JavaScript Kompendium

Daniel Herken

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Informationen sind im Internet über http://dnb.d-nb.de abrufbar.

www.bmu-verlag.de
info@bmu-verlag.de

Lektorat: Barbara Honold
Einbandgestaltung: Pro ebookcovers Angie
Druck und Bindung: Wydawnictwo Poligraf sp. zo.o. (Polen)

Taschenbuch-ISBN: 978-3-96645-057-7
Hardcover-ISBN: 978-3-96645-058-4
E-Book-ISBN: 978-3-96645-056-0

JavaScript Kompendium

Inhaltsverzeichnis

Alle Programmcodes aus diesem Buch sind über GitHub zum Download verfügbar. Dadurch müssen Sie sie nicht abtippen:
https://bmu-verlag.de/javascript-kompendium/

Außerdem erhalten Sie die eBook Ausgabe zum Buch im PDF Format kostenlos auf unserer Website:

https://bmu-verlag.de/javascript-kompendium/
Downloadcode: siehe Kapitel 10

Kapitel 1
Einleitung

In der Programmierung geht es einzig und allein darum, mit Hilfe eines Computers ein bestimmtes Ziel zu erreichen. In unserer modernen Welt ist dies tagtäglich hundert- oder sogar tausendfach der Fall. Egal ob Sie Ihre aktuellen E-Mails abrufen, im Supermarkt an der Kasse bargeldlos bezahlen oder Ihre Waschmaschine einschalten. All diese Abläufe sind in ihrem Kern nichts weiter als Computerprogramme, welche mit der physischen Welt interagieren.

Um diese Art der Interaktion herzustellen, muss zuvor jemand dem Computer beigebracht haben, wie diese Interaktion ablaufen soll. Dies wird in aller Regel durch ein Computerprogramm erreicht. All diese uns ständig umgebenden Programme müssen erstellt, gewartet und zum Großteil auch überwacht werden. Diese Aufgabe fällt im Allgemeinen einem oder mehreren Programmierern zu.

Die spannende Frage ist: Wie spannen wir den Bogen zwischen der abstrakten Idee „rufe E-Mails ab“ zu einem Programm, welches der Computer für uns ausführen kann? An sich kann der Computer nur sehr einfache Befehle verstehen und ausführen. Er kann dies allerdings viel schneller und öfter als Sie oder ich. So ist es möglich, dass Computerprogramme Aufgaben in Sekunden erledigen, für die Menschen Stunden oder sogar Tage brauchen würden.

Damit Sie als zukünftiger Programmierer effektiv mit dem Computer kommunizieren können, verwenden wir heutzutage abstrahierende Programmiersprachen wie JavaScript. Durch das Abstrahieren zwischen Befehlen, welche der Computer verstehen kann, und für uns lesbarem Programmcode sind wir in der Lage, komplexe Sachverhalte für die Maschine auszudrücken. Es wird also ein Übersetzer zwischengeschaltet, welcher unseren Code in ein für die Maschine verständliches Format überführt. Der Übersetzer wird dabei als Compiler bezeichnet.

Interessanterweise ähneln Programmiersprachen mehr oder weniger deutlich der Sprache, welche wir selbst zur Kommunikation verwenden, übersetzt ins Englische. Sich in diesen formalisierten Programmiersprachen auszudrücken, ist dabei nicht immer ganz einfach und erfordert oftmals eine disziplinierte Ausdrucksweise und die Fähigkeit, von Zusammenhängen zu abstrahieren.

Dieses Buch richtet sich sowohl an Einsteiger als auch an erfahrene Programmierer, die gerne eine weitere Programmiersprache ihrem „Wortschatz“ hinzufügen wollen. Daher wird vor jeder praktischen Anwendung jeweils auch die dazugehörige Theorie

hinreichend erklärt. Spezielles Vorwissen ist somit nicht erforderlich, ein Basiswissen in HTML und CSS ist in späteren Kapiteln allerdings von Vorteil.

Neben Theorie und Praxis wird in diesem Buch in den späteren Kapiteln eine Beispielanwendung erstellt und dann nach und nach erweitert. Zusätzlich gibt es am Ende vieler Unterkapitel Übungsaufgaben, welche Ihnen helfen sollen, das Gelernte direkt anzuwenden und zu vertiefen.

Um Ihnen das Nachverfolgen zu erleichtern und Tipparbeit zu ersparen finden sich alle Aufgaben, Lösungen und Beispielanwendungen auf GitHub unter https://github.com/danielherken/javascript_Kompendium.

Ich wünsche Ihnen viel Erfolg mit JavaScript.

Alle Programmcodes aus diesem Buch sind über GitHub zum Download verfügbar. Dadurch müssen Sie sie nicht abtippen:
https://bmu-verlag.de/javascript-kompendium/

Außerdem erhalten Sie die eBook Ausgabe zum Buch im PDF Format kostenlos auf unserer Website:

https://bmu-verlag.de/javascript-kompendium/
Downloadcode: siehe Kapitel 10

Kapitel 2

JavaScript verstehen

Lassen Sie uns zu Beginn einen genaueren Blick auf die Programmiersprache JavaScript, ihre Entstehungsgeschichte, die verschiedenen verwendeten Standards sowie die Einsatzmöglichkeiten werfen.

Besonders die Bedeutung der verschiedenen Versionen darf nicht unterschätzt werden, da Sprachfeatures nicht in jeder Version in gleicher Weise verfügbar sind.

2.1 Geschichte, Entstehung und aktuelle Einsatzmöglichkeiten

Ursprünglich wurde JavaScript 1995 als Teil des Browsers Netscape Navigator eingeführt mit dem Ziel, Webseiten und deren Funktionalität dynamisch zu gestalten. Bis dahin waren Webseiten darauf ausgelegt, statische Inhalte anzuzeigen und diese durch Links zu verknüpfen. All die dynamischen Inhalte, welche wir heute als selbstverständlich ansehen, gab es 1995 noch nicht.

Dabei war JavaScript, wie der Name bereits vermuten lässt, stark von Java beeinflusst. Trotzdem unterschieden sich die beiden Sprachen damals wie heute so stark, dass sich Wissen aus einer der Programmiersprachen nicht ohne Weiteres auf die andere übertragen lässt.

Nachdem Netscape Navigator die Integration von JavaScript eingeführt hatte, musste der damalige einzige echte Konkurrent Internet Explorer von Microsoft nachziehen. Dies geschah recht prompt, indem die sogenannte JScript-Engine in den Internet Explorer implementiert wurde. Beide Browser waren damit in der Lage, neben statischen Inhalten auch erste dynamische Elemente von Webseiten abzubilden.

Durch den wachsenden Konkurrenzdruck zwischen beiden Browsern und die fehlende Standardisierung von JavaScript überboten sich Netscape und Microsoft in immer neuen, besseren, meist aber inkompatiblen Features. Dies hatte zur Folge, dass die Webentwickler regelmäßig vor die Wahl gestellt wurden, Features entweder für jeden Browser separat zu entwickeln oder sich auf einen der beiden großen Browser zu konzentrieren. Banner mit „Best viewed in Internet Explorer“ bzw. „Best viewed in Netscape Navigator“ wurden schnell zur üblichen Norm, um den schnell wachsenden Implementierungsaufwand in Grenzen zu halten.

Durch die weitgehende Standardisierung von JavaScript konnte dieses Problem im weiteren Verlauf zwar deutlich abgeschwächt werden, allerdings sind auch aktuell

noch lange nicht alle Features des JavaScript-Standards in jedem Browser oder in jeder JavaScript-Umgebung gleichermaßen verfügbar.

Heutzutage hat JavaScript einen sehr hohen Verbreitungsgrad, und es ist daher auf praktisch jedem Gerät möglich, in einer oder anderer Form JavaScript auszuführen. Nach wie vor am häufigsten kommt JavaScript in den verschiedenen Webbrowsern wie Google Chrome, Firefox, Internet Explorer oder Safari zum Einsatz. Jeder dieser Browser bringt eine eigene sogenannte JavaScript-Engine mit, mit der sich JavaScript im Kontext einer Website oder sogar als Erweiterung des eigentlichen Browsers ausführen lässt.

In der Programmierung möchte man gleichen Code außerdem an so vielen Orten wie möglich wiederverwenden können (auch um dadurch Entwicklungskosten zu sparen). Das hat zur großen Popularität von Node.js geführt. Node.js ist eine JavaScript Laufzeitumgebung, welche auf der JavaScript Engine von Google Chrome (V8 genannt) basiert und mit der auch der serverseitige Anteil einer Anwendung in JavaScript entwickelt und als Gegenstück zu dem Code genutzt werden kann, welcher clientseitig (also im Webbrowser) läuft.

Aber die Einsatzmöglichkeiten von JavaScript enden hier noch lange nicht. Durch die Verwendung des Frameworks Electron können mithilfe von JavaScript sogar betriebssystemübergreifende Desktop-Anwendungen erstellt werden.

Mit Frameworks wie React Native lässt sich JavaScript-Code sowohl im Browser als auch in Smartphone Apps nutzen, ohne dass der Quellcode angepasst werden müsste. Ein namhafter Nutzer (und Entwickler des Frameworks) ist beispielsweise Facebook.

Im weiteren Verlauf dieses Buches werden wir uns sowohl mit JavaScript im Browser als auch auf der Serverseite befassen. Zunächst werden wir aber einen genaueren Blick auf die verschiedenen Standards (bzw. Versionen) von JavaScript werfen, da diese maßgeblich sind für die einsetzbaren Features und Sprachkonstrukte.

2.2 ECMAScript-Versionen

Nach der Einführung von JavaScript im Jahr 1995 wurde schnell klar, dass die Sprache schnell eine umfassende Standardisierung benötigt, um dem Auseinanderlaufen der Implementierung in den beiden Browsern entgegenzuwirken. Daher begann die Industrievereinigung ECMA (European Computer Manufacturers Association) 1996, einen Standard für JavaScript zu erarbeiten und fortlaufend zu verbessern. Aus markenrechtlichen Gründen konnte dabei der Name JavaScript nicht verwendet werden; die Standards werden daher unter dem Namen ECMAScript veröffentlicht.

Mit dem ECMAScript 1 Standard 1997 wurde hier der Grundstein gelegt. Er wurde stufenweise in den Jahren 1998 (ECMAScript 2) und 1999 (ECMAScript 3) erweitert. Allerdings konnte der Standard nicht mit der tatsächlichen Entwicklung in den Browsern Schritt halten und verfehlte daher vorerst das Ziel, Unterschieden in der Implementierung entgegenzuwirken. Es kam außerdem zu sogenannten „Browserwars“, in denen sowohl Microsoft als auch Netscape als Browserhersteller versuchten, durch exklusive Features die Oberhand zu gewinnen. Dabei wurde nicht alles nach den bestehenden Standards umgesetzt.

Dies wurde noch deutlicher, als die einzelnen Gruppen innerhalb der ECMA sich für den nächsten Standard ECMAScript 4 lange Zeit nicht auf eine Richtung der weiteren Sprachentwicklung einigen konnten. Ein Teil des Komitees war der Ansicht, dass JavaScript schnell deutlich bessere und tiefgreifende neue Features bekommen müsse; andere vertraten die Meinung, dass man sich auf die Einfachheit der Sprache konzentrieren solle.

Letztendlich konnte sich keine der Seiten wirklich durchsetzen. Als Kompromiss wurde die Version 4 des ECMAScript übersprungen zugunsten der Version ECMAScript 5 (veröffentlicht 2009), welche deutlich weniger neue Features enthält als ursprünglich geplant. ECMAScript 5 ist die erste Version, die heute noch relevant ist, da sie in den meisten älteren Browsern (z. B. Internet Explorer 10) vollständig verfügbar ist.

Viele der ursprünglichen Ideen aus ECMAScript 4 wurden in späteren Standards dennoch umgesetzt, wenn auch mit einiger Verzögerung. Die zehn Jahre Pause zwischen ECMAScript 3 und ECMAScript 5 hatten aber zur Folge, dass die Implementierungen innerhalb der Browser sich auseinanderentwickelten und voneinander stark unterschieden, was die Webentwicklung für lange Zeit unnötig erschwerte.

Mit ECMAScript 6 (später umbenannt in ECMAScript 2015) wurde JavaScript 2015 um weitere wichtige Sprachfeatures erweitert. Generell kann heute davon ausgegangen werden, dass alle modernen Browser sowie andere Laufzeitumgebungen (wie Node.js) den ECMAScript-2015-Standard beherrschen, was von späteren Standards keinesfalls gesagt werden kann.

Die ab 2015 jährlich erscheinenden neuen JavaScript-Updates umfassen in aller Regel einen überschaubaren, klar definierten Satz an neuen Features und Detailverbesserungen. An diesem Punkt kommt auch zum ersten Mal der Prozess des sogenannten Transpilens auf. Mit transpilen ist hierbei gemeint, JavaScript Code, der nach dem aktuellen Standard entwickelt wurde, in eine Form zu überführen, die auch Browser verstehen können, die noch auf einem älteren Standard arbeiten. Mit dem Transpilen und den dazugehörigen Tools (wie Babel) werden wir uns in einem späteren Kapitel noch genauer befassen.

Es hat sich in der JavaScript-Welt eingebürgert, die verschiedenen Standards mit unterschiedlichen Namen zu bezeichnen. ECMAScript 2019 beispielsweise wird auch ECMAScript 10 oder der Kurzform ES10 genannt. Diese Namen werden uns später in verschiedenen Konfigurationsdateien wieder begegnen.

Mit diesem Verständnis der verschiedenen Standards und ihrer Bedeutung können wir nun einen Blick auf unsere benötigte Entwicklungsumgebung werfen.

2.3 Einrichten der Entwicklungsumgebung

Generell ist nicht viel Software nötig, um JavaScript-Code auszuführen. Die hier vorgestellte Software ist für alle gängigen Betriebssysteme verfügbar, die Screenshots in diesem Buch wurden unter Windows erstellt.

In der einfachsten Form kann das Ausführen von JavaScript direkt in dem Webbrowser erfolgen, der mit jedem Betriebssystem bereits mitgeliefert wird. Für die Entwicklung hat es allerdings Sinn, einen Browser zu verwenden, welcher über gute Entwicklerwerkzeuge (sogenannte Developer Tools) verfügt, also Werkzeuge, mit denen Probleme im Quellcode leichter diagnostiziert werden können. Dafür empfiehlt sich entweder Google Chrome (https://www.google.com/chrome/) oder Mozilla Firefox (https://www.mozilla.org/de/firefox/). Zur Veranschaulichung wird hier im Buch hauptsächlich Google Chrome verwendet.

Um Google Chrome zu installieren, brauchen Sie nur das Setup von der genannten Website herunterzuladen und mit den Standardeinstellungen zu installieren.

Neben dem Webbrowser kann auch die JavaScript-Laufzeitumgebung Node.js (https://nodejs.org/) genutzt werden. Diese werden wir für die generelle Einführung in die Sprache verwenden sowie in späteren Kapiteln für das Ausführen von serverseitigem JavaScript. Zur Installation laden Sie sich die LTS-Version von Node.js für Ihr Betriebssystem herunter und installieren diese wie folgt.
Geben Sie den Ordner an, in den Node.js gespeichert werden soll (bzw. übernehmen Sie die Voreinstellung):

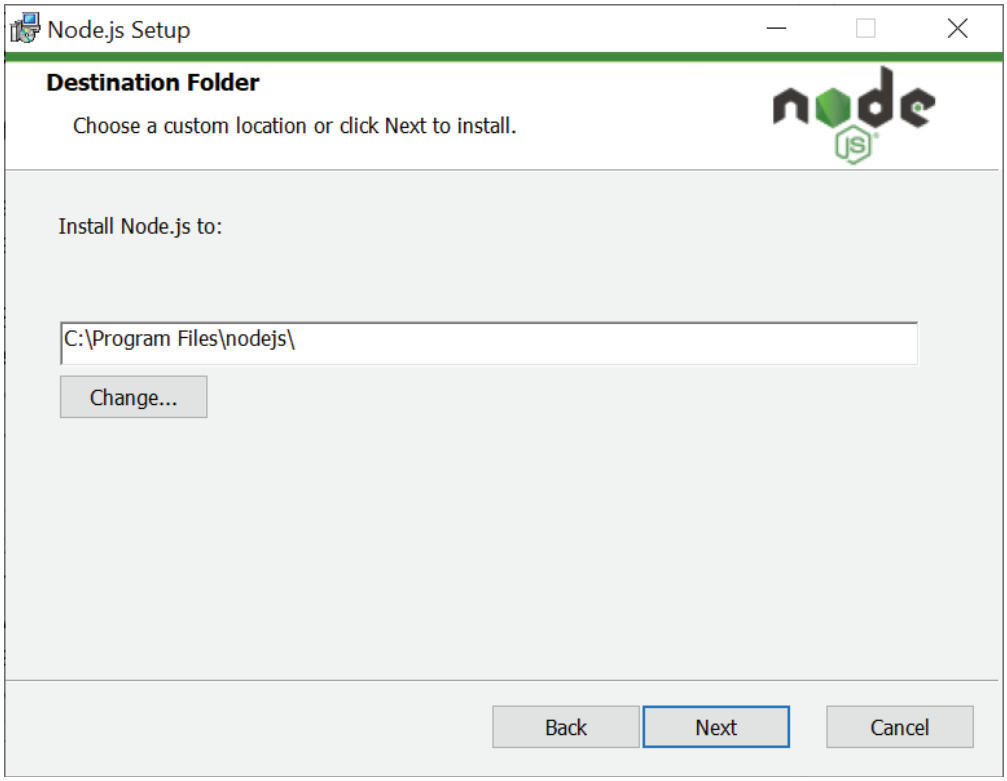

Abb. 2.1 Installation Node.js Schritt 1

Bestätigen Sie mit „Next“, welche der Features installiert werden sollen (auch hier können Sie die Voreinstellung übernehmen):

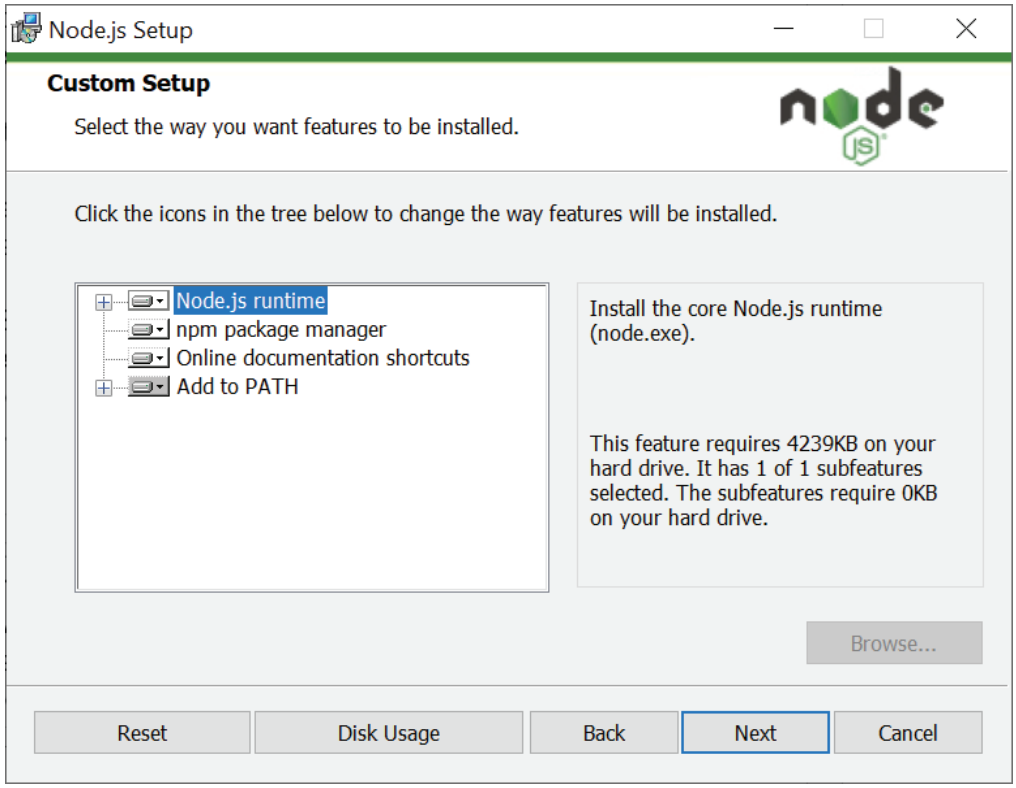

Abb. 2.2 Installation Node.js Schritt 2

Bestätigen Sie mit einem Haken, dass die benötigten Dateien automatisch installiert werden sollen:

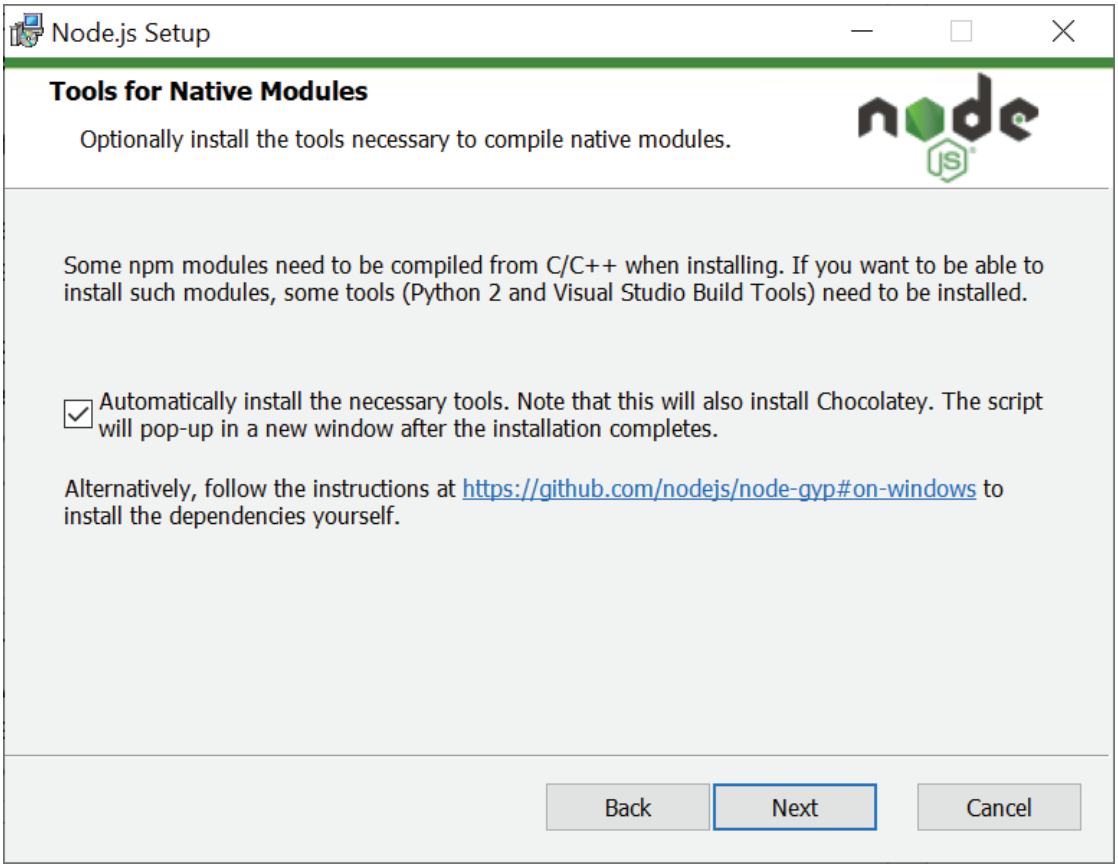

Abb. 2.3 Installation Node.js Schritt 3

Um zu prüfen, ob die Installation erfolgreich war, öffnen Sie ein Konsolenfenster und geben Sie den Befehl `node -v` ein. Daraufhin sollte die aktuelle Node.js Versionsnummer ausgegeben werden.

Unter einer Konsole versteht man einen Eingabebereich, in dem zeilenweise Befehle über die Tastatur eingegeben werden können. Unter Windows heißt dieser Bereich „Eingabeaufforderung“ und ist über den Befehl `cmd` im Windows-Startmenü zu finden. Unter macOS und Linux wird die Konsole als Terminal bezeichnet und ist dort jeweils im „Application Launcher“ zu finden.

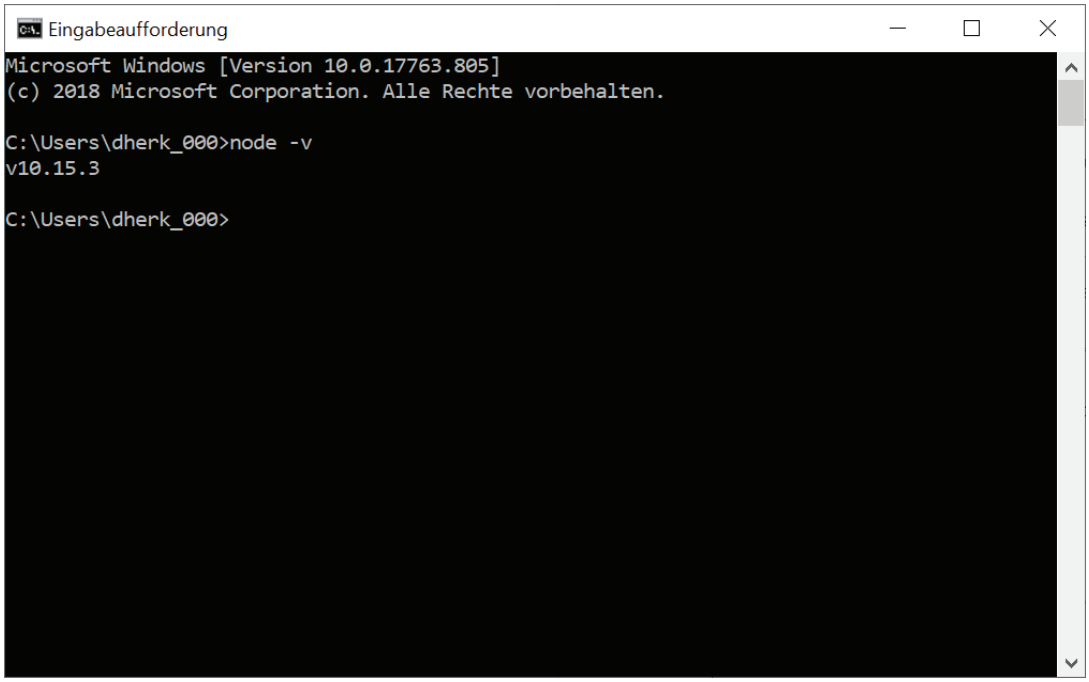

Abb. 2.4 Erfolgreiche Node.js-Installation

Zum Arbeiten mit JavaScript sind neben der Node.js Laufzeitumgebung verschiedene weitere Tools nötig. Sie brauchen einen Texteditor, mit dem Sie die eigentlichen Quellcodedateien erstellen und bearbeiten können. Im besten Fall bietet dieser Texteditor auch Syntax Highlighting; das bedeutet, dass die verschiedenen Aspekte des Sourcecodes in unterschiedlichen Farben dargestellt werden, was das Lesen des Quellcodes deutlich vereinfacht. Der deutliche Unterschied in der Lesbarkeit lässt sich gut anhand der folgenden Beispiele nachvollziehen.

```
1 const faktor1 = 10;
2 const faktor2 = 5;
3
4 // Erste Multiplikation ausführen
5 let ergebnis = faktor1 * faktor2;
6
7 const faktor3 = 7;
8
9 // Zweite Multiplikation ausführen
10 ergebnis = ergebnis * faktor3;
11
12 const faktor4 = 4;
13
14 // Dritte Multiplikation ausführen
15 ergebnis = ergebnis * faktor4;
16
17 // Ergebnis ausführen
18 console.log(ergebnis);
19
```

Abb. 2.5 Quellcode ohne Syntax Highlighting

```
1   const faktor1 = 10;
2   const faktor2 = 5;
3
4   // Erste Multiplikation ausführen
5   let ergebnis = faktor1 * faktor2;
6
7   const faktor3 = 7;
8
9   // Zweite Multiplikation ausführen
10  ergebnis = ergebnis * faktor3;
11
12  const faktor4 = 4;
13
14  // Dritte Multiplikation ausführen
15  ergebnis = ergebnis * faktor4;
16
17  // Ergebnis ausgeben
18  console.log(ergebnis);
```

Abb. 2.6 Quellcode mit Syntax Highlighting

Außerdem wird früher oder später ein Debugger benötigt, welcher es ermöglicht, das eigene Programm Zeile für Zeile auszuführen und bei jedem Schritt dessen Verhalten sowie den Inhalt von Variablen zu analysieren.

Anstatt jedes dieser Tools einzeln zu installieren und zu konfigurieren ist es daher sinnvoll, eine Entwicklungsumgebung (kurz IDE, für Integrated Development Environment) zu nutzen. In diesem Buch wird dafür Visual Studio Code (https://code.visualstudio.com/) verwendet, da es kostenlos und für alle gängigen Betriebssysteme verfügbar ist.

Visual Studio Code beinhaltet alle der vorher genannten Werkzeuge und erlaubt es daher, Quellcode zu erstellen, auszuführen und zu debuggen, ohne dass zwischen verschiedenen Softwarelösungen gewechselt werden muss. Wie genau die einzelnen Funktionen in Visual Studio Code funktionieren, werden wir in späteren Kapiteln genauer betrachten.

Zur Installation von Visual Studio Code kann das Setup für das gewünschte Betriebssystem von der Website des Herstellers heruntergeladen und mit den Standardeinstellungen installiert werden. Nach der erfolgreichen Installation können Sie Visual Studio Code öffnen und werden von der nachfolgenden Benutzeroberfläche begrüßt.

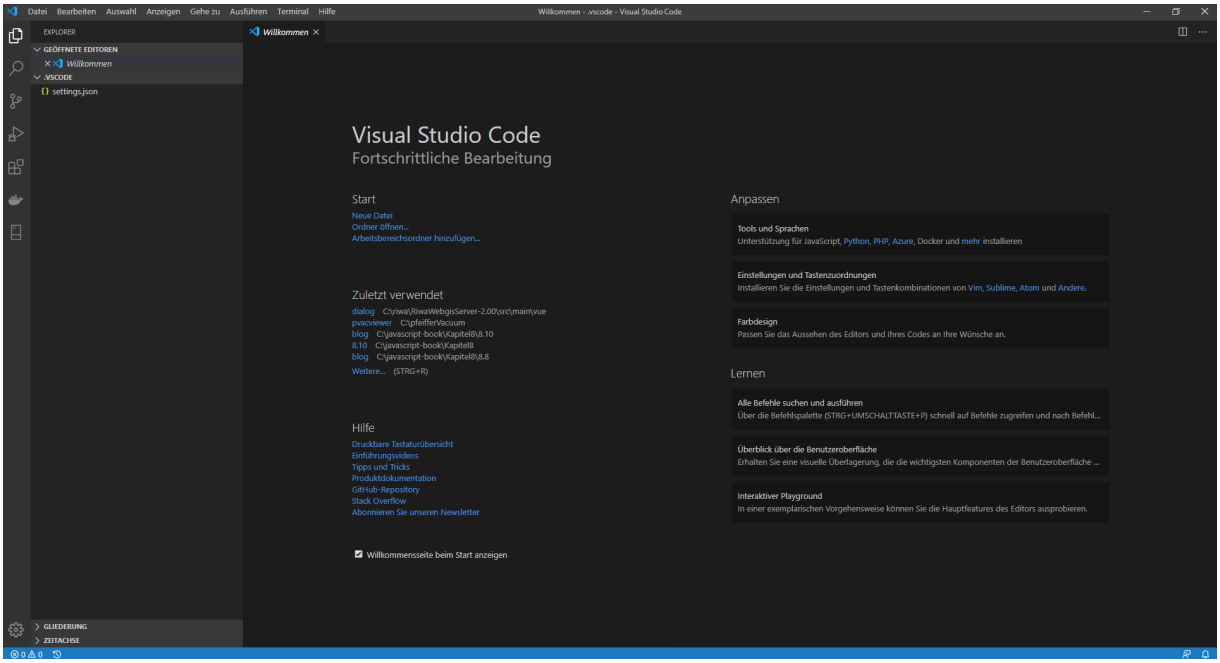

Abb. 2.7 Visual Studio Code

Mit diesen beiden Softwarepaketen ist Ihr Rechner nun voll ausgestattet, sodass Sie mit der Entwicklung beginnen können. Vorher werden wir allerdings noch kurz auf den in Node.js integrierten Packagemanager NPM eingehen.

2.4 Abhängigkeiten managen mithilfe eines Packagemanagers

Wie wir bereits gehört haben, versucht man in der Programmierung, Quellcode möglichst wiederzuverwenden. Dieses Vorgehen erstreckt sich nicht nur auf den eigenen Code. Es soll allgemein das „Rad nicht neu erfunden“ werden. Daher gibt es in der JavaScript-Community unzählige sogenannte Packages, welche man nutzen kann, um den eigenen Code um Funktionalität zu erweitern.

Diese Packages (auch Pakete genannt) müssen in das eigene Projekt integriert werden, damit auf die bestehende Funktionalität zugegriffen werden kann. Dafür gibt es aktuell zwei bekannte sogenannte Packagemanager, welche sich um die Installation der benötigten Packages kümmern. Diese Aufgabe übernimmt NPM, der Paketmanager, der Teil der normalen Node.js-Installation ist; man kann aber auch Yarn (https://yarnpkg.com) verwenden. Beide Paketmanager funktionieren nach dem gleichen Prinzip mit nahezu identischen Befehlen. Aus diesem Grund wird in diesem Buch NPM verwendet; alle Beispiele lassen sich ohne großen Aufwand auch mit Yarn nachvollziehen.

Für alle verfügbaren Pakete gibt es ein zentrales Verzeichnis, was das Auffinden geeigneter Pakete deutlich erleichtert. Dieses Verzeichnis ist über die Suchfunktion unter https://www.npmjs.com/ zu finden. Dort ist es möglich, jeweils nach einem Paketna-

men oder Schlagworten, welche die Funktion beschreiben, zu suchen. Alle dort registrierten Pakete können sowohl mit NPM als auch mit Yarn direkt im eigenen Projekt installiert werden.

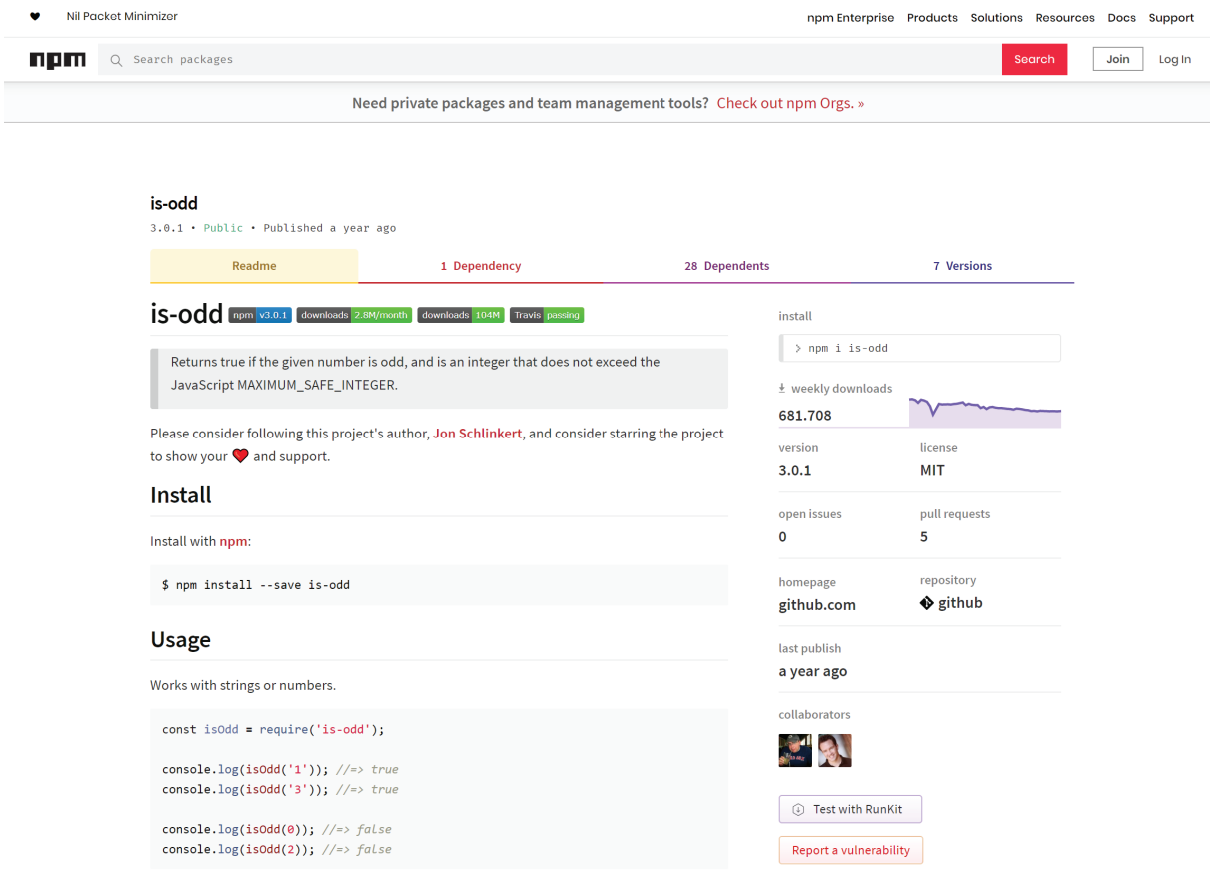

Abb. 2.8 Beispiel eines NPM-Paketes

Bei der Auswahl der Pakete für ein eigenes Projekt ist es ratsam, ein wenig Vorsicht walten zu lassen. Dies ist nötig, da zum einen Pakete Sicherheitslücken enthalten können, welche sich durch die Nutzung des Pakets in das eigene Projekt übertragen. Zum anderen besteht die Gefahr, dass schlecht gewartete Pakete veraltete Sprachkonstrukte beinhalten und möglicherweise in Zukunft nicht mehr korrekt funktionieren. Gute Pakete zeichnen sich in der Regel dadurch aus, dass sie zum einen regelmäßig gewartet werden und zum anderen über eine gute Dokumentation verfügen. Beides lässt sich direkt auf der Website des Paketes feststellen.

Um ein Paket dem eigenen Projekt hinzuzufügen, reicht ein einfacher Befehl in der Konsole: `npm install -save [Paketname]`. Um das oben gezeigte Paket `is-odd` dem eigenen Projekt hinzuzufügen, lautet der Befehl beispielsweise `npm install -save is-odd`. Das gleiche Prinzip kann zum Entfernen angewendet werden. Der Befehl hierfür lautet `npm uninstall is-odd`.

Der Paketmanager verwaltet außerdem eine Liste aller Pakete, welche für das aktuelle Projekt nötig sind. Sie findet sich in einer Datei namens **package.json**, die auch einige Informationen zum Projekt enthält (z. B. den Namen und die Version). Um alle in der Datei **package.json** hinterlegten Pakete mit einem Befehl zu installieren, reicht es aus, in der Konsole `npm install` oder die Kurzform `npm i` auszuführen.

Ein Beispiel für eine einfache **package.json** Datei mit zwei installierten Paketen (auch Abhängigkeiten genannt) findet sich im Folgenden.

```
{
  "name": "Beispiel-Projekt",
  "version": "1.0.0",
  "dependencies": {
    "is-odd": "3.0.1",
    "is-even": "3.0.1"
  }
}
```

Listing 2.1 Beispiel einer einfachen package.json

Diese package.json-Datei ist zentral in jedem JavaScript-Projekt und wird uns später noch öfter begegnen. Bevor wir nun mit dem ersten Quellcode beginnen können, müssen wir uns noch mit dem Thema Versionsverwaltung auseinandersetzen.

2.5 Einfache Versionsverwaltung mit Git

Aus der modernen Softwareentwicklung ist die Versionsverwaltung heutzutage nicht mehr wegzudenken. Mit ihrer Hilfe ist es möglich, auch in größeren Teams gleichzeitig an einer gemeinsamen Codebasis zusammenzuarbeiten, die verschiedenen Änderungen verwalten und zusammenfügen zu lassen und jederzeit die Übersicht über die Änderungen zu behalten.

Die Basis einer jeden Versionsverwaltung ist eine zentrale Stelle (in der Regel ein Server), welche über alle Änderungen zu den verschiedenen Quellcodedateien Buch führt. Dieses Verzeichnis der Änderungen wird als Historie (history) bezeichnet. Dadurch kann man jederzeit Änderungen nachvollziehen und einen bestimmten Zustand des Projektes wiederherstellen.

In diesem Buch verwenden wir die Versionsverwaltung Git, die den aktuellen Standard in großen Teilen der Branche darstellt. Die Installation von Git unterscheidet sich in den Betriebssystemen. Unter Linux und MacOS ist Git durch den jeweiligen Paketmanager des Systems mit dem Betriebssystem bereits vorhanden und verfügbar. Unter Windows kann Git für Windows (https://gitforwindows.org/) mit den Standardeinstellungen installiert und verwendet werden. Das eigentliche Arbeiten mit Git funktioniert in allen Betriebssystemen gleich.

Git verwendet einen zentralen Server, welcher ein sogenanntes Repository bereitstellt. Darin werden alle Änderungen gespeichert. Dafür kann ein eigener Git-Server verwendet werden, aber auch Server in der Cloud. Beliebte Beispiele für Anbieter von Cloud-Servern sind GitHub (https://github.com), BitBucket (https://bitbucket.org/) oder auch GitLab (https://about.gitlab.com/).

Um auch auf dem eigenen Rechner (lokal) Zugriff auf das Repository (und damit auf die einzelnen Änderungen) zu haben, ist es nötig, das Repository auf den eigenen Rechner zu kopieren. Dieser Vorgang wird als klonen bezeichnet und kann durch einen einfachen Konsolenbefehl durchgeführt werden. Zuerst wechselt man in das Verzeichnis, in welchem das lokale Repository auf dem Rechner liegen soll. Zum Wechseln des Verzeichnisses kann man in allen Betriebssystemen den Befehl `cd [Pfad zum lokalen Repository]` (z. B. `cd c:\Projekte\JavaScript`) nutzen. In diesem Verzeichnis werden alle nötigen Dateien für das Projekt inklusive der Quellcodedateien abgelegt. Dann gibt man ein: `git clone [Url zum Repository auf dem Server]`, also z. B. `git clone https://github.com/danielherken/javascript-book`.

Mit diesem Befehl werden der aktuelle Stand des Projektes sowie die komplette Historie auf den lokalen Rechner kopiert. Sie haben daher jederzeit lokal Zugriff auf alle Änderungen, die bis zum Zeitpunkt des Klonens ausgeführt und auf dem Server hinterlegt wurden. Auf Änderungen, welche andere Teammitglieder bisher nur in ihrem eigenen lokalen Repository gemacht haben, haben Sie an dieser Stelle keinen Zugriff.

Im lokalen Git-Repository, können lokale Änderungen zwei Zustände haben, „staged“ und „commit“: Den Status „staged“ haben alle Änderungen, welche prinzipiell gesichert werden sollen, aber noch nicht zur Übertragung an den Server einem Commit zugeordnet wurden. Änderungen, welche bereits einem Commit zugeordnet wurden, erhalten den Status „commit“. Diese Unterscheidung ist sinnvoll da Sie unter Umständen Dateien lokal bearbeitet haben, diese aber nicht an den Server übertragen wollen, da diese Änderungen nur Ihr lokales System betreffen oder noch nicht abgeschlossen sind.

Um eine weitere Strukturierung zu ermöglichen und um das konfliktfreie Arbeiten zu erleichtern, basiert das Arbeiten mit Git in der Regel auf sogenannten Branches. Ein Branch ist ein definierter Codezustand, mit welchem weitergearbeitet werden kann. Zumeist gibt es mindestens zwei verschiedene Branches: develop und master. Der develop-Branch stellt dabei den aktuellen Stand mit allen Änderungen dar, wohingegen der master-Branch der zuletzt in die Produktivumgebung übernommene Stand ist.

Gehen wir einmal die normale Arbeitsweise mit Git beim Implementieren eines neuen Features anhand der nötigen Git-Befehle durch:

Zunächst wird im bestehenden lokalen Repository auf den Branch gewechselt, auf welchem die neuen Änderungen basieren sollen, mit dem man also weiterarbeiten möchte. Für ein neues Feature wäre das in der Regel der develop-Branch, für eine aktuell nötige Fehlerbehebung, welche schnellstmöglich in die Produktion überführt werden soll, könnte der master-Branch sinnvoller sein.

Um auf den gewünschten Branch zu wechseln, wird der Befehl `checkout` verwendet. Für unser Beispiel werden wir hier auf den develop-Branch wechseln, von dem aus unser neues Feature entwickelt werden soll. Der komplette Befehl lautet daher `git checkout develop`.

Nun, da wir uns auf dem develop-Branch befinden, sollte vor Beginn der Entwicklung geprüft werden, ob unser lokaler Branch mit dem Branch auf dem Server übereinstimmt oder ob es neuere Änderungen auf dem Server gibt. Um den lokalen Branch mit dem Server abzugleichen, ist der Befehl `pull` einzusetzen. In unserem Beispiel wäre es daher `git pull origin develop` (`origin` ist dabei ein Schlüsselwort, das angibt, dass develop sich auf einen Branch bezieht. Alternativ könnte ein Pull z. B. auch auf ein anderes Repository unter einer anderen Url durchgeführt werden).

Von diesem aktuellen Stand werden wir nun einen neuen sogenannten Feature-Branch erstellen. Hier können anschließend alle für das Feature nötigen Änderungen durchgeführt werden, ohne dass dieser unfertige Stand die Entwicklung in anderen Branches beeinflusst. Um einen neuen Branch zu erstellen, kann erneut der `checkout`-Befehl genutzt werden: `git checkout -b branch-name`.

Auf diesem Stand kann nun die eigentliche Entwicklung durchgeführt werden. Dabei werden in den meisten Fällen Dateien angepasst und wenn nötig neue Dateien erstellt. Mit Git kann man dabei jederzeit eine Übersicht der Änderungen erhalten, indem der Befehl `status` ausgeführt wird. Hier ein Beispiel für eine `git status`-Ausgabe:

```
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   BZF.Frontend/components/ExamResultHeader.js
        deleted:    package-lock.json
```

Abb. 2.9 Git-Status-Ausgabe

Wenn alle Änderungen abgeschlossen sind, müssen diese dem Git-Server mitgeteilt werden. Dies geschieht durch einen sogenannten Commit. Einem Commit können dabei eine, mehrere oder alle Änderungen hinzugefügt werden. Um eine einzelne Datei hinzuzufügen, gibt es den Befehl `git add [Dateiname]`. Um alle vorhandenen Änderungen dem Commit hinzuzufügen, können Sie

`git add -all :/` ausführen. Um die Änderungen für eine Datei rückgängig zu machen, kann `git checkout -- [Dateiname]` genutzt werden.

Nachdem alle nötigen Dateiänderungen dem Commit hinzugefügt wurden, kann der Commit mithilfe des entsprechenden Befehls finalisiert werden. Git erwartet dabei neben der Liste aller Änderungen einen kurzen Text, welcher die Änderungen beschreibt (die sogenannte „commit message“). Dieser Text hilft, später einen bestimmten Commit in der Liste aller Commits wiederzufinden. Der komplette Befehl, um den Commit zu erstellen, lautet hier `git commit -m "Beschreibender Text"`.

Doch dieser Commit ist bisher nur auf unserem lokalen System vorhanden. Die Änderungen müssen mit einem zusätzlichen Befehl an den Git-Server übermittelt werden. Dafür kommt der push-Befehl zum Einsatz. Für unser aktuelles Beispiel wäre dies `git push origin branch-name`.

Nun sind unsere Änderungen als eigener Branch auf dem Git-Server verfügbar. Bei einem Projekt im Team wird einer der Teamkollegen die gemachten Änderungen überprüfen und, wenn es nichts zu beanstanden gibt, diese in den develop Branch übernehmen. Dies kann durch das sogenannte Mergen oder (im Falle eines externen Git-Hosts wie GitHub) über einen Pull Request geschehen. Von einem Pull Request spricht man, wenn in der Weboberfläche des Git-Hosts eine Übersicht der von Ihnen durchgeführten Änderungen dargestellt wird. Mithilfe dieser Oberfläche können Teamkollegen Ihre Änderungen kommentieren und, wenn alles zur Zufriedenheit aller erledigt wurde, genehmigen.

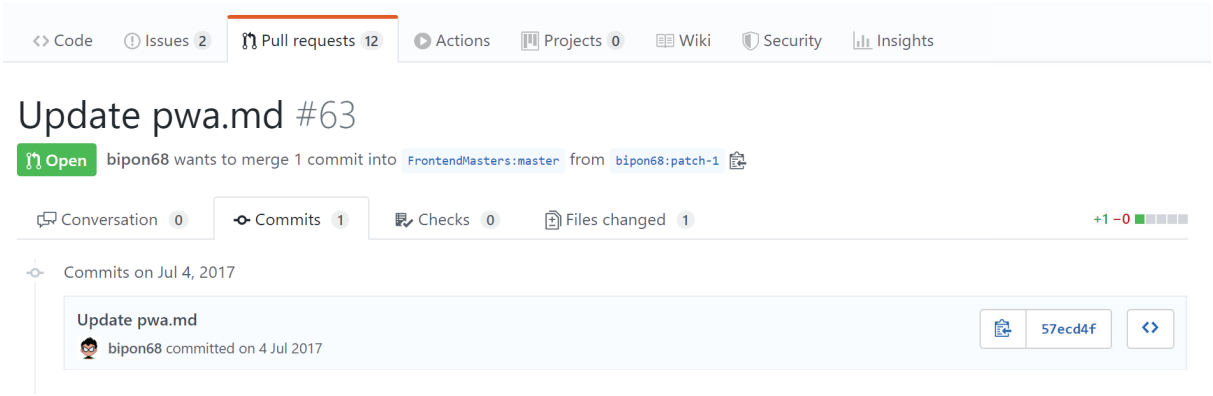

Abb. 2.10 Beispiel für einen GitHub Pull Request

Mit diesem letzten Schritt, der nötig ist, um die Entwicklung an unserem Feature abzuschließen, wird der Feature-Branch in unseren develop-Branch integriert. Wenn Sie einen Git Hoster nutzen, gibt es dafür im Pull Request einen Button; wollen Sie dies manuell ausführen, brauchen Sie den Git-Befehl `merge`. Dazu muss zunächst auf den develop-Branch gewechselt werden (`checkout`), anschließend wird der Feature-Branch in den develop-Branch überführt (`merge`) und schlussendlich werden die neuen Änderungen dem Server mitgeteilt (`push`). Die nötigen Befehle lauten wie folgt:

```
git checkout develop
git merge branch-name
git push origin develop
```

Nun sind alle für das Feature nötigen Änderungen im develop-Branch zu finden und stehen dem ganzen Team für die weitere Arbeit zur Verfügung.

Nachdem wir den grundlegenden Umgang mit der Versionsverwaltung Git kennengelernt haben, können wir uns der eigentlichen Entwicklung mit JavaScript zuwenden.

Alle Programmcodes aus diesem Buch sind über GitHub zum Download verfügbar. Dadurch müssen Sie sie nicht abtippen:
https://bmu-verlag.de/javascript-kompendium/

Außerdem erhalten Sie die eBook Ausgabe zum Buch im PDF Format kostenlos auf unserer Website:

https://bmu-verlag.de/javascript-kompendium/
Downloadcode: siehe Kapitel 10