Hochschule Karlsruhe
University of
Applied Sciences

HKA

Evaluation von Bun 1.0: Eine Verbesserung der JavaScript-Laufzeitumgebung?

Seminararbeit von

Ansgar Lichter

an der Fakultät für Informatik und Wirtschaftsinformatik

Universität:	Hochschule Karlsruhe
Studiengang:	Informatik
Professor:	Prof. Dr.-Ing. Vogelsang
Bearbeitungszeitraum:	01.10.2023 - 04.12.2023

Eidesstattliche Erklärung

Ich versichere, dass ich diese Masterthesis selbständig angefertigt, nicht anderweitig für Prüfungszwecke vorgelegt, alle benutzten Quellen und Hilfsmittel angegeben, sowie wörtliche und sinngemäße Zitate gekennzeichnet habe.

(Ort, Datum)	(Ansgar Lichter)

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

PoC Proof of Concept

Kapitel 1

Einleitung

1.1 Motivation

Die kontinuierliche Evolution von Softwaretechnologien und -umgebungen hat einen erheblichen Einfluss auf die Entwicklung und Leistung von Anwendungen. Kürzlich wurde die Version 1.0 der JavaScript Laufzeitumgebung Bun veröffentlicht, die vielversprechende Verbesserungen in Bezug auf die Leistung im Vergleich zu Node.js ankündigt [1].
JavaScript ist eine Programmiersprache, die vor allem im Kontext der Web-Entwicklung verwendet wird [2]. Aktuell erfreut sich JavaScript großer Beliebtheit. In einer Umfrage an Entwickler von Stack Overflow wurden mehr als 89.000 Entwickler befragt. JavaScript ist zum 11. Jahr in Folge die am häufigsten verwendete Programmiersprache. Mehr als 63% der befragten Entwickler haben JavaScript als beliebteste Technologie gewählt. Bei den professionellen Entwicklern ist der Anteil mit mehr als 65% sogar noch höher. Außerdem ist TypeScript, eine stark typisierte Programmiersprache, die auf JavaScript aufbaut, unter den Teilnehmer auch beliebt. Ca. 39% aller Entwickler und ca. 44% der professionellen Entwickler verwenden auch TypeScript. Damit ist TypeScript die 4. beliebteste Programmiersprache. [3] Daraus folgt, dass das Ökosystem von JavaScript eine hohe Praxisrelevanz besitzt.

JavaScript wird nicht nur für die Entwicklung im Frontend, sondern auch für die Entwicklung im Backend verwendet, ungefähr 3% der weltweit bekannten Server verwenden eine Laufzeitumgebung, die JavaScript ausführen kann. [5] Um JavaScript auf einem Server ausführen zu können, wird eine Laufzeitumgebung benötigt. Wie Abbildung 1.1 zeigt, ist Node.js die am weitesten verbreitete Laufzeitumgebung. In der gezeigten Umfrage zum Zustand von JavaScript beantworteten ca. 71% von 30.000 befragten Entwickler, dass sie Node.js als Laufzeitumgebung regelmäßig verwenden.

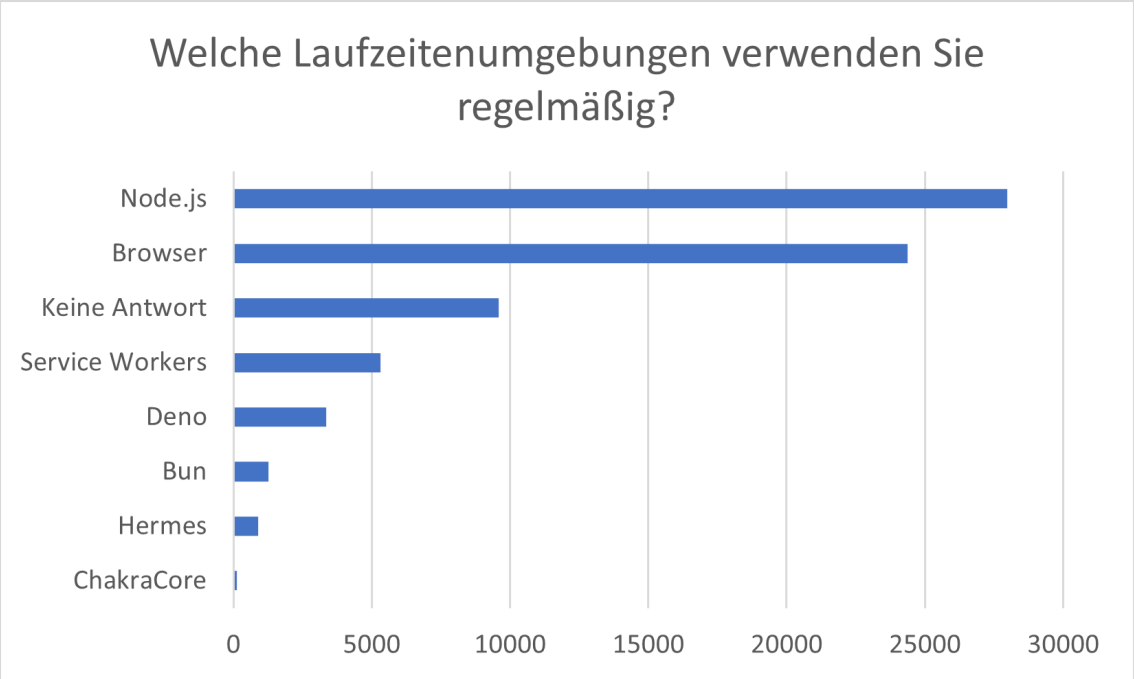

Abbildung 1.1: Nutzungsstatistik von JavaScript-Laufzeitumgebungen [4]

Nur ca. 9% der befragten Entwickler verwenden Deno und ca. 3% Bun als eine Alternative zu Node.js. [4]

1.2 Zielsetzung

Aktuell ist Node.js die Laufzeitumgebung, die am weitesten verbreitet ist. Dennoch gibt erscheinen immer wieder neue Laufzeitumgebungen für JavaScript, die versuchen Node.js zu verdrängen. Eine mögliche Alternative ist Bun, das am 9. September 2023 in der Version 1.0 veröffentlicht worden ist. Die Entwickler von Bun werben mit Features wie erheblicher Performancesteigerung, eleganten Schnittstellen und einer angenehmen Entwicklererfahrung. [1]

Das Hauptziel dieser Arbeit besteht darin, die Version 1.0 der JavaScript Laufzeitumgebung Bun einer eingehenden Evaluierung zu unterziehen. Konkret wird untersucht, ob die in den Ankündigungen versprochene signifikante Leistungssteigerung im Vergleich zu Node.js tatsächlich existiert und reproduzierbar ist. Darüber hinaus wird geprüft, inwiefern bestehende Projekte auf der Basis von Node.js mit Bun kompatibel sind. Die Ergebnisse dieser Arbeit können Entwicklern bei der Entscheidung helfen, ob sie auf Bun 1.0 migrieren sollten, und sie dabei unterstützen, die Leistung ihrer bestehenden Projekte zu verbessern. Insgesamt zielt diese Untersuchung darauf ab,

Klarheit über die Versprechungen von Bun 1.0 zu schaffen und Entwicklern fundierte Informationen für ihre Entscheidungsfindung zur Verfügung zu stellen. Dies spiegelt sich in den folgenden Leitfragen wider:

- Welche konkreten Leistungsverbesserungen können in Bun 1.0 im Vergleich zu Node.js festgestellt werden, und wie lassen sie sich quantifizieren?
- Inwiefern sind Projekte auf der Basis von Node.js kompatibel mit Bun? Wie schwierig gestaltet sich die Migration?
- Welche Herausforderungen und potenziellen Vorteile ergeben sich bei der Verwendung von Bun 1.0 im Vergleich zu Node.js für Entwickler und Projekte?

1.3 Aufbau der Arbeit

Kapitel schreiben

Kapitel 2

Grundlagen

Dieses Kapitel stellt die benötigten Grundlagen vor, die für das Verständnis der darauffolgenden Kapitel notwendig sind. Hierzu zählen die Vorstellung von Node.js und Bun sowie weiterer Grundlagen zu Performanceanalysen.

2.1 Node.js

Node.js ist ein beliebtes Tool für eine große Varianz an Projekten, darunter leichtgewichtige Webservices, dynamische Webanwendungen und Tools für die Kommandozeile. Es handelt sich um eine Open Source, platformunabhängige Laufzeitumgebung, die es ermöglicht JavaScript außerhalb des Browsers auszuführen. Node.js verwendet die V8 JavaScript Engine von Google, die auch in Google Chrome kommt. Dies ermöglicht Node.js eine hohe Performance, weshalb Unternehmen wie Netflix und Uber Node.js in ihren Softwareprojekten einsetzen. [6]

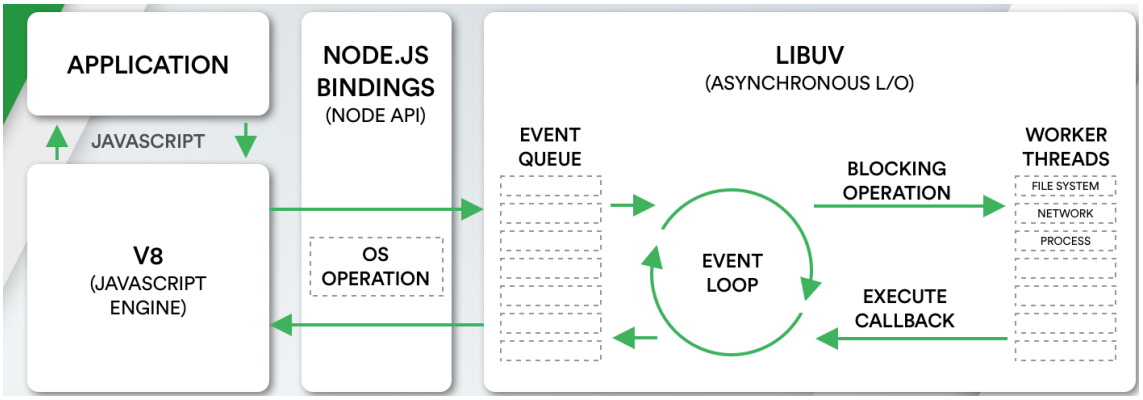

Abbildung 2.1: Node.js Architektur

Evtl. eigene Abbildung nutzen

Abbildung 2.1 zeigt die Architektur von Node.js. Grundsätzlich nutzt Node.js nur einen Thread und erstellt nicht für jede neue Anfrage einen neuen Thread. Sobald eine Applikation gestartet wird, wird in dem einzigen Thread der Node.js-Prozess gestartet. Die V8 Engine optimiert den Maschinencode zusätzlich an häufig benötigten Stellen, wobei dies nicht sofort geschieht, da die Übersetzung in Maschinencode aufgrund der Just-in-Time-Kompilierung zeitsensitive Aufgabe darstellt. Darüber hinaus ist in der Engine ein Garbage Collector integriert, der nicht mehr verwendete Objekte löscht. [7]
Für weitere Aufgaben setzt Node.js auf Bibliotheken, die fertige und etablierte Lösungsansätze für häufig benötigte Aufgaben zur Verfügung stellen. Nur für Aufgaben, für die es keine etablierte Bibliothek gibt, werden eigene Implementierungen verwendet. Im Folgenden werden die wichtigsten Komponenten vorgestellt. [7]

Event Loop
Node.js verwendet eine eventgesteuerte Architektur. Anstatt den Quellcode linear auszuführen, werden definierte Events ausgelöst, für die zuvor Callback-Funktionen registriert wurden. Dieses Konzept wird genutzt, um eine hohe Anzahl von asynchronen Aufgaben zu bewältigen. Um dabei den einzelnen Thread der Anwendung nicht zu blockieren, werden Lese- und Schreiboperationen an den Event Loop ausgelagert. Wenn auf externe Ressourcen zugegriffen werden muss, leitet der Event Loop die Anfrage weiter, und die registrierte Callback-Funktion gibt die Anfrage an das Betriebssystem weiter. In der Zwischenzeit kann Node.js andere Operationen ausführen. Das Ergebnis der externen Operation wird dann über den Event Loop zurückgeliefert. [7]
Während der Laufzeit werden viele Events erzeugt und in einer Message Queue, der Event Queue, nacheinander gespeichert. Node.js nutzt FIFO und beginnt demnach mit der Verarbeitung der ältesten Events und arbeitet sich durch die Queue, bis keine Events mehr vorhanden sind. [8]

Libuv
Der Event Loop von Node.js basiert ursprünglich auf der Bibliothek libev. Diese ist in C geschrieben und für ihre hohe Leistung und umfangreichen Features bekannt. Allerdings stützt sich libev auf native UNIX-Funktionen, die unter Windows auf andere Weise verfügbar sind. Daher dient Libuv als Abstraktionsebene zwischen Node.js und den darunter liegenden Bibliotheken für den Event Loop, um die Laufzeitumgebung auf allen Plattformen nutzen zu können. Libuv verwaltet alle asynchronen I/O-Operationen, einschließlich Dateisystemzugriffe und asynchrone

TCP- und UDP-Verbindungen. [7]

Darüber hinaus bringt Node.js den Node Package Manager (NPM) mit sich. Dieser Paketmanager ist entscheidend für den Erfolg von Node.js, da es im September 2022 mehr als 2,1 Millionen Pakete in diesem Ökosystem gibt. Es gibt somit ein Paket für nahezu alle Anwendungsfälle. Ursprünglich wurde NPM entwickelt, um Abhängigkeiten in Projekten zu verwalten, wird aber mittlerweile auch als Werkzeug für JavaScript im Frontend unterstützt. [6]

Zusammenfassend zeichnet sich Node.js durch eine eventgesteuerte Architektur und durch ein nicht blockierendes Modell für Ein- und Ausgabeoperationen aus, was es leichtgewichtig und effizient macht. Dies hat verschiedene Vor- und Nachteile.
Zu den Vorteilen gehören eine hohe Performance durch die Nutzung der V8 JavaScript Engine und die Plattformunabhängigkeit. Eine weitere Stärke ist die große und aktive Community an Entwicklern. Dank der Popularität gibt es viele etablierte Lösungsansätze, die den Entwicklungsprozess beschleunigen und vereinfachen. [6] Node.js ermöglicht die Verwendung der JavaScript-Sprache sowohl auf der Server- als auch auf der Clientseite. Dies vereinfacht die Entwicklung von Full-Stack-Anwendungen und erleichtert Entwicklern den Einstieg. [2] Das effiziente nicht blockierende I/O-Modell von Node.js ermöglicht es, mehrere gleichzeitige Anfragen effizient zu verarbeiten und eignet sich daher gut für anwendungsspezifische Aufgaben, die viele gleichzeitige Verbindungen erfordern.
Quelle einfügen
Skalierbarkeit?
Allerdings existieren auch Nachteile bei der Verwendung von Node.js. Das Single-Threaded-Modell kann bei rechenintensiven oder CPU-lastigen Aufgaben zu Engpässen führen, da es nur einen Hauptthread für die Ausführung von Code gibt [9]. Bei komplexen Anwendungen kann die Verwaltung von Callbacks und Promises zur Bewältigung von asynchronem Code kompliziert werden.
Quelle einfügen
Ein weiterer Nachteil ist, dass Node.js im Vergleich zu einigen anderen Laufzeitumgebungen über eine begrenzte Standardbibliothek verfügt, sodass Entwickler häufig auf externe Module und Pakete zurückgreifen müssen. Darüber hinaus kann die Qualität einiger NPM-Pakete variieren, und schlecht gewartete Module können zu Kompatibilitätsproblemen und Sicherheitsrisiken führen.
Quelle einfügen
Insgesamt ist Node.js eine leistungsfähige und vielseitige Laufzeitumgebung, die sich gut für bestimmte Anwendungsfälle eignet, insbesondere wenn es darum geht, skalierbare und asynchrone Anwendungen zu entwickeln.
Quelle einfügen
JavaScript programs executed by Node.js have access to a set of Node.js-specific globals like Buffer, process, and __dirname in addition to built-in modules for per-

2.2 Bun

Bun ist ein Open Source Toolkit für JavaScript. Dieses kombiniert verschiedene wichtige serverseitige Komponenten, um eine leistungsstarkes Paket zur Verfügung zu stellen. Bun ist auf macOS und Linux für die produktive Nutzung freigegeben. Unter Windows steht aktuell nur eine experimentelle Version zur Verfügung, deren Performance noch nicht optimiert worden ist. Als Alternative kann über das Windows Subsystem für Linux die veröffentlichte Linux-Version genutzt werden. [1] Ursprünglich ist Bun als ein persönliches Freizeitprojekt von Jared Sumner gestartet. Mittlerweile hat es sich als wettbewerbsfähige Alternative zu etablierten Technologien in der Webentwicklung etabliert. [10]

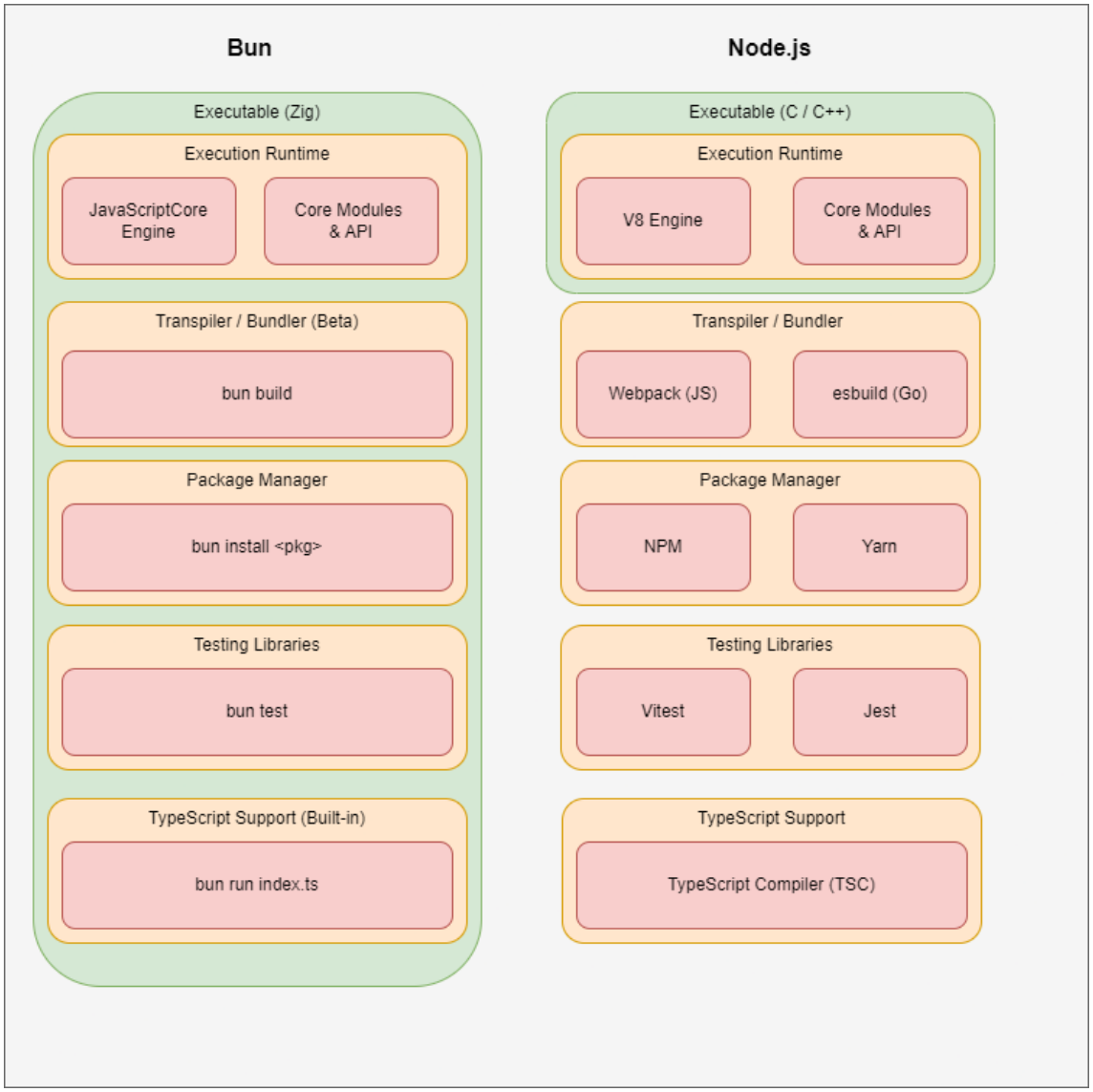

Abbildung 2.2: Comparison of the Ecosystems of Bun and Node.js

Abbildung 2.2 zeigt das Ökosystem von Bun im Vergleich zu Node.js. Im Toolkit von Bun sind folgende Komponenten enthalten: eine Laufzeitumgebung für JavaScript, ein Paketmanager wie NPM (siehe Kapitel 2.1) oder Yarn, ein Transpiler wie Babel, ein Build-Tool wie Webpack, Bibliotheken zum Testen wie Jest oder Vitest und integrierte Unterstützung für TypeScript. [1] In Node.js ist nur die Laufzeitumgebung enthalten, die anderen Komponenten müssen separat installiert werden. Dies bietet allerdings mehr Flexibilität bei der Auswahl der gewünschten Tools. Bun strebt an, das Rundum-sorglos-Tool zu sein, damit alle benötigten Funktionalitäten im Kontext von JavaScript nativ verfügbar sind. Gleichzeitig sollen dadurch die Abhängigkeiten einer Software auf Basis von Bun reduziert werden. [11]

Quellen für Abbildung oder wird das aus dem Text ersichtlich?

Quelle

Bun ist in Zig geschrieben [11]. Zig ist eine systemnahe Programmiersprache wie C und C++, die sich vor allem auf Einfachheit und Klarheit für ein besseres Verständnis konzentriert [12]. Die Entwickler haben sich aufgrund sehr guten Performance und des Speichermanagements für Zig entschieden. Anstatt der V8 Engine von Google verwendet Bun die JavaScriptCore Engine [11]. Diese ist die Engine für WebKit, die unter anderem in Apple's Safari-Browser genutzt wird [13]. In Kombination mit Zig sorgt die JavaScriptCore Engine für die bessere Performance von Bun. Dies ist auf die Architektur der JavaScriptCore Engine mit 3 Just-In-Time-Kompilern und Interpreter. Dadurch kann die Engine den Quellcode noch besser optimieren. So verbessert sich die Ausführungszeit des Quellcodes mit Bun. Darüber hinaus ermöglicht Zig mit dessen manuellem Speichermanagement und Klarheit im Kontrollfluss und Allokieren von Speicher weitere Verbesserungen der Effizienz. [11] Zusätzlich führen die beiden Komponenten noch zu drastisch reduzierten Startzeiten [11]. Das ist vor allem im Bereich des Serverless Computing ein enormer Vorteil gegenüber anderen Alternativen. Denn dies hilft die Skalierbarkeit einer Software zu verbessern, indem neue Knoten schneller hinzugezogen werden können.

Quelle

Quelle?

Node.js bietet viele Module, globale Objekte und Standard-Web-APIs an. Bun möchte eine nahtlose Integration mit Node.js anbieten. Dazu haben die Entwickler verbesserte Versionen für viele dieser Objekte implementiert. Hierzu zählen beispielsweise:

- Standard-Web-API: fetch, Request, Response,
- Module: http, https, path,
- Globale Objekte: btoa, atob. [11]

Allerdings existieren auch viele Module und globale Objekte, für die die Unterstützung teilweise oder komplett fehlt, zum Beispiel assert oder http2. Daraus folgt, dass die

Kompatibilität von bestehenden Projekten auf Basis von Node.js von den verwendeten Objekten und Modulen abhängt. [11]

2.3 Performanceanalyse

TODO

Kapitel 3

Titel 3

3.1 Section 3.1

TODO

Kapitel 4

Titel 4

4.1 Section 4.1

TODO

Kapitel 5

Titel 5

5.1 Section 5.1

TODO

Literatur

[1] JARED SUMNER und PARTOVI, ASHCON, MCDONNEL, COLIN. *Bun 1.0*, 2023. URL: \url{https://bun.sh/blog/bun-v1.0}.

[2] ETHAN BROWN. *Web development with Node and Express : leveraging the JavaScript stack*, Beijing, November 2019. URL: \url{https://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=2295093}.

[3] STACK OVERFLOW. *Stack Overflow Developer Survey 2023*, 2023. URL: \url{https://survey.stackoverflow.co/2023/#overview}.

[4] SACHA GREIF und ERIC BUREL. *State of JavaScript 2022*, 2022. URL: \url{https://2022.stateofjs.com/en-US/other-tools/}.

[5] Q-SUCCESS, Hrsg. *Usage statistics of JavaScript for websites*, 2023. URL: \url{https://w3techs.com/technologies/details/pl-js}.

[6] OPENJS FOUNDATION, Hrsg. *Introduction to Node.js*, 2022. URL: \url{https://nodejs.dev/en/learn/}.

[7] SEBASTIAN SPRINGER. *Node.js : das umfassende Handbuch*. 4., aktualisierte und erweiterte Auflage. Rheinwerk Computing. Bonn: Rheinwerk, 2022. ISBN: 9783836287654. URL: \url{http://deposit.dnb.de/cgi-bin/dokserv?id=992e511c601d4a5f84179bebaa309635&prov=M&dok_var=1&dok_ext=htm}.

[8] OPENJS FOUNDATION. *The Event Loop*, o. J. URL: \url{https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick#what-is-the-event-loop}.

[9] NIMESH CHHETRI. *A comparative analysis of node. js (server-side javascript)*. 2016. URL: \url{https://repository.stcloudstate.edu/csit_etds/5/}.

[10] MATTHEW TYSON. *Explore bun.js: The all-in-one JavaScript runtime*, 2023. URL: \url{https://www.infoworld.com/article/3688330/explore-bunjs-the-all-in-one-javascript-runtime.html}.

[11] BUN, Hrsg. *Offizielle Dokumentation*, URL: \url{https://bun.sh/docs}.

[12] ZIG SOFTWARE FOUNDATION. *Why Zig When There is Already C++, D, and Rust?*, o. J. URL: \url{https://ziglang.org/learn/why_zig_rust_d_cpp/#a-package-manager-and-build-system-for-existing-projects}.

[13] APPLE, Hrsg. *WebKit*, URL: \url{https://webkit.org/}.

Anhang A

Anhang Kapitel 1