INTERNATIONALE HOCHSCHULE

Projektarbeit

Internationale Hochschule Duales Studium

Studiengang: Informatik

Wie beeinflusst die Asynchronität von Operationen innerhalb eines Node.js Express Endpunkts die Performance in Lasttest?

Müller Korbinian

Matrikelnummer: 102302316

Adresse

Lohwald Straße 59

86356 Neusäß

Abgabedatum: 30.09.2024

Inhalt

Einleitung

In der heutigen Zeit steht die Effizienz und Performance an erster Stelle bei Software. Insbesondere, wenn es sich dabei um eine Webanwendung handelt. Das Thema wird besonders interessant, wenn diese in der Lage sein sollte, einer sehr großen Nutzerzahl standzuhalten. Eine Technologie, die entwickelt wurde, um mit den immer mehr ansteigenden Anforderungen zu Recht zu kommen, ist Node.js. Dies ist eine Plattform, die es ermöglicht, JavaScript auf dem Server auszuführen. Dies wird durch die V8 Engine von Google ermöglicht. Ein Alleinstellungsmerkmal von Node.js ist die Möglichkeit, Code asynchron, sprich nicht blockierend, auszuführen.

Diese Arbeit untersucht, wie sich die Asynchronität von Operationen innerhalb eines Node.js-Express-Endpunkts auf die Performance auswirkt. Es wird untersucht, wie und in welchem Maße die asynchrone Verarbeitung an Anfragen zu einer höheren Effizienz und einer geringeren Antwortzeit führt. Dazu wird ein Experiment durchgeführt, bei dem zwei Express.js Endpunkte implementiert werden. Einer verarbeitet die Anfragen asynchron und der andere synchron. Beide Endpunkte werden einem Lasttest unterzogen und die Ergebnisse werden miteinander verglichen.

Das Ziel dieser Arbeit ist es, die Auswirkungen der asynchronen Programmierung auf den Webserver zu quantifizieren. Dies wird durch ein praxisnahes Experiment realisiert. Die Ergebnisse helfen, eine fundierte Aussage über Vorteile und Grenzen zu treffen.

Theoretische Fundierung

Dieser Abschnitt der Arbeit beschäftigt sich mit den theoretischen Grundlagen. Die weiteren Abschnitte basieren auf dem hier dargelegten Wissen. Zunächst werden grundlegende Begriffe erläutert. In diesem Abschnitt wird erklärt, was unter Node.JS, Express, Asynchrone Programmierung und Lastentest zu verstehen sind. Im darauffolgenden Abschnitt werden die theoretischen Grundlagen erläutert, auf denen die Tests basieren.

Node.JS und Express

Node.JS ist ein JavaScript-Framework, das die durch Google entwickelte V8 Engine nutzt, um JavaScript außerhalb des Webbrowsers auszuführen. Dabei wird der Quellcode asynchron ausgeführt. Beim Start des Node.js-Prozesses wird eine Eventloop gestartet, die auf eingehende Events reagiert und diese asynchron abarbeitet (Huang and Cai, 2018, S.1-3).

Anhand dieser einzigartigen Architektur ist Node.JS in der Lage, sehr effizient zu arbeiten. Mit einer Speichergröße von 8 GB sind maximal 40000 Verbindungen mit dem Webserver möglich. Im Vergleich dazu sind bei herkömmlichen serverseitigen Programmiersprachen, wie Java oder PHP, in etwa 4000 Nutzer möglich. Dies beruht auf der Annahme, dass die herkömmlichen Programmiersprachen in etwa 2 MB

Arbeitsspeicher für jeden neuen Thread benötigen. Neben diesen Stärken weist Node auch eine Schwäche für CPU-intensive Aufgaben auf(Huang, 2020).

Laut der Entwicklerumfrage von Stackoverflow (https://survey.stackoverflow.co/2024) ist Node.js eine der begehrtesten Programmiersprachen. Kein Wunder, dass auch auf Node.JS selber bei den Web-Frameworks an Zweiterstelle ist. An siebter Stelle ist das Framework Express.JS, dieses basiert auf Node.JS. Anhand dieser Popularität wurde das Framework Express.JS für diese Arbeit ausgewählt. Die Entwickler von Express beschreiben dieses als ein schnelles und minimalistisches Web-Framework für Node.js. Es bietet eine große Anzahl an HTTPS-Funktionalitäten, dadurch wird es erleichtert, schnell und leicht robuste APIs zu entwickeln (https://expressjs.com).

Asynchrone Programmierung

Unter Asynchronität in der Programmierung versteht man, wenn der Programmcode nicht kontinuierlich ausgeführt wird. Dieser wartet, bis Daten angekommen sind oder ein Event ausgelöst wird. Dadurch werden die Systemressourcen freigegeben, wenn diese nicht benötigt werden. Diese Funktionalität ist sehr stark in JavaScript integriert (Flanagen, 2020 S.342).

Es ist wichtig zu erwähnen, dass Java und Node.JS bei der Ausführung von Quelltext nur einen Thread nutzen. Durch die asynchrone Auslegung von Node.JS ist es dennoch möglich, einen hohen Grad an Parallelität zu erreichen. Es kommt zu einer Performance, die in anderen Programmiersprachen mittels Threads erreicht wird, mit einem Bruchteil der Ressourcennutzung (Flanagen, 2020 S.583).

Lasttest und Performance-Metriken

Eines der wichtigsten Themen, das beim Testen anfällt, ist der Test der Performance. Dabei muss nicht die gesamte Software umgesetzt sein. Sobald die ersten Elemente umgesetzt sind, kann damit angefangen werden. Im Bereich der Webentwicklung ist eine der wichtigsten Fragen, welches die maximale Anzahl an Nutzern ist. Damit die Ergebnisse aussagekräftig sind, ist es essenziell, dass der Test so nah an der produktiven Umgebung ist wie möglich (Kleuker, 2019 S.355-356).

In seinem Buch beschreibt Kleuker(2019), dass eine weitere Maßnahme für die Qualitätsbewertung Metriken sind. Dabei wird ein Kriterienkatalog mit relevanten Aspekten erstellt. Im Bereich Softwareentwicklung können die Aspekte automatisiert überprüft und ausgewertet werden (Kleuker, 2019 S22-23).

Für die Evaluation in dieser Projektarbeit werden folgende Metriken ausgewertet:

- Antwortzeit: Die benötigte Zeit, die Anfrage an den Server zu schicken, diese zu bearbeiten und das Ergebnis zurückzusenden. Die Zeit wird in Millisekunden, MS, gemessen.

- Durchsatz: Die Anzahl an Requests, die in einer Sekunde verarbeitet werden können
- CPU- und Speicherverbrauch: Diese bestimmen, wie effizient der Server seine Ressourcen während der Nutzung einsetzt.

Methodik

Dieser Abschnitt der Arbeit befasst sich mit dem praktischen Vorgehen. Zunächst wird erläutert, wie die Tests strukturiert sind. Im Anschluss wird dargestellt, wie sich die Testumgebung zusammensetzt. Darauf folgt die Implementierung der API-Endpunkte.

Testumgebung

Um die Experimente durchzuführen, wird ein Webserver mit dem Node.js-FrameworkExpress.js entwickelt. Mittels des Frameworks Express.js wird ein REST-API-Service implementiert. Die API stellt zwei Endpunkte zur Verfügung, einen synchronen und einen asynchronen. Beide Endpunkte verarbeiten eingehende Anfragen auf die gleiche Weise. Nach dem Empfang des Requests wird auf die MySQL-Datenbank zugegriffen und ein SQL-Statement ausgeführt. Das Ergebnis der Datenbankabfrage wird in Form eines JSON-Objekts zurückgegeben.

Der Express.JS Server wird direkt auf dem PC ausgeführt. Hierbei wird der Port 3000 benutzt. Die Datenbank wird in einem Docker-Container gestartet. Der Datenbankcontainer wird mittels Docker Compose gestartet. Hier wird die Version 3.9 des Docker Compose File benutzt. Das MySQL-Image nutzt die Version 8.0. Die Datenbank ist auf dem lokalen Netzwerk unter dem Port 3306 erreichbar.

Die Hardware des Desktop-PCs, auf dem die Tests durchgeführt werden, besteht aus:

- Prozessor: IntelCore I9-9900 KF, nicht übertaktet
- Arbeitsspeicher: 32 GB DDR-4, 2100 GHZ
- Grafikkarte: Nvidia RTX 2080 Super

Folgendes sind die Software, die für die Entwicklung und Durchführung des Experiments genutzt worden sind:

- Betriebssystem: Windows 10 Professional
- IDE: WebStorm, JetBrains
- Docker: Version XX, 16 GB Arbeitsspeicher
- Node.js: Version 20.11.1
- Express.js: Version 10.2.4
- Artillery, Version, Software für Lasttest.

Datenbank und Datenquellen

Die Daten, die für den Lasttest genutzt werden, stammen von der Plattform Kaggle. Die Daten sind unter Laptop Prices (kaggle.com) zu erreichen. Dieser Datensatz enthält folgende Informationen zu Laptops: Hersteller, Produktname, technische Spezifikationen, Bildschirmgröße und Preis. Die Daten stehen in Form einer CSV-Datei öffentlich zum Download bereit.

Datenbankaufbau

Für die Verbindung zu der Datenbank wurde die Administrationsoberfläche von HeidiSQL benutzt. In dieser wurde eine neue Datenbank mit dem Namen Sapphire angelegt. In dieser Datenbank wurde eine Tabelle mit dem Namen laptop_prices angelegt. In dieser wurde mittels der Funktionalität, Import CSV, von HeidiSQL die Daten importiert. Die wichtigsten Felder in der neu angelegten Datenbank sind:

- Company: Der Hersteller des Laptops.
- Product: Das spezifische Laptop-Modell.
- TypeName: Der Typ des Laptops (z. B. Ultrabook, Notebook).
- Inches: Bildschirmgröße in Zoll.
- Ram: Arbeitsspeicher in GB.
- OS: Das Betriebssystem.
- Weight: Gewicht des Laptops in kg.
- Price_euros: Der Preis des Laptops in Euro.

Die Datenbank ist so strukturiert, dass alle relevanten Informationen schnell und effizient abgerufen werden können.

In der ausgeführten Abfrage werden alle relevanten Daten zu den Laptops zusammengetragen, insbesondere die komplette Systemspeichergröße. Alle Elemente werden nach den folgenden Kriterien gefiltert:

- Preis liegt zwischen 500 € und 2000 €
- Die Taktfrequenz der CPU muss über 2.5 GHz liegen
- Die Displaygröße liegt zwischen 13 und 17 Zoll
- Der Laptop besitzt mehr als 8 GB Arbeitsspeicher

Die Rechner werden nach der Systemspeicherkapazität sortiert und die ersten 10 Elemente zurückgegeben.

Aufbau der Lasttests

Der Aufbau des Lasttests ist ausschlaggebend, um realistische und aussagekräftige Resultate zu erreichen. In diesem Abschnitt wird erläutert, wie der Test aufgebaut und durchgeführt wird. Des Weiteren wird erklärt, welche Tools zum Einsatz kamen.

Simuliertes Lastszenario

Das Lastszenario wird in mehreren Schritten aufgebaut. Die Anzahl der gleichzeitigen Benutzer wird insgesamt in vier Phasen erhöht. In der ersten Phase sind es 100 simulierte Nutzer. In Phase 2 werden 250 Nutzer simuliert. In der Dritten sind es 500 Anfragen pro Sekunde. Die Belastung auf die Endpunkte wird pro Testdurchlauf 2 Minuten lang simuliert.

Insgesamt wird dieser Testablauf für jeden Endpunkt 3-mal wiederholt. Der Testablauf wird automatisiert mittels eines Batch-Scripts ausgeführt. Dieses startet die nötigen Prozesse und verwaltet die Ergebnisdateien.

Antwortzeit und Durchsatz

Diese beiden Punkte sind die wichtigsten in der Leistungsbeurteilung einer REST-API. Die Antwortzeit beschreibt die Dauer, die der Server benötigt, um die Anfrage zu verarbeiten. Für ein aussagekräftiges Ergebnis zu erreichen, berechnet Artillery sowohl die durchschnittliche Antwortzeit, die Maximale und das 95.Perzentil. Dadurch wird sichergestellt, dass die statistische Anomalität relativiert wird. Ebenso wird die maximale Antwortzeit für jede der Phasen dokumentiert.

Den maximalen Durchsatz beschreibt die Anzahl an Anfragen, die in einer Zeitspanne von einer Sekunde bearbeitet werden können. Dies wird in der letzten Phase ermittelt. In der Artillery Konfiguration wird festgelegt, dass die maximale Antwortzeit nicht mehr als 5000 MS betragen darf.

Insgesamt ist zu erwarten, dass beide Endpunkte eine ähnliche Performance besitzen, wenn der Durchsatz niedrig ist. Dies liegt daran, dass Node.js bereits asynchron arbeitet. Erst bei hohem Durchsatz wird erwartet, dass der asynchrone Endpunkt eine bessere Antwortzeit aufweist.

Hardwareauslastung

Im Testbetrieb werden die Ressourcen des Rechners kontinuierlich überprüft. Dabei wird die CPU und RAM-Nutzung des Express.js Servers überprüft und in einer CSV-Logdatei gespeichert. Um diese Daten zu erheben, wird ein Python Script erstellt und benutzt. Dieses filtert nach dem Express.js PID. Dadurch wird sichergestellt, dass die Daten nicht durch andere laufende Prozesse verfälscht werden.

CPU-Auslastung

Die CPU-Auslastung ist ein wesentliches Merkmal für die Effizienz der Anfrageverarbeitung. Da Node.js nur auf einem Thread ausgeführt wird, steigt die CPU-Nutzung mit der Intensität der Serverbeanspruchung. Hier wird erwartet, dass der asynchrone Endpunkt weniger Ressourcen benötigt als der synchrone. Dies basiert auf der Annahme, dass durch die asynchrone Verarbeitung weniger Ressourcen blockiert werden.

Arbeitsspeicher-Auslastung

Die RAM-Auslastung wurde ebenfalls gemessen, da ein hoher Speicherverbrauch auf ineffiziente Speicherverwaltung hindeuten könnte. Vor allem bei komplexen Datenbankabfragen und hoher Benutzerlast sind Schwankungen in der Speicherauslastung zu erwarten.

Implementation der Endpunkte

Die Implementation der Endpunkte erfolgt auf zwei verschiedenen Paradigmen. Einer wird synchron und der andere asynchron entwickelt. Die beiden Implementationen unterscheiden sich an zwei wichtigen Stellen. Die Callbackfunktion, die aufgerufen wird, wenn eine Anfrage an den asynchronen Endpunkt gestellt wird, ist mit dem Schlüsselwort async versehen. Dadurch wird diese Funktion aufgerufen und Node.js wartet nicht, bis die Funktion vollends ausgeführt wurde. In der Zwischenzeit können weitere Anfragen angenommen werden. Der zweite Unterschied liegt daran, dass in der Funktion das Schlüsselwort await nutzt. Dies geschieht bei der Datenbankabfrage. Dies veranlasst Node.js, die Verarbeitung der Anfrage zu pausieren, bis die Datenbank die entsprechenden Daten liefert.

Durch diese beiden Unterschiede wird der Thread nicht blockiert und Node.js kann mehrere Anfragen quasi parallel bearbeiten. Dieser Unterschied bietet eine ideale Grundlage, um die Auswirkung von asynchroner Programmierung auf die Performance einer Rest-API zu testen.

Ergebnisse

In diesem Abschnitt werden die Resultate der durchgeführten Tests detailliert dargelegt. Das Ziel ist es, die Leistung der Endpunkte zu quantifizieren. Durch die Ergebnisse können Schlüsse auf die Effizienz und Skalierbarkeit der Implementationen gezogen werden.

Testergebnis

Performance des Endpunkts

Tabelle 1der APIS, in MS; eigene Darstellung

		Synchron			Asynchron		
		mean	95%	max	mean	95%	max
100 Anfragen	1 Versuch	2	2	4	2	2	4
	2 Versuch	2	2	4	2	2	5
	3 Versuch	2	2	4	2	2	3
250 Anfragen	1 Versuch	2	3	12	2	3	7
	2 Versuch	2	3	8	2	3	8
	3 Versuch	2	3	8	2	3	8
500 Anfragen	1 Versuch	2	4	38	2	5	42
	2 Versuch	2	13	106	2	7	49
	3 Versuch	2	6	73	2	8,9	67

Die Ergebnisse, Tabelle 1, liefern interessante Erkenntnisse zur Performance von synchronen und asynchronen Endpunkten. In dem Lasttest mit niedriger Intensität liefern beide Implementationsansätze nahezu identische Resultate. Die mittlere Antwortzeit liegt bei beiden bei 2 MS, das 95 % Perzentil ist ebenfalls 2 MS. Die längste Antwortzeit lag bei der synchronen Implementierung bei 4 MS und bei der asynchronen bei 5 MS.

Bei 250 Nutzern pro Sekunde fällt auf, dass die mittlere Antwortzeit gleich bleibt. Das 95-Prozent-Perzentil erhöht sich auf 3 Millisekunden. Die maximale Antwortzeit ist bei beiden Implementationen ähnlich, bei 8 MS. Hier ist zu betonen, dass es bei der synchronen Implementierung in einem Testdurchlauf zu einer maximalen Antwortzeit von 12 MS kam. Bei der Asynchronen gab es auch einen Ausreißer, ebenfalls im ersten Durchlauf betrug die Zeit nur 7 MS.

Im letzten und auch intensivsten Test, mit 500 gleichzeitigen Nutzern, stellt sich heraus, dass die mittlere Antwortzeit ebenfalls 2 MS betrug. Beim 95 % Perzentil erreicht die synchrone Programmierung eine maximale Antwortzeit von 13 MS und die asynchrone eine Zeit von 8,9 MS. Bei der maximalen Antwortzeit liegt die synchrone Implementierung bei max. 106 MS. und die Asynchrone 67 MS. Es fällt auf, dass die asynchrone Implementierung konstantere Ergebnisse liefert.

Hardwareauslastung

Der Test mit dem synchronen Endpunkt liefert folgende Ergebnisse:

CPU-Auslastung

- **100 Nutzer**:
    - Die durchschnittliche CPU-Auslastung beträgt etwa 23–37 %, mit Spitzenwerten um 106 % und einem 95%-Perzentil von etwa 57–76 %.
- **250 Nutzer**:
    - Die CPU-Auslastung steigt deutlich, der Durchschnitt liegt bei etwa 67–99 %, und die maximale Auslastung erreicht 270 %. Das 95%-Perzentil bleibt in einem Bereich von 140–190 %.
- **500 Nutzer**:
    - Bei 500 gleichzeitigen Nutzern sehen wir die höchste CPU-Auslastung. Der Durchschnitt bewegt sich zwischen 115 % und 129 %, während die maximale CPU-Auslastung bis zu 420 % erreicht. Das 95%-Perzentil reicht von 250 % bis 312 %.

Speicherauslastung

- **100 Nutzer**:
    - Der durchschnittliche Speicherverbrauch beträgt etwa 97 MB, mit maximalen Werten von bis zu 106 MB und einem 95%-Perzentil, das sich zwischen 103–106 MB bewegt.
- **250 Nutzer**:
    - Die durchschnittliche Speicherauslastung liegt hier bei etwa 91–99 MB, wobei die Spitzenwerte bei ca. 111 MB liegen. Das 95%-Perzentil bewegt sich im Bereich von 103–106 MB.
- **500 Nutzer**:
    - Bei 500 gleichzeitigen Nutzern sehen wir eine deutlich höhere Speicherbelastung. Die Durchschnittswerte liegen zwischen 100–105,4 MB, während die Spitzen bis zu 115 MB reichen. Das 95%-Perzentil liegt bei etwa 113–114 MB.

Folgendes ist die Hardware-Auslastung für den asynchronen Endpunkt:

CPU-Auslastung

- **100 Nutzer**:
    - Die durchschnittliche CPU-Auslastung beträgt etwa 34–37 %, mit Spitzenwerten um 105 % und einem 95%-Perzentil von etwa 57–88 %.

- **250 Nutzer**:
    - Die CPU-Auslastung steigt deutlich, der Durchschnitt liegt bei etwa 78–90 %, und die maximale Auslastung erreicht 211 %. Das 95%-Perzentil bleibt in einem Bereich von 163–193 %.
- **500 Nutzer**:
    - Bei 500 gleichzeitigen Nutzern sehen wir die höchste CPU-Auslastung. Der Durchschnitt bewegt sich zwischen 110 % und 130 %, während die maximale CPU-Auslastung bis zu 361 % erreicht. Das 95%-Perzentil reicht von 241 % bis 289 %.

Speicherauslastung

- **100 Nutzer**:
    - Der durchschnittliche Speicherverbrauch beträgt etwa 91–95 MB, mit maximalen Werten von bis zu 104 MB und einem 95%-Perzentil, das sich zwischen 96,9–101 MB bewegt.
- **250 Nutzer**:
    - Die durchschnittliche Speicherauslastung liegt hier bei etwa 91–99 MB, wobei die Spitzenwerte bei ca. 111 MB liegen. Das 95%-Perzentil bewegt sich im Bereich von 107–110 MB.
- **500 Nutzer**:
    - Bei 500 gleichzeitigen Nutzern sehen wir eine deutlich höhere Speicherbelastung. Die Durchschnittswerte liegen zwischen 98,8–105,4 MB, während die Spitzen bis zu 117 MB reichen. Das 95%-Perzentil liegt bei etwa 110–116 MB.

Performance-Vergleich

Bei einem Anfragevolumen von 100 Anfragen pro Sekunde besitzen beide Endpunkte eine sehr ähnliche Performance in Hinsicht auf Antwortzeit und Durchsatz.

Beide Endpunkte sind ähnlich performant bei einem Anfragevolumen von 250 Nutzern pro Sekunde. Die Daten zeigen, dass tendenziell mehr Zeit benötigt wird, dieses Volumen abzuarbeiten, im Vergleich zu den 100 Anfragen. Im Test ist eine leichte Tendenz zu sehen, dass der synchrone Endpunkt in Extremfällen eine höhere Antwortzeit besitzt. Dadurch lässt sich auch eine leichte Tendenz erkennen, dass der asynchrone Endpunkt einen höheren Durchsatz erzielt.

Besonders interessant, bei dem letzten Test, ist die mittlere Antwortzeit, diese ist immer noch bei 2 MS. Jedoch ist das 95% Perzentil wie erwartet angestiegen. In der maximalen Antwortzeit gibt es die größten

Unterschiede. Bei dem synchronen Endpunkt sind die Werte höher als bei dem asynchronen. Das zeigt, dass der synchrone Endpunkt, wie erwartet, bei hoher Belastung effizienter ist. Der asynchrone Endpunkt liefert ziemlich konstante Ergebnisse, im Vergleich zu dem synchronen. Das bestätigt die Tendenz, dass der asynchrone Endpunkt einen höheren Durchsatz besitzt.

In der Ressourcennutzung gibt es keine signifikanten Unterschiede zwischen den beiden Implementationen. Die Nutzung der Hardware ist wie erwartet mit zunehmender Belastung gestiegen. Der Grund für eine CPU-Auslastung von über 100 % liegt an der Art, wie Python mit den Kernen umgeht. Python betrachtet nur die physischen Kerne, nicht die virtuellen. Sprich, bei einem Wert von über 100 % sind alle physischen Kerne ausgelastet und noch ein Teil der Virtuellen.

Fazit

Das Experiment hat gezeigt, dass asynchrone Operationen einen Einfluss innerhalb eines Expresses.js Servers besitzen. Die gewählten Testzentren haben gezeigt, dass die größten Unterschiede in der maximalen Antwortzeit, sprich nahe dem Limit gibt. Die Tests haben auch gezeigt, dass bei einem erwarteten maximalen Anfragevolumen von 250 Anfragen pro Sekunde, es keine Rolle spielt, wie der Endpunkt implementiert ist. Ab da liefert der asynchrone Endpunkt bessere und vorausschaubare Performance.

In zukünftigen Arbeiten sollte dieses Thema erneut aufgegriffen werden. Insbesondere sollten zukünftige Forschungen untersuchen, ob sich dieses Verhalten nur auf Windows auftritt oder auch auf Webservern. Ebenso sollten die Tests mit variablen Lasten ausgeführt werden. Eine Änderung ist es, mit mehr Nutzern zu testen, dies wird aktuell durch Node eingeschränkt, da es zu Fehlern im Node Core kommt, wenn sehr viele Akteure versuchen, gleichzeitig eine Datenbankverbindung aufzubauen. Eine andere ist es, in dem Endpunkt nicht nur die Daten mittels einer komplexen Abfrage aus der Datenbank zu laden, sondern diese Daten im Anschluss noch zu verarbeiten, dadurch wird eine höhere Last auf Node.js erzeugt.

Diese Erkenntnisse sind für alle Webentwickler essenziell, deren Ziel ist, leistungsstarke Webserver zu entwickeln.

Literaturverzeichnis

Kleuker, S. (2019). Performance- und Lasttests. In *Qualitätssicherung durch Softwaretests* (S. 355–395). Springer Fachmedien Wiesbaden. http://link.springer.com/10.1007/978-3-658-24886-4_12

Flanagan, D. (2020). *JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language*. O'Reilly Media, Incorporated. http://ebookcentral.proquest.com/lib/badhonnef/detail.action?docID=6199356

Huang, X. (2020). Research and application of Node.js core technology. In *2020 International Conference on Intelligent Computing and Human-Computer Interaction (ICHCI)* (S. 1–4). https://doi.org/10.1109/ICHCI51889.2020.00008

Huang, J., & Cai, L. (2018). Research on TCP/IP network communication based on Node.js. In *ADVANCES IN MATERIALS, MACHINERY, ELECTRONICS II: Proceedings of the 2nd International Conference on Advances in Materials, Machinery, Electronics (AMME 2018)*, Xi'an City, China (S. 040115). https://doi.org/10.1063/1.5033779

Technology | 2024 Stack Overflow Developer Survey. (2024). https://survey.stackoverflow.co/2024/technology#2-programming-scripting-and-markup-languages

Express - Node.js web application framework. (n.d.). https://expressjs.com/