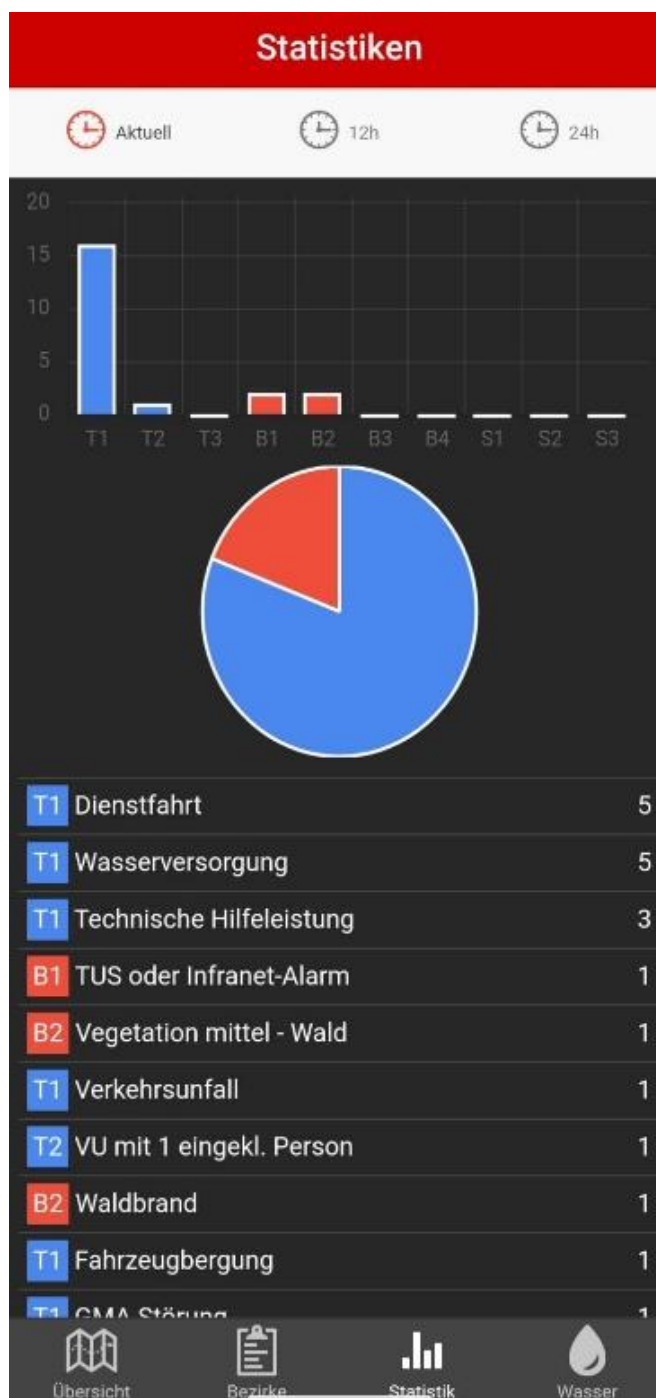


Abbildung 18: Seite mit Einsatzstatistiken mit zwei Diagrammen zu Darstellung des Einsatztyps

Quelle: Grisu NÖ App

Der letzte Menüpunkt mit dem Namen „Statistik“ zeigt die aktuelle Einsatzstatistik, die der letzten 12 Stunden und die der letzten 24 Stunden an. Die Daten werden nach Einsatzart als Balkendiagramm und als Tortendiagramm dargestellt. Zusätzlich findet sich eine Liste der Einsatz Tätigkeiten mit der Anzahl der jeweiligen Durchführungen sowie eine Summe der Aktivitäten am Ende der Seite. Eine Übersicht über die Einsatzarten ist im Anhang unter Punkt 9.2 zu finden.

Im nächsten Schritt wird die Architektur und die Implementierung der GrisuNÖ App analysiert. Die GrisuNÖ App wurde mithilfe des Ionic Frameworks und des JavaScript Frameworks AngularJS erstellt. AngularJS ist ein clientseitiges JavaScript Framework, das zur Erstellung von Single Page Applikationen verwendet werden kann (*AngularJS — Superheroic JavaScript MVW Framework*, o. J.). In AngularJS wird bei der Erstellung einer App oft das Model-View-Controller-Entwurfsmuster verwendet, wie auch im Fall der GrisuNÖ-Webapp.

In der Abbildung 19 werden die relevanten Architektureile der Grisu NÖ-Webapp, aufgeteilt in die jeweilige Kategorie des MVC-Musters, dargestellt. Die im Abschnitt „View“ aufgelisteten Templates sind HTML Dokumente, die die Struktur der einzelnen Seite der App darstellen. Wobei die drei Hauptseiten (OverviewTemplate, DistrictsTemplate und StatisticsTemplate) mit der Navigationsleiste, die im TabsTemplate definiert wird, global in der App aufrufbar sind. Weiters sind die möglichen Navigationswege zwischen den Seiten mittels strichlierter Pfeile in Abbildung 19 dargestellt.

Zu jeder HTML-Seite der App gibt es einen Controller. Dies ist eine JavaScript-Datei, in der sich ein Teil der Anwendungslogik befindet, wobei in den Controllern hauptsächlich auf *Events* wie zum Beispiel die Aktualisierung einer Seite oder die Navigation zu einer anderen Seite reagiert wird. Aber auch ein Teil der Daten wird in den Controllern gehalten, wie zum Beispiel im IncidentsListController ein Array mit den IDs der Einsätze (extendedIncidentIds).

Die eigentliche Anwendungslogik der Webapp ist im Factory DataService zu finden. Hier werden die Daten mittels HTTP-Request abgefragt, verarbeitet und dann an die Controller weitergereicht. Der zusätzlich in Abbildung 19 eingezeichnete UtilService enthält einige Funktionen, die von den Controllern verwendet werden, um zum Beispiel ein PopUp mit einer Fehlernachricht anzuzeigen.

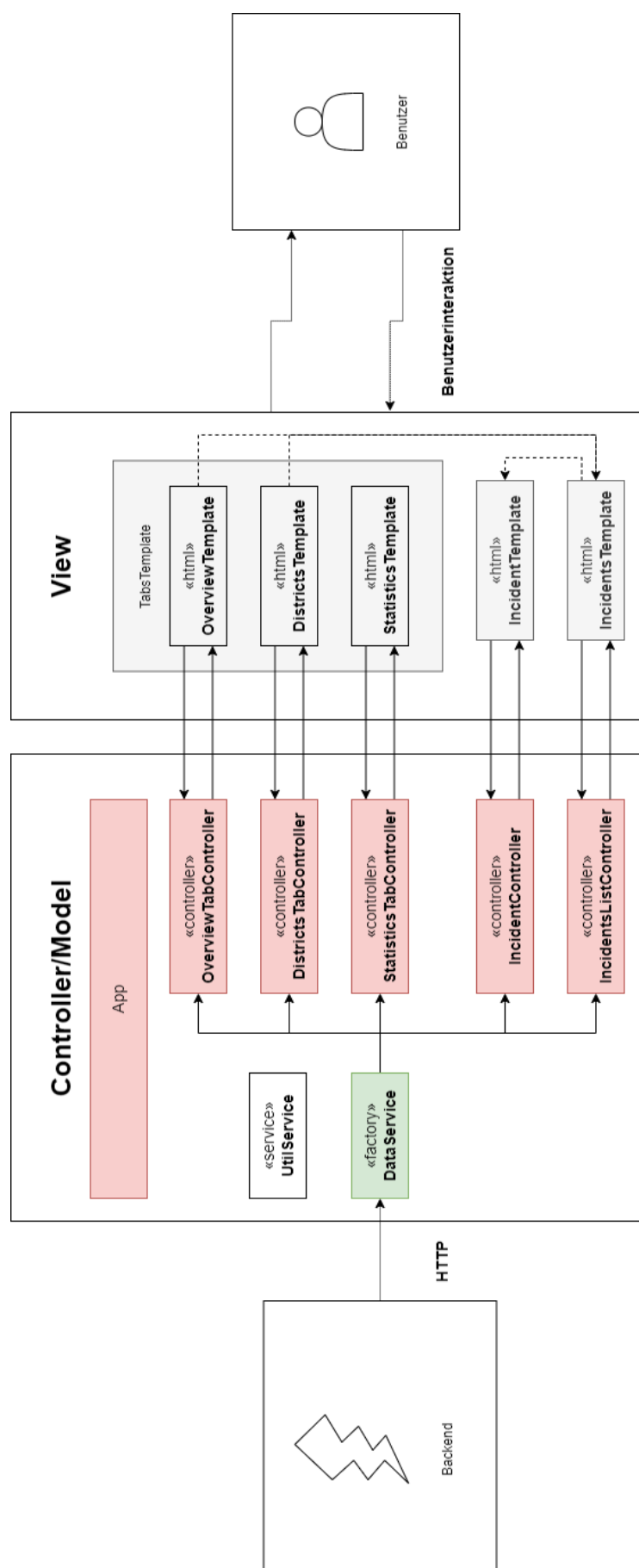


Abbildung 19: Architektur der Grisu NÖ-App

In der Factory DataService werden folgende relevante Daten abgefragt:

Name	Parameter	Beschreibung
<b>MainData</b>	-	Diese Abfrage liefert zu jedem Bezirk in NÖ die Anzahl der gerade aktiven Einsätze, die Warnstufe und die Anzahl der ausgerückten Feuerwehren. Außerdem werden statistische Werte, die gerade aktuell sind, die der letzten 12 Stunden und die der letzten 24 Stunden mitgeliefert, wobei jeweils die Summe der Einsätze und die Anzahl der Einsätze nach Alarmstufe und Einsatztext geliefert werden.
<b>ActiveIncidents</b>	districtId	In dieser Abfrage muss als Parameter die BAZ bzw. AAZ Nummer als Parameter übergeben werden. Zu dieser Bereichs-, Bezirks- beziehungsweise Abschnittsalarmzentrale wird dann eine Liste mit allen laufenden Einsätzen zurückgegeben. Diese Liste enthält zum jeweiligen Einsatz den Einsatztext, die Alarmstufe, den Einsatzort, die Zeit, das Datum und die EinsatzID.
<b>IncidentData</b>	incidentId	Über diese Abfrage können über die in der ActiveIncidents-Abfrage erhaltenen EinsatzID zusätzliche Informationen über einen Einsatz wie zum Beispiel, welche Feuerwehren wann ausgerückt sind, und noch weitere abgefragt werden.
<b>BAZInfo</b>	-	In dieser Abfrage wird der Status jeder BAZ bzw. AAZ abgefragt, wobei dieser Status „aktiv“ oder „inaktiv“ sein kann.

Tabelle 2: Alle relevanten Daten die im DataService von der HTTP-Schnittstelle abgefragt werden

Folgende Schwachstellen wurden bei der Analyse der Applikation festgestellt:

- Der Support für das Framework AngularJS wurde seit Jänner 2022 eingestellt und wird somit nicht weiter gepflegt (*AngularJS — Superheroic JavaScript MVW Framework*, o. J.).
- Zudem wird die App seit 2016 nicht mehr gewartet und auch der Build-Button auf der Github-Seite zeigt einen Error-Status an (Gassner et al., 2016).
- Weiters wurden bei der mobilen Auslieferung der App im Google Playstore und im Apple Appstore, in den Bewertungen, auf Kompatibilitätsprobleme und die geringe Geschwindigkeit der App hingewiesen (*Grisu NÖ (Feuerwehr - WASTL)*, o. J.; *Grisu NÖ (Feuerwehr - WASTL) – Apps bei Google Play*, o. J.).
- Keine automatische Datenaktualisierung. Die Seite muss immer wieder neugeladen werden, um den aktuellen Status anzeigen zu können.

### 4.3. Umsetzung der neuen Anwendung

Aus den am Ende von Kapitel 4.2 aufgezählten Schwachstellen (kein Support, keine Wartung, Geschwindigkeits- und Kompatibilitätsprobleme sowie keine automatische Datenaktualisierung) sollen die Anforderungen für die Neuimplementierung der Webapp in der Fallstudie ermittelt werden.

Bei der Umsetzung der neuen Applikation sollen folgende Anforderungen erfüllt werden:

- Reaktives Framework/Programmiersprache verwenden.
- Implementierung mit aktuellem Framework/Sprache (laufender Support)
- Die erstellte Applikation soll als statische Webanwendung ausgeliefert und rein browserseitig ausgeführt werden können.
- Erstellung einer hybriden App, um auch die Veröffentlichung im AppStore bzw. PlayStore zu ermöglichen.
- Automatische Aktualisierung der Einsatzdaten, um den neusten Stand der Informationen wiederzugeben.

Für die Neuumsetzung der App gibt es sehr viele mögliche Sprachen beziehungsweise Frameworks, die verwendet werden könnten. Um diese Auswahl einzuschränken, werden hier nur die Frameworks und Programmiersprachen, die in Kapitel 3 *Ansätze* beschrieben wurden, behandelt. In der folgenden Tabelle werden alle Frameworks und Programmiersprachen aus Kapitel 3, aufgezählt und ausgewertet, ob sie die Anforderungen erfüllen. Dabei ist zu beachten, dass die letzte Anforderung keine direkte Anforderung an das Framework/die Sprache ist, da diese eine programmatische Aufgabenstellung ist, darum ist die Anforderung in der Tabelle nicht verzeichnet. Auch auf die Darstellung der Anforderung „Reaktives Framework/Programmiersprache verwenden“ wurde verzichtet, da in Kapitel 3 nur reaktive Programmiersprachen beziehungsweise Frameworks beschrieben wurden.

Framework/Sprache	Laufender Support	Statische Auslieferung	Mobile Anwendung
<b>Fran<sup>1</sup></b>	-	-	-
<b>Yampa<sup>2</sup></b>	✓	-	-
<b>Elm<sup>3</sup></b>	✓	✓	-
<b>Flapjax<sup>4</sup></b>	-	✓	~
<b>Sodium<sup>5</sup></b>	✓	✓ <sup>6</sup>	~
<b>ReactiveX<sup>7</sup></b>	✓	✓ <sup>8</sup>	~
<b>Project Reactor<sup>9</sup></b>	✓	-	-
<b>Svelte<sup>10</sup></b>	✓	✓	✓

Tabelle 3: Erfüllung der Anforderungen (horizontale Achse) durch vorhandene Frameworks (Vertikale Achse)

Nach der in Tabelle 2 dargestellten Frameworks beziehungsweise Sprachen, scheiden Fran, Yampa und Project Reactor aus, da keine statische Auslieferung und keine mobile Veröffentlichung möglich sind. Flapjax wird ausgeschlossen, da es keinen laufenden Support für die Sprache mehr gibt. Bei Elm stellt nur die mobile Veröffentlichung ein Problem dar, da diese nur durch die manuelle Einbettung der Webapplikation in einen nativen WebView in Android und iOS möglich ist, das aber nicht empfohlen wird (*Can We Create Mobile Apps in Elm?*, 2020). Aus diesem Grund wird auch Elm nicht zur Implementierung der neuen Webapp verwendet. Bei den Frameworks Sodium TypeScript/JavaScript und RxJS (ReactiveX) gibt es die Möglichkeit, das Ionic-Framework oder ein anderes hybrides Framework einzubinden, um eine mobile Anwendung zu erstellen. Svelte bietet ein eigenes hybrides Framework an – Svelte Native.

Somit stehen in der engeren Auswahl Svelte, RxJS und Sodium TypeScript/JavaScript. Da die Dokumentation und der Community-Support von Sodium mangelhaft ist wurde diese Bibliothek ausgeschieden (*Tutorials and Docs?*, o. J.). RxJS ist die am weitesten verbreitete Bibliothek (*Most*, o. J.; *Rxjs*, o. J.; *Svelte*, o. J.). Außerdem wird RxJS von Angular unterstützt. Angular ist die nachfolgende Version von AngularJS, das bei der Grisu NÖ App verwendet wurde. Die Verwendung von Angular würde Vorteile durch die bessere Vergleichbarkeit der Struktur der Applikation bringen und es besteht bereits Know-How beim Entwickeln mit Angular und RxJS. Zudem bietet es die Möglichkeit, auch das User-Interface durch die Verwendung des hybriden Ionic Frameworks ähnlich zu gestalten. Ionic CLI unterstützt die Generierung eines Angular-Projektes mit Ionic als hybrides UI-Framework und Integration von RxJS.

<sup>1</sup> *News on Fran Releases*, o. J.

<sup>2</sup> *Yampa*, o. J.

<sup>3</sup> *Can We Create Mobile Apps in Elm?*, 2020; Czaplicki, 2012

<sup>4</sup> Meyerovich et al., 2009

<sup>5</sup> *SodiumFRP/sodium: Sodium - Functional Reactive Programming*

<sup>6</sup> Statische Auslieferung nur bei Sodium TypeScript/JavaScript möglich (*FRP Library for multiple languages*, o. J.)

<sup>7</sup> *ReactiveX*, o. J.

<sup>8</sup> Statische Auslieferung nur in RxJS, RxDart und RxPY (ReactiveX - Languages, o. J.)

<sup>9</sup> *Project Reactor*, o. J.

<sup>10</sup> *Svelte • Cybernetically Enhanced Web Apps*, o. J.; *Svelte Native • The Svelte Mobile Development Experience*, o. J.



Deshalb wurde zur Erstellung der neuen, reaktiven Webapp Angular mit RxJS und Ionic gewählt.

Bei der Umsetzung des User-Interfaces wurde darauf geachtet, die gleiche Struktur wie bei der Grisu NÖ-App umzusetzen, um den Umstieg auf die neue Applikation zu erleichtern. In der folgenden Abbildung werden die Seiten der Applikation dargestellt.

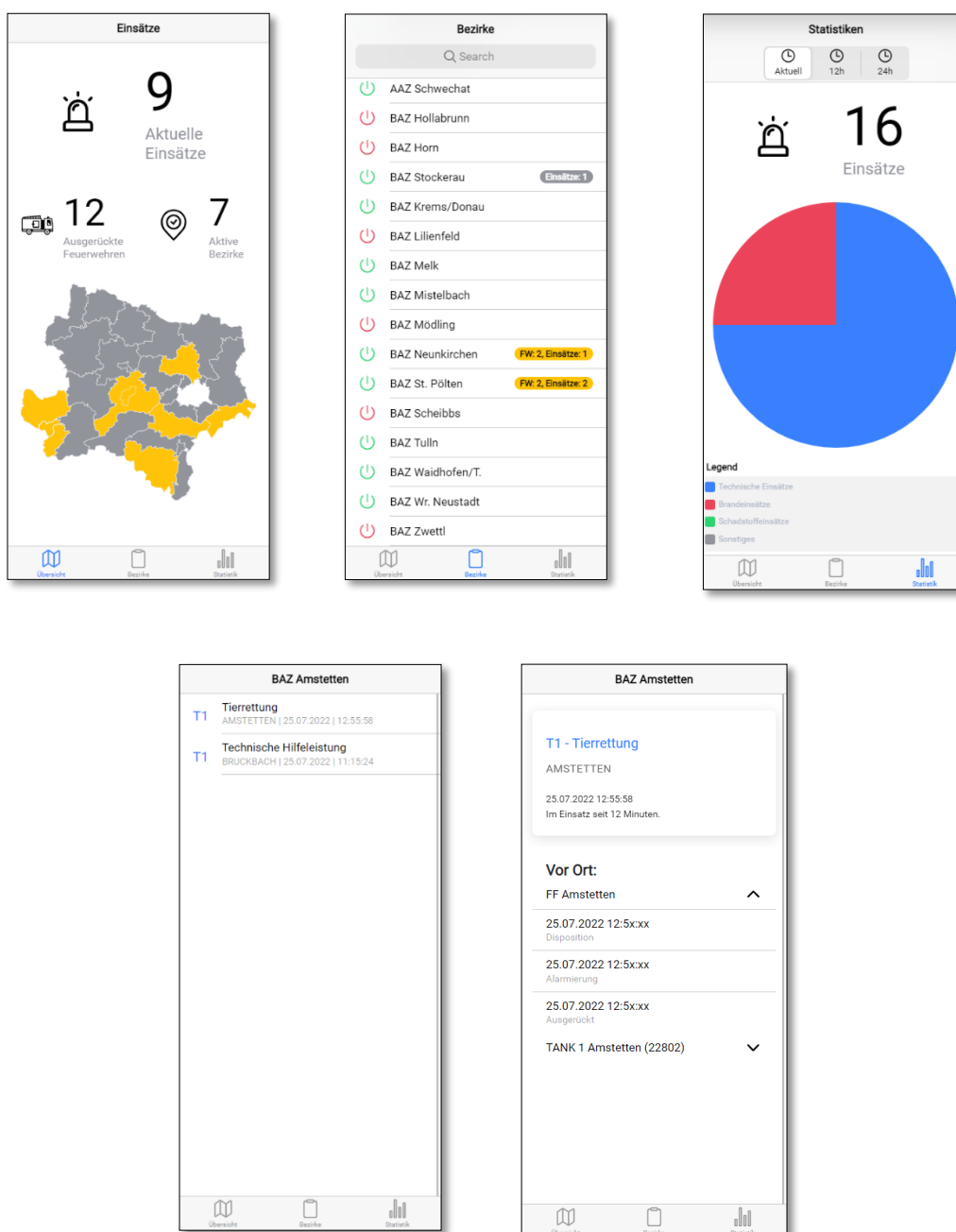


Abbildung 20: UI der Neuentwicklung

Angular arbeitet, im Gegensatz zu AngularJS (JavaScript), mit der objektorientierten Programmiersprache TypeScript (*Angular - Ahead-of-time (AOT) compilation*, o. J.). In Angular sind durch die Verwendung von TypeScript die einzelnen Komponenten der Applikation in Klassen unterteilt. Das durch die Visual Studio Code- Erweiterung classdiagram-ts automatisch generierte Klassendiagramm der Applikation ist im Anhang unter 9.3 zu finden. In der folgenden Abbildung 21 wird die Architektur der neuentwickelten Angular Anwendung dargestellt.

In Abbildung 21 wird die Architektur der neuen Anwendung, die dem MVC-Muster folgt, dargestellt. Im Bereich „View“ sind die HTML-Templates zu finden, welche die Seiten der Applikation darstellen. Da TypeScript eine stark typisierte Programmiersprache ist, wurde zu jeder Abfrage, die in Tabelle 1 dargestellt ist, ein Modell mittels Interface erstellt.

Zudem wurde bei der Entwicklung der neuen Anwendung die Bibliothek NGXS verwendet. NGXS ist ein State Management Pattern für Angular Anwendungen und ermöglicht es, den Zustand einer Anwendung zentral zu verwalten. Die aktuell angezeigten Daten und der Status der Anwendung werden im sogenannten State (in Abbildung 21 FireState) gespeichert. Dieser State stellt die Möglichkeit zur Verfügung, Daten, die im State gespeichert sind, über Actions zu beeinflussen und mittels Selektoren abzufragen. Das zentrale State Management bietet vorrangig den Vorteil eines immer vorhersehbaren und konsistenten Datenzustandes. Darüber hinaus ist der State auch der zentrale Ort für die Geschäftslogik der Applikation, den Actions. Im Falle der Neuentwicklung wäre zum Beispiel LoadMainData eine Action, die den DataService aufruft und der einen HTTP-Abfrage stellt. NGXS unterstützt RxJS und bietet so eine reaktive Abfrage der Daten im State über Observables. So werden Veränderungen im State über die Observables an die Components beziehungsweise Services weitergeben.

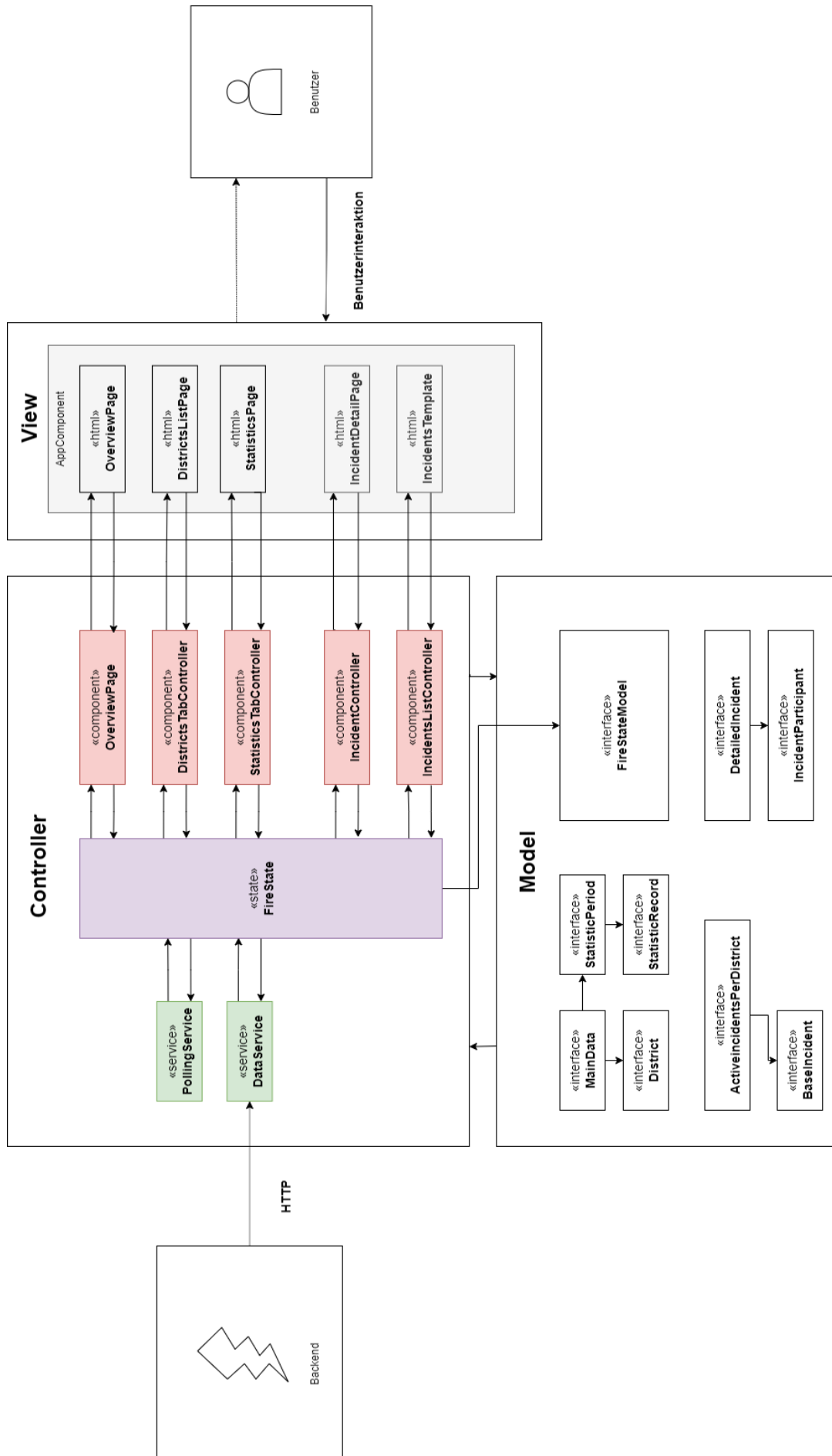
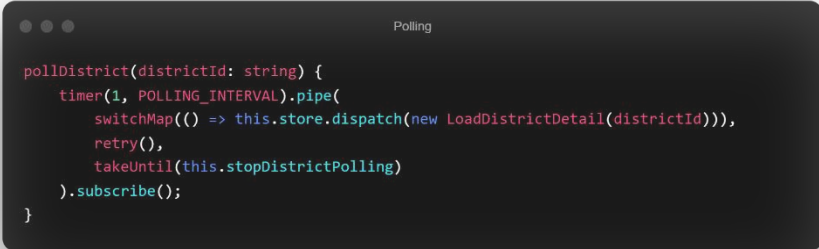


Abbildung 21: Architektur der Neuentwicklung

Überdies wurde um die Anforderung „Automatische Aktualisierung der Einsatzdaten, um den neusten Stand der Informationen wiederzugeben“ zu erfüllen, ein PollingService erstellt. Der PollingService ermöglicht es, die Daten nach einem festgelegten Zeitintervall erneut abzufragen, um die Aktualität der Daten zu erhalten. In Abbildung 23 wird ein Polling-Befehl abgesetzt.



```

Polling

pollDistrict(districtId: string) {
  timer(1, POLLING_INTERVAL).pipe(
    switchMap(() => this.store.dispatch(new LoadDistrictDetail(districtId))),
    retry(),
    takeUntil(this.stopDistrictPolling)
  ).subscribe();
}

```

Abbildung 22: Methode des Polling-Service

Die Methode pollDistrict löst zu Beginn des Aufrufs einen timer aus, der die im Folgenden beschriebenen Operatoren, im Intervall des Konstante POLLING\_INTERVAL, aufruft. In der dritten Zeile wird mittels des Operators switchMap die Action LoadDistrictDetail aufgerufen, die einen HTTP-Abfrage auslöst. Der Operator retry wiederholt den Aufruf, falls ein Fehler bei der Abfrage auftreten sollte, so lange, bis diese erfolgreich abgeschlossen wurde. Der Operator takeUntil stoppt den timer und bricht das Polling ab. Das passiert über eine eigene Methode stopPolling. In der vorletzten Zeile wird subscribed, um das Observable auszulösen.

Abschließend kann gesagt werden, dass alle Anforderungen and die neue Anwendung erfüllt wurden:

- ein reaktives Framework/Programmiersprache verwendet → RxJS
- Implementierung mit aktuellem Framework/Sprache (laufender Support) → Angular/RxJS
- Die erstellte Applikation soll als statische Webanwendung ausgeliefert und rein browserseitig ausgeführt werden können. → Angular
- Erstellung einer hybriden App, um auch die Veröffentlichung im AppStore bzw. PlayStore zu ermöglichen. → Ionic
- Automatische Aktualisierung der Einsatzdaten, um den neusten Stand der Informationen wiederzugeben. → Implantiertes Polling Service

## 4.4. Vergleich der Anwendungen

In diesem Abschnitt werden die Grisu NÖ App und die Neuentwicklung, aufbauend auf den in Kapitel 2.6 durch die Literatur erhobenen Vor- und Nachteilen reaktiver Programmierung, miteinander verglichen. Um festzustellen welche Vor- und Nachteile sich, in Hinsicht auf die reaktive Programmierung, auch bei dieser Anwendung identifizieren lassen, werden verschiedene Erhebungsmethoden verwendet.

Zur Messung der Codekomplexität wurde wie bei Holst & Gillberg die zyklomatische Komplexität nach McCabe verwendet (2020, S. 8). Wobei die zyklomatische Komplexität nach McCabe wie folgt kalkuliert wird:

$$\text{Cyclomatic complexity} = E - N + 2 * P$$

In dieser Formel steht E für die Anzahl der Kanten im Flussdiagramm, N für die Anzahl der Knoten im Flussdiagramm und P für die Anzahl der Knoten mit einem Austrittspunkt (T. J. McCabe, 1976).

Wie bei Holst & Gillberg wurde die Evaluierung bei jeder Methode einzeln durchgeführt und aus diesen Ergebnissen der durchschnittliche Wert der beiden Anwendungen berechnet. Es wurden nur Methoden miteinbezogen, die in beiden Anwendungen vorhanden sind.

Bei der Auswertung der zyklomatischen Codekomplexität wurde die Visual Studio Code Erweiterung CodeMetrics verwendet, die es erlaubt, die Komplexität von Methoden bzw. Klassen in JavaScript und TypeScript Files zu messen. (*CodeMetrics - Visual Studio Marketplace*, o. J.).

Die Messdaten der zyklomatischen Komplexität sind im Anhang unter 9.4 in einer Tabelle ersichtlich.

Folgendes Ergebnis wurde ermittelt:

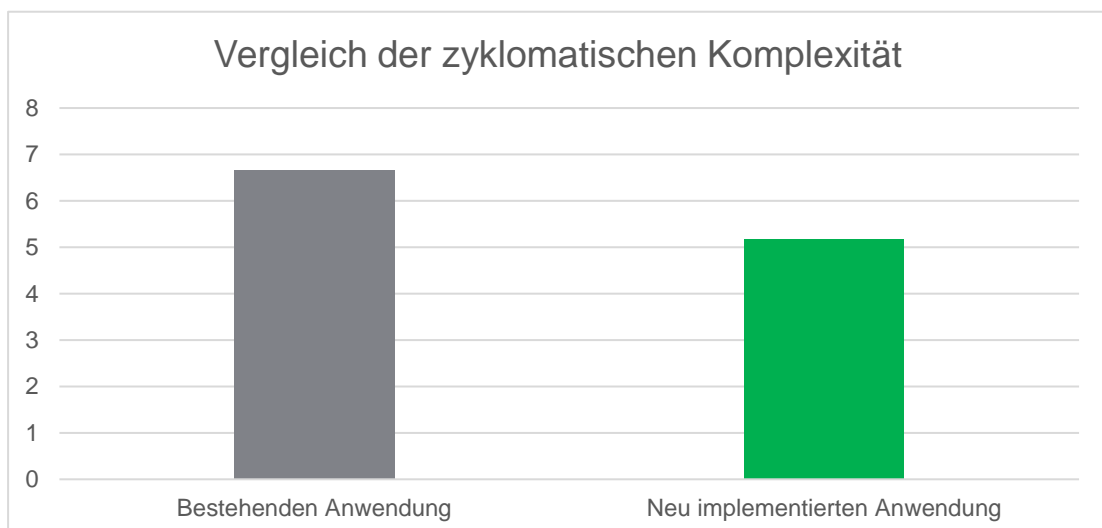


Abbildung 23: Diagramm zyklomatische Komplexität

Wie aus der Abbildung 23 entnommen werden kann, ist die zyklomatische Komplexität der neuen reaktiven Anwendung geringer als die der bestehenden Anwendung.

Für die Analyse der Codelesbarkeit sollte ursprünglich die gleichen Tools wie bei Holst & Gillberg verwendet werden (2020). Da jedoch beide Analysetools nur für Java zur Verfügung stehen und sich keine alternatives Vergleichstool finden ließ, wird die Codelesbarkeit in diesem Kapitel nicht weiter behandelt. Die Wartbarkeit des Codes wurde, wie bei der von Toczé et al. durchgeführten Studie, durch die Messung der Größe der Codebasis durchgeführt. Dazu wurden in dieser Analyse die Visual Studio Code Extension Lines of Code (LOC) verwendet, welche die Anzahl der Codezeilen aller Files eines geöffneten Projekts zählen (*Lines of Code (LOC) - Visual Studio Marketplace*, o. J.). Diese werden dann, nach ihrer Dateieindung sortiert, ausgegeben. Um das Ergebnis dieser Zählung nicht zu verfälschen, wurden alle nicht relevanten Dateien aus den Projekten gelöscht, beispielsweise die Teile der App, die nicht implementiert wurden. Die genauen Ergebnisse der LOC-Analyse sind im Anhang unter 9.5 zu finden. Bei beiden Projekten wurden nur die JS (alte Anwendung) beziehungsweise TS (neue Anwendung) Dateien gezählt, da diese den Anwendungscode enthalten.

Bei der Analyse wurden bei der bestehenden Anwendung die folgenden Zahlen ermittelt:

Zeilen in JavaScript-Dateien insgesamt	1835
davon leere Zeilen	-356
davon Kommentarzeilen	-54
<b>JavaScript-Codezeilen</b>	<b>1425</b>

Tabelle 4: Wartbarkeitsanalyse der bestehenden Anwendung

Bei der Analyse der neuen Anwendung wurde folgendes Ergebnis erzielt:

Zeilen in TypeScript -Dateien insgesamt	1105
davon leere Zeilen	-142
davon Kommentarzeilen	-161
<b>TypeScript-Codezeilen</b>	<b>802</b>

Tabelle 5: Wartbarkeitsanalyse der neu entwickelten Anwendung

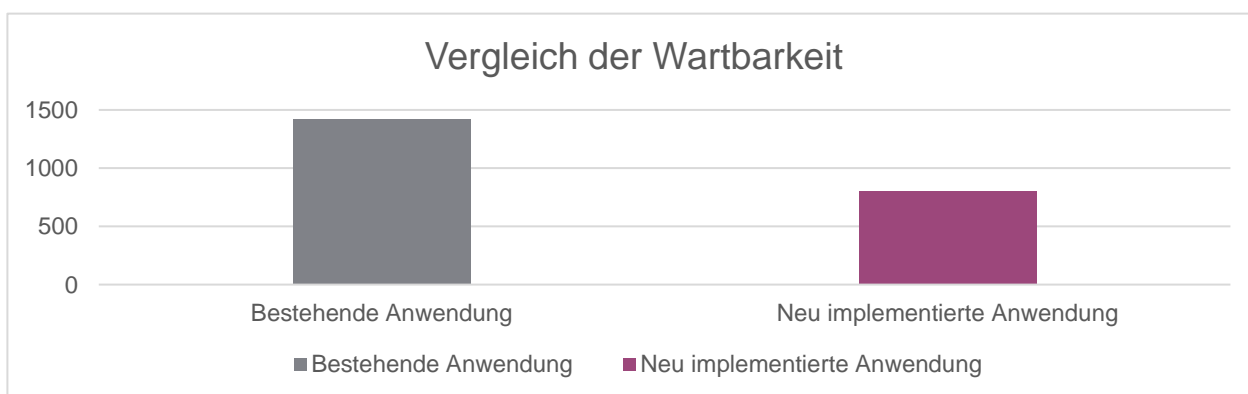
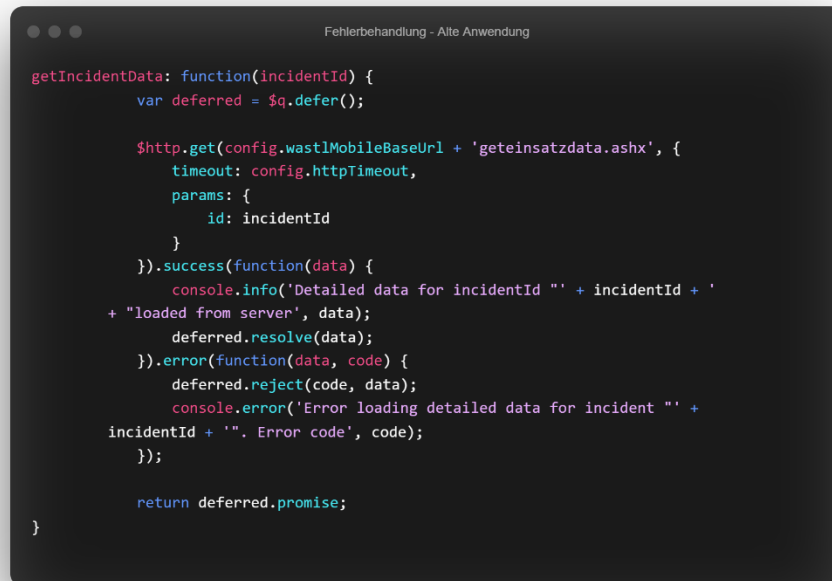


Abbildung 24: Diagramm Vergleich der Wartbarkeit über LOC

Im Vergleich wurden bei der bestehenden Anwendung um 602 Zeilen mehr Code geschrieben. Das bedeutet es wurden 1,77-mal mehr Codezeilen als bei der neu implementierten Anwendung geschrieben. Je größer die Zeilenanzahl ist, umso höher ist die Wahrscheinlichkeit, Fehler einzuführen.

Um die Fehlerbehandlung der Anwendungen zu vergleichen, wird der Code, der für die Fehlerbehandlung geschrieben wurde, bei beiden Anwendungen analysiert und gegenübergestellt.



```
Fehlerbehandlung - Alte Anwendung

getIncidentData: function(incidentId) {
    var deferred = $.defer();

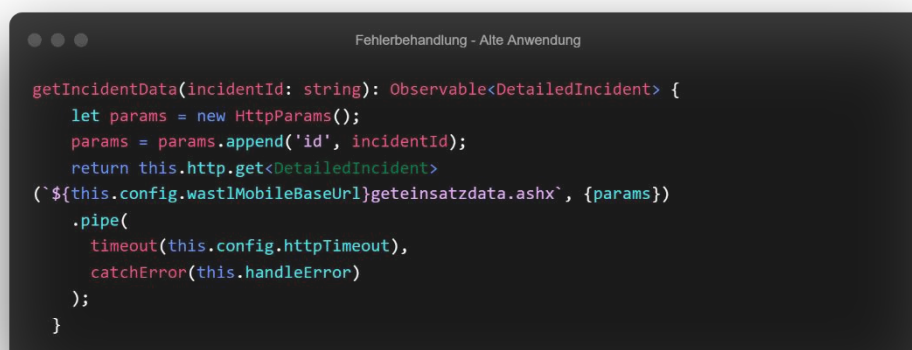
    $.http.get(config.wastlMobileBaseUrl + 'geteinsatzdata.ashx', {
        timeout: config.httpTimeout,
        params: {
            id: incidentId
        }
    }).success(function(data) {
        console.info('Detailed data for incidentId "' + incidentId + ' '
+ "loaded from server', data);
        deferred.resolve(data);
    }).error(function(data, code) {
        deferred.reject(code, data);
        console.error('Error loading detailed data for incident "' +
incidentId + '". Error code', code);
    });

    return deferred.promise;
}
```

Abbildung 25: Fehlerbehandlung - bestehende Anwendung

Bei der nicht reaktiven Anwendung werden bei dem in Abbildung 25 gezeigten HTTP-Request auftretende Fehler mit dem Callback-Error abgefangen. Dieser ruft eine anonyme Funktion auf, die ein abgeleitetes Promise ablehnt und einen Fehler in der Konsole ausgibt.

In der reaktiven Umsetzung wird mittels des Pipe-Operators der catchError-Operator hinzugefügt. Dieser löst die Methode handleError aus, wenn ein Fehler wie zum Beispiel ein Timeout auftreten sollte.



```
Fehlerbehandlung - Alte Anwendung

getIncidentData(incidentId: string): Observable<DetailedIncident> {
    let params = new HttpParams();
    params = params.append('id', incidentId);
    return this.http.get<DetailedIncident>
('`${this.config.wastlMobileBaseUrl}geteinsatzdata.ashx`, {params})
    .pipe(
        timeout(this.config.httpTimeout),
        catchError(this.handleError)
    );
}
```

Abbildung 26: Fehlerbehandlung - neu implementierte Anwendung

Der Aufbau der Fehlerbehandlung ist bei beiden Anwendungen sehr ähnlich. Aber RxJS bietet im Gegensatz zu der alten Implementierung an, den `retry`-Operator zu verwenden. Dieser versucht den HTTP-Request so lange zu wiederholen, bis dieser erfolgreich ist. Die Umsetzung eines Retries mit Callbacks wäre hier deutlich aufwändiger.

Das Debugging der beiden Applikationen wurde anhand der IDE Visual Studio Code verglichen. Bei der alten Anwendung war ein einfaches Browser Debugging über Visual Studio Code möglich. Dabei ist die IDE mit dem Browser verbunden und bietet die Möglichkeit, zu Breakpoints zu springen beziehungsweise diese zu setzen, die lokalen Variablen zu überwachen und die Call-Stacks mitzuüberwachen.

Auch bei der neuen Anwendung funktioniert diese Art des Debuggings bis zu einem gewissen Grad. Aber Operatoren und Observables können nicht debuggt werden beziehungsweise zeigen keine sinnvollen Werte an.

Um dieses Problem zu lösen, wurde die VS-Code Erweiterung „RxJS Debugging for Visual Studio Code“ verwendet. Mit dieser Erweiterung können bei den Operatoren Log-Punkte gesetzt werden. Bei diesen Logpunkten werden anschließend angezeigt, wann Subscribed, Unsubscribed oder ein Wert emittiert wird. Diese Erweiterung erleichtert das Debugging in RxJS. Trotzdem sind die Debugging-Optionen mit dem Standard-Debuggers von Visual Studio Code umfangreicher als die des RxJS Debuggers. Folgende Vor- und Nachteile ergeben sich aus der reaktiven Umsetzung der untersuchten Fallstudie:

Bezeichnung	Vorteil	Nachteil
Codekomplexität	x	
Codelesbarkeit		
Wartbarkeit	x	
Exaktere Fehlerbehandlung	x	
Debugging		x

Tabelle 6: Durch die Fallstudie ermittelte Vor- und Nachteile



## 4.5. Evaluierung und Ergebnis

In diesem Kapitel soll die Forschungsfrage beantwortet werden: Welche Vor- und Nachteile ergeben sich durch den Einsatz der *Konzepte* reaktiver Programmierung für Webanwendungen.

Dazu sollen die in Kapitel 4.4 erarbeiteten Ergebnisse, der Fallstudie, mit den Ergebnissen der Literaturrecherche verglichen und abgestimmt werden, um so festzustellen, ob es sich um einen Vorteil, einen Nachteil beziehungsweise eine Eigenschaft, die nicht eingeordnet werden kann, handelt.

In der folgenden Tabelle sind die Ergebnisse der Literaturrecherche dargestellt:

Bezeichnung	Vorteil	Nachteil
Codekomplexität	x	
Codelesbarkeit		x
Wartbarkeit	x	
Exaktere Fehlerbehandlung	x	
Debugging		x

Tabelle 7: Durch die Literaturrecherche ermittelte Vor- und Nachteile

In folgendem wird die Abstimmung, ob es sich um einen Vor- oder Nachteil handelt, durchgeführt:

### Codekomplexität

Wie in der Literatur in Kapitel 2.6 dargestellt, konnte bei der Messung der Codekomplexität der beiden Anwendungen eine geringere Komplexität bei der reaktiven Umsetzung festgestellt werden (Kapitel 4.4). Aus diesem Grund wird die geringere Codekomplexität als ein Vorteil der reaktiven Programmierung gesehen. Dies deckt sich mit den Recherchen aus der Literatur.

### Codelesbarkeit

Die Codelesbarkeit wurde in der Literatur bei der reaktiven Umsetzung als schlechter eingeordnet (Holst & Gillberg, 2020). Die praktische Lesbarkeits-Analyse konnte bei den zwei Anwendungen nicht durchgeführt werden. Aus diesem Grund wird die Codelesbarkeit im Endergebnis unten weder als Vorteil noch als Nachteil gewertet.

### Wartbarkeit

Wie bei der Codekomplexität ist die Wartbarkeit der reaktiven Anwendung in der Literatur (Kapitel 2.6) besser als die der klassischen Anwendung. Überdies ist die Wartbarkeit bei der praktischen Umsetzung der reaktiven Anwendung besser. Aus diesem Grund wird die bessere Wartbarkeit als Vorteil der reaktiven Programmierung gesehen und im Endergebnis so festgehalten.

### Exaktere Fehlerbehandlung

In der Literatur konnte bei der Verschachtelung von Fehlerbehandlungen mit reaktiven *Ansätzen* ein Vorteil gegenüber den imperativen *Ansätzen* gesehen werden (Dakowitz, 2018, S. 71). Aber da die Aufrufe in JavaScript mit Callback-Funktionen, im funktionalen Stil wie RxJS erfolgen, ist dieser Vorteil nicht gegeben. Der Nachteil kann jedoch mit Callback-Aufrufen ausgeglichen werden. Aus diesem Grund wird die exaktere Fehlerbehandlung nicht als Vor- oder Nachteil gewertet.

## Debugging

In der Literatur wird das Problem, dass die klassischen Debugger der IDEs die reaktive Programmierung noch nicht unterstützen, aufgefasst (Alabor & Stolze, 2020). Es müssen eigene Debugger installiert beziehungsweise verwendet werden. Dies wird auch in den beiden praktischen Anwendungen getestet, wobei ein reaktiver Debugger installiert wurde. Doch trotz der Installation bietet der reaktive Debugger noch nicht die Möglichkeiten eines klassischen Debuggers. Aus diesem Grund wird Debugging als Nachteil der reaktiven Programmierung gesehen.

In der folgenden Tabelle werden die Ergebnisse der Forschungsarbeit zusammengefasst dargestellt. Die Vor- und Nachteile, die sich beim Einsatz der reaktiven Programmierung ergeben.

Bezeichnung	Vorteil	Nachteil
Codekomplexität	x	
Codelesbarkeit		
Wartbarkeit	x	
Exaktere Fehlerbehandlung		
Debugging		x

Abbildung 27: Ermittelte Vor- und Nachteile von reaktiver Programmierung

## 5. Zusammenfassung und Ausblick

Das zusammengefasste Ergebnis dieser Arbeit ist, dass reaktive Programmierung Vorteile gegenüber konventionellen Programmierparadigmen hat und insbesondere bei größeren Anwendungen, bei denen die Wartbarkeit und Codekomplexität eine große Rolle spielen, als Mittel der Wahl in Betracht gezogen werden sollte. Dies leitet sich hauptsächlich aus der durchgeführten Fallstudie ab, spiegelt sich aber auch in der recherchierten Literatur wider.

Genannte Vor- und Nachteile leiten sich, bei näherer Betrachtung, entweder von der Anwendung der Konzepte reaktiver Programmierung oder dem Design der neuen Anwendung ab. Das Debugging und die Codekomplexität, leiten sich eher aus den Konzepten der reaktiven Programmierung ab. Das Debugging ist unabhängig vom Design der Anwendung und kann allgemein als Nachteil der reaktiven Programmierung gesehen werden. Die geringere Codekomplexität ergibt sich aus der kompakteren Syntax der Operatoren in RxJS und leitet sich somit ebenfalls aus den Konzepten der reaktiven Programmierung ab. Die Codekomplexität kann aber nicht als allgemeiner Vorteil gesehen werden, da sich die Syntax und die Befehlsstruktur bei anderen Ansätzen der reaktiven Programmierung unterscheiden können. Bei der Wartbarkeit spielt auch das Design der neuen Anwendung eine Rolle. Insbesondere ist das State Management mit NGXS, welches die Anwendungslogik in einem File zusammenbringt und so die Redundanz verringert, von Vorteil. In Bezug auf die Wartbarkeit lässt sich die Aussage, dass es sich um einen Vorteil handelt, ebenfalls nicht verallgemeinern.

Das Framework RxJS, mit dessen Hilfe in Kapitel 4 die Anwendung neu entwickelt wurde, lässt sich in die Kategorie „Verwandte der reaktiven Programmierung“ einordnen. Aus diesem Grund ist es fraglich, ob sich die Vor- und Nachteile des Vergleichs auch auf die Frameworks beziehungsweise Sprachen der funktionalen reaktiven Programmierung beziehen lassen.

Eine metrische Performancemessung wurde nicht durchgeführt. Nach einer qualitativen Beobachtung ist die neuentwickelte Anwendung deutlich performanter als die bestehende. Der Vergleich der Performance der beiden Anwendungen wäre ein weiterer Punkt, der in einer zukünftigen Forschungsarbeit behandelt werden könnte.

Aus diesem Grund wäre es interessant, eine weitere Anwendung mit einem funktionalen reaktiven Ansatz zu entwickeln, um zu sehen, ob sich die Vor- und Nachteile auch hier bestätigen.

Weitere interessante Themen für zukünftige Forschungsarbeiten wären:

- Relevanz von reaktiven Architekturen in modernen Cloud-Umgebungen
- Zusammenwirken eines reaktiven Frontends und eines Websocket-Backends in Hinblick auf Echtzeit Datenaustausch

## 6. Literaturverzeichnis

- Alabor, M., & Stolze, M. (2020). Debugging of RxJS-based applications. In *Proceedings of the 7th ACM SIGPLAN International Workshop on Reactive and Event-Based Languages and Systems* (S. 15–24). Association for Computing Machinery.  
<https://doi.org/10.1145/3427763.3428313>
- Angular—Ahead-of-time (AOT) compilation. (o. J.). Abgerufen 25. Juli 2022, von  
<https://angular.io/guide/aot-compiler>
- AngularJS — Superheroic JavaScript MVW Framework. (o. J.). Abgerufen 24. Juli 2022, von  
<https://angularjs.org/>
- Bainomugisha, E., Carreton, A. L., Cutsem, T. van, Mostinckx, S., & Meuter, W. de. (2013). A survey on reactive programming. *ACM Computing Surveys*, 45(4), 52:1-52:34.  
<https://doi.org/10.1145/2501654.2501666>
- Berry, G., & Gonthier, G. (1992). The Esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2), 87–152.  
[https://doi.org/10.1016/0167-6423\(92\)90005-V](https://doi.org/10.1016/0167-6423(92)90005-V)
- Blackheath, S. (2016). *Functional Reactive Programming*. Simon and Schuster.
- Bonér, J. (2016). *Reactive Microservices Architecture: Design Principles For Distributed Systems*. O'Reilly. <https://go.lightbend.com/reactive-microservices-architecture-design-principles-for-distributed-systems-oreilly>
- Bonér, J., Farley, D., Kuhn, R., & Thompson, M. (2014, September 16). *Das Reaktive Manifest*. reactivemanifesto. <https://www.reactivemanifesto.org/de>
- Can we create Mobile apps in elm? - Request Feedback. (2020, Oktober 7). Elm.  
<https://discourse.elm-lang.org/t/can-we-create-mobile-apps-in-elm/6363>
- CodeMetrics—Visual Studio Marketplace. (o. J.). Abgerufen 26. Juli 2022, von  
<https://marketplace.visualstudio.com/items?itemName=kisstkondoros.vscode-codemetrics>
- Czaplicki, E. (2012). Elm: Concurrent FRP for Functional GUIs. *Undefined*.  
<https://www.semanticscholar.org/paper/Elm-%3A-Concurrent-FRP-for-Functional-GUIs-Czaplicki/1791a8a278b83c54425d7581cb45320feba5f4b0>

- Czaplicki, E. (2016, Januar 4). *New Adventures for Elm*. Elm. <https://elm-lang.org/news/new-adventures-for-elm>
- Czaplicki, E., VonderHaar, A., & Chugh, R. (2016, Januar 29). *Elm Software Foundation*. <https://foundation.elm-lang.org/>
- Dakowitz, P. (2018). *Comparing reactive and conventional programming of Java based microservices in containerised environments* [Thesis, Hochschule für angewandte Wissenschaften Hamburg]. <https://reposit.haw-hamburg.de/handle/20.500.12738/8321>
- Die Bezirke Niederösterreichs—Land Niederösterreich*. (o. J.). Abgerufen 19. Juli 2022, von [https://www.noel.gv.at/noel/Alle\\_Bezirke\\_Niederoesterreichs.html](https://www.noel.gv.at/noel/Alle_Bezirke_Niederoesterreichs.html)
- Elliott, C., & Hudak, P. (1997). Functional reactive animation. *Proceedings of the second ACM SIGPLAN international conference on Functional programming*, 263–273. <https://doi.org/10.1145/258948.258973>
- Elliott, C. M. (2009). Push-pull functional reactive programming. *Proceedings of the 2nd ACM SIGPLAN symposium on Haskell*, 25–36. <https://doi.org/10.1145/1596638.1596643>
- FF Krems | Warn- und Alarmstufenliste*. (o. J.). Abgerufen 19. Juli 2022, von <https://www.feuerwehr-krems.at/CodePages/Wastl/wastlmain/ShowOverview.asp>
- Functional reactive animation | Proceedings of the second ACM SIGPLAN international conference on Functional programming*. (o. J.). Abgerufen 24. Juni 2022, von <https://dl.acm.org/doi/10.1145/258948.258973>
- Gassner, A., Kolmann, P., & Scheidl, C. (2016, März 11). *Grisu-NOE - Git*. <https://github.com/Grisu-NOE/mobile-app>
- Google. (2022). *Angular—Reactive forms*. <https://angular.io/guide/reactive-forms>
- Grisu NÖ (Feuerwehr—WASTL)*. (o. J.). App Store. Abgerufen 25. Juli 2022, von <https://apps.apple.com/us/app/grisu-n%C3%B6-feuerwehr-wastl/id961696829>
- Grisu NÖ (Feuerwehr—WASTL) – Apps bei Google Play*. (o. J.). Abgerufen 25. Juli 2022, von [https://play.google.com/store/apps/details?id=at.lex.grisu.noel&hl=de\\_AT&gl=US](https://play.google.com/store/apps/details?id=at.lex.grisu.noel&hl=de_AT&gl=US)
- Holst, G., & Gillberg, A. (2020). *The impact of reactive programming on code complexity and readability: A Case Study*. <http://urn.kb.se/resolve?urn=urn:nbn:se:miun:diva-39510>
- Introduction to Ionic | Ionic Documentation*. (o. J.). Abgerufen 25. Juli 2022, von <https://ionicframework.com/docs>

- Jourdan, M., Lagnier, F., Maraninchi, F., & Raymond, P. (1994). A multiparadigm language for reactive systems. *Proceedings of 1994 IEEE International Conference on Computer Languages (ICCL'94)*, 211–218. <https://doi.org/10.1109/ICCL.1994.288379>
- Krill, P. (2006, Oktober 16). *Flapjax on the griddle for Web apps—News—Digital Arts*. <https://www.digitalartsonline.co.uk/news/creative-lifestyle/flapjax-on-griddle-for-web-apps/>
- Lines of Code (LOC)—Visual Studio Marketplace*. (o. J.). Abgerufen 26. Juli 2022, von <https://marketplace.visualstudio.com/items?itemName=lyzerk.linecounter>
- McCabe, C., & Timmins, F. (2005). How to conduct an effective literature search. *Nursing Standard*, 20(11), 41–48.
- McCabe, T. J. (1976, Dezember). *A Complexity Measure | IEEE Journals & Magazine | IEEE Xplore*. <https://ieeexplore.ieee.org/document/1702388>
- Meyerovich, L. A., Guha, A., Baskin, J., Cooper, G. H., Greenberg, M., Bromfield, A., & Krishnamurthi, S. (2009). *Flapjax: A Programming Language for Ajax Applications*. <https://cs.brown.edu/~sk/Publications/Papers/Published/mgbcgbk-flapjax/>
- Most*. (o. J.). Npm. Abgerufen 25. Juli 2022, von <https://www.npmjs.com/package/most>
- News on Fran Releases*. (o. J.). Abgerufen 25. Juli 2022, von <http://conal.net/fran/news.htm>
- NÖ Landesfeuerwehrverband—WASTL / GRISU - System*. (o. J.). Abgerufen 25. Juli 2022, von <https://www.noe122.at/fachinfos/informationstechnologie/wastl-grisu-system>
- Project Reactor*. (o. J.). Abgerufen 25. Juli 2022, von <https://projectreactor.io/>
- ReactiveX*. (o. J.). Abgerufen 9. Juni 2022, von <https://reactivex.io/>
- Runeson, P., & Höst, M. (2008). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*. <https://doi.org/10.1007/s10664-008-9102-8>
- Rxjs*. (o. J.). Npm. Abgerufen 25. Juli 2022, von <https://www.npmjs.com/package/rxjs>
- Schögel, M., & Tomczak, T. (2009). Fallstudie. In C. Baumgarth, M. Eisend, & H. Evanschitzky (Hrsg.), *Empirische Mastertechniken: Eine anwendungsorientierte Einführung für die Marketing- und Managementforschung* (S. 79–105). Gabler Verlag. [https://doi.org/10.1007/978-3-8349-8278-0\\_3](https://doi.org/10.1007/978-3-8349-8278-0_3)

- Shibanai, K., & Watanabe, T. (2018). Distributed functional reactive programming on actor-based runtime. *Proceedings of the 8th ACM SIGPLAN International Workshop on Programming Based on Actors, Agents, and Decentralized Control*, 13–22.  
<https://doi.org/10.1145/3281366.3281370>
- SodiumFRP/sodium: Sodium—Functional Reactive Programming (FRP) Library for multiple languages.* (o. J.). Abgerufen 4. Juli 2022, von <https://github.com/SodiumFRP/sodium>
- Solid-js.* (o. J.). Npm. Abgerufen 25. Juli 2022, von <https://www.npmjs.com/package/solid-js>
- Sperber, M. (2001). *Developing a Stage Lighting System from Scratch*. 36, 122–133.  
<https://doi.org/10.1145/507635.507652>
- Svelte.* (o. J.). Npm. Abgerufen 25. Juli 2022, von <https://www.npmjs.com/package/svelte>
- Svelte.* (2022). <https://svelte.dev/>
- Svelte Native • The Svelte Mobile Development Experience.* (o. J.). Abgerufen 25. Juli 2022, von <https://svelte-native.technology/>
- Toczé, K., Vasilevskaya, M., Sandahl, P., & Nadjm-Tehrani, S. (2016). Maintainability of functional reactive programs in a telecom server software. *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 2001–2003.  
<https://doi.org/10.1145/2851613.2851954>
- Tutorials and docs?* (o. J.). Sodium FRP Forum. Abgerufen 25. Juli 2022, von <http://sodium.nz/t/tutorials-and-docs/164>
- Übersicht Einsatzarten.* (o. J.). Feuerwehr Münchendorf. Abgerufen 19. Juli 2022, von <http://www.ff-muenchendorf.at/bürgerinformation/einsatzarten/>
- Wan, Z., & Hudak, P. (2000). Functional reactive programming from first principles. *Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation*, 242–252. <https://doi.org/10.1145/349299.349331>
- Yampa.* (o. J.). Hackage. Abgerufen 25. Juli 2022, von [//hackage.haskell.org/package/Yampa](http://hackage.haskell.org/package/Yampa)
- Yin, R. K. (2017). *Case Study Research and Applications: Design and Methods*. SAGE Publications.

## 7. Tabellenverzeichnis

Tabelle 1: Forschungsfragen mit Kapitel und Methodik.....	9
Tabelle 2: Alle relevanten Daten die im DataService von der HTTP-Schnittstelle abgefragt werden.....	38
Tabelle 3: Erfüllung der Anforderungen (horizontale Achse) durch vorhandene Frameworks (Vertikale Achse) .....	40
Tabelle 4: Wartbarkeitsanalyse der bestehenden Anwendung.....	46
Tabelle 5: Wartbarkeitsanalyse der neu entwickelten Anwendung.....	46
Tabelle 6: Durch die Fallstudie ermittelte Vor- und Nachteile .....	48
Tabelle 7: Durch die Literaturrecherche ermittelte Vor- und Nachteile .....	49
Tabelle 8: Zuordnung Alarmzentrale zu Bezirk.....	58
Tabelle 9: Technischer Einsatz – Einsatzarten.....	59
Tabelle 10: Brandeinsatz – Einsatzarten.....	59
Tabelle 11: Schadstoffeinsatz - Einsatzarten .....	59
Tabelle 12: Messwerte der zyklomatischen Komplexität .....	60

## 8. Abbildungsverzeichnis

Abbildung 1: Beispiel „propagation of change“ .....	11
Abbildung 2: Zustandsgraph .....	12
Abbildung 3: Evaluierungsmodelle .....	13
Abbildung 4: Merkmale von reaktiven Systemen.....	17
Abbildung 5: Fran-Beispiel .....	21
Abbildung 6: Yampa-Beispiel .....	21
Abbildung 7: Elm-Beispiel .....	22
Abbildung 8: Flapjax-Beispiel.....	23
Abbildung 9: Sodium-Beispiel .....	24
Abbildung 10: RxJS-Beispiel.....	26
Abbildung 11: Reactor-Beispiel.....	26
Abbildung 12: Svelte-Beispiel .....	27
Abbildung 13: Screenshots aus der für einen PC-Webbrowser optimierten WASTL-App.....	29
Abbildung 14: Übersichtsseite Einstiegsseite Grisu NÖ-Webapp mit Einsatzkarte und der aktuellen Einsatzstatistik.....	31
Abbildung 15: Auflistung der Einsätze der Bereichsalarmszentrale Amstetten.....	32
Abbildung 16: Detailseite eines Einsatzes.....	33
Abbildung 17: Auflistung aller Bereichs-, Bezirks- und Abschnittsalarmszentralen.....	34
Abbildung 18: Seite mit Einsatzstatistiken mit zwei Diagrammen zu Darstellung des Einsatztyps .....	35
Abbildung 19: Architektur der Grisu NÖ-App.....	37
Abbildung 20: UI der Neuentwicklung .....	41
Abbildung 21: Architektur der Neuentwicklung .....	43
Abbildung 22: Methode des Polling-Service .....	44
Abbildung 23: Diagramm zyklomatische Komplexität .....	45
Abbildung 24: Diagramm Vergleich der Wartbarkeit über LOC .....	46
Abbildung 25: Fehlerbehandlung - bestehende Anwendung .....	47
Abbildung 26: Fehlerbehandlung - neu implementierte Anwendung.....	47



Abbildung 27: Ermittelte Vor- und Nachteile von reaktiver Programmierung .....	50
Abbildung 28: Automatisch generiertes Klassendiagramm .....	60
Abbildung 29: Lines of Code - Messung / neu implementierte Anwendung .....	61
Abbildung 30: Lines of Code - Messung / bestehende Anwendung.....	61

## 9. Anhang

### 9.1. Auflistung der politischen Bezirke und deren Zugehörigkeit zu den Abschnitts- und Bereichs- bzw. Bezirksalarmzentralen

Politischer Bezirk ( <i>Die Bezirke Niederösterreichs - Land Niederösterreich, o. J.</i> )		Abschnitts- und Bereichs- bzw. Bezirksalarmzentrale
<b>Amstetten</b>		BAZ Amstetten (Bereichsalarmzentrale)
<b>Baden</b>		BAZ Baden
<b>Bruck an der Leitha</b>		BAZ Bruck/Leitha / AAZ Schwechat
<b>Gänserndorf</b>		BAZ Gänserndorf
<b>Gmünd</b>		BAZ Gmünd
<b>Hollabrunn</b>		BAZ Hollabrunn
<b>Horn</b>		BAZ Horn
<b>Korneuburg</b>		BAZ Stockerau
<b>Krems an der Donau (Statutarstadt)</b>		BAZ Krems/Donau (Bereichsalarmzentrale)
<b>Krems</b>		BAZ Krems/Donau (Bereichsalarmzentrale)
<b>Lilienfeld</b>		BAZ Lilienfeld
<b>Melk</b>		BAZ Melk
<b>Mistelbach</b>		BAZ Mistelbach
<b>Mödling</b>		BAZ Mödling
<b>Neunkirchen</b>		BAZ Neunkirchen
<b>St. Pölten (Stadt)</b>		BAZ St. Pölten (Bereichsalarmzentrale)
<b>St. Pölten</b>		BAZ St. Pölten (Bereichsalarmzentrale) / AAZ Purkersdorf
<b>Scheibbs</b>		BAZ Scheibbs
<b>Tulln</b>		BAZ Tulln / AAZ Klosterneuburg
<b>Waidhofen an der Thaya</b>		BAZ Waidhofen/T.
<b>Waidhofen an der Ybbs (Statutarstadt)</b>		BAZ Amstetten (Bereichsalarmzentrale)
<b>Wiener Neustadt (Statutarstadt)</b>		BAZ Wr. Neustadt (Bereichsalarmzentrale)
<b>Wiener Neustadt</b>		BAZ Wr. Neustadt (Bereichsalarmzentrale)
<b>Zwettl</b>		BAZ Zwettl

Tabelle 8: Zuordnung Alarmzentrale zu Bezirk

## 9.2. Übersicht Einsatzarten

Technische Einsätze	Beschreibung
<b>T1</b>	Einfache technische Einsätze wie z.B.: Beseitigen von Hindernissen, Fahrzeugbergung, Auspumparbeiten
<b>T2</b>	Einsätze zur Menschenrettung wie z.B. nach Verkehrsunfällen oder anderen Unfällen
<b>T3</b>	Einsätze mit mehreren eingeklemmten Personen wie z.B. Autobusunfall, Eisenbahnunglück.

Tabelle 9: Technischer Einsatz – Einsatzarten  
Quelle: *Übersicht Einsatzarten*, o. J.

Brandeinsätze	Beschreibung
<b>B1</b>	Kleinere Einsätze wie Müllbehälterbrand, Brandverdacht, TUS Alarm, u.ä.
<b>B2</b>	Brände, bei denen ein Löschzug erforderlich ist und voraussichtlich Atemschutz eingesetzt werden muss, wie z.B. Wohnungs- oder Kellerbrand
<b>B3</b>	Brände, bei denen mehr als ein Löschzug erforderlich sind, z.B. Brand eines Wohnhauses, kleinerer Gewerbebetrieb, Dachstuhlbrand u.ä.
<b>B4</b>	Brände, bei denen mehr als zwei Löschzüge erforderlich sind, z.B. Brand eines landwirtschaftlichen Objektes, Brand eines Industrieobjektes u.ä.

Tabelle 10: Brandeinsatz – Einsatzarten  
Quelle: *Übersicht Einsatzarten*, o. J.

Schadstoffeinsätze	Beschreibung
<b>S1</b>	Kleiner Schadstoffeinsatz wie Ölspur u.ä.
<b>S2</b>	Örtlicher Chemieunfall wie kleineren Gewässerschäden, Austritt von Schadstoffen in deren Bereich nur mit Schutzstufe 3 gearbeitet werden kann
<b>S3</b>	Größerer Ölaustritt, Tankwagenunfall, Chemieunfall mit Umweltschäden u.ä.

Tabelle 11: Schadstoffeinsatz - Einsatzarten  
Quelle: *Übersicht Einsatzarten*, o. J.

### 9.3. Klassendiagramm der Neuentwicklung

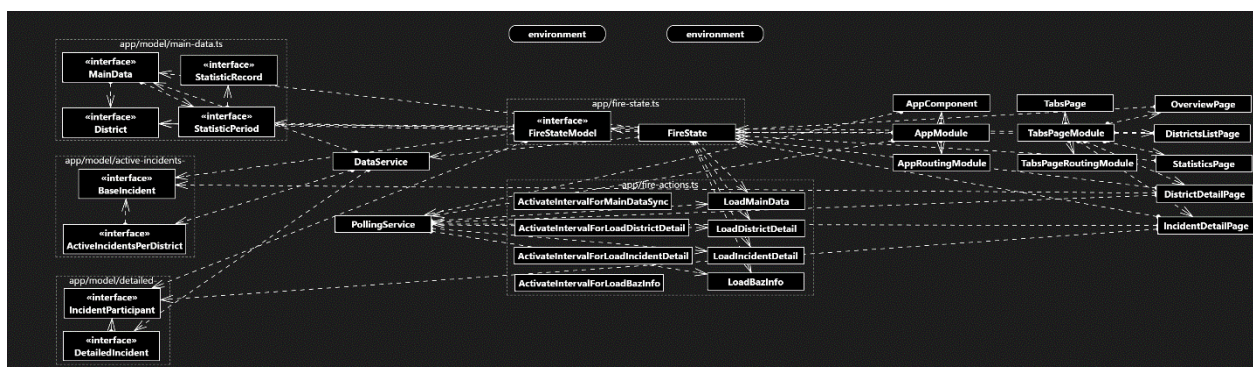


Abbildung 28: Automatisch generiertes Klassendiagramm  
Quelle: Visual Studio Marketplace, o. J..

### 9.4. Tabelle mit den Messwerten der zyklomatischen Komplexität

Alte Anwendung		Neue Anwendung	
Namen	CYC <sup>11</sup>	Namen (neue Applikation)	CYC <sup>13</sup>
getMainData()	7	getMainData()	1
getActiveIncidents()	4	loadDistrictDetail()	1 + 3
getIncidentData()	4	loadIncidentDetail ()	1 + 3
getBazInfo()	6	getBazInfo()	1
processMainData()	15	loadMainData()	13
processBazInfo()	4	loadBazInfo()	8
Summe	40	Summe	31
Durchschnitt	6.67	Durchschnitt	5.17

Tabelle 12: Messwerte der zyklomatischen Komplexität

<sup>11</sup> CYC steht für Cyclomatic Complexity – zyklomatische Komplexität

## 9.5. Wartbarkeitsmessung

```

LineCount GrisunO

=====
EXTENSION NAME : linecounter
EXTENSION VERSION : 0.2.7
-----
count time : 2022-05-26 13:12:03
count workspace : c:\Users\Saturn\Documents\GitHub\mobile-app
total files : 110
total code lines : 3114
total comment lines : 88
total blank lines : 527

statistics
| extension| total code| total comment| total blank|percent|
|-----|
| .html| 732| 1| 67| 24|
| .js| 1835| 54| 356| 59|
| .svg| 15| 0| 0| 0.48|
| .scss| 270| 3| 60| 8.7|
| .json| 37| 0| 0| 1.2|
| .md| 94| 14| 16| 3.0|
| | 40| 4| 5| 1.3|
| .xml| 82| 12| 19| 2.6|
| .yaml| 9| 0| 4| 0.29|
|-----|

```

Abbildung 30: Lines of Code - Messung / bestehende Anwendung

```

LineCount Neue Anwendung

=====
EXTENSION NAME : linecounter
EXTENSION VERSION : 0.2.7
-----
count time : 2022-05-26 13:10:22
count workspace : c:\Users\Saturn\Documents\GitHub\fireservice-webapp
total files : 76
total code lines : 14465
total comment lines : 328
total blank lines : 263

statistics
| extension| total code| total comment| total blank|percent|
|-----|
| .json| 12383| 3| 8| 86|
| .scss| 122| 159| 50| 0.84|
| .ts| 1105| 142| 161| 7.6|
| .html| 572| 1| 17| 4.0|
| .svg| 139| 0| 0| 0.96|
| .css| 0| 0| 0| 0.0|
| .js| 73| 14| 7| 0.50|
| .md| 15| 5| 11| 0.10|
| | 56| 4| 9| 0.39|
|-----|

```

Abbildung 29: Lines of Code - Messung / neu implementierte Anwendung