Vergleich moderner Webtechnologien zum Zwecke der Programmierung eines Zeitmanagementsystems zugeschnitten für Musiker in Form einer Server-Client Applikation

Bachelorarbeit

eingereicht von: **Simon Andreas Brunner**
Matrikelnummer: 11778939

im Fachhochschul-Bachelorstudiengang Wirtschaftsinformatik (0470)
der Ferdinand Porsche FernFH

zur Erlangung des akademischen Grades eines

Bachelor of Arts in Business

Betreuung und Beurteilung: DI Eszter Geresics-Földi, MSc BSc

Wiener Neustadt, September 2022

Ehrenwörtliche Erklärung

Ich versichere hiermit,

1. dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Inhalte, die direkt oder indirekt aus fremden Quellen entnommen sind, sind durch entsprechende Quellenangaben gekennzeichnet.
2. dass ich diese Bachelorarbeit bisher weder im Inland noch im Ausland in irgendeiner Form als Prüfungsarbeit zur Beurteilung vorgelegt oder veröffentlicht habe.

Lienz, 23.09.2022

Unterschrift

Creative Commons Lizenz

Das Urheberrecht der vorliegenden Arbeit liegt beim Autor. Sofern nicht anders angegeben, sind die Inhalte unter einer Creative Commons „Namensnennung – Nicht-kommerziell – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz“ (CC BY-NC-SA 4.0) lizenziert.
Die Rechte an zitierten Abbildungen liegen bei den in der jeweiligen Quellenangabe genannten Urheber*innen.

Die Kapitel <1 bis 3> der vorliegenden Bachelorarbeit wurden im Rahmen der Lehrveranstaltung „Bachelor Seminar 1“ eingereicht und am 31.8.2022 als Bachelorarbeit 1 angenommen.

Kurzzusammenfassung: Vergleich moderner Webtechnologien zum Zwecke der Programmierung eines Zeitmanagementsystems, zugeschnitten für Musiker, in Form einer Server-Client Applikation

Musikschaffende unterliegen zunehmend Problemen des Zeitmanagements. Die vorliegende Arbeit beschäftigt sich damit, Datenbanktechnologien, Backendtechnologien und Frontendtechnologien im Hinblick auf die Implementierung einer Server-Client Applikation zum synchronen Zeitmanagement von Musikern und Veranstaltern gegeneinander zu stellen und die dafür geeignetsten auszuwählen. Für jeden Teilaspekt des sogenannten Solution Stacks wurden jeweils vier moderne Technologien vorgestellt, deren Stärken und Schwächen klassifiziert und durch die der Skizze der Applikation entnommenen Kriterien nach ihrer Eignung gereiht. Dabei wurde das Weighted Scoring Model unter der Anwendung des „Direct Scoring“ verwendet. Die Kombination aus MongoDB, ExpressJS, Angular und NodeJS erwies sich nach der Recherche und der Anwendung der wissenschaftlichen Methode als der geeignetste Solution Stack. Um die Rahmenbedingungen abzugrenzen, wurde die Applikation sowohl in funktionalen wie in nicht-funktionalen Anforderungen definiert. Use-Cases und ein Entity-Relationship Model ergänzten die Grenzen und sorgten für eine klare Definition der Kriterien an den Solution Stack.

Schlagwörter:

Solution-Stack, Musikermanagementsoftware, Server-Client-Applikation, Datenbanktechnologien, Backendtechnologien, Frontendtechnologien

Abstract: Comparison of modern web technologies for the purpose of programming a time management system, tailored for musicians, in the form of a server-client application.

Music creators are increasingly subject to time management problems. The present work deals with the comparison of database technologies, backend technologies and frontend technologies with regard to the implementation of a server-client application for synchronous time management of musicians and event organizers and the selection of the most suitable ones. For each aspect of the so-called solution stack, four modern technologies were presented, their strengths and weaknesses classified, and ranked according to their suitability using criteria taken from the application outline. In doing so, the Weighted Scoring Model was used with the application of "Direct Scoring". The combination of MongoDB, ExpressJS, Angular and NodeJS was found to be the most suitable solution stack after researching and applying the scientific method. To delineate the framework, the application was defined in both functional and non-functional requirements. Use cases and an entity-relationship model supplemented the boundaries and provided a clear definition of the criteria for the solution stack.

Keywords:

solution-stack, artist-management software, server-client-application, databasetechnologies, backend technologies, frontend technologies

Inhaltsverzeichnis

3. EINLEITUNG 5

2. GRUNDLAGEN UND AKTUELLER STAND DER WISSENSCHAFT UND FORSCHUNG 6

2.1 VORSTELLUNG MODERNER DATENBANKTECHNOLOGIEN 11
2.1.1 PostgreSQL 11
2.1.2 MongoDB 13
2.1.3 MariaDB 15
2.1.4 SQLite 17
2.2 VORSTELLUNG MODERNER BACKENDTECHNOLOGIEN 18
2.2.1 Java Spring 18
2.2.2 Node.js + ExpressJS 20
2.2.3 Python 23
2.2.4 C# 24
2.3 VORSTELLUNG MODERNER FRONTENDTECHNOLOGIEN 26
2.3.1 React 26
2.3.2 Angular 28
2.3.3 Svelte 35
2.3.4 VueJS 37

3. DER BAND MANAGEMENT SERVER (BMS) 38

3.1 EINLEITUNG UND DEFINITION 38
3.2 ENTITY RELATION SHIP MODELL DES BMS 44
3.3 ANFORDERUNGSKATALOG ABSTRAHIERT AUS USE CASES, ER UND PRODUKTBESCHREIBUNG 45
3.3.1 Funktionale Kernanforderungen 45
3.3.2 Nichtfunktionale Anforderungen 47
3.4 KRITERIENLISTE MIT GEWICHTUNG AUF BASIS DES ANFORDERUNGSKATALOGS 48

4. SKIZZE DES WEIGHTED SCORING MODELS 49

4.1 ANWENDUNG DES WEIGHTED SCORING MODELS AUF DIE UNTER PUNKT 2 VORGESTELLTEN DATENBANK-, BACKEND- UND FRONTENDTECHNOLOGIEN 50
4.1.1 Anwendung des WSM auf die unter Punkt 2 beschriebenen Datenbanktechnologien 50
4.1.2 Anwendung des WSM auf die unter Punkt 2 beschriebenen Backendtechnologien 52
4.1.3 Anwendung des WSM auf die unter Punkt 2 beschriebenen Frontendtechnologien 55
4.2 ANALYSE DER ERGEBNISSE UND DEFINITION DES ENTSTANDENEN STACKS 58
4.2.1 Ergebnisse des WSM: Datenbanktechnologien 58
4.2.2 Ergebnisse des WSM: Backendtechnologien 58
4.2.3 Ergebnisse des WSM Frontendtechnologien 58
4.2.4 Stackdefinition 58

5. CONCLUSIO UND BEANTWORTUNG DER FORSCHUNGSFRAGE 59

LITERATURVERZEICHNIS: 60

ABBILDUNGSVERZEICHNIS 63

3. Einleitung

Zur Programmierung gelangt eine Server-Client Applikation, welche betriebssystem-unabhängig das monochrone Management von Musikgruppen und einzelnen Musikern zum Zwecke hat.
Eine Applikation dieser Art muss naturgemäß über das Internet erreichbar sein, da die Datensynchronisierung ohne dies unmöglich wäre. Programme oder Applikationen übernehmen im Grunde genommen meistens ähnliche Hauptaufgaben.
Daten werden von einem User, einem anderen Programm oder einer Maschine in ein System geführt, dort verarbeitet und an anderer Stelle wieder ausgegeben. Dabei können drei Hauptbereiche unterschieden werden. Daten müssen persistent gespeichert, abgerufen und manipuliert werden können. Logiken und Algorithmen sind von Nöten, um einen vorhandenen Datensatz einem Zieldatensatz anzupassen und Anfragen zu verwalten. Für Programme, welche von Personen bedient werden sollen, ist ein User Interface oder kurz UI unabdingbar.
Bei klassischen Desktop Applikationen, wie zum Beispiel MS-Excel sind die Programmschichten der Datenpersistenz, der Logik und der Präsentation in einer Einheit und Instanz an einem Gerät verwirklicht. Das Programm verwaltet dabei seinen eigenen Mikrokosmos aus Userdaten und kommuniziert dabei nicht mit anderen Rechnern.
Webapplikationen hingegen verteilen diese Funktionalitäten. Zur Datenpersistenz wird eine Datenbanktechnologie benötigt. Ein Webserver ist essenziell, um Anfragen von Benutzern und Maschinen zu verarbeiten sowie Daten von der Datenbank auszulesen, zu schreiben oder auch zu überschreiben oder zu löschen und an diverse Präsentationsschichten weiterzugeben. Die Visualisierung dieser Daten sowie die Steuerung des Servers wird über eine Frontend Applikation an den Client Geräten realisiert. Der Verbund dieser Softwarekomponenten wird „Solution-Stack“ oder einfach nur „Stack“ genannt.
Ziel der Arbeit ist der Vergleich von Backend-, Frontend-, und Datenbanktechnologien zum Zwecke der Wahl der Geeignetsten in Anbetracht der Implementierung einer Server-Client Applikation zum Zeitmanagement einzelner Musiker, welche mehreren Gruppierungen angehörig sind. Dazu werden für jeden Aufgabenbereich jeweils vier moderne Softwarekomponenten vorgestellt und anschließend auf die Kompatibilität im Verbund mit anderen gewichtet. Solution Stacks wie „LAMP“ oder „WINS“ werden aufgrund ihrer Betriebssystemabhängigkeit nicht in den Vergleich miteinbezogen, da dies dem gesetzten Ziel zum Widerspruch läge. Eine genauere Definition des Sollproduktes (fortan Band Management Server) ist zweckmäßig, um die Softwarekomponenten an einem gegebenen Ziel zu messen („For Fast and Secure Sites | Jamstack“ o. J.)

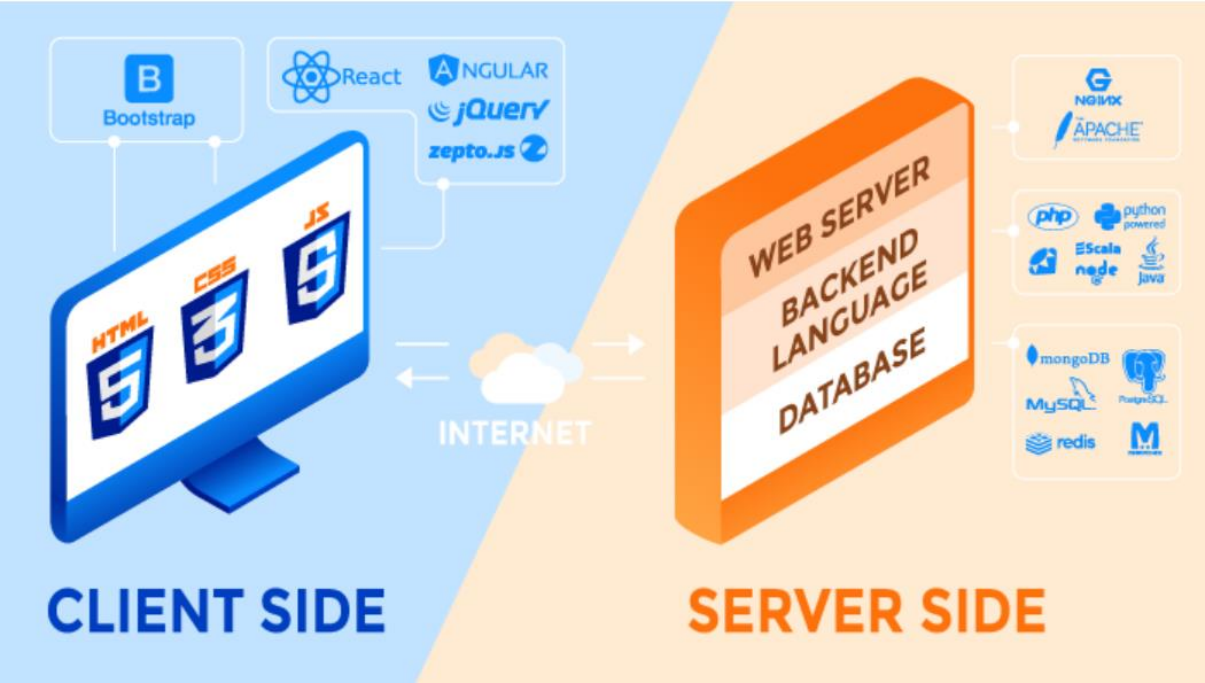

Abbildung 1 Solution Stack
Quelle: Rubygarage.org 2022

Die vorliegende Arbeit befasst sich mit der folgenden Forschungsfrage:

Welche Datenbank-, Backend- und Frontendtechnologien sind zur Implementierung einer Server-Client Applikation zur musikerformationsübergreifenden Zeitplanung die Geeignetsten?

2. Grundlagen und aktueller Stand der Wissenschaft und Forschung

Im Gegensatz zur klassischen Desktop Anwendung werden Webapplikationen im sogenannten „Server-Client“ Modell erstellt. Server und Client kommunizieren dabei über den „Representational State Transfer“ oder kurz REST. Der Server bietet dabei REST-APIs (API = Application Programming Interface) an, welche vom Client angesprochen werden können – damit können unter anderem CRUD Prozesse (Create, Read, Update, Delete) und Multipart Requests (z.B. für einen Fileupload) oder auch AJAX Calls (AJAX = Asynchronus Javascript and XML) umgesetzt werden. In Richtung Server -> Client wird die Kommunikation über sogenannte „Web-Sockets“ ermöglicht. Dabei überwacht der Client eine Schnittstelle und setzt beim Eintreffen eines Events eine hinterlegte Logik (Callback Function) in Gang.

Wird am Client Gerät der URL (Uniform Resource Locator) des Webservers eingegeben, so sendet der Clientrechner einen Priorität-Request (Priorität = Hypertext Transfer Protocol) an den Webserver, welcher in einem Priorität-Response den HTML (Hyper Text Markup Language) Quellcode an den Client ausgibt. Dabei stellt der Browser der Clientmaschine die Präsentationsschicht dar, während der Server den Großteil der Businesslogik und die Datenpersistenz übernimmt. Der große Vorteil in der Webanwendung liegt darin, dass beliebig viele Clientgeräte betrieben werden können, und diese dafür in der Regel lediglich einen Webbrowser sowie eine aufrechte Verbindung zum Internet benötigen. In der Fachsprache wird der serverseitige Teil der Applikation Backend und der clientseitige Teil Frontend genannt. (Helmich o. J.)

Da jede aktive „Session“ eines Users eine zusätzliche Last für den Server bedeutet, hat sich das Design clientseitig hin zur SPA (Single Page Application) gewandt. Eine SPA besteht aus einem einzigen HTML Quelldokument und lädt die korrespondierenden Inhalte „lazy“ – also dynamisch nach. Der „State“ – also der Zustand der jeweiligen Session wird dabei nicht mehr am Server, sondern am Client gespeichert, wodurch eine aufwändige Emulation von States über Cookies durch den Server obsolet wird. Dadurch kann die Last für den Server bei hohen Userzahlen drastisch reduziert werden, da durch das Persistieren von Daten auf einen Webstorage durch den Client jeder Serverknoten antworten kann. Somit ist der Webclient vom Server relativ unabhängig und kann auf Userinteraktionen selbständig reagieren, was die Anzahl an Priorität Cycles um ein Vielfaches reduziert, und dabei schnellere Responsezeiten zur Konsequenz hat und folglich die „Userexperience“ verbessert. Durch die Tatsache, dass es sich lediglich um ein einziges HTML Dokument handelt, kann auf serverseitiges Routing verzichtet werden, wodurch eine permanente Web-Socket Verbindung ermöglicht wird. Frontendframeworks wie „Angular“, „Reactjs“ und „Vue.js“ sind Werkzeuge, welche auf Single Page Applications spezialisiert sind. Durch das Vorhalten des States am Client sind SPAs bis zu einem gewissen Grad auch offline nutzbar. Werden keine Daten vom Server benötigt, kann die Frontendapplikation Standalone genutzt werden. Seit 2015 ist auch die Umsetzung einer SPA in Form einer PWA (Progressive Web-App) möglich – diese stellt eine Symbiose aus einer responsiven Web-Applikation und einer nativen Applikation dar. Implementierte Service Workers übernehmen dabei das Caching für die Offline-Funktionalitäten (einmal abgerufene Daten werden dabei lokal vorgehalten und damit auch offline verfügbar gemacht). PWAs können direkt aus dem Browser mittels „Add to Homescreen“ Funktion installiert werden und umgehen damit eine sonst obligatorische Anwesenheitspflicht in einem App-Store, setzen jedoch das HTTPS-Protocol voraus. Das App Icon und alle anderen Inhalte, welche für die native Applikation notwendig sind, werden dabei aus einer Manifest Datei bezogen.

PWAs können Elemente des Client-Betriebssystems ansprechen, was reinen SPAs nicht möglich ist. So können PWAs zum Beispiel Push Notifications an den Homescreen des Smartphones oder Tablets ausgeben. („Single-page Webanwendungen“ 2015)(Liebel o. J.)

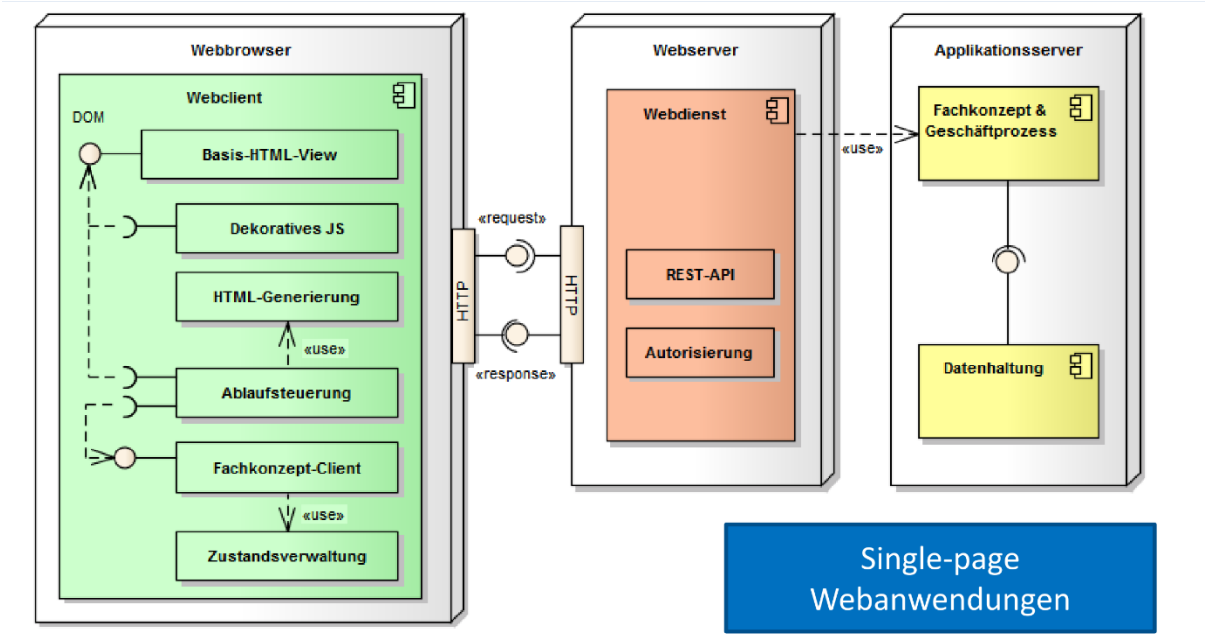

Abbildung 2 SPA Scheme
Quelle: https://www.smf.de/pdf/Single-page_Webanwendungen_2015.pdf

Als Transferobjekte haben sich Daten im JSON (Javascript Object Notation) Format etabliert. Die Vorteile gegenüber dem älteren XML liegen sowohl in der möglichen Typisierung wie auch in der Lesbarkeit. JSON Dateien sind unabhängig von den eingesetzten Programmiersprachen und können mittels „Parser“-Libraries in beinahe allen gängigen Sprachen verwendet werden. JSON kann von Javascript-basierenden Webapplikationen direkt konsumiert werden, wodurch hier der „Parse“ Prozess entfällt. („Getting Started Step-By-Step“ o. J.)

```
{
  "productId": 1,
  "productName": "A green door",
  "price": 12.50,
  "tags": [ "home", "green" ]
}
```

Abbildung 3 JSON Example
Quelle: https://json-schema.org/learn/getting-started-step-by-step

Moderne Webanwendungen setzen vermehrt auf die Programmierparadigmen „Event-Driven-Architecture“ und „Reactive-Programming“. Das bedeutet, dass die Datenverarbeitung auf sogenannten „Streams“ basiert. Tritt eine Änderung der Daten in Kraft, wird diese automatisch im Ausführungsmodell propagiert. Dies ermöglicht eine konsequente Trennung der Businesslogik, der Datenhaltung und der Darstellung. Ein Beispiel einer Applikation könnte wie folgt aussehen:

State Management: NgRx Store
Frontendframework: Angular
Reactive Library: RxJs

Der Store beinhaltet einen Ring-Pattern. Er besteht aus „Actions“, „Reducers“, „Effects“ und „Selectors“. Dabei dürfen ausschließlich Reducers das State-Object verändern. Eine Implementierung einer Datenladung könnte wie folgt aussehen:

Eine Service Class der Applikation „dispatched“ in ihrem Constructor die Action „loadData“ an den Store. „loadData“ triggert den Effect „loadData$“, welcher über eine weitere Service Class (ApiService) über die REST-API das Backend nach Daten abfragt. Kommen diese Daten über den Priorität-Response retour wird die Action „loadDataSuccess“ mit dem Body des Priorität-Responses als Parameter „dispatched“. Eine Reducer Class überwacht das Actions Observable des Stores und wird in der Pipe durch die Methode „ofType(loadDataSuccess)“ aktiv. Der Reducer verändert nun das State Object, welches dieser als „initialState“ implementiert.

Bsp: initialState = {data: []}. Nach der Manipulation könnte das State Object wie folgt aussehen:

```
state = {
    data: [
            myData: ‚yes‘,
            myBoolean: false,
        ]
}
```

Eine Klasse DataService könnte nun über den Selector des Stores sich auf das Read-Only Observable „myData$“ „subscriben“ und dieses asynchron an Components der Applikation ausliefern. Das Frontendframework „Angular“ bietet dafür mit dem Databinding über die „Async Pipe“ auf Subkomponenten und der Möglichkeit das Service „DataService“ an jeder Stelle mittels Dependency Injection zu implementieren bereits die vollständige Funktionalität. Die Action „loadData“ kann an jeder Stelle des Programms „dispatched“ werden, „myData$“ gibt aber immer den aktuellen Wert an alle Subscriber weiter, wodurch die Datenrepräsentation an allen Stellen der Software immer „Up to Date“ ist. Dadurch wird eine Zentralisierung der Datenhaltung und des Application-States geschaffen. Verschiedene Darstellungen der Daten können durch RxJs Operatoren wie z.B. „.pipe()“ und „map()“ realisiert werden, „zeigen“ jedoch immer auf dasselbe Ausgangsobjekt, wodurch Diskrepanzen zur Laufzeit ausgeschlossen werden können. („NgRx“ o. J.; „Introduction | RxJS - Javascript library for functional reactive programming.“ o. J.; Buckel 2012)

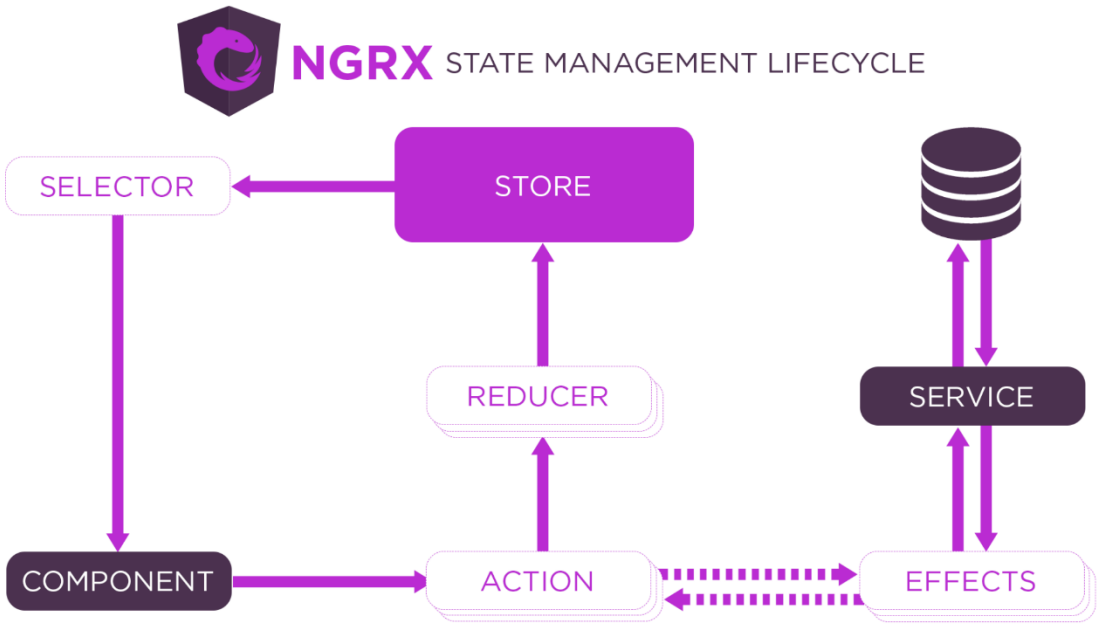

Abbildung 4 NGRX State Management Lifecycle
Quelle: https://ngrx.io/guide/store

2.1 Vorstellung moderner Datenbanktechnologien

2.1.1 PostgreSQL

Als ORDBMS (Objektrationales Datenbankmanagementsystem) verwendet Postgres weitgehend den SQL-Standard. Postgres ist ACID-konform und wurde seit dem Jahr 1996 stetig weiterentwickelt. Es werden sowohl erweiterbare Datentypen, Aggregate als auch Operatoren unterstützt. Laut eines Versuches der Universität Athen in Kooperation mit der Universität von London ist PostgreSQL im Punkto Performanz seinem Konkurrenten MongoDB weit überlegen. („PostgreSQL: About“ o. J.)(Makris u. a., o. J.)

Pro:

PostgreSQL ist das bekannteste Datenbankmanagementsystem, weswegen die meisten externen Tools perfekt integriert sind. Selbst für Big Data Anwendungen hat sich das System bewehrt. Die Technologie wird seit 1995 entwickelt und bietet daher äußerst ausgereifte Funktionen und Schnittstellen wie JDBC (Java Database Connectivity) und ODBC (Open Database Connectivity). Queries können parallelisiert werden und damit auf alle verfügbaren Prozessorkerne aufgeteilt werden, was die Performanz enorm steigert. PostgreSQL bietet

Rich SQL an und kann auch mit unstrukturierten Daten, wie zum Beispiel aus einer MongoDB-Instanz umgehen. Das System ist partitionierfähig und kann dem entsprechend auf mehrere Server aufgeteilt werden. Der bei weitem am häufigsten genannte Vorteil von PostgreSQL ist die Effizienz seines zentralen Algorithmus, der viele Datenbanken, die als fortschrittlicher angepriesen werden, übertrifft. Dies ist besonders nützlich, wenn sie mit großen Datenmengen arbeiten, bei denen E/A-Prozesse sonst zu einem Engpass werden können. PostgreSQL ist sehr flexibel, so können Funktionen mit vielen serverseitigen Sprachen, darunter C, R, Java, Ruby, Perl und Python, geschrieben werden. („Comparing 3 Open Source Databases: PostgreSQL, MariaDB, and SQLite | Opensource.Com“ o. J.)

+ etabliert
+ effizient
+ flexibel
+ ausgereift
+ parallelisierte Queries
+ Rich SQL
+ partitionierfähig
+ sicher
+ Open Source

Kontra:

PostgreSQL komprimiert gespeicherte Daten nicht. Das führt zu einer hohen Speicherauslastung und kann bei großen Datenmengen zu Performanzeinbußen führen. Zum jetzigen Zeitpunkt inkludiert PostgreSQL keine Machine Learning Funktionen und zwingt den Entwickler zum Verwenden externer Software. Postgres kann mit großen Datenmengen sehr gut umgehen, für kleine Datenmengen existieren allerdings schnellere Tools. Clustering kann zurzeit nur durch Third Party Plugins realisiert werden, auch wenn es Ziel ist, diese Funktionen in Postgres zu implementieren, so ist dies bis jetzt noch nicht geschehen. („Comparing 3 Open Source Databases: PostgreSQL, MariaDB, and SQLite | Opensource.Com“ o. J.)

- Hohe Speicherauslastung
- kein integriertes Machine Learning
- für kleinere Datenmengen gibt es schnellere Tools
- Clustering erfordert Third Party Software

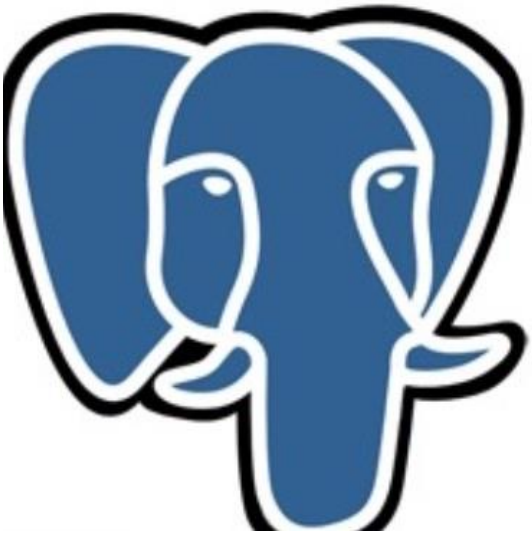

Abbildung 5 PostgreSQL Logo
Quelle: postgresql.org

2.1.2 MongoDB

MongoDB ist ein dokumentenbasiertes NoSQL Datenbankmanagementsystem, welches von den Firmen MongoDB Inc. Und Atlas seit 2007 entwickelt und gewartet wird. Das erste lauffähige Inkrement erschien im Jahr 2009. Die Technologie verwendet JSON-ähnliche Strukturen. MongoDB ist das verbreitetste NoSQL Datenbankmanagementsystem weltweit. Seit 2018 wird MongoDB nichtmehr als Open Source Software, sondern unter der proprietären SSPL Lizenz zur Verfügung gestellt – diese zwingt Entwickler, wenn sie Dienste von MongoDB an Dritte bereitstellt, zur Veröffentlichung jedweder Software, Schnittstellen sowie des gesamten Quellcodes. (Horowitz o. J.). Die Software ist darauf ausgelegt auf mehreren Servern gleichzeitig zu laufen und stellt dabei die Datenkonsistenz über die Verfügbarkeit. Um dennoch eine hohe Verfügbarkeit zu gewährleisten, legt MongoDB Replikationen durch Sharding auf mehreren Servern an, was jedoch in einem gewissen Zeitfenster einen nach einem bestätigten Schreibzugriff getätigtem Lesezugriff inaktuelle Daten zuspielt. Im Gegensatz zu relationalen Datenbanken, bei denen die Struktur eines Eintrages durch das Schema der Tabelle genau vorgegeben ist, besitzt MongoDB Schemafreiheit – das bedeutet, dass sich selbst Anzahl und Art der Attribute von Einträgen in derselben Collection voneinander unterscheiden können. (Trelle 2014) („Schemaless Database | MongoDB Blog“ o. J.)

Pro:

Dem dokumentenbasierten Organisationselement von MongoDB entspringt auch ihr größter Vorteil, die Flexibilität. Werden nachträglich Änderungen an der Struktur oder den Schemas einer relationalen Datenbank getätigt, führt dies oft zu sehr aufwändigen Migrationen. Im Falle von MongoDB werden in einem Objekt einfach andere Attribute abgespeichert. Ein weiterer Vorteil von MongoDB ist das sogenannte Sharding – Datensets werden auf n Server verteilt – dies führt zu einer Lastverteilung und erhöht die Stabilität. Das Datenbankmanagementsystem akzeptiert des Weiteren auch „ad-hoc"-Queries. Das System nutzt auch RAM caching, was zu einer erheblichen Performanz Steigerung führt, solange der Server über die entsprechende Hardware verfügt.

+ Flexibel
+ ad-hoc Queries
+ Ram caching
+ Sharding
+ JSON Synergien

Kontra:

MongoDB unterstützt aufgrund seiner Architektur „joins" nur äußerst begrenzt. Sie müssen manuell über Code implementiert werden, was erhebliche Zeiteinbußen zur Folge und auch negative Auswirkungen auf die Performanz der Datenbank hat. Des Weiteren der Architektur geschuldet, werden eine Vielzahl von Daten dupliziert und redundant gehalten, was sich negativ auf die Wartbarkeit sowie die Speicherinanspruchnahme auswirkt. Werden Dokumente nicht perfekt indiziert führt dies zu Datendiskrepanzen, Fehlern und die Abfragelaufzeit wird deutlich erhöht. Das Sharding, auch wenn es Vorteile hat, führt zu einem Zeitfenster nach einem Schreibzugriff, in welchem folgende Lesezugriffe Legacy Data, also nicht den aktuellen Stand liefern – dies wird auch als Eventual Consistency bezeichnet. MongoDB ist in seiner Standardkonfiguration äußerst unsicher, so stehen Daten ungeschützt im Internet und lassen sogar teilweise Schreibzugriffe zu. (Makris u. a., o. J.)

- keine Joins möglich
- Redundante und duplizierte Datenhaltung
- hohe Speicherinanspruchnahme
- Eventual Consistency

- standardgemäß äußerst unsicher
- SSPL Lizenz zwingt zur Veröffentlichung von Quellcode

Abbildung 6 MongoDB Logo
Quelle: https://www.mongodb.com/

2.1.3 MariaDB

MariaDB ist ein Open-Source-Datenbankmanagementsystem, welches auf relationalen Datenbanken aufbaut. Die Technologie wird seit 2009 von der MariaDB Corporation entwickelt und ist für Microsoft Windows, Linux, MacOS, Solaris und OpenBSD erhältlich. MariaDB wurde unter dem Aspekt entwickelt, eine sichere und hoch leistungsfähige Betriebsweise zu garantieren. So werden Daten auf Clustern Ende-zu-Ende verschlüsselt. Das System ist über die GPLv2 weiterhin auch für kommerzielle Zwecke kostenlos erhältlich. MariaDB verwendet Standard SQL und gleicht in seiner Anwendung MySQL. Seit der Erscheinung des MariaDB-ColumnStore sind auch Big-Data-Technologien in MariaDB verfügbar, womit eine massive parallele Abfrageverteilung und Datenladungen umgesetzt werden können. Der ColumnStore kann mit relationalen Engines gemeinsam verwendet werden und unterstützt somit zusätzlich zur spaltenorientierten Speicherung auch die Möglichkeit zur herkömmlichen relationalen Speicherung. (Kekäläinen o. J.) („MariaDB“ 2022) („Comparing 3 Open Source Databases: PostgreSQL, MariaDB, and SQLite | Opensource.Com“ o. J.)

Pro:

MariaDB ist für hoch performante Anwendungen konzipiert und unterstützt über den ColumnStore distribuierte Datenhaltung wie Abfrage und das im Gegensatz zu MySQL auch für den Enterprise Sektor kostenlos. Die MariaDB Corporation veröffentlicht regelmäßige

Sicherheitsupdates, was zeigt, dass der Entwickler um Sicherheit bemüht ist. Migrationen von MySQL nach MariaDB sind sehr schnell umzusetzen, da beide Produkte eine hohe Kompatibilität aufweisen. („Benefits and Challenges of Maria DB vs MySQL“ 2016)

+ distribuierte Datenhaltung via Column Store
+ Open Source
+ sicher
+ flache Lernkurve

Kontra:

Im Vergleich mit PostgreSQL ist MariaDB weniger performant. Speziell das Caching der Datenbank entspricht nicht mehr dem neuesten Stand der Technik. Einige negative Aspekte haben sich durch den ColumnStore gelöst, speziell im Augenmerk der Big Data Anwendung, jedoch ist die Datenbank ohne dieses optionale Upgrade den meisten seiner Konkurrenten unterlegen. Auch wenn es initial vom Hersteller versprochen wurde, ist die Migration von MySQL nach MariaDB nichtmehr komplett kompatibel und erfordert daher doch einiges an Refactoring. („Benefits and Challenges of Maria DB vs MySQL“ 2016)

- mäßig performant
- ohne ColumnStore keine „First Tier Technologie“
- nichtmehr komplett mit MySQL kompatibel

Abbildung 7 MariaDB Logo
Quelle: https://de.wikipedia.org/wiki/MariaDB

2.1.4 SQLite

SQLite ist die wohl am häufigsten implementierte Datenbank-Engine der Welt, da sie von vielen gängigen Webbrowsern, Betriebssystemen und Mobiltelefonen übernommen wurde. Die Software wird vom SQLite-Team seit 2000 ursprünglich als leichtgewichtige Abspaltung von MySQL entwickelt. Inzwischen handelt es sich im Gegensatz zu vielen anderen Datenbanken nicht mehr um eine Client-Server-Engine – vielmehr ist die gesamte Software in jede Implementierung eingebettet. Auf eingebetteten oder verteilten Systemen befindet sich auf jedem Rechner eine vollständige Implementierung der Datenbank. Dies kann die Leistung von Datenbanken erheblich beschleunigen, da es die Notwendigkeit von Aufrufen zwischen den Systemen reduziert. („SQLite Home Page“ o. J.) („SQLite“ 2021)

Pro:

Für kleine Anwendungen ist SQLite optimal. Da es extrem klein ist, kann es ohne zeitaufwändige Workarounds in einer Vielzahl von eingebetteten Systemen implementiert werden. Die geringe Größe macht das System schnell. Durch die Verringerung der Größe der Datenbank und der zugehörigen Verarbeitungssoftware müssen weniger Daten verarbeitet werden. Seine weite Verbreitung bedeutet auch, dass SQLite wahrscheinlich die kompatibelste Datenbank ist, die es gibt. Dies ist besonders wichtig, wenn ein System in Smartphones integrieren werden soll. („Comparing 3 Open Source Databases: PostgreSQL, MariaDB, and SQLite | Opensource.Com“ o. J.)

+ schnell bei kleinen Datenbanken
+ hohe Kompatibilität

Kontra:

Die geringe Größe von SQLite bedeutet, dass einige Funktionen fehlen, die in größeren Datenbanken zu finden sind. So fehlt z. B. eine integrierte Datenverschlüsselung, die zum Standard geworden ist, um die häufigsten Online-Hackerangriffe zu verhindern. Die weite Verbreitung und der öffentlich verfügbare Code machen die Arbeit mit SQLite zwar einfach, vergrößern aber auch die Angriffsfläche. Obwohl der Single-File-Ansatz von SQLite Geschwindigkeitsvorteile mit sich bringt, gibt es keine einfache Möglichkeit, eine Mehrbenutzerumgebung mit diesem System zu implementieren. („Comparing 3 Open Source Databases: PostgreSQL, MariaDB, and SQLite | Opensource.Com“ o. J.)

- große Systeme nicht möglich
- große Angriffsfläche
- keine integrierte Datenverschlüsselung
- keine Mehrbenutzerumgebung möglich

Abbildung 8 SQLite Logo
Quelle: https://de.wikipedia.org/wiki/SQLite

2.2 Vorstellung moderner Backendtechnologien

2.2.1 Java Spring

Spring ist ein von VMWare entwickeltes und äußerst etabliertes Opensource-Framework zur Erstellung von Webapplikationen für die Java-Plattform. Spring stellt eine Komplettlösung dar, deren Konnektivität, Sicherheit und Performanz im Websektor ungeschlagen ist. Java verwendet als Laufzeitumgebung seine eigene virtuelle Maschine, das JRE (Java Runtime Environment) und ist deshalb komplett Betriebssystem-unabhängig. („Spring Makes Java Simple.“ o. J.)

Pro:

In seiner Standardkonfiguration implementiert Spring lediglich das Plain Old Java Object, jegliche Erweiterung muss vom Entwickler bewusst gesetzt werden. Somit ist die Basiskonfiguration „Light Weight“ und wächst erst mit den Anforderungen an die Applikation selbst. Komponenten werden lose über Dependancy Incjection gekoppelt, das bedeutet, eine Komponente, welche eine andere Komponente als Abhängigkeit braucht, weiß selbst nicht, woher die notwendige Komponente kommt. Dies führt zu einer losen Struktur. Weiters erlaubt Spring die Interaktion mit jeglichen auf Spring basierenden Applikationen – eine Spring Boot Applikation kann problemlos mit einer Spring Cloud Applikation kombiniert werden.

Auch der Spring Lifecycle von Komponenten ist durch die Methoden init() und destroy() vorgegeben und verständlich. Des Weiteren überwacht Spring Third Party Komponenten, was erheblich zur Sicherheit der Applikation beiträgt, und gilt auch sonst als eines der sichersten Frameworks. Speziell rechenintensive Requests kann Spring performant abarbeiten. Spring fördert in seiner Architektur die Implementierung von Unit Tests, was bei deren Umsetzung die Applikation stabilisiert. („Java Spring Pros and Cons - Javatpoint“ o. J.)

+ Sicher
+ Lose Koppelung von Komponenten
+ Sehr gute Konnektivität (JDBC, Spring Boot, Spring Cloud)
+ Nur so groß wie nötig
+ Unit Test freundlich
+ Flexibel
+ Hoch performant bei rechenintensiven Szenarien
+ Reaktiv
+ Langzeiterfahrungen
+ Stark typisiert

Kontra:

Spring ist ein über Jahre gewachsenes Framework, dessen Komplexität um ein Vielfaches höher ist als z.B. jene einer Node Umgebung und beherbergt dadurch eine sehr steile Lernkurve. Dieser Umstand wird durch das Fehlen von spezifizierten Guide Lines und die Parallelität diverser Mechanismen sogar noch gesteigert. Weiters verlässt sich das Framework immer noch auf XML-Definitionen, diese funktionieren, sind allerdings nichtmehr ganz zeitgemäß und erschweren dem Entwickler die Arbeit durch das Fehlen der offensichtlichen Typisierung (.dtd muss gelesen werden) und durch die generell schwerere Lesbarkeit von XML im Vergleich zu JSON. („Java Spring Pros and Cons - Javatpoint“ o. J.)

- Sehr steile Lernkurve, hohe Komplexität
- Teilweise veraltet

Abbildung 9 Spring Logo
Quelle: spring.io 2022

2.2.2 Node.js + ExpressJS

Node.js ist eine serverseitige Javascript-Laufzeitumgebung, welche von Ruby's „Event Machine" und Python's „Twisted" beeinflusst wurde. Node.js arbeitet „Eventdriven" – das heißt Node.js verarbeitet Events als ein Laufzeitkonstrukt und nicht als Library. Im Gegensatz zu üblichen Server-Laufzeitumgebungen, in denen am Anfang des Scripts Callback-Funktionen definiert sind, welche dann mit einem dedizierten blockierenden Aufruf gestartet werden, tritt Node.js nach dem Ausführen des Eingabeskripts in die Event-Schleife ein und verlässt diese, wenn es keine Callback-Funktionen mehr auszuführen gibt. Node.js wurde speziell für die Entwicklung skalierbarer Web-Applikationen konzipiert.

Mit dem auf NodeJS basierendem Framework „ExpressJS" kann Node.js um einige Tools erweitert werden, welche das Erstellen moderner Webapplikationen einfacher gestaltet. (Node.js o. J.)

Pro:

Node.js verfügt über ein riesiges Ökosystem, erreichbar über den „Node Package Manager" NPM, welcher mehr als 800 000 Libraries enthält. Da Node auch auf Javascript basiert, können jegliche Javascript Libraries verwendet werden, was Problemlösungen in ihrem zeitlichen Aufwand verkürzt. PayPal, Microsoft, SAP, und IBM sind die Hauptfinanzmittelsteller für Node, dadurch wird sowohl Long Term Support als auch künftig Relevanz geboten. Der größte Vorteil des RTE liegt aber in seiner Skalierbarkeit – Die Software kann mit multiplen Verbindungen umgehen und priorisiert dabei eingehende Requests nach zu erwartender Response Zeit. Node-Zyklen blockieren eingehende Requests nicht, sie führen lediglich eine Callback-Funktion, basierend auf eingehenden Events aus –

somit können n-fach Knoten im Mainloop registriert werden, was die mögliche Nutzerzahl einer Applikation nicht länger beschränkt.

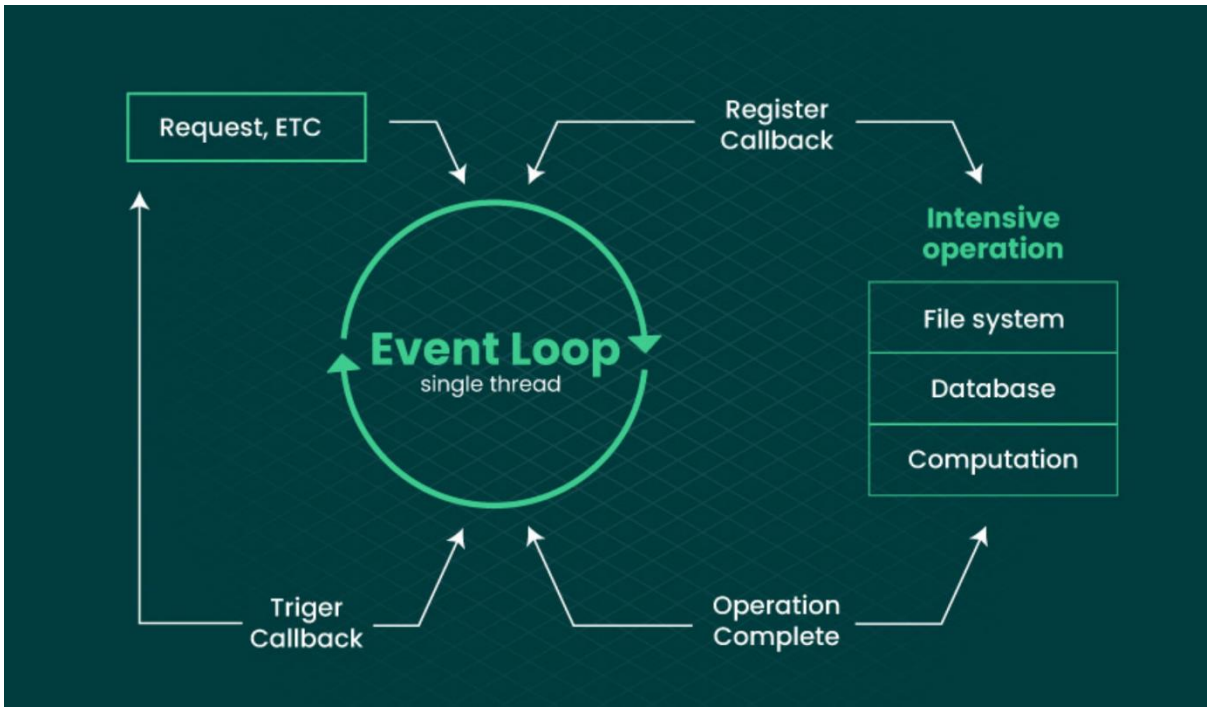

Abbildung 10 Node Process Cycle
Quelle: https://forbytes.com/blog/nodejs-pros-and-cons/

Node hat in der Standardkonfiguration nur ein Ausmaß von 25.4MB, das Runtime Environment ist auf Simplizität ausgelegt und erlaubt durch die Verwendung von Javascript die Sprachengleichheit von Backend und Frontend und ermöglicht somit auch Frontend Entwicklern, ohne weitere Einarbeitungszeit, die Implementierung von serverseitig ausgeführtem Code. Diese Tatsache führt auch zu einer nahtlosen Kommunikation mit dem Frontend. Wird lediglich der Aspekt Input/Output herangezogen, ist Node aufgrund der Single Thread Architektur jeder anderen RTE überlegen. („Exploring Node.Js Pros and Cons“ 2021)

+ Große Community
+ Flache Lernkurve
+ Hoch skalierbar
+ Schnellste I/O
+ Requests können architekturbedingt nicht blockieren
+ ExpressJs vereinfacht die Erstellung eines Webservers deutlich

Kontra:

Node führt Callback-Funktionen asynchron aus, dies kann schnell zur sogenannten „Callback Hell“ ausarten. Dabei handelt es sich um einen Zustand der Applikation, in der diese kaum noch wartbar ist, da die Reihenfolge, in welcher die Callback-Funktionen ausgeführt werden, nicht statisch vorgegeben ist. Auch wenn es darum geht rechenintensive Requests auszuführen, ist Node die falsche Wahl. Erhält Node einen CPU-intensiven Request, wird dieser von der Software mit der höchsten Priorität eingestuft. Dies führt, auch Aufgrund der Single-Thread Architektur, zur Verlangsamung des RTE. Erst 2018 wurde ein optionales Modul eingeführt, welches Node die Möglichkeit bietet, auf Multithreading-Technologie zurückzugreifen und Javascript parallel auszuführen. Dennoch bleibt Node bei rechenintensiven Operationen im direkten Vergleich hinter seinen Konkurrenzprodukten. Weiters sei angemerkt, dass die große Community zwar für zahlreiche Libraries sorgt, doch entsprechen diese oft nicht den gängigen Coding-Standards und führen manchmal zu duplizierten Codesegmenten. („Exploring Node.Js Pros and Cons“ 2021)

- Callback Hell
- Malus bei rechenintensiven Operationen
- Vorsicht bei der Implementierung von Third Party Libs geboten
- ohne Typscript Konfiguration nicht typensicher

Abbildung 11 NodeJS Logo
Quelle: https://nodejs.org/ 2022

Abbildung 12 Express Logo
Quelle: expressjs.com 2022

2.2.3 Python

Python ist ein seit 1991 von der Python Software Foundation entwickelte Programmiersprache, vorzugsweise für Serverseitige Implementierung. Python setzt dabei auf einen gut lesbaren und knappen Programmierstil und wurde mit dem Ziel der Übersichtlichkeit entwickelt. So kommt Python mit einer geringeren Anzahl an syntaktischen Konstrukten aus als andere Programmier- und Skriptingsprachen. Die Sprache lässt sowohl objektorientierte, funktionale als auch aspektorientierte Programmierparadigmen zu. („Welcome to Python.Org“ o. J.) („Python (Programmiersprache)“ 2022)

Pro:

Im Gegensatz zu anderen höheren Programmiersprachen ist Python einfach zu lesen, zu schreiben und zu erlernen. Datentypen werden von Python automatisch zur Laufzeit zugewiesen, dadurch wird der geschriebene Code im Vergleich zu anderen Backendsprachen reduziert und die Lesbarkeit erhöht, ohne den Funktionsumfang einzuschränken. Dies steigert auch die Produktivität der Programmierer. Python verfügt nativ über sehr große Anzahl an Softwarebibliotheken, was die Verwendung von Third Party Bibliotheken vielfach obsolet macht. Die Software ist quelloffen und kostenlos verfügbar und kann auf eine sehr große und aktive Community zurückgreifen. Python kann auf jeder Plattform ausgeführt werden. (Szkaradek o. J.)

+ Plattform unabhängig‘
+ Open Source
+ flache Lernkurve
+ keine Third Party Libraries notwendig
+ große und aktive Community

Kontra:
Python ist in der Ausführung von Code nicht schnell, da es sich um eine dynamisch typisierte und interpretierte Programmiersprache handelt. So werden Prozesse durch die zeilenweise Ausführung verlangsamt. Python hat auch eine geringe Speichereffizienz. Ein weiterer Nachteil der just in Time Compilation ist, dass ein Programm zur Laufzeit unerwartete Fehler haben kann, da die Daten einer Variablen nicht statisch sind, sondern sich jederzeit ändern können, sich Laufzeitfehler nur schwer vermeiden lassen. Auch ist es nicht möglich eine

Speicheroptimierung vorzunehmen. Auch für Unit Tests bietet Python kein dediziertes Framework. (Szkaradek o. J.)

- geringe Speichereffizienz
- Speicheroptimierung nicht möglich
- mäßig performant
- dynamisch interpretiert und daher anfällig für Laufzeitfehler
- Unit Tests sind nur schwer umsetzbar

Abbildung 13 Python Logo
Quelle: https://de.wikipedia.org/wiki/Python_(Programmiersprache)

2.2.4 C#

C# ist eine objektorientierte Programmiersprache und wurde ursprünglich für das .NET Framework von der Firma Microsoft entwickelt. Auch wenn die Programmiersprache an und für sich plattformunabhängig ist, beschränkt der starke Hang sowie die Optimierung für .NET und die Tatsache, dass C# historisch beinahe exklusiv für Windows entwickelt wurde, die Praktikabilität der Anwendung außerhalb einer Windowsumgebung. Erst seit der Einführung von .Net Core wird offizieller Support für Linux und Mac OSX geboten. Die Sprache orientiert sich stark an den Sprachen Java, C++ und Delphi und unterstützt seit Version 2.0 Generics, Partials, Iterators und Generators. Wie auch Java, verwendet C# einen eingebauten Garbage Collector, welcher die Speicherverwaltung übernimmt. (BillWagner o. J.) („C-Sharp“ 2022)

Pro:

C# ist eine Hochsprache, das bedeutet, dass ihr Syntax der menschlichen Sprache ähnelt. Dies führt zu höheren Abstraktionsebenen vom Maschinencode und birgt den Vorteil, dass Entwickler sich auf das Wesentliche, die Implementierung von Funktionen konzentrieren können. Da C# mit C und C++ verwand und an Java angelehnt ist, fällt es Programmierern, welche erstere beherrschen, sehr einfach in die Sprache einzusteigen. Des Weiteren ist C# statisch typisiert, das bedeutet, dass standardmäßig eine Typenüberprüfung zur Kompilierzeit durchgeführt wird und Typenfehler früh erkannt werden, und somit nicht auf die Ausführungsumgebung übertragen werden können. Microsoft bietet eine umfangreiche Dokumentation für C# und .NET an und widmet sich den .NET-Ressourcen wie dem Common Type System, den Compilern und den Möglichkeiten der asynchronen Programmierung. C# bietet zur Applikationslaufzeit gegenüber Java einen kleinen Performanzvorsprung. („The Good and the Bad of C# Programming“ o. J.)

+ einfacher Syntax
+ flache Lernkurve
+ typensicher
+ sehr gut dokumentiert
+ performanter als Java

Kontra:
C# ist stark von .NET abhängig, so muss für jede Plattform eine eigene Laufzeitumgebung verwendet werden, sowie der Code an die Plattform angepasst werden. .NET stellt dabei zwar alle Ressourcen zur Verfügung, C# Code allein erhält allerdings ein Manko in seiner Flexibilität. Auch wenn die Sprache selbst eine flache Lernkurve aufweist, so führt der Gebrauch, bedingt durch die sonstigen Einschränkungen, kaum an .NET, welches eine erheblich steilere Lernkurve aufweist, kaum herum. („The Good and the Bad of C# Programming“ o. J.)

- .NET abhängig
- nicht selbstständig plattformunabhängig
- mit .NET sehr umfangreich

Abbildung 14 C# Logo
Quelle: https://commons.wikimedia.org/wiki/File:Csharp_Logo.png

2.3 Vorstellung moderner Frontendtechnologien

2.3.1 React

React ist eine 2013 von Facebook Inc. Entwickelte und nunmehr von Meta gewartete Open Source Frontend-Javascript Library zur Erstellung von Singlepage Applikationen. Dabei wird ein großer Wert auf einen unidirektionellen Datenfluss gelegt. React ist darauf ausgelegt, deklarative und effiziente Benutzeroberflächen zu implementieren. React baut auf Komponenten auf, welche selbst kleine, isolierte Code Snippets sind, aus denen sich dann komplexe User Interfaces zusammenstellen lassen. („React – A JavaScript Library for Building User Interfaces“ o. J.)

Pro:
React ist eine JavaScript Library mit einer vergleichsweise flachen Lernkurve. Entwickler, welche bereits Erfahrungen mit JavaScript haben, sollten keine Probleme damit haben, die Konzepte hinter React zu verstehen. Wie auch Angular, setzt React dabei auf das Component Driven Design – es werden kleine Komponenten erstellt, welche an jeder Stelle der Applikation wieder verwendet werden können. Dies führt zu einer übersichtlichen, durchgängigen Applikation und fördert das „Single Responsibility“ Paradigma. Weiters setzt React auf ein „Virtual DOM“ (DOM = Document Object Model), das bedeutet, eine virtuelle Abbildung des tatsächlich im Browser existierenden DOMs wird in den Speicher geladen. Wird eine Komponente dynamisch nachgeladen, wird sie vorerst in den virtuellen DOM projiziert. Der

DOM wird von React dann in Zyklen durch den virtuellen DOM überschrieben, was erhebliche Performanzsteigerungen zur Folge hat. Dieser virtuelle DOM ist durch Chrome und Firefox Extensions für den Entwickler einsehbar, was das Debugging deutlich vereinfacht. Ein weiterer Vorteil des virtuellen DOMs ist, dass Suchmaschinen JavaScript-lastige Webseiten nicht selten blacklisten. Da der virtuelle DOM aber nur auf einem Server existiert, welcher dem Browser lediglich eine HTML5 Seite ausgibt, werden React Anwendungen von Suchmaschinen eher gefunden. React basiert komplett auf JavaScript und erlaubt damit die Implementierung zahlreicher Third Party Libraries, welche ebenfalls auf JavaScript basieren. („Pros and Cons of ReactJS - Javatpoint“ o. J.)

+ Flache Lernkuve
+ Komponenten Architektur
+ Hoch Performant
+ Gute Konnektivität zu bestehenden JS Libraries
+ SEO (Search Engine Optimization) freundlich

Kontra:
Die Update Frequenz von React ist so hoch, dass ein ständiges Mitlernen der Entwickler eine logische Konsequenz bildet. Die hohe Frequenz führt auch dazu, dass die Dokumentation nur selten wirklich dem Stand der Dinge entspricht, da der Publisher mit dem Schreiben der Dokumentation der Entwicklung der Library hinterherhinkt. React ist, wie gesagt, eine Javascript Library und kein volles Frontend Framework. Es bietet somit nur Lösungen für die Schichten betreffend des User Interfaces an, wodurch weitere Technologien für die Entwicklung einer Applikation zwingend notwendig werden. Durch die Verwendung einer eigenen syntaktischen Erweiterung von HTML, welches dann um JavaScript erweitert wird (JSX) entsteht unübersichtlicher Code, auch die Lernkurve wird davon negativ beeinflusst. („Pros and Cons of ReactJS - Javatpoint“ o. J.)

- Nicht typisiert
- Zu hohe Frequenz der Inkremente
- Kaum aktuelle Dokumentation
- Rein für die Schicht des User Interfaces verwendbar
- JSX als proprietärer Syntax

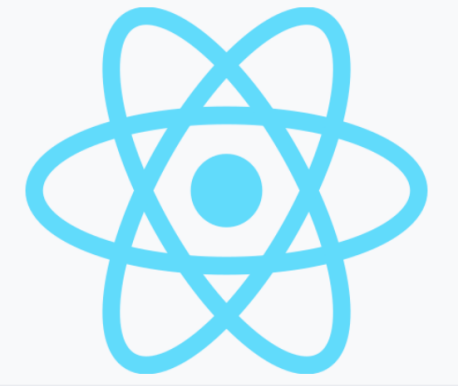

Abbildung 15 Reactjs Logo
Quelle: Reactjs.org 2022

2.3.2 Angular

Anular ist eine seit 2014 entwickelte Open Source Plattform zur Programmierung moderner Web-Applikationen, welche auf das JavaScript-Superset „Typescript“ aufgebaut ist und befindet sich derzeit auf Version 14. Das Framework wird von einer Community von 1,7 Millionen Programmieren entwickelt, die Alphabet-Tochter Google LLC hat dabei den größten Anteil. Das Hauptaugenmerk legt das Framework dabei auf ein komponenten-basiertes Design und damit auch auf die Skalierbarkeit seiner Applikationen. Im Gegensatz zu ReactJS, welches per Definition eine Javascript-Library darstellt, kommt Angular als Framework, also als Komplettpaket. Angular enthält eine Sammlung von abgestimmten JavaScript Libraries, unter anderem werden mit ihnen das clientseitige Routing, Formularabwicklung und Validierung sowie Server-Client Kommunikation geboten. Das Erstellen und Testen von Komponenten sowie die Aktualisierung des geschriebenen Codes wird dabei durch eine Reihe von Entwicklertools unterstützt. Durch das Komponentendesign und dessen starke Kapselung, die vorgegebenen Konventionen im stark objektorientierten Fokus und das Trennen von Logik und Anzeige wird die Skalierung von kleinen Applikationen bis auf ein Enterprise-Level deutlich vereinfacht. Über sogenanntes Databinding können Informationen von der HTML Datei an die Businesslogik der Typescript Datei übergeben und auch aufgenommen werden. Dabei wird auch das sogenannte „Two-Way-Databinding“ unterstützt, welches einem Public Member der Typescript Datei erlaubt sowohl von extern, also vom User über die Eingabe, als auch von der Businesslogik direkt überschrieben zu werden. Lazy-Loading, welches wahlweise implementiert werden kann, sorgt dafür, dass nur die für die Applikation im Moment wichtigen Module geladen werden. Module die initial nicht geladen wurden laden dann per Bedarf „lazy“ nach, was wiederum zu einer Performanzsteigerung im

Browser führt. Durch die gängige Erweiterung des Frameworks mit der observable-basierten Library „RxJS“ werden dem Programmierer sogenannte Observable-Streams zur Verfügung gestellt, welche eine komplett asynchrone Datenverarbeitung ermöglichen und der Performanz der Applikation zuträglich sind. Eine weitere übliche Erweiterung stellt der sogenannte Ngrx Store dar, welcher auf Basis eines globalen „State“ Observables, welches in einem Redux Pattern verändert werden kann, einen Kerndatenspeicher am Client vorhält und so Ladezeiten auf ein Minimum reduziert. Auch Desktopanwendungen können mit Angular schnellstens realisiert werden, da das Frontend durch das Hinzufügen von Service-Workern sehr schnell in eine Electron-Wrapper Instanz gepackt werden kann – die so erstellte Desktopapplikation wird defacto in einem Electron-Chromium Browser headless gerendert, während die Service-Workers Logikschnittstellen der Priorität-Verbindung ersetzen. Daten können allerdings nicht ohne eine Backendanbindung geladen werden. Electron-Desktop-Applikationen sind plattformunabhängig. („Angular“ o. J.) („RxJS - Introduction“ o. J.) („NgRx - @ngrx/store“ o. J.) („Electron | Plattformübergreifende Desktop-Anwendungen mit JavaScript, HTML und CSS entwickeln.“ o. J.)

Die essenziellen Bausteine einer Angular Applikation:

Components:
Components sind in sich geschlossene Programmteile und bestehen aus einer TypeScript Klasse mit einem „@Component“ Dekorator, einer optionalen HTML Datei, welche von Angular als „Template“ bezeichnet wird und optionalen Styles Dateien, welche je nach Setting „.css“, „.sass“ oder „.scss“ sein können. Weiters kann das standardmäßig integrierte Jasmin/Karma Testframework mit Dateien der Endung „.spec.ts“ angesteuert werden, wodurch auch Philosophien wie „Test-Driven-Design“ umsetzbar sind. Components können als kleine Teile der Benutzeroberfläche oder als gekapselter Programmabschnitt verstanden werden und sind streng hierarchisch organisiert. Components können wiederum in anderen Components wiederverwendet werden, indem sie über ihren CSS-Selektor angesprochen werden. Der HTML Code, der wahlweise inline in der .ts Datei definiert sein kann, oder eben als Pfad mit Verweis auf die beigelegte HTML Datei umgesetzt sein kann, wird von Angular mit eigener Syntax erweitert, welcher es erlaubt, Werte dynamisch aus der Component einzufügen. Das Document Object Model (DOM) wird bei einer eintretenden Veränderung des Datenstandes von Angular automatisch aktualisiert. Um dabei die Performanz zu steigern können über Dependency Injection im Constructor der .ts Datei, das von Angular bereitgestellte ENUM „ChangeDetectionStrategy“ und die Klasse „ChangeDetectorRef“

Änderungen im Template bewusst forciert werden, sodass Variablen nicht in jedem Digest Cycle überprüft werden müssen.

Dependency Injection

Mittels Dependency Injection können Services und Klassen in Components injected werden, Angular übernimmt dabei die Instanziierung automatisch.

Modules

In Modules können Imports, Exports und Injektoren organisiert werden. Modules sind Klassen, welche mit dem „@NgModule“ Decorator versehen werden. Ein Module deklariert die ihm zugeordneten Components, Directives, Pipes und Services und steuert deren Öffentlichkeitsstatus über Exports. Modules organisieren wie Templates kompiliert und Services zur Laufzeit initialisiert werden und importieren wiederum Services und Components aus anderen Modules, welche in den darunterliegenden Components als Abhängigkeiten definiert sind. Dabei beschreibt ein Module einen in sich geschlossenen Funktionsblock, der selbstständig Third-Party-Libraries importieren kann, was erneut zu einer starken Kapselung führt und den Wartungsaufwand einer Applikation, sowie den eventuell unnötigen globalen Import von Third-Party-Libraries verringert. Jede Angular Applikation verfügt über ein „Root-Module“, in welchem sämtliche Sub-Modules und Router-Modules registriert werden. Die Erfahrung zeigt, dass das Implementieren eines „CoreModule“, welches einen CoreService, einen ApiService und eine Commons-Library trägt, erhebliche Erleichterungen mit sich bringt. Hierbei können im ApiService alle Priorität CRUD (Create Read Update Delete) Methoden abgebildet sein. Der CoreService könnte die Kommunikation mit dem Ngrx Store abwickeln und „readonly“ Datenstreams für die gesamte Applikation bereitstellen, was wiederum Daten und Logik trennt und für erhöhte Wartbar- und Skalierbarkeit sorgt. Da das CoreModule in jedem weiteren Module importiert wird, können in der CommonsLibrary definierte Methoden, Pipes und Interfaces von allen Komponenten referenziert werden.

Directives

Built-in Directives

Mit „NgClass“ und „NgStyle“ lassen sich direkt im HTML über die Bindung mit einem im TypeScript definierten Public Member CSS-Klassen und Styles dynamisch setzen.

Mit „NgModel“ kann z.B. einem Formular ein Two-Way-Databinding wiederum auf einen Public Member der TypeScript Datei referenziert werden.

Component Directives

Analog zu dem CSS Selector einer Component, liefert Angular standartgemäß nützliche Component Direktives aus. „ng-container“ sorgen für Struktur im HTML und werden, im Gegensatz zum klassischen „div“ nicht selbst im DOM gerendert, sondern nur deren jeweiligen Child-Elemente. „ng-template“ erspart durch die einmalige Definition eines HTML-Knotens bei wiederkehrender Anwendung die Repetition des Snippets, wodurch Änderungen auch wieder nur an einer Stelle umzusetzen sind.

Structural Directives

Angular besitzt nativ über eine Vielzahl von Struktur Direktiven, die dazu dienen Logikelemente im Template zu implementieren. So können z.B. mit „*ngIf“ und „*ngSwitchCase“ bzw „*ngSwitch“ konditionale Logiken und mit „*ngFor“ Iterationen direkt im HTML File untergebracht werden. Die in Angular nativen Direktiven können dabei benutzerdefiniert erweitert werden.

Attribute Directives

Das Verhalten und Aussehen von DOM-Elementen kann mit sogenannten „Attribute Directives“ gesteuert und verändert werden. Der Vorteil liegt darin, dass die Direktive nur einmal geschrieben werden muss und auf eine Vielzahl von Elementen angewendet werden kann.

Angular CLI

Mit dem Angular Comand Line Interface können Components und Services erstellt, sowie „build“ und „test“ Befehle einfach und fehlerfrei ausgeführt werden.

Pro:

Angular bietet eine komponentenbasierte Architektur und erhöht damit die Code-Qualität unter anderem auch durch die hohe Wiederverwendbarkeit dieser Komponenten.

Komponenten erhöhen die Lesbarkeit des Codes und verringern so den Wartungsaufwand einer Applikation. Unit-Test können mit dem eingebauten Jasmin/Karma Framework einfach umgesetzt werden, und über die Angular CLI auch leicht in Continuous Integration Tools wie GitLab oder Jenkins aufgerufen werden. Die Verwendung von TypeScript, auch wenn dieses im Endeffekt zu ECMAScript (JavaScript) transpiliert wird, sorgt während der Implementierung für ein typensicheres Handling und ermöglicht eine Priorität (Integrated Development Environment) – Unterstützung, welche mit VanillaJS nicht möglich ist. Da es sich bei TypeScript um ein JavaScript Superset handelt, kann auch mit JavaScript gearbeitet werden, was für neue Entwickler eine Erleichterung darstellt. Das AngularFlex Module erlaubt responsives Bootstrapping einer Applikation ohne Third Party Libraries wie Bootstrap, und bietet somit ein Komplettpaket zum Entwickeln moderner Web-Applikationen für Mobile und Desktop. Trotz der relativ steilen Lernkurve von RxJS und der Tatsache, dass es für Angular beinahe eine Notwendigkeit darstellt, ist RxJS eine hochmoderne Technologie am Zahn der Zeit, die, wenn einmal beherrscht, das Verarbeiten von Datenströmen und asynchronen Abläufen mehr als nur marginal vereinfacht, da die integrierte Library mit ihren Observable-Streams und Subscriptions dem Broadcast-Promise System herkömmlicher JavaScript-Applikationen durch ihre Stabilität und State-Sicherheit, speziell in Kombination mit dem Ngrx-Store Module, weit überlegen ist. Angular wurde mit dem Ziel geschaffen, komplett plattformunabhängig zu sein und verfolgt ein „Mobile-First“ Paradigma. Die in Angular gebotene hierarchische Dependency Injection entkoppelt die Komponenten von ihren Abhängigkeiten, indem diese parallelisiert ausgeführt werden, was Angular zu einem High-Performance Framework macht. Der in Angular integrierte Ivy Compiler, welcher aus dem geschriebenen TypeScript und HTML Code für den Browser interpretierbaren Code generiert, verfügt über die sogenannte „Tree-Shaking“ Funktion, welche unerhebliche Programmteile während der Rendering-Phase von eben jener ausschließt und damit selbst komplexe Anwendungen schnell laden lässt. Ivy kompiliert dabei Ahead of Time (AOT), das heißt, die Applikation wird direkt im Build-Prozess kompiliert. Dadurch können Kompilierungszeiten verringert werden. Als weiterer Vorteil sei die Involviertheit Googles erwähnt, welches Long Term Support (LTS) für Angular garantiert. Ein immenser Vorteil liegt weiters in der nahtlosen Kompatibilität mit Angular Material. Angular Material ist eine getestete Sammlung von UI Komponenten, welche auf Basis von Studien über das Userverhalten entwickelt wurden. Angular Material schreibt gewisse Design-Guidelines vor, dies führt zu einer konsistenten Applikation und Userführung, was es wiederum einem User erlaubt, Programme intuitiv zu bedienen. Jede von Google entwickelte Software mit User Interface verwendet diese. In Anbetracht des Marktanteils von Google sind es die meisten User gewohnt, Material Komponenten zu bedienen. (Team o. J.) Angular erhält in etwa alle sechs Monate ein neues Update, welches über das Angular CLI mit einem einfachen Befehl nahtlos bestehende

Applikationen auf den neusten Stand der Technik hievt. Trotz der Tatsache, dass es sich um ein relativ junges Framework handelt, existiert bereits ein mächtiges Ökosystem an Angular Applikationen und Code Snippets, auch die Priorität Integration ist mit Open Source Plugins für z.B. die freie Priorität Visual Studio Code gewährleistet. Auch wenn der Electron Wrapper nicht proprietär für Angular ist, ist eine Implementierung für Desktop Standalone Applikationen mit Angular und Ngrx Store einfach. Direktiven erweitern HTML Syntax und Funktionen für einen schnellen, gekapselten und wartbaren Umgang mit Templates. („The Good and the Bad of Angular Development“ o. J.) („NgRx - @ngrx/store“ o. J.) („Electron | Plattformübergreifende Desktop-Anwendungen mit JavaScript, HTML und CSS entwickeln.“ o. J.)

+ Komponentenbasiert
+ hohe Wartbarkeit
+ IDE Unterstützung
+ Integriertes Bootstrapping via AngularFlex Module
+ TypeScript
+ Cross Plattform
+ RxJS integrierbar zum asynchronen Datenhandling
+ Ivy Compiler (Schnelle Just in Time Kompilierung)
+ Hierarchische Dependency Injection
+ Google LTS
+ reaktiv
+ Getestete und konsistente UI-Komponenten mit Angular Material
+ Alle sechs Monate wird ein neues Inkrement veröffentlicht
+ Einfaches Command Line Interface
+ Großes Ökosystem
+ Einfache Umsetzung einer Web-Applikation als Desktop-Applikation mittels Electron
+ Erweiterter HTML Syntax mittels Direktiven

Kontra:
Im Vergleich zu anderen JavaScript Frameworks und Libraries weist Angular eine deutlich steilere Lernkurve auf, was neuen Entwicklern den Einstieg deutlich erschwert. Die auch wenn noch so vorteilhaften Elemente wie Modules, Dependency Injection, Components, Services, Templates und Direktiven sind in dieser Form nur im Angular Framework enthalten, was für einen Einsteiger oder Umsteiger eine komplett neue Herangehensweise an die Frontendentwicklung darstellt. Auch die beinahe zwingende Verwendung von RxJS erschwert

den Einstieg in das Framework deutlich. Auch die zwingende Verwendung von TypeScript, trotz der erheblichen Vorteile der Sprache, stellt zu Beginn eine Hürde dar, da sie, auch wenn dem JavaScript entlehnt, deutlich davon abweicht und das Erlernen dem Erlernen einer neuen Programmiersprache gleichkommt. Weiters ist das Angular Framework komplex, und durch das komponentenbasierte Design ergeben sich oft viele Aufgaben für vergleichsweise kleine Programmfunktionen. Auch ein Update einer im Vorgänger AngularJS programmierten Applikation, ist nur durch einen zeitweise hybriden Ansatz, bei welchem Angular Komponenten ein Downgrade auf AngularJS erhalten, bis alle AngularJS Komponenten auf Angular umgestellt sind, nur um anschließend alles gemeinsam wieder Upzugraden, stellt einen enormen Aufwand dar. („The Good and the Bad of Angular Development“ o. J.)

- Steile Lernkurve
- RxJS erforderlich
- TypeScript
- Komplex
- Migration vom Vorgänger AngularJS ist nur mit sehr hohem Aufwand möglich

Abbildung 16 Angular Logo
Quelle: Angular.io 2022

2.3.3 Svelte

Svelte ist eine JavaScript-Library zur Programmierung von reaktiven Single-Page-Applikationen welche, anders als andere bekannte Vertreter der SPA Frameworks, während der Laufzeit der Applikation nicht auf frameworkspezifische Elemente zugreift. Svelte wird vom Compiler in reinen JavaScript-Code transpiliert und verliert dadurch jedwede Abhängigkeit zu externen Software-Libraries. Es wird, anders als bei ReactJS und Angular, auf einen Shadow-DOM, welcher über Changedetection-Strategien Zustandsänderungen wahrnimmt, und so den DOM selektiv verändert verzichtet, und damit die Prozessorlast am Clientgerät reduziert. Während Angular, wie auch ReactJS, die Rechenlast auf den Browser übertragen, überträgt Svelte diese Schritte in den Kompilierprozess. Die Bibliothek stellt damit einen radikal neuen Ansatz für das Erstellen von Benutzeroberflächen dar. („Svelte • Cybernetically Enhanced Web Apps“ o. J.) („Svelte (Framework)“ 2022)

Pro:
Svelte kommt ohne hochkomplexe State-Management-Systeme wie Ngrx-Store aus, da das State-Managment bereits nativ in der Programmiersprache vorhanden ist. Komponenten werden automatisch exportiert und können ohne das Schreiben von zusätzlichem Code in der gesamten Applikation verwendet werden, was Exportfehler gänzlich ausschließt und die Implementierung einer Frontendapplikation beschleunigt. Komponentenattribute können zudem über einfache Shortcuts an Subkomponenten weitergegeben werden und erhöhen die Leserlichkeit bei gleichzeitiger Minderung potenzieller Tippfehler. Svelte ist Stand 2022 das beliebteste Frontend Framework, und die Community wächst stetig an. Wie Angular auch, werden mit Svelte Animationen und Effekte ausgeliefert, welche für ein modernes Web-Framework unabdingbar sind und ansonsten über Third-Party-Libraries eingebunden werden müssten. Durch das Hinzufügen eines Dollarzeichens kann eine reaktive Anweisung auf eine Variable gebunden werden, was die reaktive Programmierung und damit auch die Skalierbarkeit der Applikation vereinfacht. („Why Svelte Is the Next Big Thing in JavaScript Development“ o. J.)

+ eingebautes State Management
+ auto export von Komponenten
+ Attribute Short Cuts
+ skalierbar

+ reaktiv

Kontra:

Der Stil der Verwendung von DOM-Events verwendet eine spezifische Syntax und weicht komplett vom Standard-JavaScript Syntax ab. Entwickler, welche bereits Erfahrung in anderen JavaScript-basierenden Frameworks haben, müssen sich auf die von Svelte definierte Syntax verständigen. Das Framework unterstützt keine Referenzupdates und Arraymutationen, womit Entwickler sicherstellen müssen, dass Arrays neu zugewiesen werden und damit die Benutzeroberfläche aktualisiert wird. Weiters ist Svelte ein sehr junges Framework und hat daher noch keine große Community, womit die Unterstützung im Problemfall durch das Internet erschwert wird. Ebenfalls dem Alter geschuldet, werden viele Plugins und Integrationen, welche für eine komplexe Frontendanwendung von Nöten sind, noch nicht unterstützt. Aufgrund des Fehlens eines Virtual-DOMs und des Transpilierens zu Plain JavaScript sind Svelte-Anwendungen zur Laufzeit vollgepackt mit JavaScript – dies hat einen negativen Einfluss auf die Search Engine Optimierung. („Why Svelte Is the Next Big Thing in JavaScript Development“ o. J.)

- noch sehr jung und unausgereift
- spezifische Syntax für DOM-Events
- keine Referenzupdates sowie Arraymutationen
- kleine Community
- fehlende Integrationen für Enterpries Applikationen
- fehlende Plugins für Enterprise Applikationen
- schlechte SEO-Optimierung

Abbildung 17 Svelte Logo
Quelle: https://de.m.wikipedia.org/wiki/Datei:Svelte_Logo.svg

2.3.4 VueJS

VueJS ist ein Model-View-View-Model JavaScript-Framework zur Entwicklung von Single-Page-Applikationen mit einer flachen Lernkurve. Aus- und Eingaben sind dabei direkt an die Datenquelle gekoppelt und werden so als reaktive Elemente eingebunden. Wie auch Angular, setzt VueJS dabei auf Komponenten, Direktiven und den Double Curly Syntax zum HTML-Databinding. Das Framework wurde 2013 von Evan You entwickelt und von Adobe, Behance, Alibaba, Gitlab und Xiaomi übernommen. VueJS erlaubt auch die Erstellung von Desktopapplikationen über das Progressive Web App Design. VueJS ist Angular und ReactJS sehr ähnlich und baut im Wesentlichen auf den selben Strukturen auf. („Vue.Js - The Progressive JavaScript Framework | Vue.Js“ o. J.) („Vue.js“ 2022)

Pro:

VueJS ist sehr kompakt, so kommt die Zip-Datei mit 18KB aus und fördert so auch die SEO und die UX. Wie Angular und React auch, setzt das Framework für die Zustandsänderungen im DOM auf einen Virtual-DOM, womit nicht der ganze DOM, sondern nur der betreffende Teil neu zu rendern ist, und ist dabei performanter. VueJS verfügt über eine solide Dokumentation, was den Einstieg erleichtert und verwendet Standard HTML und JavaScript Syntax, womit die Lernkurve deutlich flacher als bei ReactJS und Angular ausfällt. („The Good and the Bad of Vue.Js Framework Programming“ o. J.)

+ flache Lernkurve
+ sehr kompakt
+ SEO optimiert
+ Virtual Dom
+ höchst performant
+ gut dokumentiert
+ reaktiv
+ große Community

Kontra:

VueJS ist speziell im chinesischen Sprachraum stark verbreitet. Die Diskussionen, wie auch die Offenlegung möglicher Lösungsansätze erfolgt somit oft in chinesischer Sprache und

erschwert damit den Zugang für Entwickler, deren Fähigkeiten sich diese Sprache entzieht. Die Reaktivität von VueJS beruht auf dessen Komponenten und wird nur ausgelöst, wenn ein User die jeweilige Komponente ansteuert. In der Regel sind Webapplikationen aus einer Vielzahl von Komponenten zusammengebaut und der erforderliche Trigger wird in VueJS Komponenten oft unzureichend an Sibling- und Parent-Komponenten weitergegeben, was zu einer Reduktion der Reaktivität führt. Das Framework ist nicht für Enterprise Applikationen ausgelegt. („The Good and the Bad of Vue.Js Framework Programming“ o. J.)

- Sprachbarriere
- nicht typisiert
- eingeschränkte Reaktivität
- nicht für Enterprise Applikationen ausgelegt

Abbildung 18 VueJS Logo
Quelle: https://de.m.wikipedia.org/wiki/Datei:Vue.js_Logo_2.svg

3. Der Band Management Server (BMS)

3.1 Einleitung und Definition

Die Musikbranche vom Laien bis zum professionellen Musiker leidet unter Organisationsproblemen. Von der Big Band über Musikkapellen, vom Jazz Trio zur Heavy Metal Band, alle werden von Unpünktlichkeit und Terminkollisionen der einzelnen Mitglieder

tangiert.
In meiner 15-jährigen Tätigkeit als Musiker in den Bereichen „Melodic Death Metal“ und „Hip-Hop“ wurden mir diese Probleme als Interpret auf der Bühne und bei Proben, sowie auch in der Führung meines eigenen Tonstudios am eigenen Leib zu Teil. Dieses Problem wurde mir auch von befreundeten Musikern, Tonstudioinhaber, Kapellmeistern und Musikproduzenten bestätigt.

Um eine einheitliche Plattform zu schaffen, welche korrespondierende Termine in den verschiedenen Formationen gegeneinander validiert, liegt im 21. Jahrhundert die Implementierung einer proprietären Softwarelösung nahe. Um den Zugang für Musikschaffende zu vereinfachen, muss das System plattformunabhängig sein. Um die Daten unter den Usern mobil zu synchronisieren, wird auf das Konzept der Server-Client Applikation zurückgegriffen. Diese soll sowohl als SPA als auch als PWA an die Clientgeräte ausgeliefert werden.

Es soll ein Prototyp mit der Kernfunktionalität erstellt werden, um ihn bei großen Onlineplattformen zum Zwecke der Distribution vorzustellen.

Im Rahmen dieser Arbeit handelt es sich um ein fiktives Software Produkt.

Die Kernfunktionen sind:

- Erstellen, bearbeiten, löschen und validieren von Terminen – Musikformationsübergreifend
- Usergruppen in Musikformationen
  (Bsp.: Musiker, Techniker, Merch, Marketender etc.)
- Suche nach geeigneten Personen für Musikformationen
- Suche, Verkaufe, Tausche
- Messaging System zum schnellen Austausch
- Social Media News Feed
- Veranstalterkartei und Bewertung

Das Kernelement soll dabei der User (=Musiker) bilden. Er kann Musikgruppen beitreten, Musikgruppen erstellen und löschen und andere User zu seinen Musikgruppen hinzufügen beziehungsweise sie von diesen entfernen. User können anderen Usern und Musikgruppen folgen, dies führt dazu, dass Posts jener auf der Social Media News Feed Seite angezeigt

werden. User wie auch Musikgruppen können Termine anlegen und den Terminen eine Priorität zuweisen. Wird ein Termin erstellt, werden die betreffenden User informiert und müssen die Teilnahme über ein Captcha bestätigen. Die Applikation arbeitet nach dem FIFO-Prinzip (First In First Out). Das heißt, wird ein Termin erstellt, welcher mit einem bestehenden Termin eines eingeladenen Users kollidiert, wird eine Warnung ausgegeben - die betreffenden User können dann allerdings auf Basis der Priorität der kollidierenden Termine eine Zusage beziehungsweise Absage einleiten. In beiden Fällen wird der jeweilige Terminersteller und auf Wunsch auch alle Eingeladenen über den Ausgang informiert. Termine enthalten neben Datum, Uhrzeit und Dauer auch eine Information über die verwendete Zeitzone und können wahlweise in UTC, Local und Host angezeigt werden. UTC bezieht sich dabei auf GMT 0, Local auf den Standort des Musikers, ausgelesen über die Uhrzeit des (Mobil) Gerätes und Host bezieht sich auf den Standort des Terminerstellers. Weiters enthalten Termine Informationen über den Standort und eine Beschreibung sowie einen Titel. User enthalten neben Name, E-Mail-Adresse, Passwort, Genre(s) und Adresse auch eine Liste von Tätigkeiten, welche der User als seine Fähigkeiten listet. Diesen Tätigkeiten kann auch eine der folgenden Erfahrungsstufen beigefügt werden: Hobbyist, Intermediate, Semiprofessional und Professional. Musikgruppen sind User, welche um folgende Felder erweitert werden: User List, Homebase (im Normalfall der Standort des Proberaums), Genre(s), Bandname und Branchen (User Untergruppen). Erstellt eine Musikgruppe einen Termin, kann die Propagierung dessen auf dedizierte Branchen beschränkt werden (beispielweise muss ein Kartenverkäufer bei einer Probe nicht anwesend sein). Musikgruppen und User können nach anderen Musikgruppen, Usern und Gütern suchen, Güter zum Verkauf oder Tausch anbieten und Anzeigen über ihre Verfügbarkeit schalten. Veranstalter können, wie Musiker und Musikgruppen auch, nach verfügbaren Musikgruppen suchen, Termine vom Typ „Veranstaltung“ erstellen und Musikgruppen und User dazu einladen. Termine vom Typ Veranstaltung können nach Beendigung von Usern und Musikgruppen bewertet werden. User, Musikgruppen und Veranstalter können Neuigkeiten auf der Social Media News Feed Seite teilen. Musiker, Veranstalter und Musikgruppen können miteinander mittels Chatfunktion kommunizieren, wobei die Kommunikation innerhalb einer Musikgruppe einem Gruppenchat, sortiert nach Branchen, gleichkommt.

Für spätere Inkremente der Software sind weitere Module vorgesehen. Es sind Premium User vorgesehen, welche auf folgende Funktionen Zugriff haben sollen. So soll es für Musiker und Musikgruppen möglich sein, ihre Instrumente, Betriebsmittel und sonstiges Equipment in einer Liste zu pflegen und bei Bedarf direkt von einem Drittanbieter nachzubestellen. Weiters ist ein Modul zur Gagenkalkulation vorgesehen, welches über externe APIs Preise von Unterkünften

und Treibstoff ermittelt und über dies die Mindestgage errechnet. Auch ein länderspezifischer Einkommens- und Umsatzsteuerrechner soll dem System in späterer Folge hinzugefügt werden. Die Erweiterbarkeit stellt somit ein immanentes Kriterium an jegliche zur Verwendung kommenden Technologien.

Use Cases:

- Ein User kann einen User-Account erstellen
- Ein User kann seinen User-Account bearbeiten
- Ein User kann seinen User-Account löschen
- Ein User kann einen Veranstalter erstellen
- Ein User kann einen von ihm/ihr erstellten Veranstalter bearbeiten
- Ein User kann einen von ihm/ihr erstellten Veranstalter löschen
- Ein User kann eine Musikgruppe erstellen
- Ein User kann eine von ihm/ihr erstellte Musikgruppe verändern
- Ein User kann User aus einer von ihm/ihr erstellten Musikgruppe ausschließen
- Ein User kann eine von ihm/ihr erstellte Musikgruppe löschen
- Ein User kann einen Termin erstellen
- Ein User kann einen von ihm/ihr erstellten Termin verändern
- Ein User kann einen von ihm/ihr erstellten Termin löschen
- Ein User kann einen von einer Musikgruppe erstellten Termin annehmen oder absagen
- Ein User kann anderen Usern und Musikgruppen folgen
- Ein User kann Informationen mit Usern, welche ihm/ihr folgen, teilen
- Ein User kann einem anderen User oder einer Musikgruppe private Nachrichten schicken
- Ein User kann von einem anderen User oder einer Musikgruppe private Nachrichten empfangen
- Ein User kann nach Usern und Musikgruppen im System suchen
- Ein User kann zum Zwecke, einer Bandfindung eine Anzeige schalten
- Ein User kann eine von ihm/ihr erstellte Anzeige bearbeiten
- Ein User kann eine von ihm/ihr erstellte Anzeige löschen
- Ein User kann zum Zwecke des Verkaufs oder Tausches von Gegenständen eine Anzeige schalten
- Ein User kann zum Zwecke des Verkaufs oder Tausches von Gegenständen eine Anzeige bearbeiten

- Ein User kann zum Zwecke des Verkaufs oder Tausches von Gegenständen eine Anzeige löschen
- Ein User kann eine Veranstaltung anlegen
- Ein User kann eine von ihm/ihr erstellte Veranstaltung bearbeiten
- Ein User kann eine von ihm/ihr erstellte Veranstaltung löschen
- Ein User kann nach Ablauf derer und bei Teilnahme, Termine, welche von Veranstaltern erstellt wurden, bewerten
- Eine Musikgruppe kann Termine erstellen
- Eine Musikgruppe kann einen von ihr erstellten Termin verändern
- Eine Musikgruppe kann einen von ihr erstellten Termin löschen
- Eine Musikgruppe kann anderen Usern und Musikgruppen folgen
- Eine Musikgruppe kann Informationen mit Usern und Musikgruppen, welche ihr folgen, teilen
- Eine Musikgruppe kann einen User einladen
- Eine Musikgruppe kann einen User, welchen sie beinhaltet, entfernen
- Eine Musikgruppe kann Subgruppen (Branchen) anlegen
- Eine Musikgruppe kann ihr zugehörige User in bestehenden Branchen verteilen
- Eine Musikgruppe kann ihr zugehörigen Usern Rechte zusprechen
- Eine Musikgruppe kann ihr zugehörigen Usern Rechte absprechen
- Eine Musikgruppe kann eine Veranstaltung anlegen
- Eine Musikgruppe kann eine von ihr erstellte Veranstaltung bearbeiten
- Eine Musikgruppe kann eine von ihr erstellte Veranstaltung löschen
- Eine Musikgruppe kann zum Zwecke der Mitgliedersuche Anzeigen schalten
- Eine Musikgruppe kann von ihr erstellte Anzeigen zum Zwecke der Mitgliedersuche bearbeiten
- Eine Musikgruppe kann von ihr erstellte Anzeigen zum Zwecke der Mitgliedersuche löschen
- Eine Musikgruppe kann Anzeigen zum Zwecke der Anpreisung von Gütern erstellen
- Eine Musikgruppe kann von ihr erstellte Anzeigen zum Zwecke der Anpreisung von Gütern bearbeiten
- Eine Musikgruppe kann von ihr erstellte Anzeigen zum Zwecke der Anpreisung von Gütern löschen
- Eine Musikgruppe kann an Terminen teilnehmen
- Eine Musikgruppe kann nach Ablauf derer und bei Teilnahme, Termine welche von Veranstaltern erstellt wurden bewerten
- Ein Veranstalter kann Termine erstellen

- Ein Veranstalter kann von ihm/ihr erstellte Termine bearbeiten
- Ein Veranstalter kann von ihm/ihr erstellte Termine löschen

3.2 Entity Relation Ship Modell des BMS

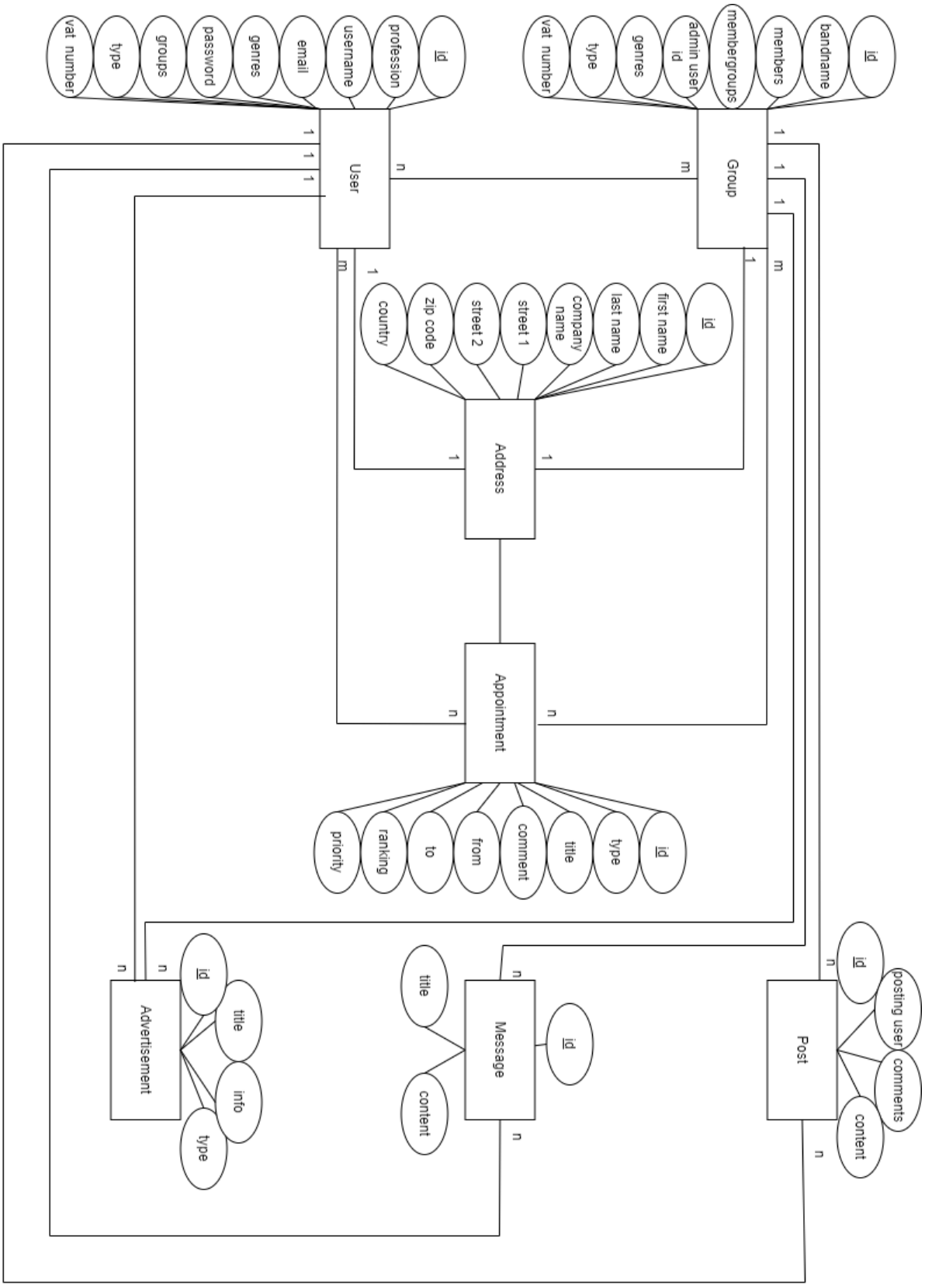

Abbildung 19 Entity Relation Ship Model BMS

3.3 Anforderungskatalog abstrahiert aus Use Cases, ER und Produktbeschreibung

3.3.1 Funktionale Kernanforderungen

- User sind die Hauptentität der Anwendung
  Jegliche Operation an Daten setzt eine gültige Authentifizierung als User voraus. User können Musiker, Veranstalter und Administratoren sein.
- Registrieren von Usern
  Ein User kann sich mit einer gültigen E-Mail-Adresse, einem gültigen Passwort und seiner Wohnadresse sowie einer optionalen Umsatzsteuernummer registrieren. Nach automatischer Validierung der Syntax wird auf der Datenbank vom System eine neue Entität der Klasse User erstellt. Schlägt die Validierung fehl, so wird ein visuelles Feedback ausgegeben.
- Einloggen von Usern
  Ein User, welcher bereits im System registriert ist, kann sich mit der hinterlegten E-Mail-Adresse und seinem persönlichen Passwort in das System einloggen. Wird vom System kein zugehöriger User gefunden oder das eingegebene Passwort stimmt mit dem hinterlegten Passwort nicht überein, so erfolgt ein visuelles Feedback an den User.
- Löschen von Usern
  Ein User, welcher im System eingeloggt ist, kann seine Entität aus der Datenbank des Systems löschen. Dies stellt einen irreversiblen Eingriff dar, weshalb der User im Vorfeld über die Auswirkungen informiert wird, und auch dazu aufgerufen wird die Eingabe zu quittieren.
- Bearbeiten von Userdaten
  Ein User, welcher im System eingeloggt ist, kann seine eigenen Daten verändern.

- Administrationsrechte
  User des Typs „Administrator" sind, wenn sie eingeloggt sind, dazu befähigt, sämtliche Entitäten des Systems einzusehen, zu verändern und zu löschen
- Musikgruppen als Sekundarentitäten
  Musikgruppen sind Gruppen von Usern.
- Musikgruppen erstellen
  Authentifizierte User können Entitäten vom Typ „Musikgruppe" erstellen. Dabei soll es möglich sein, der Musikgruppe einen Bandnamen zuzuweisen, Mitglieder

einzuladen und diesen Rollen, Rechte und eine Branche zuzuweisen und optional eine Umsatzsteuernummer für die Musikgruppe zu hinterlegen

- Musikgruppen bearbeiten
  Authentifizierte können Musikgruppen, in welchen sie dazu autorisiert sind, bearbeiten, Mitglieder einladen und entfernen, sowie deren Rollen, Rechte und Branche ändern. Der Ersteller einer Musikgruppe kann bearbeiten, jedoch selbst nicht von anderen bearbeitet werden.
- Musikgruppen löschen
  Ersteller einer Musikgruppe können von ihnen erstellte Musikgruppen löschen. Dieser Vorgang ist irreversibel, der Ersteller muss daher den Vorgang quittieren.
- Termine erstellen
  User und Musikgruppen können Termine erstellen. Dabei muss ein Zeitrahmen in der Form „von“ – „zu“, eine Priorität und ein Ort eingetragen werden.
- Termine bearbeiten
  Ersteller von Terminen, sowie autorisierte User einer Musikgruppe, welche einen Termin erstellt haben, können den Termin bearbeiten.
- Löschen von Terminen
  Ersteller von Terminen, sowie autorisierte User einer Musikgruppe, welche einen Termin erstellt haben, können den Termin löschen.
- Erstellen von Veranstaltungen
  Veranstalter, Musiker und Musikgruppen können Termine vom Typ „Veranstaltung“ erstellen.
- Bearbeiten von Veranstaltungen
  Veranstalter, Musiker und Musikgruppen können von ihnen erstellte Termine vom Typ „Veranstaltung“ bearbeiten.
- Löschen von Veranstaltungen
  Veranstalter, Musiker und Musikgruppen können von ihnen erstellte Termine vom Typ „Veranstaltung“ löschen.
- Termine validieren
  Das System muss bei der Erstellung eines Termins mögliche Kollisionen mit bestehenden Terminen der eingeladenen User und Musikgruppen validieren und gegebenenfalls durch visuelles Feedback auf eine Kollision aufmerksam machen.
- Termine quittieren
  Erstellte Termine müssen von geladenen Usern angenommen oder abgesagt und durch ein Captcha quittiert werden.

3.3.2 Nichtfunktionale Anforderungen

- Die Datensicherheit der User muss gewährleistet sein.
- Das System muss betriebssystemunabhängig sein.
- Die Implementierung des Systems sollte so einfach wie möglich sein. (Ausnutzen von Synergieeffekten der verwendeten Technologien)
- Es finden keine rechenintensiven Prozesse statt, die Applikation beschränkt sich im Wesentlichen auf CRUD Prozesse der Datenbank, diese müssen performant abgehandelt werden können.
- Die Architektur muss einen Rich Client vorsehen, um die Serverlast zu reduzieren
- Die Applikation muss gekapselt sein und spätere Erweiterungen und Skalierungen zulassen.
- Das System sollte weitgehend auf Third-Party-Libraries verzichten, um die Wartbarkeit zu garantieren.
- Es wird angenommen, dass das System erst ab 10 000 Usern seine Funktionen im vollen Umfang nutzbar machen kann – die Auslegung als Enterprise Anwendung ist dringend vorzusehen.
- Das System muss als Progressive Web-Applikation ausgeführt sein und mittels „Add to Home Screen“ Funktion für mobile und stationäre Geräte installiert werden können, um Abhängigkeiten von App-Stores zu vermeiden, und den Zugang zur Applikation für User zu erleichtern, wie auch den späteren Verkauf durch Distributoren zu erleichtern.
- Das System muss einen „Mobile First“ Ansatz verfolgen.
- Das Layout der Präsentationsschicht muss für mobile wie auch stationäre Geräte angepasst sein.
- Die Userführung soll durch Icons intuitiv gestaltet werden.
- Sowohl in der Mobilversion wie auch in der Desktopversion muss die Navigation durch das System von jedem Teil der Präsentationsschicht aus möglich sein.

3.4 Kriterienliste mit Gewichtung auf Basis des Anforderungskatalogs

Kriterium	**Gewichtung (0-9)**
Multiuserfähigkeit	9
Platfformunabhängikeit	9
Performanz	7
Wartungsarmut	5
Javascript Synergien	3
Datensicherheit	9
Distributionsfähigkeit	9
PWA-Fähigkeit	8
Reaktive Programmierung	8
Übersetzbarkeit	5
Implementierungsgeschwindigkeit	5
Implementierungssicherheit	6
Lernkurve	4
Typisierung	6
Frontend Statemangement	6
Objektorientierte Architektur	7
Mobilfähigkeit	9
Standalonefähigkeit	5
Community Hilfe	7
Hohe Dokumentationsqualität	5
Backend CPU Performanz	2
Backend CRUD Performanz	9
Ausgereiftheit	6

4. Skizze des Weighted Scoring Models

Um die unter Punkt 2 beschriebenen Technologien zu vergleichen, kommt das sogenannte „Weighted Scoring Model zum Einsatz. Hierbei ist anzumerken, dass nicht alle unter Punkt 3.4 gelisteten Kriterien auf alle Technologien zutreffen. Zum Beispiel macht es keinen Sinn das Kriterium der reaktiven Programmierung auf Datenbanktechnologien anzuwenden, wie es auch keinen Sinn macht Datensicherheit als Kriterium für Frontend Technologien zu verwenden. Daher ist der erste Schritt des hier treffenden Weighted Scoring Model, den verschiedenen Technologien die für sie relevanten Kriterien zuzuweisen. Im zweiten Schritt muss die Gewichtung der Kriterien von den derzeit skalaren Werten 0-9 in ein prozentuelles Verhältnis gesetzt werden. Dabei muss die Summe der Gewichtungen der Kriterien 100% ergeben. Die Gewichtung der Kriterien ist damit über die verschiedenen Systeme teilweise gegeben, womit überschneidende Kriterien in jedem Teil des Stacks derselben Anforderung unterliegen. Im dritten Schritt werden in Form einer Tabelle die „konkurrierenden" Technologien zusammen mit den Kriterien aufgetragen und pro Kriterium in Prozent im sogenannten „Direct Ranking" bewertet. Das bedeutet, dass die Gewichtung der jeweiligen Technologie direkt zugeordnet wird – dies geschieht auf Basis der Erfahrungen, welche sowohl beim Erarbeiten des Punktes 2 dieser Arbeit und seiner Subpunkte nötigem Fachwissen und der miteinhergehenden Literaturrecherche wie auch auf persönlichen Erkenntnissen im beruflichen Umfeld resultiert. In der letzten Zeile der Tabelle wird mit Hilfe der Formel $\sum_{c=C(0)}^{c(C(C.length-1))} \mathrm{w(c)} * \mathrm{w(t)}$ der sogenannte „Score" der jeweiligen Technologien ermittelt. (w = weight; c = criteria; t = technologie; C = criteria list;). Diejenigen Technologien aus Datenbanksystemen, Backend und Frontend mit dem höchsten Score werden dabei zu einem Solution Stack zusammengeführt was direkt zur Beantwortung der Forschungsfrage sowie dem Belegen beziehungsweise Widerlegen der Eingangshypothese führt.

4.1 Anwendung des Weighted Scoring Models auf die unter Punkt 2 vorgestellten Datenbank-, Backend- und Frontendtechnologien

4.1.1 Anwendung des WSM auf die unter Punkt 2 beschriebenen Datenbanktechnologien

Aus der Kriterienliste aus Punkt 3.4 sind für Datenbanksysteme folgende Kriterien relevant:

- Ausgereiftheit
- Backend CRUD-Performance (=CRUD Performance)
- Hohe Dokumentationsqualität
- Implementierungssicherheit
- Implementierungsgeschwindigkeit
- Distributionsfähigkeit
- JavaScript Synergien
- Wartungsarmut
- Performanz
- Multiuserfähigkeit

Umrechnung der relevanten Kriterien in ein Prozentuelles Verhältnis:

Kriterium	**Skalar Wert (w)**	**Prozentwert (w)**
Ausgereiftheit	6	9,4%
CRUD-Performance	9	14,1%
Dokumentationsqualität	5	7,8%
Implementierungssicherheit	6	9,4%
Implementierungsgeschwindigkeit	5	7,8%
Distributionsfähigkeit	9	14,1%
Javascript Synergien	3	4,7%
Wartungsarmut	5	7,8%
Performanz	7	10,9%
Multiuserfähigkeit	9	14,1%
Gesamt;	64	100%

Weighted Scoring

Kriterium	**Gewichtung**	**PGSQL**	**MoDB**	**MaDB**	**SQLite**
Ausgereiftheit	9,4%	100	90	60	50
CRUD-Performance	14,1%	90	100	80	50
Dokumentationsqualität	7,8%	90	90	90	90
Implementierungssicherheit	9,4%	90	80	80	10
Implementierungsgeschwindigkeit	7,8%	50	70	70	90
Distributionsfähigkeit	14,1%	70	90	70	0
JavaScript Synergien	4,7%	0	100	100	0
Wartungsarmut	7,8%	60	50	60	60
Performanz	10,9%	90	80	50	50
Multiuserfähigkeit	14,1%	100	100	100	0
Weighted Scores:	**100%**	**79,93%**	**86,67%**	**75,72%**	**36,86**

Legende:

PostgreSQl = PGSQL

MongoDB = MoDB

MariaDB = MaDB

SQLite = SQLite

Ergebnis des WSM:

Technologie	**Score**
1. MongoDB	86,67%
2. PostgreSQL	79,93%
3. MariaDB	75,72%
4. SQLite	36,86%

4.1.2 Anwendung des WSM auf die unter Punkt 2 beschriebenen Backendtechnologien

Aus der Kriterienliste aus Punkt 3.4 sind für Backendtechnologien folgende Kriterien relevant:

- Multiuserfähigkeit
- Plattformunabhängigkeit
- Performanz
- Wartungsarmut
- JavaScript Synergien
- Datensicherheit
- Distributionsfähigkeit
- Reaktive Programmierung
- Implementierungsgeschwindigkeit
- Implementierungssicherheit
- Lernkurve
- Objektorientierte Architektur
- Community Hilfe
- Hohe Dokumentationsqualität
- Backend CPU-Performanz
- Backend CRUD-Performanz
- Ausgereiftheit

Umrechnung der relevanten Kriterien in ein Prozentuelles Verhältnis:

Kriterium	**Skalar Wert (w)**	**Prozentwert (w)**
Multiuserfähigkeit	9	8,2%
Plattformunabhängigkeit	9	8,2%
Performanz	7	6,4%
Wartungsarmut	5	4,5%
Javascript Synergien	3	2,7%
Datensicherheit	9	8,2%
Distributionsfähigkeit	9	8,2%
Reaktive Programmierung	8	7,2%
Implementierungsgeschwindigkeit	5	4,5%
Implementierungssicherheit	6	5,5%
Lernkurve	4	3,6%
Objektorientierte Architektur	7	6,4%

Community Hilfe	7	6,4%
Hohe Dokumentationsqualität	5	4,5%
Backend CPU-Performanz	2	1,8%
Backend CRUD-Performanz	9	8,2%
Ausgereiftheit	6	5,5%
Gesamt	110	100%

Weighted Scoring

Kriterium	**Gewichtung**	**JavaS**	**NJs**	**Py**	**C#**
Multiuserfähigkeit	8,2%	100	100	100	100
Plattformunabhängigkeit	8,2%	100	100	100	0
Performanz	6,4%	80	90	70	80
Wartungsarmut	4,5%	60	50	50	60
JavaScript Synergien	2,7%	0	100	0	0
Datensicherheit	8,2%	90	90	90	90
Distributionsfähigkeit	8,2%	80	90	90	60
Reaktive Programmierung	7,2%	100	100	0	100
Implementierungsgeschwindigkeit	4,5%	60	90	100	60
Implementierungssicherheit	5,5%	100	90	60	90
Lernkurve	3,6%	20	60	100	30
Objektorientierte Architektur	6,4%	100	90	0	100
Community Hilfe	6,4%	100	100	100	100
Hohe Dokumentationsqualität	4,5%	100	90	50	80
Backend CPU-Performanz	1,8%	100	80	70	70
Backend CRUD-Performanz	8,2%	80	100	90	70
Ausgereiftheit	5,5%	90	90	80	100
Gesamt	100%	84,89%	91,03%	70,98%	73,15%

Legende:

JavaS = Java Spring

NJs = NodeJs

Py = Python

C# = C#

Ergebnis des WSM:

Technologie	Score
1. NodeJs	91,03%
2. Java Spring	84,89%
3. C#	73,15%
4. Python	70,98%

4.1.3 Anwendung des WSM auf die unter Punkt 2 beschriebenen Frontendtechnologien

Aus der Kriterienliste aus Punkt 3.4 sind für Frontendtechnologien folgende Kriterien relevant:

- Multiuserfähigkeit
- Plattformunabhängigkeit
- Performanz
- Wartungsarmut
- Datensicherheit
- PWA-Fähigkeit
- Reaktive Programmierung
- Übersetzbarkeit
- Implementierungsgeschwindigkeit
- Implementierungssicherheit
- Lernkurve
- Typisierung
- Frontend Statemanagement
- Mobilfähigkeit
- Standalonefähigkeit
- Community Hilfe
- Hohe Dokumentationsqualität
- Ausgereiftheit

Umrechnung der relevanten Kriterien in ein Prozentuelles Verhältnis:

Kriterium	**Skalar Wert (w)**	**Prozentwert (w)**
Multiuserfähigkeit	9	7,6%
Plattformunabhängigkeit	9	7,6%
Performanz	7	5,9%
Wartungsarmut	5	4,2%
Datensicherheit	9	7,6%
PWA-Fähigkeit	8	6,7%
Reaktive Programmierung	8	6,7%
Übersetzbarkeit	5	4,2%
Implementierungsgeschwindigkeit	5	4,2%
Implementierungssicherheit	6	5%
Lernkurve	4	3,4%
Typisierung	6	5%
Frontend Statemanagement	6	5%

Mobilfähigkeit	9	7,6%
Standalonefähigkeit	5	4,2%
Community Hilfe	7	5,9%
Hohe Dokumentationsqualität	5	4,2%
Ausgereiftheit	6	5%
Gesamt	**119**	**100%**

Weighted Scoring

Kriterium	**Gewichtung**	**React**	**NG**	**Sv**	**Vue**
Multiuserfähigkeit	7,6%	100	100	100	100
Plattformunabhängigkeit	7,6%	100	100	100	100
Performanz	5,9%	100	100	90	90
Wartungsarmut	4,2%	100	80	100	100
Datensicherheit	7,6%	100	100	80	80
PWA-Fähigkeit	6,7%	80	100	60	60
Reaktive Programmierung	6,7%	100	100	70	60
Übersetzbarkeit	4,2%	100	100	100	100
Implementierungsgeschwindigkeit	4,2%	100	70	90	100
Implementierungssicherheit	5%	90	100	70	70
Lernkurve	3,4%	70	30	70	100
Typisierung	5%	0	100	0	0
Frontend Statemanagement	5%	80	100	60	60
Mobilfähigkeit	7,6%	100	100	100	100
Standalonefähigkeit	4,2%	50	100	0	0
Community Hilfe	5,9%	100	90	60	60
Hohe Dokumentationsqualität	4,2%	90	100	70	60
Ausgereiftheit	5%	90	90	60	60
Gesamt	100%	88,12%	94,43%	73,44%	73,79%

Legende:

React = ReactJS

NG = Angular

SV = Svelte

Vue = VueJS

Ergebnis des WSM:

Technologie	Score
1. Angular	94,43%
2. ReactJS	88,12%
3. VueJS	73,79%
4. Svelte	73,44%

4.2 Analyse der Ergebnisse und Definition des entstandenen Stacks

4.2.1 Ergebnisse des WSM: Datenbanktechnologien

Für die sehr spezifische in Punkt 3 deklarierte Anwendung und der daraus resultierenden Kriterien geht MongoDB mit 86,67% Erfüllung der gestellten Anforderungen als klarer Favorit aus dem WSM hervor. Dabei sei aber angemerkt, dass auch PostgreSQL, wenn auch etwas abgeschlagen zweiter, mit seinen Merkmalen und Fähigkeiten der Anwendung gerecht werden würde. Dasselbe gilt auch für MariaDB. Einer Verwendung von SQLite ist auf Basis des WSM sowie dessen Fähigkeiten für die definierte Applikation abzuraten.

4.2.2 Ergebnisse des WSM: Backendtechnologien

NodeJS geht mit 91,03% der Erfüllung der deklarierten Anwendung als Favorit aus dem WSM hervor. Java Spring wäre dennoch eine gute Alternative. Die Diskrepanz der Punkte erklärt sich wie auch schon bei der Datenpersistenz hauptsächlich durch die Spezifikation der Applikation. Die Verwendung von C# und Python für das Implementieren des BMS ist nach Ergebnissen des WSM nicht ratsam. Neben dem Ergebnis des WSM, erfüllt C# das dediziert gewünschte Kriterium der Plattformunabhängigkeit nicht. Python wird „Just in Time“ kompiliert, wodurch die Typensicherheit gefährdet ist, was einer Applikation auf Enterprise Niveau zu Lasten kommt. Das WSM bestätigt NodeJS als das am besten geeignete System für die Backendkomponente der definierten Applikation.

4.2.3 Ergebnisse des WSM Frontendtechnologien

Trotz seiner steilen Lernkurve und dem Malus in der Implementierungsgeschwindigkeit geht Angular mit 94,43% Erfüllung der deklarierten Kriterien abgeschlagen als Favorit aus dem WSM hervor. Da es sich bei Angular um ein komplettes Frontendframework und nicht nur um eine Library handelt, liegen die Vorteile auf der Hand. ReactJS ist dennoch eine umfangreiche JavaScript Library, deren Existenzberechtigung vor allem in Applikationen mittlerer Größe liegt. Zwischen VueJS und Svelte liegen lediglich insignifikante Diskrepanzen, eine Verwendung für kleine performante Applikationen ist ratsam. Für die vorgegebene Applikation, welche dediziert auf Enterprise Niveau liegt, ist Angular jedoch seinen Konkurrenten vorzuziehen.

4.2.4 Stackdefinition

Den Ergebnissen geht der sogenannte MEAN-Stack als der geeignetste Solution Stack für das Band Management System hervor. **M**ongeDB **E**xpressJS **A**ngular **N**odeJS profitiert dabei durch JavaScript und JSON-Synergien und ist vor allem durch seine Performanz bei CRUD-Operationen für das BMS die geeignetste Lösung aus den Vorgestellten Technologien. PostgreSQL, Java Spring und Angular ist ebenfalls ein relevantes Stack, dieses würde jedoch vor allem bei CPU-intensiven Anliegen von Vorteil sein.

5. Conclusio und Beantwortung der Forschungsfrage

Der Stand der Technik und der Wissenschaft im Jahr 2022 erlaubt die Umsetzung des unter Punkt 3 beschriebenen Systems mit nahezu jeder der unter Punkt 2 vorgestellten Technologien. Die Definition der Applikation und der daraus resultierenden Kriterien kann jedoch mit Hilfe des Weighted Scoring Models eine Bessereignung einzelner Stack Komponenten zu Tage fördern. Die Eingangshypothese, der geeignetste Solution Stack für das BMS ist der MEAN-Stack, erfolgte aus jahrelanger Erfahrung im Web-Development und das, obwohl ich den MEAN-Stack selbst nicht beruflich verwende, da die Applikationen, die ich beruflich betreue andere Anforderungen haben als das unter Punkt 3 beschriebene System.

Dem Ergebnis der Recherche und der Anwendung des Weighted Scoring Models geht hervor, dass die Eingangshypothese **korrekt** war.

Der MEAN-Stack ist der geeignetste Solution Stack zur Implementierung einer Server-Client Applikation zur musikformationsübergreifenden Zeitplanung.

Literaturverzeichnis:

„Angular“. o. J. Zugegriffen 12. Juli 2022. https://angular.io/.

„Benefits and Challenges of Maria DB vs MySQL“. 2016. 5. Dezember 2016. https://tutorialsweb.hosting/benefits-and-challenges-of-maria-db-vs-mysql/.

BillWagner. o. J. „C# Docs - Get Started, Tutorials, Reference.“ Zugegriffen 12. Juli 2022. https://docs.microsoft.com/en-us/dotnet/csharp/.

Buckel, Thomas. 2012. „Zum Potential von Event-Driven Architecture für komplexe Unternehmensnetzwerke“. Multikonferenz Wirtschaftsinformatik 2012 Tagungsband der MKWI 2012. file:///C:/Users/info/Downloads/Beitrag389.pdf.

„Comparing 3 Open Source Databases: PostgreSQL, MariaDB, and SQLite | Opensource.Com“. o. J. Zugegriffen 12. Juli 2022. https://opensource.com/article/19/1/open-source-databases.

„C-Sharp“. 2022. In *Wikipedia*. https://de.wikipedia.org/w/index.php?title=C-Sharp&oldid=223402874.

„Electron | Plattformübergreifende Desktop-Anwendungen mit JavaScript, HTML und CSS entwickeln.“ o. J. Zugegriffen 12. Juli 2022. https://www.electronjs.org/.

„Exploring Node.Js Pros and Cons“. 2021. Forbytes. 8. September 2021. https://forbytes.com/blog/nodejs-pros-and-cons/.

„For Fast and Secure Sites | Jamstack“. o. J. Jamstack.Org. Zugegriffen 12. Juli 2022. https://jamstack.org/.

„Getting Started Step-By-Step“. o. J. JSON Schema. Zugegriffen 12. Juli 2022. https://json-schema.org/learn/getting-started-step-by-step.html.

Helmich, Martin. o. J. „RESTful Webservices (1): Was ist das überhaupt?“ Reseller Hosting, Webhosting für Agenturen & Freelancer. Zugegriffen 12. Juli 2022. https://www.mittwald.de/blog/webentwicklung-design/restful-webservices-1-was-ist-das-uberhaupt.

Horowitz, Eliot. o. J. „[License-review] Approval: Server Side Public License, Version 1 (SSPL v1)“. Zugegriffen 12. Juli 2022. http://lists.opensource.org/pipermail/license-review_lists.opensource.org/2018-November/003826.html.

„Introduction | RxJS - Javascript library for functional reactive programming.“ o. J. Zugegriffen 12. Juli 2022. https://xgrommx.github.io/rx-book/.

„Java Spring Pros and Cons - Javatpoint“. o. J. Www.Javatpoint.Com. Zugegriffen 12. Juli 2022. https://www.javatpoint.com/java-spring-pros-and-cons.

Kekäläinen, Otto. o. J. „MariaDB Foundation“. MariaDB.Org. Zugegriffen 12. Juli 2022. https://mariadb.org/.

Liebel, Christian. o. J. „Progressive Web Apps, Teil 1: Das Web wird nativ(er)“. Developer. Zugegriffen 12. Juli 2022. https://www.heise.de/blog/Progressive-Web-Apps-Teil-1-Das-Web-wird-nativ-er-3733624.html.

Makris, Antonios, Giannis Spiliopoulos, Konstantinos Tserpes, und Dimosthenis Anagnostopoulos. o. J. „Performance Evaluation of MongoDB and PostgreSQL for spatio-temporal data“. CEUR-WS.org. file:///C:/Users/info/Downloads/BMDA_3.pdf.

„MariaDB“. 2022. In *Wikipedia*. https://de.wikipedia.org/w/index.php?title=MariaDB&oldid=223942326.

„NgRx“. o. J. Zugegriffen 12. Juli 2022. https://ngrx.io/.

„NgRx - @ngrx/store“. o. J. Zugegriffen 12. Juli 2022. https://ngrx.io/guide/store/.

Node.js. o. J. „Node.Js“. Node.Js. Zugegriffen 12. Juli 2022. https://nodejs.org/en/.

„PostgreSQL: About“. o. J. Zugegriffen 12. Juli 2022. https://www.postgresql.org/about/.

„Pros and Cons of ReactJS - Javatpoint“. o. J. Www.Javatpoint.Com. Zugegriffen 12. Juli 2022. https://www.javatpoint.com/pros-and-cons-of-react.

„Python (Programmiersprache)“. 2022. In *Wikipedia*. https://de.wikipedia.org/w/index.php?title=Python_(Programmiersprache)&oldid=224447029.

„React – A JavaScript Library for Building User Interfaces“. o. J. Zugegriffen 12. Juli 2022. https://reactjs.org/.

„RxJS - Introduction“. o. J. Zugegriffen 12. Juli 2022. https://rxjs.dev/guide/overview.

„Schemaless Database | MongoDB Blog“. o. J. MongoDB. Zugegriffen 12. Juli 2022. https://www.mongodb.com/blog/post/why-schemaless.

„Single-page Webanwendungen“. 2015. W3L AG. https://www.smf.de/pdf/Single-page_Webanwendungen_2015.pdf.

„Spring Makes Java Simple.“ o. J. Spring. Zugegriffen 12. Juli 2022. https://spring.io/.

„SQLite“. 2021. In *Wikipedia*. https://de.wikipedia.org/w/index.php?title=SQLite&oldid=209224887.

„SQLite Home Page“. o. J. Zugegriffen 12. Juli 2022. https://www.sqlite.org/index.html.

„Svelte • Cybernetically Enhanced Web Apps“. o. J. Zugegriffen 12. Juli 2022. https://svelte.dev/.

„Svelte (Framework)“. 2022. In *Wikipedia*. https://de.wikipedia.org/w/index.php?title=Svelte_(Framework)&oldid=222299066.

Szkaradek, Tomasz. o. J. „Pros and Cons of Python“. Zugegriffen 12. Juli 2022. https://thecodest.co/blog/pros-and-cons-of-python.

Team, Angular Components. o. J. „Angular Material“. Angular Material. Zugegriffen 12. Juli 2022. https://material.angular.io/.

„The Good and the Bad of Angular Development“. o. J. *AltexSoft* (blog). Zugegriffen 12. Juli 2022. https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/.

„The Good and the Bad of C# Programming“. o. J. *AltexSoft* (blog). Zugegriffen 12. Juli 2022. https://www.altexsoft.com/blog/c-sharp-pros-and-cons/.

„The Good and the Bad of Vue.Js Framework Programming“. o. J. *AltexSoft* (blog). Zugegriffen 12. Juli 2022. https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/.

Trelle, Tobias. 2014. „MongoDB. Der praktische Einstieg“. In *MongoDB. Der praktische Einstieg*, 21. Heidelberg.

„Vue.js“. 2022. In *Wikipedia*. https://de.wikipedia.org/w/index.php?title=Vue.js&oldid=224218352.

„Vue.Js - The Progressive JavaScript Framework | Vue.Js“. o. J. Zugegriffen 12. Juli 2022. https://vuejs.org/.

„Welcome to Python.Org“. o. J. Python.Org. Zugegriffen 12. Juli 2022. https://www.python.org/.

„Why Svelte Is the Next Big Thing in JavaScript Development“. o. J. Zugegriffen 12. Juli 2022. https://naturaily.com/blog/why-svelte-is-next-big-thing-javascript-development.

Abbildungsverzeichnis

Abbildung 1 Solution Stack 6
Abbildung 2 SPA Scheme 8
Abbildung 3 JSON Example 9
Abbildung 4 NGRX State Management Lifecycle 11
Abbildung 5 PostgreSQL Logo 13
Abbildung 6 MongoDB Logo 14
Abbildung 7 MariaDB Logo 15
Abbildung 8 SQLite Logo 18
Abbildung 9 Spring Logo 20
Abbildung 10 Node Process Cycle 21
Abbildung 11 NodeJS Logo 22
Abbildung 12 Express Logo 22
Abbildung 13 Python Logo 24
Abbildung 14 C# Logo 26
Abbildung 15 Reactjs Logo 28
Abbildung 16 Angular Logo 34
Abbildung 17 Svelte Logo 36
Abbildung 18 VueJS Logo 38
Abbildung 19 Entity Relation Ship Model BMS 44