FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Fachochschule Aachen, Campus Jülich

Fachbereich 09 - Medizintechnik und Technomathematik
Studiengang Angewandte Mathematik und Informatik

Seminararbeit

Analyse der Eignung der TypeScript-Frameworks Angular, React und Vue im Umfeld des WZL der RWTH Aachen University

Autor:
Luca Außem, 3576632

Betreuende:
Prof. Dr. Philipp Rohde
Walburga Hobbie

Aachen, 15. Dezember 2024

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Seminararbeit mit dem Thema „Analyse der Eignung der TypeScript-Frameworks Angular, React und Vue im Umfeld des WZL der RWTH Aachen University“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Ich verpflichte mich, ein Exemplar der Seminararbeit fünf Jahre aufzubewahren und auf Verlangen dem Prüfungsamt des Fachbereiches Medizintechnik und Technomathematik auszuhändigen.

Name: Luca Außem

Aachen, den 15. Dezember 2024

Unterschrift des Studenten

Abstract

Browserbasierte Anwendungen, auch „Webapplikationen“ genannt, sind Anwendungen, welche über den Browser ausgeführt werden. Diese bieten eine Vielzahl an Vorteilen, wie Plattformunabhängigkeit, einfache Wartung und kosteneffiziente Entwicklung. Diese Vorteile sorgen unter anderem für die wachsende Beliebtheit solcher Software. Dem Entwickler stehen dabei verschiedene Werkzeuge zur Verfügung, um solche Applikationen zu realisieren.

Netflix, Gmail und Facebook sind populäre Beispiele von solchen Webanwendungen. Auch wenn diese auf derselben technischen Grundlage basieren, gibt es einen kritischen Unterschied: das Framework, welches zur Entwicklung benutzt wurde. Während in den letzten Jahren viele für Developer zur Verfügung stunden, zählen Angular, React und Vue dabei zu den am weitesten verbreitetsten.

Am Werkzeugmaschinenlabor WZL der RWTH Aachen University werden viele solcher Webanwendungen von den MATSE entwickelt und für verschiedene Aufgaben verwendet. Hierbei stellt sich nun die Frage, welches dieser Frameworks sich für Projekte in diesem Umfeld am besten eignet, oder ob ein solches Framework für die aufgestellten Projekte überhaupt nötig bzw. effizient ist.

Abkürzungsverzeichnis

Abkürzung	Bedeutung
AJAX	Asynchronous JavaScript and XML
CLI	Command Line Interface
CSRF	Cross-Site Request Forgery
CSS	Cascading Style Sheets
DOM	Document Object Model
DRY	Don't Repeat Yourself
HTML	Hypertext Markup Language
IEEE	Institute of Electrical and Electronics Engineers
IQS	Informations-, Qualitäts- und Sensorsysteme in der Produktion
JS	JavaScript
JSX	JavaScript XML
KISS	Keep It Short and Simple
MATSE	Mathematisch-technischer Softwareentwickler
MPA	Multi-Page Application
PHP	PHP: Hypertext Preprocessor
PWA	Progressive Web Application
RWTH	Rheinisch-Westfälische Technische Hochschule
SFC	Single-File Component
SPA	Single-Page Application
TS	TypeScript
TSX	TypeScript XML
WZL	Werkzeugmaschinenlabor
XSS	Cross-Site-Scripting
XSSI	Cross-Site Script Inclusion

Inhaltsverzeichnis

1 Einleitung

Für die Entwicklung von Webapplikationen stehen dem Entwickelnden verschiedene Werkzeuge bereit, um das geplante Softwareprojekt zu realisieren. Eines dieser Werkzeuge ist das „Framework“, welches ein Grundgerüst für die Applikation darstellt und dem Entwickelnden bei der Entwicklung dynamischer Webapplikationen durch verschiedene Funktionen unterstützt. Bei der Wahl des Frameworks für die Entwicklung von Webapplikationen gibt es eine Vielzahl an Optionen, aus denen man wählen kann. Von lightweight Frameworks, welche nur die nötigsten Funktionen besitzen, bis zu heavyweight Frameworks, welche den Entwickelnden in vielen Bereichen unterstützen und viele weitere Funktionen – sogar oft für Prozesse, die nicht direkt mit der grundlegenden Entwicklung in Verbindung stehen – bereitstellen, ist alles vertreten. Dabei stellt sich nicht selten die Frage, welches Framework man für das nächste Projekt verwenden soll.
Bezüglich dieser Frage gibt es eine Vielzahl an Diskussionen, welche eine allgemeingültige Antwort produzieren möchten. Die drei Frameworks, welche dabei meistens miteinander verglichen werden, sind Angular, React und Vue. Dabei werden die Stärken und Schwächen der einzelnen Frameworks sowie deren Einsatzbereiche gegenübergestellt, um so das „beste“ dieser Frameworks ausfindig zu machen.
Trotz des Ziels, eine allgemeingültige Antwort auf diese Frage zu geben, wird diese sehr oft nur für bestimmte Umfelder bzw. Projektziele beantwortet. Ein online Beitrag beschreibt es dabei mit

„[. . .] and it depends upon the developer’s goal and business requirements of the project, which one they should choose.“ [1]

Es lässt sich also folgern, dass die Auswahl des Frameworks von dem gegebenen Umfeld abhängt und es somit keine allgemeingültige Antwort für das „beste“ Framework gibt.
Aus diesem Grund beschäftigt sich folgende Arbeit mit der Bestimmung des passenden Frameworks für ein gegebenes Umfeld. Dieses Umfeld ist das Werkzeugmaschinenlabor WZL der RWTH Aachen University und bezieht sich dabei genauer auf die MATSE-Ausbildung am Lehrstuhl für Informations-, Qualitäts- und Sensorsysteme in der Produktion. Während der MATSE-Ausbildung am WZL werden zum Großteil Projekte in dem Bereich der Webentwicklung vergeben. Zum jetzigen Zeitpunkt wird dafür kein Framework verwendet, jedoch wurden in den vergangenen Monaten immer mehr Projekte als Single-Page-Webanwendung entwickelt, weshalb die Frage, welches Framework für solche Projekte geeignet wäre, immer präsenter wird. Die folgende Analyse beschäftigt sich dabei mit den oben genannten Frameworks: Angular, React und Vue.
Mittels verschiedener Analysemethoden soll dabei ein finaler Nutzwert für jedes dieser Frameworks ermittelt werden, um damit eine finale Entscheidung über die Eignung dieser Frameworks treffen zu können. Hierbei liegt der Fokus darauf, eine angepasste Auswahl für das gegebene Umfeld zu treffen und dabei verschiedenste Einflüsse zu berücksichtigen. Die dabei festgestellten potenziellen Nachteile der Frameworks werden dabei auch in die Entscheidung mit einbezogen. Somit ist das Ziel, eine fundierte Auswahl eines dieser Frameworks, welche auf die Anforderungen des Umfeldes angepasst ist und zeitgleich dabei eine nachhaltige Entscheidung darstellt.

2 Theoretische Grundlagen

2.1 Webapplikation – Definition

Eine Webapplikation ist eine Software, die auf einem Server gehostet wird. Dieser stellt die Software so zur Verfügung, dass sie clientseitig mit einem Browser aufrufbar ist. Der Nutzende kann hierbei über die URL im verwendeten Browser auf die jeweilige Nutzeroberfläche der Applikation zugreifen. Der Browser des Nutzenden steuert dabei die Nutzeroberfläche durch clientseitige Skripte, während die serverseitige Architektur für die Verarbeitung der Daten zuständig ist. Somit lassen sich nutzerseitig auf Funktionen zugreifen, welche ansonsten die Installation eines Programms benötigt hätten [2].

2.1.1 Typen von Webapplikationen

Webapplikationen lassen sich in verschiedene Typen unterteilen. Diese unterscheiden sich dabei im Abruf der Daten bzw. der generellen Funktionalitäten. Die Typen besitzen dabei ihre eigenen Vor- und Nachteile. Da dieses Kapitel nur eine Einleitung in das Thema „Webapplikationen“ geben soll, werden diese jedoch nicht genannt bzw. verglichen.
Single-Page Applications (SPA) werden beim erstmaligen Seitenaufruf geladen. Für die gesamte Nutzungszeit wird die weitere Nutzerinteraktion mittels dem AJAX-Konzept realisiert [3]. Somit wird die Seite, wie der Name bereits sagt, während der Nutzung nicht geändert, sondern die Inhalte der Nutzeroberfläche werden dynamisch geändert.
Multi-Page Applications (MPA) laden während der Benutzung mehrfach neu. Sollen Daten geändert werden, so wird die Seite neu geladen bzw. es wird auf eine andere Seite weitergeleitet und die Daten vom Server abgefragt. Die Ladezeiten hängen dabei stark vom Server bzw. den für das server-side rendering verwendeten Methoden ab [3].
Progressive Web Applications (PWA) stellen eine Kombination aus traditionellen Websites und nativer Software dar. Sie können somit über den Browser, aber auch über eine auf dem Gerät installierte App zugegriffen werden. Dabei gibt es auch die Möglichkeit, diese ohne bestehende Internetverbindung über die installierte App abzurufen [4].

2.1.2 Unterschiede zu nativer Software und statischen Websites

Der Unterschied zu nativer Software ist hierbei, dass diese nur für eine spezielle Nutzerumgebung, zum Beispiel für ein bestimmtes Betriebssystem, entwickelt wurde. Zudem muss sie separat auf dem Computer installiert werden. Im Gegensatz dazu ist für die Verwendung von Webapplikationen nur die Installation eines beliebigen Internetbrowsers nötig, über den man auf die Applikation zugreift. Da Webapplikationen im Kern standardmäßige Websites sind, müssen auch hierbei die Unterschiede zwischen einer solchen Website und einer Webapplikation erläutert werden. Die Unterschiede beziehen sich hierbei auf die Interaktivität mit dem Nutzer. Websites sind dabei meist statisch und zeigen nur einen vorher definierten Inhalt an, welcher jedoch auch durch gewisse Faktoren, z. B. die eingestellte Sprachen, eine gewisse Dynamik besitzen kann. Webapplikationen hingegen besitzen komplexere Funktionen, bspw. Anfragen an eine Datenbank, und eine erhöhte Nutzerinteraktivität. Mit dem sich verändernden Design von modernen Websites verschmelzen diese Grenzen jedoch langsam, da auch standardmäßige Websites eine immer mehr wachsende Dynamik bekommen [5].

2.1.3 Vor- und Nachteile

Die Vor- und Nachteile der Nutzung von Webapplikationen im Gegensatz zu nativer Software lassen sich schließlich wie folgt beschreiben: Webapplikationen erreichen einfacher eine größere Anzahl an Nutzenden. Da die Software über einen Server zur Verfügung gestellt wird, lässt sich diese einfacher warten und für die Nutzenden updaten. Auch die Sicherheit wird dabei verbessert, da Daten zentral gespeichert werden und Sicherheitsupdates schnell für die Nutzenden installiert werden können. Durch Erhöhung der Serverressourcen lässt sich die Software auch relativ einfach skalieren. Da solche Software über den Browser abgerufen wird, ist jedoch eine Internetverbindung notwendig und die Leistung der Software wird durch den Browser eingeschränkt. Auch auf die Hardware des Nutzenden lässt sich nur beschränkt zugreifen, was, je nach Projekt, eventuelle Einschränkungen der Funktionalität zur Folge haben könnte. Sicherheitsrisiken können hierbei beispielsweise durch *Cross-Site-Scripting (XSS)* und Datenlecks entstehen [6].
Inwiefern diese Vor- und Nachteile gegeneinander abzuwiegen sind, bleibt dem Entwickelnden für das jeweilige Projekt überlassen. Es lässt sich jedoch festhalten, dass solche Software für viele Umfelder entscheidende Vorteile gegenüber nativer Software mit sich bringt, was sich in der steigenden Beliebtheit dieser Art von Software zeigt.

2.1.4 Methoden der Entwicklung

Auch der Weg, solche Webapplikationen zu entwickeln, kann sehr unterschiedlich sein. Zum einen lässt sich eine solche Applikation mit reinem HTML, CSS und JavaScript schreiben, womit der Entwickelnde volle Kontrolle über die Funktionen innerhalb der Anwendung besitzt. Soll der Entwicklungsprozess vereinfacht werden, lassen sich hierbei externe Bibliotheken verwenden, um komplexere Funktionen einfacher in die eigene Software zu integrieren.
Andererseits lässt sich auch ein Framework verwenden, welches bereits viele Funktionalitäten für häufig auftretende Probleme bereitstellt. Solche Frameworks fangen bereits bei vordefinierten Styles bzw. HTML-Komponenten, wie zum Beispiel Bootstrap, an, können jedoch auch weit über einfache Funktionalitäten hinausreichen.

2.2 Frameworks in der Webentwicklung

2.2.1 Definition

Wie der Name schon sagt, stellt ein Framework ein Grundgerüst für eine Software dar, welches eine vorgefertigte Struktur und viele Funktionen bzw. Werkzeuge für die Entwicklung der Software bereitstellt. Somit muss der Entwickelnde eine Software nicht von Grund auf neu programmieren, sondern kann auf diese Funktionalitäten zurückgreifen, was den Entwicklungsprozess nicht nur verkürzt, sondern auch effizienter und resistenter gegenüber Fehlern gestaltet.
Durch die oft verfügbare Wiederverwendbarkeit von Komponenten innerhalb eines Frameworks, folgt die Entwicklung somit zeitgleich dem Design-Prinzip *Don't Repeat Yourself (DRY)* und mittels der Verringerung des nötigen Codes, welcher geschrieben werden muss, *Keep It Short And Simple (KISS)*. *Convention over Configuration* sagt aus, dass nur spezifische und unkonventionelle Elemente durch den Entwickelnden selbst konfiguriert werden sollen. Indem Frameworks bereits bekannte Verfahren als Standardmethoden implementieren, folgen sie i. d. R. auch diesem Prinzip [7]. Ein Framework für eine Webapplikation, auch Webframework genannt, ist hierbei ein Framework, welches als Grundgerüst für eine Webanwendung fungiert.
Webframeworks lassen sich hierbei weiter in zwei verschiedene Arten einteilen: *Server-Side Frameworks* [8] sind Frameworks, welche Funktionalitäten für die Verarbeitung der Daten und weiterer Server Logik, wie URL Mapping, bereitstellen und aus diesem Grund im Backend einer Webapplikation verwendet werden. *Client-Side Frameworks* [9] stellen im Gegensatz dazu die Nutzeroberfläche,

das Frontend, dar. Bei Webapplikationen ist diese Darstellung dynamisch und basiert oft auf Daten, die vom Backend bereitgestellt werden.

2.2.2 Überblick über die zu analysierenden Frameworks

Wie bereits in Kapitel 1 genannt, werden in dieser Analyse die drei Frameworks Angular, React und Vue analysiert und miteinander verglichen. Um eine kurze Übersicht über diese zu geben, folgt nun eine kurze Einleitung in jedes dieser Frameworks.

Angular wurde erstmals 2010 unter dem Namen „AngularJS“ veröffentlicht. Seit 2014/2015 befindet es sich in der Version 2.0 und wird auch nur „Angular“ bzw. „Angular 2.0“ genannt. Es wurde von dem Google Mitarbeiter Miško Hevery im Rahmen eines Nebenprojekts entwickelt und wird auch mittlerweile von Google gewartet [10].
Es stellt viele Werkzeuge für den Entwickelnden zur Verfügung, welche eine effiziente Entwicklung ermöglichen. Durch die Menge an Funktionalität wird Angular als das heavyweight Framework bezeichnet, welches dadurch jedoch auch für robusten Code sorgt und Angular daher oft für large-scale Applikationen verwendet wird [11].

React wurde 2013 von Jordan Walke entwickelt, welcher Mitarbeiter von Facebook war, wo das Framework inzwischen auch weiterentwickelt wird. Durch die Implementierung eines virtuellen *Document Object Model (DOM)*, welches den DOM nur an geänderten Stellen neu rendert, weist React eine hohe Performance auf [12].
Nach der Definition eines Frameworks ist React jedoch keines, sondern ausschließlich eine Bibliothek. Im Vergleich mit Angular und Vue wird React jedoch nicht selten auch als Framework bezeichnet. Innerhalb dieser Analyse wird für die Einfachheit React somit auch als Framework bezeichnet.

Vue ist ein Framework, welches von dem Google Mitarbeiter Evan You entwickelt wurde. Dieses Framework entstand aus dem Ziel, eine lightweight Version von AngularJS bereitzustellen. Es wurde 2014 veröffentlicht und befindet sich momentan in der 3. Version, auch „Vue3“ genannt [13]. Vue ist dabei für seine besondere Anfängerfreundlichkeit bekannt, was sich durch die flache Lernkurve auszeichnet. Es besitzt zudem eine starke Flexibilität, sodass es sich für viele verschiedene Anwendungsfälle nutzen lässt [14]. Wie React besitzt auch Vue ein virtuelles DOM, was zu der bekannten Performance des Frameworks beiträgt [15].

2.3 Analytische Verfahren

2.3.1 Anforderung

Die nächsten Analyseverfahren beschäftigen sich unter anderem mit der Erfassung der verschiedenen Anforderungen and die Frameworks. Dafür sollte der Begriff „Anforderung“ jedoch zuerst genau definiert werden. Laut IEEE wird eine Anforderung im Kontext der Softwareentwicklung dabei wie folgt definiert:

> *„Requirements represent the core functionality that software must deliver in order to be useful and meet business process or user goals.“* [16]

Wird in dieser Analyse eine Anforderung angesprochen, so bezieht sich diese auf die genannte Definition.

2.3.2 Umfeldanalyse

Mithilfe einer Umfeldanalyse werden die Rahmenbedingungen, Einflüsse und Faktoren untersucht, welche speziell für das gegebene Umfeld von Bedeutung sind. Hierdurch soll die Dynamik des Umfelds genauert definiert werden können, sodass eine fundierte Entscheidung, welche dieses gegebene Umfeld berücksichtigt, getroffen werden kann.
Hierbei werden verschiedene Faktoren gesammelt, welche einen Einfluss auf die Entscheidung haben könnten. Die Faktoren lassen sich danach in folgende Gruppen einteilen: *Sachliche Faktoren*, welche nicht-personelle Einflüsse, darstellen. *Soziale Faktoren* sind Personen(-gruppen), welche einen gewissen Einfluss besitzen. Diese versteht man unter dem Begriff „Stakeholder“, welcher in Abschnitt 2.3.3 weiter definiert wird. *Interne Faktoren* befinden sich dabei innerhalb des genannten Umfelds und *externe Faktoren* liegen außerhalb des Umfelds, was in der Analyse jedoch nicht berücksichtigt wird. Aus diesen Faktoren lassen sich schließlich Anforderungen ableiten, welche in späteren Schritten der Analyse benötigt werden.
Wie durch einen Online-Beitrag von dem Softwareunternehmen Asana [17] beschrieben, wird dieses Analyseverfahren nur noch selten angewendet, da es mit anderen Verfahren – unter anderem der Stakeholderanalyse – verschmilzt und somit überflüssig wird. Die Umfeldanalyse wird bei dieser Analyse jedoch dafür verwendet, einen Grundstein zu legen und einen generellen Einblick in das Umfeld zu geben.

2.3.3 Stakeholderanalyse

Innerhalb der Softwareentwicklung gibt es eine Vielzahl an Akteuren, welche Teil dieses Prozesses sind. Dies sind dabei nicht nur die Entwickelnden, sondern auch Projektleitende, Auftraggebebende usw. Diese müssen bei der Analyse berücksichtigt werden, da diese einen starken Einfluss auf die Projekte haben können. Diese Akteure werden dabei auch *Stakeholder* genannt.
Die Definition des Begriffs „Stakeholder“ ist dabei nicht eindeutig definiert und kann, je nach Kontext, anders verstanden werden. Die Definition, welche am bekanntesten ist und am häufigsten verwendet wird, definiert Stakeholder jedoch folgendermaßen:

> *„Any group or individual who is affected by or can affect the achievement of an organization's objectives.“* [18]

Stakeholder können dabei in zwei Gruppen eingeteilt werden: Die *internen Stakeholder*, welche beispielsweise Teil des Teams oder Unternehmens sind und *externe Stakeholder*, welche Parteien außerhalb des Lehrstuhlumfelds, wie zum Beispiel Kunden oder externe Auftraggebende, sind. Auch hier werden externe und interne Stakeholder nicht unterschiedlich behandelt.

Mittels einer Stakeholderanalyse lassen sich die Interessen und der Einfluss dieser identifizieren, sodass ihre Interessen während der finalen Diskussion berücksichtigt werden können. Das verwendete Verfahren orientiert sich dabei an dem Verfahren nach dem Organisationshandbuch des Bundesministeriums des Inneren und für Heimat [19]. Bei diesem werden die verschiedenen Stakeholder identifiziert und deren Beziehungen untereinander dargestellt. Die einzelnen Stakeholder werden schließlich mit diesen Daten bezüglich der *Betroffenheit*, dem *Interesse*, dem *Einfluss* und den *Erwartungen* analysiert und bewertet.
Anhand dieser Analyse lassen sich nicht nur Anforderungen auf Basis der vorhandenen Stakeholder definieren. Die einzelnen Faktoren, welche für die jeweiligen Stakeholder analysiert wurden, dienen zudem für die Bestimmung der Gewichtung einzelner Anforderungen. Dies stellt somit sicher, dass bei der finalen Bewertung der Frameworks die wichtigsten Anforderungen mehr berücksichtigt werden, was schließlich zu einer höheren Akzeptanz des Ergebnisses führt.

2.3.4 Risikoanalyse

Bei der Risikoanalyse werden mögliche Risiken identifiziert und bewertet. Hierzu wird erneut ein Verfahren aus dem Organisationshandbuch des Bundesministeriums des Inneren und für Heimat verwendet [20].
Der Begriff „Risiko“ lässt sich dabei als „aus der Unvorhersehbarkeit der Zukunft resultierende, durch ‚zufällige‘ Störungen verursachte Möglichkeiten, von geplanten Zielwerten abzuweichen“ [21], definieren. Laut der hier genannten Definition müsste ein Risiko somit nicht gezwungenermaßen einen negativen Einfluss besitzen.
Zuallererst müssen die relevanten Risiken erkannt werden, wozu es verschiedene Methoden gibt. In dieser Analyse wird die Kopfstandtechnik verwendet, bei der die Kernfrage der Risikoidentifikation einfach umgekehrt wird. Nachdem die Risiken gesammelt wurden, werden diese schließlich bewertet, indem diese mittels eines Vergleichs mit den anderen Frameworks eingestuft und nach der Priorität, dieses Risiko zu vermeiden, quantifiziert werden. Für die Einstufung werden dafür die verschiedenen Risiken im Vergleich mit den anderen Frameworks sortiert und je nach Platzierung dem Wert 0.0, 0.5, oder 1.0 zugeordnet. Bei der Priorisierung dieser Risiken werden die einzelnen Risiken miteinander verglichen und je nach Anzahl der Risiken einem Wert zwischen 0 und 100 zugeordnet. Zum Schluss wird den gegebenen Risiken mit der Formel

$$\text{Risikofaktor} = \sum \text{Einstufung}_i \cdot \text{Priorität}_i \tag{2.1}$$

ein Wert zugeteilt, welcher im späteren Verlauf der Analyse verwendet werden kann, um den Nutzwert hinsichtlich der Risiken anzupassen.

2.3.5 Nutzwertanalyse

Nachdem die Anforderungen definiert wurden, werden diese mithilfe der Nutzwertanalyse bewertet. Hierbei werden die Frameworks anhand von verschiedenen Kriterien, welche in den vorherigen Analyseschritten definiert wurden, bewertet. Diese Kriterien besitzen eine gewisse Gewichtung, welche in der Stakeholderanalyse bestimmt wird.
Die hierbei befolgten Schritte beziehen sich dabei auf den Beitrag der Munich Business School [22]. Dabei werden die Kriterien identifiziert und deren Gewichtungen bestimmt, was bereits in den Analysen davor zu einem Großteil geschehen ist. Die Bewertung wird bestimmt, indem die Frameworks hinsichtlich der Anforderungen verglichen und jeweils eingestuft werden, wodurch diese dabei den Wert 0.0, 0.5, oder 1.0 zugeordnet bekommen. Die Bewertung lässt sich somit folgendermaßen bestimmen:

$$\text{Bewertung}_i = \text{Maximalpunktzahl} \cdot \text{Einstufung}_i. \tag{2.2}$$

Die Maximalpunktzahl wird dabei im späteren Verlauf in Abhängigkeit von der Anzahl der Anforderungen bestimmt. Mittels dieser Kriterien werden die Frameworks bewertet und der Gesamtnutzen durch die Formel

$$\text{Gesamtnutzen} = \sum \text{Gewichtung}_i \cdot \text{Bewertung}_i \tag{2.3}$$

berechnet, sodass man schließlich eine Entscheidung mittels der Ergebnisse treffen kann.
Für diese Analyse wird diese Formel jedoch angepasst, sodass diese die Ergebnisse der Risikoanalyse berücksichtigt. Daraus ergibt sich somit folgende finale Formel

$$\text{Gesamtnutzen} = \left(\sum \text{Gewichtung}_i \cdot \text{Bewertung}_i\right) - \text{Risikofaktor}. \tag{2.4}$$

Diese Formel beinhaltet die Ergebnisse von den vorher durchgeführten Analysen. Somit lassen sich die Frameworks final bewerten, um so eine fundierte Entscheidung treffen zu können.

3 Analyse

3.1 Umfeld

3.1.1 Das Werkzeugmaschinenlabor WZL der RWTH Aachen University

Das Werkzeugmaschinenlabor WZL der RWTH Aachen University ist ein Forschungsinstitut in der Produktionstechnik. Dieses ist in die Lehrstühle für *Produktionssystematik*, *Werkzeugmaschinen* und *Informations-, Qualitäts- und Sensorsysteme in der Produktion* unterteilt, in welchen die MATSE-Ausbildung angeboten wird. Der dritte Lehrstuhl, auch kurz „IQS“ genannt, ist hierbei der Lehrstuhl, dessen Umfeld nun untersucht werden soll. Da in Bezug auf Softwareprojekte nur selten eine Zusammenarbeit mit den anderen Lehrstühlen existiert, werden diese für diese Analyse nicht beachtet.
Das Umfeld ist durch eine starke Dynamik geprägt. Die wissenschaftlichen Mitarbeitenden sind meist nur während ihres Promotionsvorhabens im Rahmen der Forschungsprojekte des WZL in diesem tätig und MATSE haben nach abgeschlossener Ausbildung zwar die Möglichkeit, als studentische Hilfskraft weiterhin in dem Umfeld tätig zu sein, jedoch gibt es keine festen Stellen, welche speziell für MATSE vorgesehen sind. Es ergibt sich somit ein Mitarbeiterpool, welcher einem häufigen Wechsel unterliegt und somit unterschiedliche Wissensstände mit sich bringt. Des Weiteren herrscht eine rege Zusammenarbeit mit externen Partnern. Dies führt zu diversen möglichen Quellen für Softwareprojekte, sodass Projekte in verschiedenen Bereichen, wie der Darstellung von Daten und Speicherung und Abruf von Datenbankdaten, entstehen können.
Aufgrund dieser Dynamik erfordert dieses Umfeld somit eine gewisse Flexibilität, um sich somit auf die oft ändernden Gegebenheiten bzw. Projektanforderungen anpassen zu können. Durch diese Flexibilität lassen sich diese Projekte schließlich schneller und effizienter entwickeln.

3.1.2 Projekte der MATSE

Das Aufgabengebiet der MATSE am Lehrstuhl IQS umfasst in erster Linie die Entwicklung von Webapplikation und statischen Websites. Die Wartung und Weiterentwicklung von bereits entwickelter Software spielt besonders bei erfahreneren MATSE in höheren Lehrjahren eine wichtige Rolle. Wichtig anzumerken ist, dass diese Tools ohne Framework entwickelt wurden und somit auf reinem HTML, CSS, JavaScript und PHP basieren. Somit ist oft ein tiefes Verständnis dieser Technologien erforderlich, damit dies effizient möglich ist.
Die Umsetzung der Projekte liegt zu einem großen Teil in der Verantwortung der MATSE. Diese haben die Freiheit, selbst über die verwendeten Technologien und Werkzeuge entscheiden zu können. Im Kontext dieser Analyse bezieht sich das Werkzeug auf das Framework, welches für die Projekte verwendet werden soll. Dieses soll jedoch passend zum Umfeld gewählt werden. Ansonsten müssen aber keine Einschränkungen berücksichtigt werden. Zwar gibt es zusätzliche technische Beschränkungen, da diese jedoch frameworkunabhängig sind, werden diese im Rahmen dieser Analyse nicht weiter erläutert und berücksichtigt.

3.1.3 Faktoren

Bei den oben genannten Faktoren, kann nun, wie bereits beschrieben, zwischen *intern* und *externe* sowie zwischen *sozialen* und *sachlichen* unterschieden werden. Aus diesen Faktoren können im weiteren Verlauf der Analyse Stakeholder und Anforderungen abgeleitet werden, welche bei der Bewertung der Frameworks berücksichtigt werden. Folgende Matrix ergibt sich:

	sachlich	**sozial**
intern	- ehem. Projekte - techn. Infrastruktur	- Entwicklerteam - Projektleitung - Ansprechpartner
extern		- Kooperationspartner - Projektleitung

Abbildung 3.1: Visualisierung der Umfeldeinflüsse

3.2 Stakeholder

3.2.1 Auflistung der Stakeholder

Die Projekte der MATSE besitzen meist nur kleine Teams, weshalb es nur eine kleine Menge an Stakeholdern gibt. Diese Teams bestehen dabei aus einer kleinen Gruppe aus *Entwickelnden*, welche an der Programmierung der Software beteiligt sind. Die *Projektleitung* besteht i. d. R. aus einer Person, von welcher der Projektauftrag kommt und welche in Rücksprache mit den Entwickelnden Verbesserungsvorschläge auf Basis des aktuellen Stands der Software gibt. Somit verschmilzt in diesem Fall die Rolle der Projektleitung mit der der Auftraggeber. Der *Betreuer* der MATSE beobachtet hierbei den Prozess, lässt jedoch den MATSE freien Handlungsspielraum und greift nur bei Herausforderungen ein. In wenigen Projekten existiert auch eine Zusammenarbeit mit der *IT*, welche z. B. Nutzungsberechtigungen für interne APIs verwaltet.

3.2.2 Analyse der Stakeholder

Entwickelnde: Die Entwickelnden besitzen die Hauptrolle in den Projekten. Somit werden sie durch die Wahl der verwendeten Werkzeuge besonders stark betroffen, da sie diese im Anschluss für die Entwicklung der Software verwenden. Daraus folgt auch ein hohes Interesse. Der Einfluss ist hierbei relativ hoch, da diese die verwendeten Werkzeuge selbst wählen und dabei einen großen Handlungsspielraum besitzen.
Bei der Wahl des Frameworks erwarten diese somit, dass mit diesem einfach zu arbeiten ist und sich dieses auch für zukünftige Projekte verwenden lässt.

Projektleitung: Die Projektleitung besitzt das höchste Interesse und den höchsten Einfluss am Projekt, welche sich hierbei jedoch eher auf die Funktionalität der Software und nicht auf die verwendeten Technologien beziehen. Da diese zudem in den meisten Fällen nicht am Projekt mitentwickelt, ist sie somit nicht von der Wahl des Frameworks betroffen.
Die einzigen Erwartungen, die sich definieren lassen, sind die, dass das Projekt durch das verwendete Framework keinen negativen Einfluss erhält und es keine funktionalen Einschränkungen gibt.

Betreuender: Der Betreuende bzw. Ausbildende der MATSE steht zwischen den Entwickelnden und der Projektleitung. Da dieser für die MATSE verantwortlich ist, besitzt er dementsprechend auch hohes Interesse. Betroffen ist er nur, wenn er von den Entwickelnden nach Hilfe gefragt wird bzw. wenn es Herausforderungen während des Projekts gibt.

IT: Die IT hat bei den Projekten das geringste Interesse und ist wenig und eher neutral davon betroffen, da sie nur bei einzelnen Features unterstützt. Durch ihre administrative Rolle besitzt sie den größten Handlungsspielraum bei der Implementierung von Technologien.
Wirkliche Erwartungen gibt es hierbei nicht. Die einzige Anforderung, die hierbei im Raum steht, ist die, dass das Framework für das Werkzeug geeignet ist, welches angebunden werden muss.

3.2.3 Visualisierung

Stakeholder	Grad der Betroffenheit	Interesse	Einfluss	Erwartungen
Betreuender	0	++	++	Zielgerichtete Auswahl
Entwickelnde	++	++	+	Funktionalität & Effizienz
IT	-	- -	- -	Kompatibilität
Projektleitung	0	+	+	Funktionalität

Tabelle 3.1: Stakeholder Zusammenfassung

Mit diesen Daten lassen sich die Anforderungen der Stakeholder bestimmen. Die von den Stakeholdern abgeleiteten Anforderungen werden dabei in einem späteren Schritt genauer erläutert. Die hier gesammelten Daten lassen sich außerdem in einer Vierfeldmatrix visualisieren, sodass das Gesamtbild etwas klarer wird.

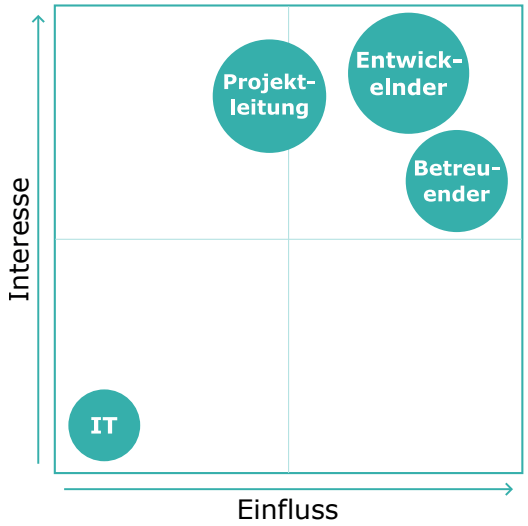

Abbildung 3.2: Stakeholder-Matrix

3.3 Risikoanalyse

3.3.1 Identifikation der Risiken

Wie bereits in Abschnitt 2.3.4 beschrieben, wird für diesen Teil der Analyse die Kopfstandtechnik verwendet. Somit lässt sich die Fragestellung „Wie muss das Framework gewählt werden, um die daraus resultierenden Risiken für zukünftige Projekte zu maximieren?“ definieren. Aus dieser Fragestellung kann man folgende Lösungen ableiten:

Abhängigkeit von Kollegen: Der Code ist generell schwer verständlich und das Framework ist so komplex, dass man als Anfänger auf die Hilfe von anderen angewiesen ist.

Hohe Schwierigkeit: Die Entwicklung mit dem Framework ist für die Entwickelnden anfangs sehr mühsam, sodass eine Menge Vorarbeit notwendig ist, um mit diesem effizient arbeiten zu können.

Kurzlebig: Es ist wahrscheinlich, dass einzelne Features vom Framework bzw. das Framework an sich sehr schnell nicht mehr unterstützt wird.

Unflexibilität: Es ist schwierig, neue Funktionen zu implementieren und das Framework lässt sich nur sehr schwer für Bereiche einsetzen, für die es nicht direkt gedacht ist.

Verwundbar: Es ist sehr einfach, unsichere Features zu implementieren, insbesondere wenn keine manuelle Fehlerprävention oder zusätzliche Sicherheitsmaßnahmen durchgeführt werden.

Aus diesen Lösungen lassen sich nun folgende potenzielle Risiken ableiten, die im späteren Teil der Analyse für die einzelnen Frameworks jeweils bewertet werden sollen:

Dysfunktionalität: Das Framework besitzt nur wenige Grundfunktionen. So müssen viele weitere Funktionen mittels externer Bibliotheken eingebunden bzw. selbst programmiert werden.

Komplexität: Es ist schwierig, sich in das Framework einzuarbeiten. So ist für jede neue Funktion eine Menge an Recherche notwendig, sodass die Implementierung lange dauert.

Kurzlebigkeit: Das Framework wird nicht mehr lange mit (sicherheitsrelevanten) Updates versorgt, sodass die App relativ schnell vulnerabel wird.

Unsicherheit: Das Framework unterstützt den Entwickelnden nicht dabei, sicheren Code zu schreiben.

3.3.2 Risikobewertung

Den nun definierten Risiken werden zwei Faktoren zugeordnet. Je nach Einstufung wird dem Risiko ein Wert im Bereich 0 bis 1 zugeordnet. Je nach Priorität, welche das Risiko besitzt, erhält es einen Wert zwischen 0 und 100.
Die Einstufung ist hierbei abhängig von dem Framework. Somit wird vorerst nur der Schadensfaktor für die jeweiligen Anforderungen bestimmt.

Risiko	Schadensfaktor
Dysfunktionalität	25
Komplexität	100
Kurzlebigkeit	50
Unsicherheit	75

Tabelle 3.2: Schadensfaktoren der Risiken

3.4 Anforderungen an das Framework

3.4.1 Anforderung mit Bezug zum Umfeld

Betrachtet man das Umfeld, so wird klar, dass das Framework für eine dynamische Umgebung geeignet sein muss, in der relativ oft neue Personen auf die Codebase zugreifen. Somit ist es von Vorteil, einen einfach aufgebauten Code zu besitzen. Der Entwickelnde hat zwar einen starken Einfluss darauf, jedoch ist es auch teilweilse abhängig von dem Framework. Zeitgleich ist es auch hilfreich, wenn Mitarbeitende Erfahrungen mit dem jeweiligen Framework haben, sodass ein Austausch von Informationen stattfinden kann.
Da oft mit sensiblen Daten gearbeitet wird und oft auf interne Schnittstellen zurückgegriffen wird, ist somit nicht unrelevant, dass das Framework den Entwickelnden dabei unterstützten sollte, robusten und sicheren Code zu schreiben.
Durch die Betrachtung den Umfeldes lassen sich somit folgende Anforderungen definieren, welche das Framework hierbei erfüllen sollte:

- Flexibilität
- Interne Erfahrung
- Lesbarkeit
- Sicherheit

3.4.2 Anforderung mit Bezug zu den Stakeholder

Wie in Unterkapitel 3.2 festgestellt, gibt es eine Vielzahl an Stakeholdern, welche einen gewissen Einfluss auf besitzen. Betrachtet man die Stakeholder und deren Erwartungen, ergeben sich folgende Anforderungen:
Es sollte schnell in einen neuen Mitarbeiterkreis eingebunden werden können und auch für eine Vielzahl an verschiedenen Projekten ausgerichtet sein. Eine nicht allzu steile Lernkurve sollte dabei vorhanden sein, um einen einfachen Einstieg für neue MATSE, welche i. d. R. kaum Erfahrung im Bereich der Programmierung besitzen, zu gewährleisten. Bei der Übergabe von Projekten an andere MATSE, ist der Faktor des leichten Verstehens von bereits geschriebenem Code nicht zu vernachlässigen.
Eine andere Anforderung ist die Zukunftssicherheit. Nicht nur soll das Framework lange mit (sicherheitsrelevanten) Updates unterstützt werden, sodass Projekte mit längerem Lifecycle nicht ggf. neugeschrieben werden müssen. Es liegt auch im Interesse der MATSE, Erfahrungen mit Werkzeugen zu sammeln, welche später im Berufsleben, also nach Abschluss der Ausbildung, gefragt sind.
Das Framework sollte zudem für die Projekte passend gewählt werden. So sollte für eine kleine, simple Webapplikation beispielsweise kein komplexes Framework mit vielen Funktionen gewählt werden, welche am Ende nicht benötigt werden.
Betrachtet man schließlich die Gesamtheit der Stakeholder, ergeben sich folgende Anforderungen:

- Anfängerfreundlichkeit
- Angepasstheit
- Funktionalität
- Zukunftssicherheit

3.4.3 Anforderung mit Bezug zu zukünftigen Projekten

Es lassen sich weitere Anforderungen bezüglich zukünftiger, sich noch nicht in Planung befindlicher Projekte festlegen. Dabei soll berücksichtigt werden, welche Komplexität diese aufweisen und welche Funktionalitäten diese besitzen könnten.
Natürlich lassen sich die Spezifikationen von zukünftigen Projekten nicht garantiert definieren. Somit sollen diese hier nur auf Basis von ehemaligen Projekten abgeschätzt werden. Gleichzeitig werden diese Anforderungen etwas weniger stark gewertet, um eventuelle Unsicherheiten zum Teil ausgleichen zu können.
Die Software besitzt sehr oft einen langen Lifecycle mit mehreren Feature-Requests. Zudem wird nicht allzu selten Software für einen ähnlichen Zweck leicht angepasst und wiederverwendet, was sich zum Großteil durch qualitativen Code vereinfachen lässt. Das verwendete Framework kann jedoch auch einen nicht kleinen Einfluss darauf nehmen.
Somit lassen sich schließlich folgende Anforderungen definieren:

- Erweiterbarkeit und Wiederverwendbarkeit
- Wartbarkeit

3.4.4 Zusammenfassung der Anfordungen

Schließlich lassen sich die Anforderungen zu einer einfacheren Übersicht wie folgt zusammenfassen, damit diese in der Diskussion bewertet werden können:

Anfängerfreundlichkeit: Die meisten MATSE, welche eine Ausbildung anfangen, haben relativ wenig Erfahrung. Je einfacher das Framework hierbei zu verstehen ist, desto besser ist es in diesem Fall für eine effiziente Entwicklung.

Angepasstheit: Das Framework sollte für die typischen Aufgabenbereiche der MATSE passend gewählt werden. So sollte kein Framework gewählt werden, welches viele komplexe Funktionen besitzt, daher auch schwieriger zu lernen ist, welche man jedoch nicht für die Projekte benötigt.

Funktionalität: Ein Framework, welches von sich aus mehr Funktionen bereit stellt, ist sicherer, da der Entwickelnde keine inoffiziellen Bibliotheken installieren muss, welche womöglich Sicherheitslücken aufweisen. Durch vorhandene Funktionalität, lässt sich das Framework auch flexibler für viele verschiedene Typen von Projekten einsetzen.

Interne Erfahrung: Will man sich in ein solches Framework einarbeiten, so ist dies einfacher, wenn man Ansprechpartner hat, die einem bei Problemen helfen könnten.

Lesbarkeit: Ein klarer und lesbarer Code lässt sich einfacher verstehen und erweitern. So können auch andere Entwickelnde einfacher an der Codebase arbeiten und die Software so um neue Funktionen erweitern.

Sicherheit: Sicherheitsfunktionen, welche von dem Framework bereitgestellt werden, sorgen für sicherere Software, welche u. a. für die Arbeit mit sensiblen Daten geeignet ist.

Wartbarkeit & Erweiterbarkeit: Soll eine Applikation um eine Funktion erweitert werden, so ist es hilfreich, wenn man diese ohne größere Probleme in ein bereits fertiggestelltes Projekt einbinden kann, ohne dieses besonders stark umstrukturieren zu müssen.

Zukunftssicherheit: Nutzt man ein Framework für ein Projekt, so sollte dieses noch für eine längere Zeit unterstützt werden. Dies ist nicht nur für die Sicherheit wichtig, sondern auch für die Ausbildung der MATSE, welche dieses Wissen somit im späteren Berufsleben gebrauchen können.

3.5 Nutzwert – Vorbereitung

3.5.1 Vorbemerkung

Natürlich besitzt nicht jede Anforderung die gleiche Gewichtung wie alle anderen. So könnte eine Anforderung wichtiger sein als eine andere. Daraus folgt schließlich, dass, wie bei einer Nutzwertanalyse üblich, jede Anforderung einen individuellen Faktor – die Gewichtung – besitzt, welcher den Einfluss auf die finale Wertung bestimmt.
Es sollen nun die einzelnen Anforderungen in Hinblick auf ihre ihrer Wichtigkeit analysiert werden, sodass der Nutzwert in die finale Diskussion mit einbezogen werden kann.

3.5.2 Gewichtungen

Betrachtet man die Ergebnisse aus Unterkapitel 3.2, so lassen sich den verschiedenen Anforderungen eine gewisse Gewichtung zuordnen. Ziel der Gewichtung ist dabei, die Stakeholder mit dem größten Einfluss zufriedenzustellen.
Somit lassen sich die definierten Anforderungen wie folgt bewerten:

Anforderung	Gewichtung (%)
Anfängerfreundlichkeit	15%
Angepasstheit	20%
Funktionalität	10%
Interne Erfahrung	5%
Lesbarkeit	10%
Sicherheit	15%
Wartbarkeit & Erweiterbarkeit	10%
Zukunftssicherheit	15%

Tabelle 3.3: Gewichtung der Anforderungen

3.5.3 Gesamtpunktzahl

In der vorherigen Analyse ließen sich insgesamt acht verschiedene Anforderungen identifizieren. Addiert man für jede Anforderung dabei 100 Punkte zur Gesamtpunktzahl, ergibt sich eine Gesamtpunktzahl von 800 Punkten. Diese Punktzahl kann dabei nur erreicht werden, indem ein Framework in allen Anforderungen und Risiken am besten abschneidet.
Eine weitere Variante wäre, eine feste Punktzahl zu verwenden, welche unabhängig von der Anzahl der Anforderungen ist, z. B. 1000. Im Rahmen dieser Analyse wird jedoch das erstere Verfahren verwendet

3.5.4 Überschneidungen zwischen Anforderungen und Risiken

Betrachtet man die definierten Anforderungen und Risiken, so fallen gewisse Überschneidungen auf. Dabei beziehen sich *Dysfunktionalität* und *Funktionalität* bzw. *Unsicherheit* und *Sicherheit* auf das jeweils denselben Aspekt. Um diese nicht doppelt zu werten, werden somit *Dysfunktionalität* und *Unsicherheit* aus der Bewertung der Risiken entfernt und nur die jeweiligen Anforderungen gewertet.

4 Diskussion der Frameworks

4.1 Angular

4.1.1 Erfüllung der Anforderungen

Anfängerfreundlichkeit: Angular erleichtert es Entwickelnden durch eine ausführliche Dokumentation, sich in das Framework einzuarbeiten. Da Angular jedoch einige komplexere Funktionen wie Routing und Services besitzt, ist die Lernkurve steiler als bei anderen Frameworks. Zudem ist Angular auf TypeScript ausgerichtet, was die Lernkurve erneut erhöht, da dieses etwas komplexer ist als JavaScript. Zwar kann auch JavaScript verwendet werden, dafür muss Angular jedoch zuerst konfiguriert werden.

Angepasstheit: Betrachtet man die typischen Projekte der MATSE, so stellt man fest, dass Angular mit seiner tiefen Funktionalität für viele Projekte zu komplex wäre. Nur wenige Projekte besitzen dabei eine Tiefe, welche für Angular passend wäre, sodass Angular für das Umfeld meist schon zu heavyweight wäre.

Funktionalität: Wie bereits erwähnt, besitzt Angular eine Menge an Funktionen, welche von Haus aus mit dem Framework kommen. Außerdem können APIs von Google, wie z. B. Google Maps und Google Pay, über vorangefertigte Komponenten eingebunden werden. Auch werden über die Angular CLI viele Möglichkeiten zum Testen, Deployen und Warten bereitgestellt [23].

Interne Erfahrung: Im direkten Umfeld gibt es nur einen Entwickelnden, welcher bereits Projekte mit Angular umgesetzt hat. Dieser ist dabei jedoch nicht jederzeit unter der Woche erreichbar, weshalb interne Hilfe somit nur beschränkt verfügbar ist.

Lesbarkeit: Der Code von Angular ist immer in mehrere Dateien unterteilt. So gibt es eine strenge Teilung von Logik, Struktur und Styling in Form der .ts-, .html- und .css-Dateien. Zudem wird ein detaillierter Styleguide bereitgestellt [24], sodass der Code eine – vorausgesetzt, man folgt diesen Konventionen – konsistente Struktur aufweist. Jedoch besitzt Angular viel Boilerplate Code, was besonders für Anfänger sehr komplex wirkt.

Sicherheit: Angular wird mit mehreren Sicherheitsfunktionen gegen typische Sicherheitslücken ausgeliefert [25]. So werden möglicherweise injizierte Script-Tags durch Sanitationsverfahren herausgefiltert, welche man zudem konfigurieren kann. Es werden auch HTTP-Sicherheitslücken bezüglich *Cross-Site Request Forgery (CSRF)* und *Cross-Site Script Inclusion (XSSI)* abgesichert.

Wartbarkeit & Erweiterbarkeit: Komponenten werden einzelnen .html- und .css-Dateien zugeordnet [26]. Innerhalb einer .ts-Datei können dabei Funktionen für verschiedene Komponenten deklariert werden. Dabei lassen sich exportierte Funktionen aus anderen .ts-Dateien importieren, sodass man auch diese wiederverwenden kann.
Die Aufteilung in Komponenten erleichtert das Hinzufügen und Verbessern von Funktionen. Zudem lässt sich mit einem einzigen Befehl das Projekt updaten, wobei außerdem das weitere Vorgehen währenddessen über die Kommandozeile erläutert wird [27].

Zukunftssicherheit: Angular erfreut großer Beliebtheit. Mit knapp 95.000 Stars bzw. 1.500 Issues auf GitHub und 38.000 Stellenanzeigen (auf dem amerikanischen Markt) [28] lässt sich davon ausgehen, dass Angular noch für eine lange Zeit unterstützt wird. Da es außerdem von Google zur Verfügung gestellt wird, ist ein langer Lebenszyklus fast schon garantiert.

4.1.2 Bestimmung der Risikofaktoren

Risiko	Schadensfaktor	Platzierung	Einstufung	Risikofaktor
Komplexität	100	1	1.00	100.00
Kurzlebigkeit	50	2	0.50	25.00

Tabelle 4.1: Risikofaktoren von Angular

Die Schwäche von Angular ist in diesem Kontext die Komplexität des Frameworks. Durch die Menge an Funktionen schneidet Angular zwar besonders gut in der Funktionalität ab, jedoch geht dies für die meisten Projekte schon über das hinaus, was tatsächlich benötigt wird. Dies führt schließlich zu einem unnötig hohen Einarbeitungsaufwand für die jeweiligen Projekte.
Nutzt man schließlich die Formel 2.1, so ergibt sich ein Risikofaktor von **125.00**.

4.1.3 Bestimmung des Nutzwerts

Anforderung	Gewichtung	Platzierung	Einstufung	Punktzahl
Funktionalität	0.10	1	1.00	80
Sicherheit	0.15	1	1.00	120
Wartbarkeit & Erweiterbarkeit	0.10	1	1.00	80
Interne Erfahrung	0.05	2	0.50	20
Lesbarkeit	0.10	2	0.50	40
Zukunftssicherheit	0.15	2	0.50	60
Anfängerfreundlichkeit	0.15	3	0.00	0
Angepasstheit	0.20	3	0.00	0

Tabelle 4.2: Nutzwerte von Angular

Angular zeichnet sich durch seine Funktionalität aus, welche ohne zusätzliche Bibliotheken bereitgestellt wird. Durch die Aufteilung der Komponenten bzw. deren Bestandteile, lässt sich geschriebener Code einfacher nachvollziehen und somit zu verbessern und ermöglicht daher eine nachhaltige Lösung.
Aufgrund von Angulars Komplexität ist dieses Framework jedoch für viele Projekte eine eher ungünstige Wahl. Insbesondere für neue MATSE stellt Angular mit hoher Wahrscheinlichkeit eine relativ große Herausforderung dar, da es durch die Vielzahl an Funktionen und Konzepten am schwierigsten zu erlernen ist.
Durch die Formel 2.3 lässt sich somit nun ein Gesamtnutzwert von **400.00** bestimmen. Betrachtet man dabei die Risiken und verwendet somit die Formel 2.4, ergibt sich eine finale Punktzahl von **275.00**.

4.2 React

4.2.1 Erfüllung der Anforderungen

Anfängerfreundlichkeit: Wie bereits erwähnt, ist React kein wirkliches Framework, sondern eine Bibliothek. Dabei kommt häufig *JavaScript XML (JSX)* bzw. *TypeScript XML (TSX)* zum Einsatz, wodurch Entwickelnde einen direkten Bezug zu JavaScript erhalten, was zu einem zeitgleich wachsendem Verständiss von JavaScript führt und somit besonders für Anfänger hilfreich ist [29]. React unterstützt dabei sowohl JavaScript und TypeScript, was den Einstieg etwas einfacher machen könnte, falls man noch keine Erfahrung mit TypeScript haben sollte. Jedoch können für Anfänger reactspezifische Konzepte wie State, Props und Hooks etwas schwieriger zu verstehen sein [30][31][32].

Angepasstheit: React ist nach Vue wohl am besten an das Umfeld angepasst. Es ist nicht allzu komplex und bietet eine hohe Flexibilität, was ideal für die verschiedenen Projektarten ist und somit eine nachhaltige Lösung darstellt. Außerdem bringt es Anfängern dabei die grundlegenden Konzepte bei der Programmierung mit JavaScript näher, was auch außerhalb von React wichtig ist.

Funktionalität: React stellt an sich nur die Funktionalität für das komponentenbasierte Rendern des UIs bereit. Weitere Funktionen wie das Zustandshandling von Komponenten spielen dem Rendern des UIs zu [30]. Werden weitere komplexere Funktionen benötigt, so muss auf externe Bibliotheken zurückgegriffen werden.
Jedoch gibt es hierbei weniger Limitierungen, welche durch ein Framework eventuell gegeben werden, da React unter anderem nur für das Renders des UIs zuständig ist.

Interne Erfahrung: React ist von den drei Frameworks das einzige, bei welchem es im direkten Umfeld keine näheren Erfahrungen gibt. Somit sind die Entwickelnden momentan auf die offizielle Dokumentation bzw. andere externe Ressourcen angewiesen.

Lesbarkeit: Wie bei den anderen Frameworks arbeitet React mit einem komponentenbasierten System. Die Struktur der Komponenten wird jedoch mittels return zurückgegeben und dem DOM hinzugefügt. Dies erhöht somit im Gegensatz zu z. B. Angular, wo die Struktur in einer separaten .html-Datei gespeichert wird, die Anzahl an Einrückungen. Dies könnte dafür sorgen, dass die generelle Codestruktur etwas schwieriger zu folgen wäre [33].
React verwendet zudem .jsx-Dateien, wodurch man HTML und JavaScript, bzw. TypeScript mittels .tsx-Dateien, innerhalb einer Datei kombinieren kann, was es einfacher macht, die Struktur der Komponenten zu lesen [29].

Sicherheit: React verhindert automatisch XSS-Attacken, indem es jeden Wert, welcher in eine Komponente eingefügt wird, automatisch escaped und somit zu einem reinen String konvertiert, bevor dieser gerendert wird [34]. Damit Werte weiter gesanitized werden, müssen jedoch weitere Bibliotheken verwendet werden.
Reacts Sicherheitsfunktionen verhalten sich dabei gleich wie die von Vue, wodurch man diese auf eine gemeinsame Stufe stellen kann.

Wartbarkeit & Erweiterbarkeit: React besitzt Komponenten, welche vollständig unabhängig vom Rest des Projekts sind. Somit lassen sich diese ggf. für die Implementierung von neuen Funktionen einfacher wiederverwenden bzw. durch die Kapselung warten. Zudem existieren aufgrund der Popularität von React viele Drittanbieter-Tools, welche weitere Funktionen zur Verfügung stellen.
Durch die möglicherweise relativ etwas schwächere Lesbarkeit des Codes, könnte es jedoch etwas schwieriger sein, solche Projekte zu warten.

Zukunftssicherheit: Wie bei Angular steht hinter React ein großes Unternehmen. Zudem ist es mit knapp 222.000 Stars, 800 Issues und 80.000 Stellenanzeigen auf dem amerikanischen Markt am weitesten verbreitet [28]. Man kann daher sagen, dass React in diesem Punkt mit Abstand am besten abschneidet.

4.2.2 Bestimmung der Risikofaktoren

Risiko	Schadensfaktor	Platzierung	Einstufung	Risikofaktor
Komplexität	100	2	0.50	50.00
Kurzlebigkeit	50	3	0.00	0.00

Tabelle 4.3: Risikofaktoren von React

Da React nur eine Bibliothek ist, besitzt es keine tiefen Funktionalitäten, wie es bspw. bei Angular der Fall ist. Einige von Reacts Konzepten können jedoch besonders für Anfänger schwer zu verstehen sein und die Verwendung von JSX bzw. TSX stellt eine weitere Herausforderung dar, die man beim Lernen von React zuerst bewältigen muss.
Wie zuvor verwendet man erneut die Formel 2.1. Der Risikofaktor, welchen man somit für React erhält, beträgt schließlich **50.00**.

4.2.3 Bestimmung des Nutzwerts

Anforderung	Gewichtung	Platzierung	Einstufung	Punktzahl
Zukunftssicherheit	0.15	1	1.00	120
Angepasstheit	0.20	2	0.50	80
Anfängerfreundlichkeit	0.15	2	0.50	60
Funktionalität	0.10	2	0.50	40
Sicherheit	0.15	2	0.50	60
Interne Erfahrung	0.05	3	0.00	0
Lesbarkeit	0.10	3	0.00	0
Wartbarkeit & Erweiterbarkeit	0.10	3	0.00	0

Tabelle 4.4: Nutzwerte von React

Reacts Stärken liegen in der Popularität und – besonders auf das Umfeld angewendet – der vergleichsweise einfachen Verwendung, sodass React nicht nur einfacher als etwa Angular zugänglich ist, sondern auch eine Wissensgrundlage für die generelle Webprogrammierung legt.
Durch 2.3 erhält man einen Gesamtnutzwert von **360.00**, welcher mit der Beachtung der Risiken und somit durch Verwendung der Formel 2.4 einen finalen Wert von **310.00** erhält.

4.3 Vue

4.3.1 Erfüllung der Anforderungen

Anfängerfreundlichkeit: Auch Vue besitzt eine sehr detailreiche Dokumentation. Bei dieser werden jedoch auch auf weitere Lernmaterialien, wie Videokurse und auch Community-Foren, verwiesen. Zudem wird unter anderem auch eine Anleitung zum Migrieren der Projekte von Vue2 auf Vue3 bereitgestellt [35].
Wie bei React kann auch bei Vue sowohl JavaScript, als auch TypeScript ohne große Konfiguration verwendet werden. Somit ist auch hier der Einstieg für Anfänger etwas leichter. Auch durch die simple Struktur der einzelnen Komponenten wirkt Vue dabei intuitiver als React.

Angepasstheit: Da Vue für nicht allzu große Projekte geeignet ist, passt Vue in diesem Punkt eher in das Umfeld als Angular und React. Durch seine Einsteigerfreundlichkeit ist es zudem eine passende Option besonders für neue MATSE, welche meist generell noch wenig Erfahrung besitzen.

Funktionalität: Vue stellt viele grundlegende Funktionen für die Entwicklung dynamischer Webapplikationen zur Verfügung, welche man auch in Angular vorfindet. Komplexere Funktionen wie Routing und State Management lassen sich durch verschiedene Bibliotheken einbinden [36][37]. Jedoch zielt Vue eher darauf ab, besonders lightweight zu sein.

Interne Erfahrung: Im Vergleich zu den anderen beiden Frameworks besitzt Vue hierbei die meiste Erfahrung im Umfeld. Zum jetzigen Zeitpunkt gibt es zwei Entwickelnde, welche Erfahrung mit Vue haben. Davon ist immer mindestens einer erreichbar, sodass es i. d. R. jederzeit verfügbare Hilfe gibt.

Lesbarkeit: Auch hier werden die Komponenten in verschiedene Dateien aufgeteilt. Es ist wie in Angular möglich, Logik, Struktur und Styling in verschiedene Dateien aufzuteilen, jedoch bietet Vue die Möglichkeit an, diese Komponenten in einer einzigen .vue-Datei zu speichern und eine einzelne Komponente somit etwas kompakter zu gestalten. Diese Komponente wird als *Single-File Component (SFC)* bezeichnet [38].
Im Vergleich zu Angular existiert in Vue generell weniger Boilerplate-Code, was den Code somit etwas verständlicher macht. Durch diese Reduzierung an Code wirken Vue-Projekte generell etwas kompakter. Ein Beispiel dafür wäre, dass Direktiven bzgl. des Renderns von Inhalten in Vue direkt in die Tags geschrieben werden [39].
Auch hier werden Konventionen bereitgestellt, sodass der Code und das Projekt eine einheitliche Struktur aufweist und somit für mehr Personen einfacher zu verstehen ist [40].

Sicherheit: Ähnlich wie Angular stellt auch Vue ein paar Funktionen bereit, um den entwickelten Code sicherer zu gestalten. So werden Strings und dynamische Attribute automatisch escaped, sodass Code-Injection verhindert werden kann. HTTP bezogenen Risikien, wie CSRF und XSSI werden dabei nicht von Vue behandelt, da diese zum Backend gehören.[41]
Dies ist wie bei den Sicherheitsfunktionen von React, weshalb sich diese bzgl. dieser Anforderung gleich einstufen lassen.

Wartbarkeit & Erweiterbarkeit: Durch die Einteilung in SFC wird die Wartung deutlich vereinfacht. Diese sorgt nicht nur für eine bessere Kapselung der Daten und somit für ein einfacheres Debugging. Dabei sind die relevanten Bestandteile der Komponenten innerhalb einer einzigen Datei, was ein ständiges Navigieren im Code-Editor verringert und direkt einen Überblick über die jeweilige Komponente gibt.

Zukunftssicherheit: Im Gegensatz zu Angular und React steht hinter Vue kein großes Unternehmen, sondern es wird von seiner Community weiterentwickelt. Auch wenn dieses Frameworks in den letzten Jahren an Beliebtheit gewinnt, fällt dieses Framework bezüglich der Zukunftssicherheit etwas schlechter aus als Angular und React.
Mit knapp 45.000 Stars, 1.000 Issues und 17.000 Stellenanzeigen (amerikanischer Markt) ist Vue auch hier am wenigsten weit verbreitet, was sich mit der jetzigen Entwicklung in den nächsten Jahren jedoch ändern könnte [28].

4.3.2 Bestimmung der Risikofaktoren

Risiko	Schadensfaktor	Platzierung	Einstufung	Risikofaktor
Kurzlebigkeit	50	1	1.00	50.00
Komplexität	100	2	0.50	50.00

Tabelle 4.5: Risikofaktoren von Vue

Vues Schwäche liegt hierbei in dem kleineren Ökosystem als bei den anderen Frameworks. Zwar besitzt es eine Vielzahl an Funktionen, jedoch werden zum jetzigen Zeitpunkt durch die geringere Popularität etwas weniger externe Bibliotheken angeboten, welche die Funktionalität noch erweitern könnten.
Schließlich erhält man mit der Formel 2.1 einen finalen Risikofaktor von **100.00**.

4.3.3 Bestimmung des Nutzwerts

Anforderung	Gewichtung	Platzierung	Einstufung	Punktzahl
Anfängerfreundlichkeit	0.15	1	1.00	120
Angepasstheit	0.20	1	1.00	160
Interne Erfahrung	0.05	1	1.00	40
Lesbarkeit	0.10	1	1.00	80
Funktionalität	0.10	2	0.50	40
Sicherheit	0.15	2	0.50	60
Wartbarkeit & Erweiterbarkeit	0.10	2	0.50	40
Zukunftssicherheit	0.15	3	0.00	0

Tabelle 4.6: Nutzwerte von Vue

Die Simplizität und Anfängerfreundlichkeit dieses Frameworks ist die Stärke von Vue. Dieses kann somit schnell erlernt werden, wodurch es ohne lange Einarbeitung für die Entwicklung von Projekten verwendet werden kann.
Mittels Formel 2.3 besitzt auch Vue einen Gesamtnutzwert von **540.00**. Dieser ergibt durch Formel 2.4 einen finalen Wert von **440.00**.

5 Fazit

5.1 Zusammenfassung der Ergebnisse

In dieser Seminararbeit wurde mithilfe von verschiedenen Analyseverfahren eine vergleichende Bewertung der Frameworks Angular, React und Vue durchgeführt. Dabei wurden unter Berücksichtigung des Umfeldes und der Interessen verschiedener Stakeholder diverse Anforderungen an die Frameworks definiert und anhand ihrer Relevanz gewichtet. Um potenzielle Schwächen der Frameworks in die Bewertung einzubeziehen, wurden schließlich verschiedene Risiken analysiert und in der finalen Bewertung beachtet. Indem man die Ergebnisse dieser Analysen verwendete, konnte somit festgestellt werden, welches dieser Frameworks sich am ehesten für das Umfeld des WZL der RWTH Aachen University eignet.
Am Ende dieser Analyse konnte sich schließlich Vue durchsetzen. Auch wenn es die geringste Popularität der verglichenen Frameworks besitzt, zeigt sich seine Stärke in der flachen Lernkurve. Somit eignet sich Vue für die Entwicklung von kleinen Projekten, welche im Umfeld hauptsächlich benötigt werden und durch die kompakte Struktur lässt es sich auch über einen längeren Zeitpunkt besser warten und erweitern.
React konnte sich mit seiner Flexibilität und seinem breiten Ökosystem hinter Vue einordnen. Zwar ist es für die verschiedenen Arten von Projekten geeignet, jedoch ist es aufgrund mancher seiner Konzepte schlechter für Anfänger geeignet, weshalb es nicht am besten auf das Umfeld angepasst ist.
Angular reiht sich daher unten ein. Es besitzt eine tiefe Funktionalität und ermöglicht es somit, besonders große und robuste Software zu schreiben. In dem hier betrachteten Umfeld ist eine solche Robustheit jedoch nicht benötigt, weshalb sich Angular am wenigsten für die Projekte in diesem Umfeld eignen würde.

Somit kann die Frage, welches der TypeScript-Frameworks – Angular, React und Vue – sich für das Umfeld des WZL der RWTH Aachen University am besten eignet, mit **Vue** beantwortet werden. Dieses Ergebnis lässt sich dabei als Grundstein auf Basis von theoretischen Ergebnissen sehen. Inwiefern sich dies schließlich in der Praxis bewahrheitet, kann sich nur mit der Zeit zeigen.

5.2 Reflexion und Ausblick

Natürlich gibt es in dieser Analyse viel Raum für Optimierungen. Es wurde viel mit der Dokumentation der Frameworks und weiteren Beiträgen gearbeitet, weshalb in diese Analyse zum Großteil Theorie eingeflossen ist. Indem man sich über einen längeren Zeitraum in die Frameworks einarbeitet, ließen sich diese auch anhand der Praxis genauer vergleichen. Beispielsweise könnte man das gleiche Projekt in den Frameworks umsetzen und den geschriebenen Code bzw. die Erfahrungen, welche man bei der Entwicklung gemacht hat, in die Analyse mit einbeziehen. Desweiteren wäre eine zusätzliche Möglichkeit, die Analyse um ein mögliches Endergebnis zu erweitern. Wie bereits in Abschnitt 3.1.2 erwähnt, wurden alle vorherigen Projekte ohne jegliches Framework umgesetzt. Somit könnte man auch über die Eignung der Variante „kein Framework“ diskutieren.
Trotzdem wurde das Ziel, die drei bekannten Frameworks in einem ersten Schritt hinsichtlich ihrer theoretischen Eignung für das gegebene Umfeld miteinander zu vergleichen, erreicht und eine fundierte Grundlage für weitere Untersuchungen bzw. praktische Anwendungen geschaffen.

A Literatur

[1] Jenna Thorne. *Angular vs React vs Vue: Core Differences*. 2023. URL: https://blog.openreplay.com/single-page-apps-vs-multiple-page-apps/.

[2] *What is a Web Application?* Amazon. URL: https://aws.amazon.com/de/what-is/web-application/.

[3] Mohit Joshi. *Single Page Applications (SPA) Vs. Multi-Page Applications (MPA)*. 2023. URL: https://www.browserstack.com/guide/angular-vs-react-vs-vue.

[4] *What's the Difference Between PWA & SPA*. Mozilla. URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Guides/What_is_a_progressive_web_app.

[5] LTS Group. *A Detailed Comparison Between A Web App Vs Website*. 2023. URL: https://www.linkedin.com/pulse/detailed-comparison-between-web-app-vs-website-lts-group-vietnam-4j2fc.

[6] *Web App: 8 Web Application Vorteile*. Bright Solutions. URL: https://www.brightsolutions.de/blog/web-app-web-application/.

[7] *Webframeworks – Überblick und Klassifizierung*. Ionos. 2022. URL: https://www.ionos.de/digitalguide/websites/web-entwicklung/webframeworks-ein-ueberblick/.

[8] *Server-side web frameworks*. Mozilla. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks.

[9] *Introduction to client-side frameworks*. Mozilla. URL: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction.

[10] Dave Gavigan. „The History of Angular“. In: *Medium* (2018).

[11] Vlad Koval. „Is Angular dead?“ In: *Medium* (2024).

[12] *React Introduction*. GeeksforGeeks. 2024. URL: https://www.geeksforgeeks.org/reactjs-introduction/.

[13] MadhushaPrasad. „Vue.js history“. In: *Medium* (2023).

[14] *Vue.js – Ways of Using Vue*. URL: https://vuejs.org/guide/extras/ways-of-using-vue.

[15] *Vue.js – Rendering Mechanism*. URL: https://vuejs.org/guide/extras/rendering-mechanism.

[16] *Software Requirements Specifications*. IEEE Computer Society. URL: https://www.computer.org/resources/software-requirements-specifications.

[17] *Umfeldanalyse: Definition und Umsetzung im Überblick!* Asana. 2024. URL: https://asana.com/de/resources/project-environment-analysis.

[18] R. Edward Freeman. *Strategic Management: A Stakeholder Approach*. Pitman Publishing, 1984.

[19] *Organisationshandbuch – Stakeholderanalyse*. Bundesministerium des Innern und für Heimat. URL: https://www.orghandbuch.de/Webs/OHB/DE/OrganisationshandbuchNEU/4_MethodenUndTechniken/Methoden_A_bis_Z/Stakeholderanalyse/Stakeholderanalyse_node.html.

[20] *Organisationshandbuch – Risikoanalyse*. Bundesministerium des Innern und für Heimat. URL: https://www.orghandbuch.de/Webs/OHB/DE/OrganisationshandbuchNEU/4_MethodenUndTechniken/Methoden_A_bis_Z/Risikoanalyse/risikoanalyse_node.html.

[21] Frank Romeike. *Risikomanagement*. Springer Gabler, 2018.

[22] *Nutzwertanalyse einfach erklärt*. Munich Business School. URL: https://www.munich-business-school.de/l/bwl-lexikon/nutzwertanalyse.

[23] *Angular – CLI*. Google. URL: https://angular.dev/tools/cli.

[24] *Angular – Angular coding style guide*. Google. URL: https://angular.dev/style-guide.

[25] *Angular – Secruity*. Google. URL: https://angular.dev/best-practices/security.

[26] *Angular – Components*. Google. URL: https://angular.dev/guide/components.

[27] *Angular – Keeping your Angular projects up-to-date*. Google. URL: https://angular.dev/update.

[28] Andrei Neagoie. *Angular vs React vs Vue: The Best Framework for 2024 is...* 2024. URL: https://zerotomastery.io/blog/angular-vs-react-vs-vue/.

[29] *React – Writing Markup with JSX*. Meta. URL: https://react.dev/learn/writing-markup-with-jsx.

[30] *React – State: A Component's Memory*. Meta. URL: https://react.dev/learn/state-a-components-memory.

[31] *React – Props*. Meta. URL: htthttps://react.dev/reference/react/Component#props.

[32] *React – Built-in React Hooks*. Meta. URL: https://react.dev/reference/react/hooks.

[33] *React – Your First Component*. Meta. URL: https://react.dev/learn/your-first-component.

[34] *React – JSX Prevents Injection Attacks*. Meta. URL: https://legacy.reactjs.org/docs/introducing-jsx.html#jsx-prevents-injection-attacks.

[35] *Vue.js – Migration Build*. URL: https://v3-migration.vuejs.org/migration-build.

[36] *Vue Router*. URL: https://router.vuejs.org/.

[37] *What is Vuex?* URL: https://vuex.vuejs.org/.

[38] *Vue.js – Single-File Components*. URL: https://vuejs.org/guide/scaling-up/sfc.

[39] *Vue.js – Conditional Rendering*. URL: https://vuejs.org/guide/essentials/conditional.html.

[40] *Vue.js – Conventions and Best Practices*. URL: https://vuejs.org/guide/reusability/composables.html#conventions-and-best-practices.

[41] *Vue.js – Security*. URL: https://vuejs.org/guide/best-practices/security.

B Tabellenverzeichnis

C Abbildungsverzeichnis