Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Henning Kahl

Entwicklung eines graphischen Webseiten-Editors basierend auf Google Blockly

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Henning Kahl

Entwicklung eines graphischen Webseiten-Editors basierend auf Google Blockly

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: : Prof. Dr. Ing. Birgit Wendholt
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 1. Dezember 2017

Henning Kahl

Thema der Arbeit
Entwicklung eines graphischen Webseiten-Editors basierend auf Google Blockly

Stichworte
Editor, Webseiten, Drag & Drop, Programming for Non-Programmers, HTML, Visuelle Programmiersprachen, Blockbasierte Programmierung, Google Blockly, Problembasiertes Lernen, EZWEB-Editor

Kurzzusammenfassung
Die vorliegende Bachelorarbeit untersucht, ob ein visueller Drag & Drop-Webseiten-Editor auf Basis der blockbasierten Programmiersprache Google Blockly einen einfachen Einstieg in die Webseitenerstellung ermöglicht. Für eine Validierung wurde dieses Konzept prototypisch umgesetzt, von Probanden getestet und in einer Umfrage untersucht. Das Ergebnis zeigt, dass dieser Ansatz durchaus Potential bietet.
Henning Kahl

Title of the paper
Development of a graphical web page editor based on Google Blockly

Keywords
Editor, Webseiten, Drag & Drop, Programming for Non-Programmers, HTML, Visual Programming, Blockbased Programming, Google Blockly, Problem Based Learning, EZWEB-Editor

Abstract
The present bachelor thesis examines the question whether creating a visual drag and drop website-editor based on the block-based programming language Google Blockly offers a simple start into website-creation. For validation this concept was implemented prototypically, tested with the help of propositi and examined in a survey. The result shows that this approach offers potential.

Inhaltsverzeichnis

1 Einführung

Wir befinden uns im 21. Jahrhundert, einer Zeit, in der Webseiten nicht mehr aus dem Alltag wegzudenken sind. Ein Internetauftritt stellt hierbei das online-Schaufenster eines Unternehmens dar. Ist dieses veraltet oder fehlt es, so wird von potentiellen Kunden schnell zur Konkurrenz mit vorhandenem Internetauftritt gegriffen. Daher liegt eine gewisse Verantwortung beim Unternehmen, diesen Auftritt zu erstellen und zu pflegen. Da oft keine Kenntnisse im Bereich der Webseitenerstellung vorhanden sind, werden fehlerhafte, wenig benutzerfreundliche und langsame Internetauftritte erstellt, die zumeist auch nicht für den Einsatz auf mobilen Endgeräten geeignet sind. Gerade der Einsatz eben dieser mobilen Endgeräte im Konsumverhalten hat in den vergangenen Jahren stark zugenommen. Vgl. [21, 15, 20]

Im besten Fall wird von Seiten der Unternehmen auf professionelle Dienstleister zurückgegriffen. Aber auch die Erstellung durch Homepage-Baukästen hat signifikant zugenommen, da hier verstärkt durch Anbieter solcher Baukästen geworben wurde. Diese haben diverse Vor- und Nachteile, die hier später in dieser Arbeit näher erläutert werden.

An dieser Stelle sei ein Zitat zu bringen, welches unterstreicht, dass im Rahmen der Digitalisierung auch die Wirtschaft das Potential und die Notwendigkeit von Programmierkenntnissen als grundlegende Fähigkeit erkannt hat. Diese Erkenntnis zeigt, dass dies nicht nur auf ausgebildete Informatiker, sondern auch für diverse andere Arbeitnehmergruppen zutrifft.

> *„Schon seit Jahren wird von der Digitalisierung gesprochen und wie sie Unternehmen verändert. Aber inzwischen sind damit nicht mehr nur Apple oder Google gemeint, sondern auch der deutsche Mittelstand. Heute brauchen nicht mehr nur große Soft- oder Hardware-Unternehmen Entwickler und Programmierer für neue Technologien, sondern fast jede Firma. Sogar viele Gärtnereien, zum Beispiel, haben eine Internetseite, viele Firmen brauchen eine App oder einen Onlineshop, um im Markt konkurrenzfähig zu bleiben. [...]*“ - Katharina Heckendorf [14]

Diese Abhandlung beschäftigt sich im Folgenden mit einem neuen innovativen Ansatz, der Unternehmern und Privatpersonen den Einstieg in die Erstellung von Webseiten erleichtern soll und den Weg ebnen soll, diese eigenständig erstellen und verwalten zu können, ohne

einen großen Zeit- und Kostenaufwand vorauszusetzen. Erkenntnisse über die Umsetzung eines solchen Ansatzes sollen erforscht, prototypisch implementiert und durch eine Umfrage validiert werden. Da das Lernen und Lehren zum Erstellen von Webseiten für Einsteiger oft eine große Hürde darstellt, soll an dieser Stelle beleuchtet werden, weshalb es notwendig ist, eine neue Form des Editierens zu generieren. Bisherige Technologien wie z.B. Code-Editoren werden als wenig einsteigerfreundlich empfunden und bieten zwar maximale Freiheit beim Erstellen von Internetauftritten, jedoch können viele Einstellungen schnell überfordern und zum Beispiel fehlende Autovervollständigung zu Fehlern führen.

Der zu erforschende Ansatz soll aufzeigen, wie ein leichter Einstieg auch weiterführend in Hinsicht auf die Arbeit mit Code-Editoren umsetzbar ist, damit Baukasten-Webseiten, welche normalerweise keinen Einblick in die Welt des Codes dahinter gewähren lassen, erweitert werden können. Beispielsweise um sinnvolle, nicht enthaltene Erweiterungen oder um die Möglichkeit, einen Internetauftritt von Anfang an erstellen zu können.

An dieser Stelle gibt ein Zitat vom Sprachen-Lernen durch Filme im Originalton anderer Sprachen mit Untertiteln, die ein- und ausblendbar sind, eine Idee zur Umsetzbarkeit.

> *„Der Hauptgrund für das erfolgreiche Sprachen lernen der Skandinavier ist ihr englischsprachiger Medienkonsum. Skandinavier sehen viele Filme und TV-Serien aus Amerika auf Englisch, während die Italiener fast alles aus Amerika auf Italienisch übersetzen. Dies ist ein wichtiger Unterschied, weil es nicht nur bedeutet, dass die Skandinavier dem Englischen schon früh in ihrem Leben ausgesetzt sind, sondern auch regelmäßig. Wie lange jemand einer Fremdsprache ausgesetzt ist, bestimmt in der Regel sein Verstehensniveau, die Fähigkeit, die Sprache auch dann zu verstehen, wenn schnell gesprochen wird und die Fähigkeit, Laute zu imitieren.*“- Luca Lampariello [9]

Der geplante Editor soll neben der eigentlichen Bearbeitungsansicht auch eine Code-Preview erhalten. Diese Code-Preview soll, wie im Zitat beschrieben, an dieser Stelle den Code als Untertitel zu der eigentlich graphischen Bearbeitungsansicht abbilden.

Die Einarbeitung für Einsteiger in den Bereich der Webseiten-Erstellung mit Hilfe von Code-Editoren mag mitunter sehr aufwendig und erschlagend sein. Als Voraussetzung sind neben HTML-Kenntnissen auch die Einarbeitung in die jeweiligen Editoren an sich erforderlich. Entnommen aus diversen Programmheften von Volkshochschulen betragen Zeit und Kosten für das Erlernen der Grundlagen zum Erstellen von Webseiten durchschnittlich 10-15 Stunden bei Kosten von mind. 120€ pro Kurs. An dieser Stelle sei erwähnt, dass ein Mehrwert von online-basierten Lernplattformen hier offensichtlich ist, da diese zumeist kostenfrei sind und

die Tage, an denen gelernt werden kann, frei einteilbar sind. Weiterhin muss keine Wegstrecke auf sich genommen werden.

Bei anderen Modellen wie dem WYSIWYG-Editor und / oder Baukasten-Editoren steht zweifelsfrei der Vorteil, ohne Code-Kenntnisse Webseiten erstellen zu können, im Vordergrund. Kritisch zu sehen ist hier aber, dass die Technologie dahinter fehleranfällig sein kann und i.d.R. langsamer als selbstgeschriebener Code ist, da dieser oft allgemein gehalten für gängige Probleme geschrieben wurde. Weiterhin kann es sicherheitskritisch sein, dass keine Kontrolle über den Code im Hintergrund besteht. Nicht regelmäßig gepflegter Code kann so zu vermeidbaren Sicherheitslücken im System führen, hier muss jedoch eine Abhängigkeit und ein großes Vertrauen an den Ersteller dieser Editoren vorausgesetzt werden.

Der in dieser Arbeit behandelte Ansatz soll vermeiden, dass Abhängigkeiten gebildet werden, da der Nutzer das Endresultat selber schreibt. Hierbei darf vom Nutzer jedoch natürlich nicht die eigenständige Weiterbildung fehlen.

Nicht zu vergessen, sollte auch erwähnt werden, dass Baukasten-Editoren sehr starr sind. Es wird i.d.R. ein vorgefertigtes Template verwendet und angepasst. Auch die mobile Ansicht, sofern vorhanden, ist meist wenig wandelbar. Daher soll es ebenfalls ein Ziel dieser Arbeit sein, Design durch CSS anpassbar zu gestalten und Bibliotheken wie Bootstrap und Materialize einzubinden zu können.
An dieser Stelle bot es sich an, den Ansatz, visuelle Elemente zu verschieben und zu verbinden, in Kombination mit Fehlervermeidung in einem neuen Konzepteditor zu vereinen, um die Vorteile der o.g. WYSIWYG-Ansätze, sowie von Code-Editoren zusammenführen zu können.

Ein weiterer Punkt der als Mehrwert (wie auch in anderen Editoren) mit einfließen soll, ist die Steigerung der Arbeitsgeschwindigkeit. Elemente mit großem textuellem Umfang sollen schnell mit der Maus als Gesamteinheit verschoben und dupliziert werden können, anstatt sich mit Tastenkürzeln o.ä. herumzuschlagen. Vorgefertigte Code-Blöcke, die in den Kategorien verankert seinen sollen, spielen hierbei ebenfalls mit ein und geben weiteres potentielles Lernmaterial. Vgl. 3.1.1
Um dem Nutzer den Einstieg zu erleichtern, sollen die Constraints der einzelnen Blöcke hinsichtlich Ihrer Konnektoren und Zugehörigkeiten farblich als Empfehlung gekennzeichnet werden. Jedoch sollen keine direkten strukturellen Restriktionen eingebaut werden. Der Editor wird somit kontextfrei behandelt.

Um später auch Reverse-Engineering unterstützen zu können, kommen hier erste Ansätze zum Tragen. Beispielsweise die Code-Preview, welche neben der HTML-Preview und der Bearbeitungsansicht existieren, soll zu Beginn nur readonly sein, jedoch durch eine integrierte

Importfunktion die Möglichkeit geben, Fremdcode einbringen zu können.

Zusammengefasst geht dieser Arbeit nun die Zielsetzung voraus, einen onlinebasierten visuellen Drag & Drop-Webseiten-Editor zu erstellen, der mit Hilfe der blockbasierten Programmiersprache Google Blockly[1] einen einfachen Einstieg in die Webseitenerstellung bieten soll. Validiert wird dieser durch eine Umfrage. Der zu erstellende Editor soll unter dem Namen EZWEB-Editor[2], abgeleitet von „Easy Web“ typische Probleme wie Flüchtigkeitsfehler, das Vergessen der schließenden Tags sowie der Anführungszeichen bei den Attributen dieser Tags gegenüber den Code-Editoren, welche im Allgemeinen maximale Freiheit beim Erstellen von Webseiten bilden, verhindern. Auch entsteht oft unübersichtlicher Quellcode, begründet durch fehlende oder falsche Einrückung, welche hier kompensiert werden soll. Dem Lernen zu Grunde liegen dabei auch eine Informationsquelle und weiterführende Informationen. Hier sollen integrierte Tutorials den Einstieg sowie das Verständnis des grundlegenden Aufbaus von Webseiten schaffen und Hinweise für weiterführende Angebote bereitgestellt werden. Unter diesen Gesichtspunkten soll der EZWEB-Editor mit einer geeigneten Lehrumgebung vor den o.g. Fehlern schützen.

Im Folgenden wird zunächst in Kapitel 2 ein Einblick in die Grundlagen der verschiedenen verwendeten Technologien und Lehransätze eingegangen, die es ermöglichen einen Editor, mit der in der Einführung beschriebenen Zielsetzung umzusetzen. In Kapitel 3 werden die funktionalen und nicht-funktionalen Anforderungen an den zu entwickelnden EZWEB-Editor definiert. Anschließend wird in Kapitel 4 die entwickelte Lösung beschrieben. Dabei wird aufgezeigt, welche der Anforderungen und Konzepte tatsächlich im EZWEB-Editor umgesetzt wurden und wie diese realisiert worden sind. In Kapitel 5 wird der entwickelte EZWEB-Editor evaluiert und die Erkenntnisse der Ergebnisse in den Zusammenhang mit der Zielsetzung gebracht. Die Erkenntnisse liefern hierbei neben dem positiven Ergebnis, dass der EZWEB-Editor die Zielsetzung weitestgehend erfüllt, auch Anregungen für die Weiterentwicklung, die im Ausblick neben der Zusammenfassung in Kapitel 6 beschrieben werden.

[1] https://developers.google.com/blockly
[2] http://ezweb-editor.de

Danksagung

An dieser Stelle möchte ich die Gelegenheit nutzen, all denjenigen zu danken, die mich während der Anfertigung dieser Bachelorarbeit unterstützt und motiviert haben.

Zuerst gebührt mein Dank Frau Prof. Birgit Wendholt, die meine Bachelorarbeit betreut und begutachtet hat, sowie mir mit vielen hilfreichen Anregungen zur Seite stand. Außerdem möchte ich mich bei Herrn Prof. Dr. Stefan Sarstedt für die Bereitschaft zur Erstellung des Zweitgutachtens bedanken.

Ebenfalls möchte ich mich bei meinem Kommilitonen Torben-Dennis Mader bedanken, der mir mit viel Interesse, Geduld und Hilfsbereitschaft zur Seite stand und durch konstruktive Diskussionen maßgeblich dazu beigetragen hat, dass dieses Bachelorarbeit in dieser Form vorliegt.

Ein besonderer Dank gilt auch allen und Teilnehmerinnen und Teilnehmern meiner Umfrage, ohne die diese Arbeit nicht in dieser Form hätte entstehen können. Nicht zuletzt gebührt meiner Familie und meiner Freundin Dank, die mir persönlich zur Seite gestanden haben, wenn ich Ihre Hilfe benötigt habe.

2 Grundlagen und vergleichbare Arbeiten

Um einen Überblick über den aktuellen Stand der Forschung zu erhalten, wird in diesem Kapitel zunächst im Abschnitt 2.1 die Idee „Programming for Non-Programmers“ erläutert und mit Hilfe der Methode problembasiertes Lernen in 2.2 zu einem Lehransatz gebracht. Weiterhin soll durch Vergleichen verschiedener visueller Programmierkonzepte eine geeignete graphische Programmiersprache in 2.3 gefunden werden, mit der sich die HTML-Struktur abbilden lässt. Um den Umfang, die Eigenschaften und die Abgrenzung zu anderen HTML-Editoren herauszuarbeiten, werden in 2.4 aktuelle Webseiten Editoren näher analysiert und miteinander verglichen. Aus diesen Kapiteln soll in 2.5 eine Grundlage für die Erstellung des EZWEB-Editors resultieren. Dabei soll anhand der gewonnen Erkenntnisse geklärt werden, welche Eigenschaften für die Lösung gelten sollen.

2.1 Programming for Non-Programmers

Programming for Non-Programmers, im folgenden PfNP, ist das Keyword für das Lehren von Programmierkenntnissen an die Zielgruppe der Nichtprogrammierer. Zu diesem Begriff ist ein Anstieg an Lehr- und Lernangeboten in den letzten Jahren zu verzeichnen. Bereits renommierte Universitäten wie Stanford[1], MIT[2], Hassoplattner Institut[3] und HOOU[4] haben sich mit diesem Thema beschäftigt und eigene Angebote zum Erlernen von Programmiersprachen bereitgestellt. Zudem sei erwähnt, dass ebenfalls private Anbieter Lehrangebote bereitstellen, besonders hervorzuheben seien hier Codecademy[5] und code.org[6], die sich auf den Bereich der Webentwickler spezialisiert haben und CodeCombat[7] welches mit Hilfe eine spielerischen Ansatzes die Programmierung lehrt. Die Nachfrage nach Angeboten lässt sich auch auf die Anwendung von Software zurückführen, die für einen großen Teil der Menschen fest im Alltag

[1] https://see.stanford.edu/Course/CS106A
[2] https://ocw.mit.edu/index.htm
[3] https://open.hpi.de/courses
[4] https://www.hoou.de
[5] https://www.codecademy.com
[6] https://code.org
[7] https://codecombat.com

verankert ist, sei es das Verwenden eines Messengers wie Whatsapp[8] oder ein Shopping-Webseitenaufruf auf Amazon[9]. Weiterhin zeigen auch steigende Smartphoneverkäufe, dass der Umgang mit neuer Technik und dessen Software mehr und mehr in den Alltag Einzug erhalten hat [10, 11]. Gleichzeitig haben Studien ergeben, dass die Anzahl an Webseiten im Verlauf der Jahre stark zugenommen hat [13]. Die Abbildung 2.1 zeigt insbesondere, dass es in den Jahren 2012-2015 einen starken Zuwachs gegeben hat.

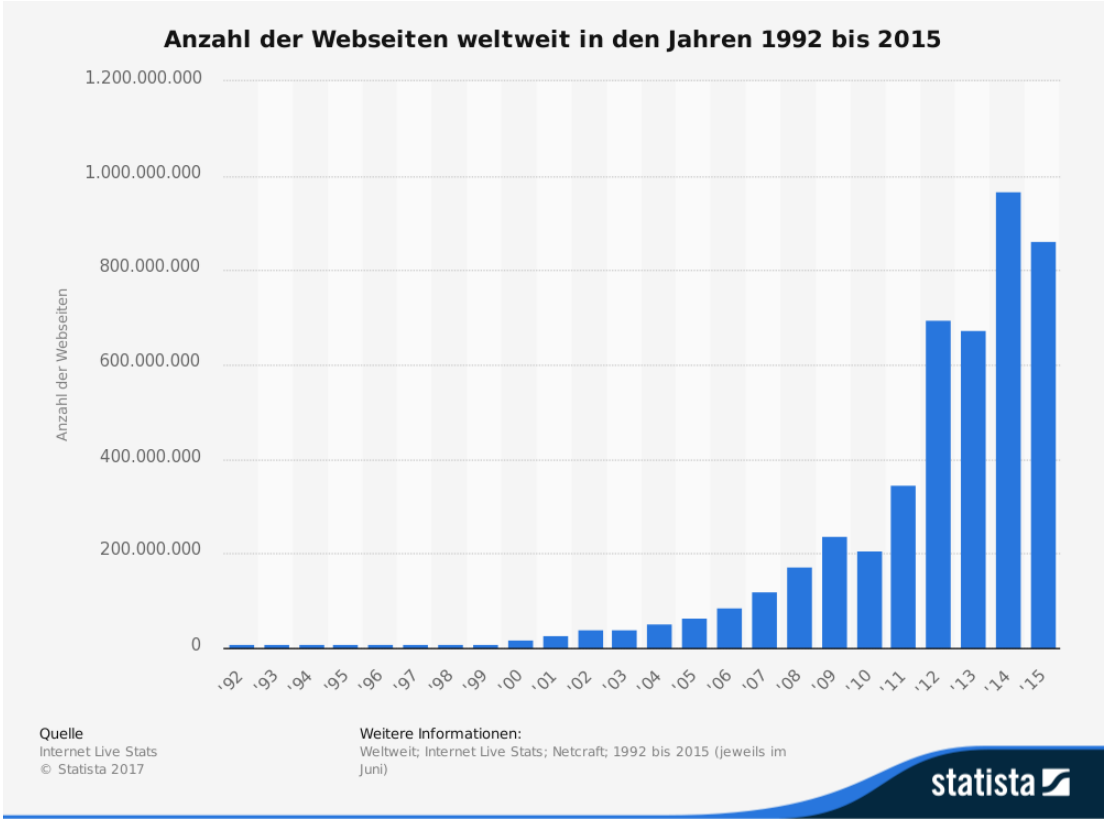


Abbildung 2.1: Anzahl der Webseiten weltweit in den Jahren 1992 bis 2015 Quelle: Internet Live Stats. (n.d.). Anzahl der Webseiten weltweit in den Jahren 1992 bis 2015. In Statista - Das Statistik-Portal. Zugriff am 6. November 2017, von https://de.statista.com/statistik/daten/ studie/290274/umfrage/anzahl-der-webseiten-weltweit/.

Daher resultiert ein steigender Bedarf nach der Fähigkeit zur Webseitenentwicklung auch für Nichtprogrammierer. Doch im Gegensatz zur Nutzung und Bedienung von Software, wie Internetauftritten, die i.d.R. viele beherrschen, steht die Erstellung dieser, da ein großer Auf-

[8] https://www.whatsapp.com
[9] https://www.amazon.de

wand zum Lernen der Grundlagen aufgebracht werden muss. Hier zählen die Einarbeitung in Semantik und Syntax von webseitenspezifischen Sprachen sowie Editoren. Des Weiteren auch das Erarbeiten von Designgrundlagen und Konventionen und zudem die Einhaltung von Standards und Sicherheitsaspekten. Den Aufwand, eine digitale Visitenkarte zu erstellen, nehmen viele daher nicht auf sich. Auch bekannte Menschen aus Politik und Wirtschaft regen an, Programmieren zu erlernen und erachten diese Fähigkeit als elementar, wie folgenden Zitate belegen.

> „*We believe that coding should be a required language in all schools.*“ - Tim Cook [6]

> „*I think everybody in this country should learn to program a computer. Learn a computer language. Because it teaches you how to think.*“ - Steve Jobs [8]

> „*Im Übrigen - wir sprechen heute so viel über Algorithmen - denke ich, dass es auch eine der grundlegenden Notwendigkeiten ist, dass im Schulunterricht das Programmieren zumindest ansatzweise gelernt wird, damit man weiß, wie Algorithmen zustande kommen.*“ - Angela Merkel, Rede von Bundeskanzlerin Merkel zur Eröffnungsveranstaltung der 30. Medientage am 25. Oktober 2016[1]

PfNP hat bereits eine lange Geschichte. Die erste erziehungsorientierte, funktionale Programmiersprache war Logo. Logo wurde 1967 von Seymour Papert und Wally Feurzeig entworfen, um Kindern durch Selbstentdecken die Welt der Programmierung spielerisch näher zu bringen [7]. Mit Hilfe von primitiven Befehlen, Funktionen und Prozeduren konnte in einer graphischen Ausgabe eine Schildkröte bewegt werden. Für diese sog. Turtle-Grafik ist Logo berühmt geworden. Mit Hilfe der Schildkröte, die einen Stift hält, konnte auf einem Blatt Papier gezeichnet werden. Befehle zum Zeichnen bestehen aus einer Richtungsangabe und einer Längeneinheit. Aber neben der Zeichenfunktion, welche einen großen Teil dieser Umgebung ausmachte, konnten auch andere Ausgaben, wie beispielsweise Listen oder Sätze, auf der Oberfläche erzeugt werden.

Neben dieser Sprache sind processing[10], Arduino[11] und vvvv[12] gute Beispiele für Programmierkonzepte für Nichtprogrammierer. Processing kam bereits 2001 auf den Ansatz des vereinfachten Lehrens und erweiterte Java um eine stark verkürzte Syntax. Damit schufen Ben Fry

[10] https://processing.org
[11] https://www.arduino.cc
[12] https://vvvv.org

und Casey Reas eine Programmierumgebung zum Erlernen der grundlegenden Fähigkeiten des Programmierens. Processing wurde für die Zielgruppe der Künstler und Designer erschaffen. Es verfügt heute über eine große sowie aktive Community und bietet daher sehr guten Support bei Fragen zur Unterstützung bei Problemen.

Auch im Bereich der technischen Informatik gibt es z.B. für die Mikrocontroller-Programmierung schon Lernangebote. Besonders hervorzuheben sei hier Arduino aus dem Do-it-yourself-Bereich. Dessen Entwicklungsumgebung basiert auf processing und soll technisch weniger versierten Laien den Zugang zur Programmierung von Mikrocontrollern ermöglichen. Die Programmierung erfolgt hierbei in einer C bzw. C++ ähnlichen Programmiersprache. Um den Anwendern den Zugang zum Programmieren zu erleichtern bleiben Header-Dateien und technische Details, sowie umfangreiche Bibliotheken weitestgehend vor diesen verborgen.

Auch innerhalb der Grafikprogrammierung gibt es Lernumgebungen, um die Kenntnisse der Echtzeit-Grafikanimationen zu erlernen und Prototyping in diesem Bereich durchzuführen. An dieser Stelle sei das blockbasierte Toolkit vvvv[12], 1998 von der Firma MESO erstellt, als Beispiel genannt.

> „*vvvv ist eine hybride visuelle / textuelle Live-Programmierumgebung für einfaches Prototyping und Entwicklung. Es wurde entwickelt, um die Handhabung großer Medienumgebungen mit physischen Schnittstellen, Echtzeit-Grafikanimationen, Audio und Video zu erleichtern, die mit vielen Benutzern gleichzeitig interagieren können.*“ - vvvv group, vvvv.org - Startseite, 25.10.2017

Im Gegensatz zu Ansätzen des Lehrens für Nichtprogrammierer spezieller Zielgruppen gibt es generell auch Ansätze im Bereich der Webseitenerstellung. Dort haben sich bereits diverse Anbieter formiert, um grundlegende Kenntnisse über webseitenspezifische Sprachen zu vermitteln. Beispiele für Anbieter sind hier z.B. w3schools[13] mit einem dem problembasierten Lernen ähnlichen Ansatz. Getreu der Herangehensweise: „Hier ist der Code, schaue Dir die Ausgabe an; spiele damit herum und erkenne was sich ändert“ werden auf dieser Plattform die Grundlagen für viele webbasierte Sprachen wie HTML, CSS, php, JavaScript usw. in Form von Tutorials angeboten. Jedoch wird dem Nutzer hier wenig Unterstützung für Fehler- und Syntaxverständnis geboten. Diese Erkenntnisse müssen ausschließlich eigenverantwortlich gewonnen werden.

Blockbasierte Programmierumgebungen, die einen Einstieg in die Welt der Software-Programmierung mit Übersetzung zu diversen Sprachen wie JavaScript, Python, php, Lua und Dart lehren, seien ebenfalls in den Bereich von PfNP einzuordnen. Hier zu nennen sei vor Allem

[13] https://www.w3schools.com

Blockly von Google. Dort werden Kenntnisse über Schleifen, Bedingungen, Variablen und Ausgaben vermittelt. Um einen Eindruck in den Leistungsumfang zu erhalten dient die folgende Grafik 2.2. In dieser Grafik befindet sich auf der linken Seite die Toolbox. In Kategorien sortiert liegen die Blöcke für den Workspace, der sich in der Mitte befindet. Auf dem Workspace sind aktuell Blöcke hinterlegt, die miteinander verbunden eine while-Schleife bilden, welche als Ausgabe drei Mal „Hello World!“ hat. Auf der rechten Seite ist nun die Übersetzung der Blöcke in die Sprachen JavaScript, Python, php, Lua und Dart erkennbar. Die Übersetzung kann mit Hilfe eines Drop-Down-Menüs gewechselt werden. Weiterhin befindet sich auf dem Workspace ein Papierkorb, der es ermöglicht, auf dem Workspace abgelegte Blöcke zu löschen. Zuletzt befindet sich unten rechts ein Play-Button, der als Funktion die Kompilierung und Ausführung des Codes ermöglicht.

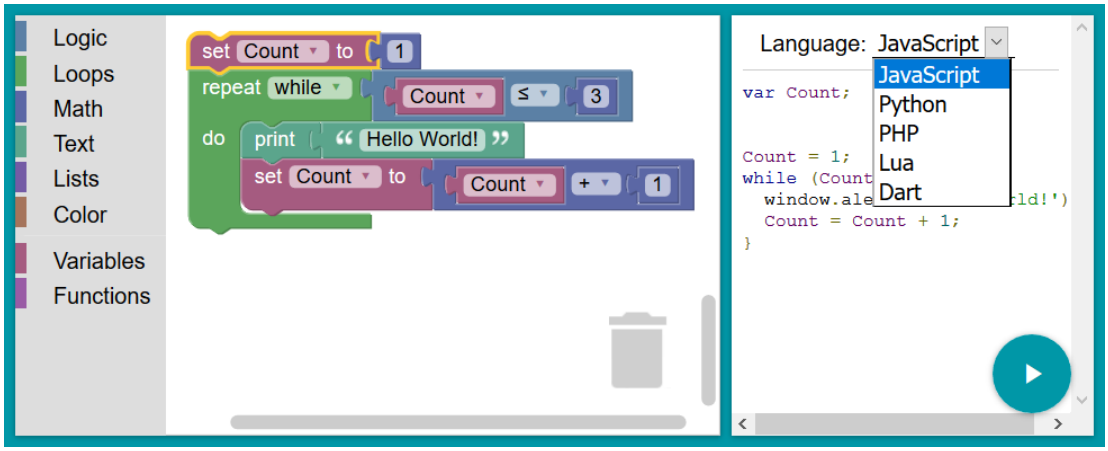


Abbildung 2.2: Google Blockly - Funktionsumfang Quelle: eigenes Werk.

Aus den verschiedenen Lehransätzen für Software-, Hardware- sowie Grafikprogrammierung ist zu erkennen, das PfNP in allen Bereichen Einzug hält. All diesen genannten Tools und Lernumgebungen ist gemeinsam, dass Laien als Zielgruppe ein einfacher Einstieg mit steiler Lernkurve, intuitiven Bedienelementen, einer geeigneten Lernumgebung und weiterführende Informationen über das darüberhinausgehende Lernen bereitgestellt wird. Diese Zielsetzung ist vergleichbar mit der Zielsetzung dieser Arbeit, daher sollen diese Eigenschaften auch mit Fokus auf die Entwicklung von Webseiten für den EZWEB-Editor gelten.

2.2 Problembasiertes Lernen

Um Nichtprogrammierern das Wissen verhältnismäßig komplexer Sachverhalte zur Webseitenentwicklung zu vermitteln, soll ein geeigneter Lernansatz gefunden werden.

Außerdem soll in diesem Kapitel problembasiertes Lernen (abgekürzt PBL) als Methodik erklärt werden. In diesem Zusammenhang wird die Frage geklärt, warum es sinnvoll ist, PBL in den EZWEB-Editor einfließen zu lassen. Der Begriff des problembasierten Lehrens und Lernens ist in den vergangenen Jahren zu dem Leitkonzept eines kognitiv aktivierenden und Selbständigkeit fördernden Unterrichts geworden [22]. Die Kernidee besteht darin, sich Kompetenzen durch eigenständige Problemlösung anzueignen [22]. Die klassischen Formen beruhen dabei auf dem Prinzip, Probleme mit Hilfe kleiner Gruppen und tutorieller Unterstützung zu lösen und zu besprechen [22]. Das Ziel ist es hierbei, transferfähiges Wissen sowie fachspezifische Lern- und Denkstrategien zu erwerben [22]. Ursprünglich kommt dieser Ansatz aus der Medizin und hatte den Antrieb, eine fachspezifische effektive Problemlösekompetenz zu vermitteln und die Motivation beim Lernen zu steigern. Näher beschrieben werden die Ziele von PBL in dem Buch „Didaktik und Evaluation in der Psychologie“ von Günter Krampen und Hermann Zayer:

> *„1. Das zu erwerbende Wissen soll strukturiert für den Gebrauch in einem Anwendungskontext vermittelt werden.*
> *2. Lernende sollen eine effektive, fachspezifische Problemlösekompetenz erwerben.*
> *3. Wissenerwerbsprozesse sollen auch die Aneignung von Kompetenzen im Bereich des selbstgesteuerten Lernens beinhalten.*
> *4. Die Motivation beim Lernen soll gesteigert werden.“* [18]

Zur Umsetzung dieser Ziele empfehlen die Autoren, verschiedene Elemente miteinander zu kombinieren. Zuallererst soll eine authentische Problemstellung als Ausgangsmaterial für die Wissenserwerbsprozesse vorausgesetzt werden. Weiterhin soll das Lernen innerhalb von Kleingruppen stattfinden, welche durch Tutoren, die den Lernprozess moderieren und unterstützen, geleitet werden. Zwischen den Kleingruppentreffen sollen Phasen des selbstgesteuerten Lernens erfolgen [18].

Da die kollaborative Umsetzung des EZWEB-Editors nicht Teil der Zielsetzung war, soll an dieser Stelle der Ansatz PBL, wie oben beschrieben, für den EZWEB-Editor abgewandelt werden. Hierbei wird das Lernen in Kleingruppen auf den Wissenserwerb im alleinigen Eigenstudium verlegt. Die tutorielle Unterstützung soll mit Hilfe von authentischen Problemstellungen in Form von Tutorials, näher beschrieben in 3.1.5, stattfinden.

Eine umfassende Auswertung diverser Studien zu den Effekten von PBL konnte einen positiven Lerneffekt gegenüber dem des traditionellen Lernens belegen [17]. Anhand dieser Erkenntnis kann der Einsatz von PBL in abgewandelter Form begründet werden.

Im E-Learning wird aufgrund dieser Effekte PBL abgewandelt in diversen online-Lernplattformen verwendet. Als Beispiel sei hier w3schools[13] zu nennen. Diese verwenden neben dem Wissenserwerb durch Tutorials ebenfalls ein Forum zur Kommunikation zwischen Lernenden.

Daher soll der EZWEB-Editor unter Berücksichtigung der oben genannten Abwandlungen die Lernmethode PBL verwenden.

2.3 Graphische Programmiersprachen

Visuelle bzw. graphische Programmiersprachen sind formale Sprachen mit visueller Syntax und Semantik, mit dessen Hilfe Programmierkonzepte wie Programmcode und Programmabläufe abgebildet werden können. In diesem Kapitel sollen das Verständnis für visuelle Programmiersprachen und deren Historie erläutert sowie die Arten und Ausprägungen von diesen hinsichtlich Ihrer Eigenschaften anhand des heutigen Standes für den zu konzipierenden EZWEB-Editor analysiert werden. Dabei soll evaluiert werden, welches dieser Konzepte am geeignetsten für den zu entwickelnden EZWEB-Editor ist.

Um zu verstehen, weshalb visuelle Programmiersprachen eine lange Historie haben und bereits seit Ende der 50er Jahre entwickelt werden, muss man zunächst verstehen, dass codebasierte Programmiersprachen i.d.R. für Laien schwer erlernbar sind. Für gewöhnlich kostet es viel Zeit und Arbeit, die Syntax und Semantik einer Sprache zu erlernen, sich mit der Programmierumgebung vertraut zu machen und Fehlererkennungsprozesse für einen selbst zu automatisieren. Motiviert durch diese Hürden entwickelte Haibt bereits Ende der 1950er Jahre ein System zur automatischen Generierung von Flussdiagrammen aus Fortran- und Assembler-Programmen [23]. 1963 folgte ein Zeichenprogramm namens Sketchpad, entwickelt von Ivan E. Sutherland, welches als erstes brauchbares System mit graphischer Interaktion bezeichnet wurde [23]. 1969 entwarf Ellis et al. das System Grail, welches Programme direkt aus maschinenlesbaren Flussdiagrammen generieren konnte [23]. Mitte der 1970er Jahre erschlossen sich durch bessere Hardware-Komponenten viele neue Anwendungsgebiete für den Computer [23]. Zu dieser Zeit wurde sich verstärkt damit auseinander gesetzt, wie der Einsatz von Grafik in der Programmierung Vorteile bringen könnte [23]. 1975 entwickelte Davic C. Smith eine visuelle Programmierumgebung mit dem Namen Pygmalion, die bis heute als Meilenstein der graphischen Programmierung gilt [23]. An dieser Stelle lässt sich auch

wieder vvvv[12] (vgl. 12) einreihen.

Heutzutage werden die bekanntesten Ausprägungen visueller Programmiersprachen in die zwei großen Kategorien „blockbasiert“ und „flussbasiert“ eingeteilt. In der flussbasierten Programmierung werden Anwendungen durch Netzwerke von „Black-Box“ Prozessen, die Daten über vordefinierte Verbindungen, sog. Kanten, durch Message Passing übertragen, definiert. Black-Box-Module können endlos wiederverbunden werden, um verschiedene Anwendungen abbilden zu können [5]. Diese Art der Programmierung ist komponentenorientiert und ermöglicht die Abbildung von Schleifen, logischen Verbindungen und Gattern. Aber auch klassischer Programmcode kann hiermit abgebildet werden. Zu modernen flussbasierten Sprachen gehören NoFlo[14], javaFBP[15] und Flowhub[16].

Blockbasierte Programmiersprachen ermöglichen die Abbildung von Programmcode wie Schleifen, Bedingungen sowie Eingabe- und Ausgabeparametern, etc. Mit Hilfe von Blöcken, die z.B. Programmcodes durch Verbindungselemente sequenziell und hierarchisch abbilden, kann Software erstellt werden. Hierbei werden in der Regel keine konkreten Syntaxkenntnisse vorausgesetzt. Diese blockbasierten Sprachen stehen für einen einfachen Zugang zur Programmierung, der von Programmieranfängern erfolgreich genutzt und zunehmend auch als Möglichkeit gesehen werden kann, nicht-professionellen Programmierern das Gestalten von Informatiksystemen zu ermöglichen [16]. Als Gründe hierfür werden u.a. die intuitive Bedienung, schnelle Erfolgserlebnisse und ein breites und kontextualisiertes Anwendungsspektrum genannt, welche zu einer großen Beliebtheit unter jungen Programmierern führt und sich als Kernmerkmale in den populären Programmierumgebungen Scratch[17], Snap! (BYOB)[18] und Google Blockly[1] wiederfindet [16].

An dieser Stelle soll in Hinblick auf die Abbildung der HTML-Struktur auf den EZWEB-Editor der signifikant hervorstechende Unterschied zwischen flussbasierten und blockbasierten Programmiersprachen genannt werden. Die bei dem flussbasierten Ansatz auftretende Modellierung erfolgt über den Datenfluss. Die Transformation der Daten findet durch die Kette von Datenflüssen statt. Da HTML kein Datenfluss ist oder als ein solcher repräsentiert werden kann, ist dieser Ansatz für den EZWEB-Editor ungeeignet. Die Abbildung einer Baumstruktur, wie sie auch in HTML vorhanden ist, ähnelt jedoch in vielerlei Hinsicht derer von blockba-

[14] https://noflojs.org
[15] https://github.com/jpaulm/javafbp
[16] https://flowhub.io
[17] https://scratch.mit.edu
[18] http://snap.berkeley.edu

sierten Sprachen. Daher sollen nun hier geeignete blockbasierte Sprachen hinsichtlich Ihrer Eigenschaften näher miteinander verglichen werden.

Zu den großen Anbietern von blockbasierten Sprachen zählen Scratch[17], Snap![18] und Google Blockly[1], die allesamt auch für den Bildungsbereich entwickelt worden sind. Der Leistungsumfang von Scratch, 2007 vom MIT[19] veröffentlicht, war zunächst sehr einfach gehalten. Mit Hilfe von Blöcken konnten grundlegende Konzepte wie Schleifen und Bedingungen von Programmiersprachen erforscht und erprobt werden. Mit diesen Elementen konnten Spiele, Animationen und Geschichten erschaffen werden. Ihr Ziel war es, Neueinsteiger, besonders Kinder und Jugendliche, mit den elementaren Konzepten der Programmierung vertraut zu machen. Die Herangehensweise beruht hierbei auf einem spielerischen Ansatz. Aus der Entwicklerdokumentation geht hervor, dass in Scratch bis zur Version 2.0 keine eigenen Blöcke definiert werden konnten. Diese Tatsache änderte sich ab der Version 2.0 durch die Kooperation mit BYOB. Jedoch können bis heute nur Blöcke in vereinfachter Form, ohne Ein- und Ausgabeparameter, erstellt werden. Das 2010 aus Scratch heraus entstandene BYOB (engl. Abk. von Build Your Own Blocks) sollte Scratch um komplexere und abstraktere Konzepte, welche dort zugunsten der Kindertauglichkeit bisher fehlten, erweitern [12]. Bis zum Auseinanderdriften dieser Sprachen basierte BYOB ebenfalls auf dem in Squeak[20] implementieren Quellcode von Scratch und behielt sich die Aufwärtskompatibilität bei [12]. Jedoch blieb die Beschränkung der online-Verfügbarkeit durch das Fehlen eines geeigneten webfähigen Abspielers [12]. BYOB wurde nach der Version 3.1.1 umbenannt und ist heute besser unter dem Namen Snap! (BYOB) bekannt [12]. Die neuste Version beruht auf JavaScript und ist daher auch für gängige Plattformen online verfügbar [12]. Die Entwicklerdokumentation hinsichtlich der Erstellung von Blöcken auf der Codeebene ist in Snap! (BYOB) nicht nennenswert vorhanden. Die Eigenerstellung von Blöcken beschränkt sich dabei auf den visuellen Rahmen, denn Blöcke können lediglich mit Hilfe von Pop-Up-Menüs erstellt und benannt werden.

Um die ausgeprägteste dieser Sprachen, Google Blockly, zu betrachten soll im Folgenden ein Einblick in den Funktionsumfang und die Entwicklung eigener Inhalte innerhalb dieser Sprache gegeben werden. An dieser Stelle seien zudem die verschiedenen Konnektoren der Blöcke von Google Blockly in der Grafik 2.3 aufzuzählen. Die Grafik zeigt dabei einen zusammengesteckten Block aus diversen Unterblöcken. Man spricht hier von der Komposition von Blöcken. Zu erkennen sind die Verbindungselemente vorheriger Block- / nächster-Block-Konnektor, Puzzle-Konnektor und Input-Konnektor. Neben diesen Verbindungselementen besteht auch die Möglichkeit, einem Block ein Drop-Down-Feld mitzugeben, diesem einen Namen zuzuweisen,

[19] http://web.mit.edu
[20] http://squeak.org

sowie ein Feld mit Informationen bereitzustellen. Ein Merkmal der Verbindungen ist, dass nicht alle belegt sein müssen. Die Sprache an sich wurde vollständig in JavaScript geschrieben und weist daher Plattformunabhängigkeit auf. Die Entwicklerdokumentation ist insbesondere für die Eigenerstellung von Blöcken sehr umfangreich. Auch fällt in dieser Dokumentation auf, dass die Sprache von vornherein dafür ausgelegt war, Blöcke um eigene JavaScript-Funktionen zu ergänzen. Diese können unter anderem für die Ausgabe von Programmiersprachen genutzt werden. Ein weiterer Aspekt dieser Sprache ist die Funktion der parallelen Betrachtung von Code und Blockansicht. Näher beschrieben bedeutet die Funktion der parallelen Betrachtung, dass es Blockly den Nutzern ermöglicht, eine Textübersetzung ihres Block-Codes einsehbar zu machen. Um aufzuzeigen, dass das Konzept von Blockly bereits an mehreren Stellen evaluiert worden ist, sei hier die erfolgreiche Implementation eines dezentralen Ansatzes für die Programmierung interaktiver Anwendungen mit JavaScript und Blockly als Beispiel genannt [19]. Assaf Marron, Gera Weiss und Guy Wiener beschrieben in dem Artikel zu dieser Implementation das Ergebnis als einen Proof-of-Concept für eine Programmierumgebung mit natürlicher und intuitiver Handhabung, welches interessante Merkmale der Verhaltensprogrammierung aufweist [19].

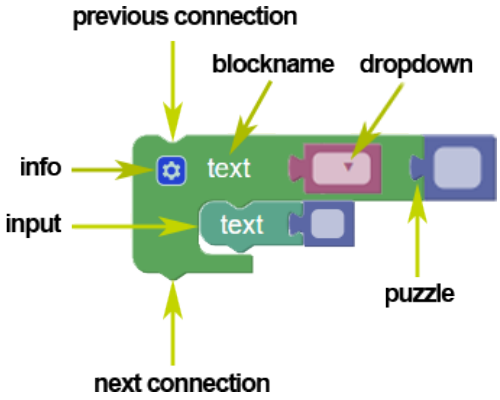


Abbildung 2.3: Google Blockly - Konnektoren Quelle: eigenes Werk.

Zusammenfassend lässt sich sagen, dass HTML nicht auf Scratch abbildbar ist, da keine komplexen, parametrisierbaren Blöcke unterstützt werden. Des Weiteren bieten weder Scratch noch Snap! (BYOB) die volle Unterstützung von Cross-Browser-Kompatibilität (CBK) oder Plattformunabhängigkeit, noch einen geeigneten Entwicklungsrahmen für die Abbildung von Code zu den Blöcken. Weiterhin hat die Untersuchung ergeben, dass, im Gegensatz zu Google Blockly, Snap! (BYOB) aufgrund der nicht ausreichenden Entwicklerdokumentation nur mit viel Aufwand auf HTML abzubilden wäre.

2.4 Entwicklungswerkzeuge für Webseiten

Im Gegensatz zu den visuell unterstützenden Ansätzen in Kapitel 2.3, gibt es im Bereich der Webseitenerstellung zwar Editoren, die auf den visuellen Ansatz abzielen und Webseiten in einem vorgegebenen Rahmen produzieren lassen, jedoch hierbei kein Wissen über HTML und andere webseitenspezifische Sprachen vermitteln. Für das Erlernen dieser Sprachen müssen Anfänger auf unkomfortable Code-Editoren in Kombination mit Tutorials zurückgreifen. Inwieweit Code-Editoren Unterstützung für Anfänger bieten und visuelle Konzepte kein Wissen über den Hintergrund lehren, soll im folgenden Kapitel geklärt werden.

Zuvor sollen nun generelle Konzepte von Editoren mit ihren Unterschieden und grundsätzlichen Eigenschaften aufgezeigt und analysiert werden. Zunächst seien hier die verschiedenen Arten der Editoren zu nennen: Code-Editor, Homepage-Baukasten / Website-Builder und In-Site-Editing. All diesen Editoren gemeinsam ist die Zielsetzung, eine vollwertige Webseite erstellen zu können oder Änderungen an diesen vorzunehmen. Dabei unterscheidet sich die jeweilige Herangehensweise jedoch stark.

Bei Code-Editoren wie Notepad++[21] und Sublime Text[22] muss der Programmcode selbst geschrieben werden. Dabei müssen Kenntnisse über Syntax und Semantik bis ins Detail beherrscht werden. Mittlerweile können Code-Editoren diverse Hilfestellungen zum Programmieren bereitstellen, welche den Umgang mit diesen deutlich vereinfachen. Zu deren Mehrwerten zählen Tipps und Hinweise auf Programmierfehler, integrierte Dateibrowser, Coderefactoring, Autokorrektur, integrierte Anbindung an Versionsverwaltungssysteme wie git[23], sowie semiautomatische Zeichen- und Fehlerkorrektur. I.d.R. stellen nur die kostenpflichtigen Editoren wie beispielsweise Produkte von JetBrains[24] oder Adobe Dreamweaver[25] solche Features zur Verfügung, die aufgrund des hohen Preises nicht für den Einstieg, sondern hauptsächlich für den professionellen Einsatz geeignet sind. Bekannte und viel eingesetzte Editoren wie Notepad++[21] oder Sublime Text[22] stellen diese Funktionen nicht, bedingt oder lediglich über externe Plugins bereit.

Daneben gibt es den Ansatz der WYSIWYG-Editoren. Zu diesen zählen online verfügbare Homepage-Baukästen wie z.B. 1&1 MyWebsite[26] und Jimdo[27]. Bei diesen lassen sich mit Hilfe von vordefinierten Templates, die es mit Inhalt zu befüllen gilt, Webseiten erstellen. Oft liegen

[21] https://notepad-plus-plus.org
[22] https://www.sublimetext.com
[23] https://git-scm.com
[24] https://www.jetbrains.com
[25] http://www.adobe.com/de/products/dreamweaver.html
[26] https://hosting.1und1.de/homepage-erstellen
[27] https://de.jimdo.com

hierbei jedoch Einschränkungen hinsichtlich der Abwandlung des grundlegenden Aufbaus und der optischen Gestaltung vor. Weiterhin ist die Anpassung der dazugehörigen mobilen Ansicht zumeist wenig individuell oder bereits festgelegt, wenn nicht sogar weggelassen worden. Ebenfalls zählen offline verfügbare Website-Builder zu den WYSIWYG-Editoren, welche nach dem gleichen Schema wie Homepage-Baukästen funktionieren. Ebenfalls gibt es in der Kategorie des WYSIWYG das In-Site-Editing. Der Begriff des In-Site-Editing meint, dass Anwender innerhalb von Foren Kommentare oder Einträge mit Textbetonungen, Styleoptionen oder Dateianhängen verfassen oder bearbeiten können. Bekannt hierfür ist z.B. Wikipedia[28]. Um weitere Hinweise für notwendige Eigenschaften über den zu entwickelnden EZWEB-Editor zu erhalten, stellt die Tabelle 2.1 die Eigenschaften der Editortypen vergleichend nebeneinander.

Editortyp		**Vorteile**	**Nachteile**
Code-Editor		+ maximale Freiheit + individuelle Gestaltung	- wenig einsteigerfreundlich - Autovervollständigung meist nur bei kostenpflichtigen Editoren
WYSIWYG	Baukasten / Builder	+ i.d.R. keine Codekenntnisse notwendig + online Editoren benötigen keiner Installation	- veraltungsgefährdet - ggf. langsam - u.a. Sicherheitskritisch
	In-Site-Editing	+ i.d.R. keine Codekenntnisse notwendig	- keine Neuerschaffung - eingeschränkter Umfang

Tabelle 2.1: Editortypen im Detail

Neben den in Tabelle 2.1 genannten Eigenschaften, kann es bei Website-Buildern älterer Versionen vorkommen, dass ein sehr langer Code produziert wird. Bilder können dabei in viele Unterbilder zerlegt werden, somit deutlich mehr Speicherplatz belegen und daher auch für längere Ladezeiten sorgen. Weiterhin können oft nicht nachvollziehbare Dinge passieren, wie zum Beispiel die abweichende Verschiebung von Elementen auf der Oberfläche. Bei vielen WYSIWYG-Editoren kann es als Nachteil angesehen werden, dass der Code dahinter i.d.R. nicht bearbeitet werden kann und kein Wissen hierüber vermittelt wird. Daher können ggf. nicht alle Bedürfnisse über den Rand der WYSIWYG-Editoren hinaus erfüllt werden. Der WYSIWYG-Ansatz ist somit nicht als optimale Lösung für die Problemstellung zu sehen und bestärkt darin, einen neuen Ansatz zu konzipieren und zu prüfen. Zudem kann bei diesen Angeboten oft keine Suchmaschinenoptimierung durchgeführt werden.

[28] https://www.wikipedia.de

2.5 Fazit

Die im Abschnitt 2.1 beschriebene Historie an Ansätzen für Nichtprogrammierer zeigt auf, dass PfNP bereits sehr lange ein Thema ist. Dabei wird deutlich, dass eine Vereinfachung des Lernen von komplexen Programmiersprachen und -sachverhalten notwendig ist. Aus dieser Erkenntnis heraus, soll der EZWEB-Editor Nichtprogrammierern grundlegende Kenntnisse im Bereich der Webseitenentwicklung lehren und als neues Tool in dieser Fachrichtung eingereiht werden.

PBL beschrieben in Abschnitt 2.2 zeigt auf, dass diese Methode ein vielversprechender Ansatz zum Lernen ist. Eine Auswertung diverser Studien zu diesem Thema zeigt, dass PBL einen positiven Lerneffekt gegenüber traditionellen Lernmethoden aufweist. PBL soll daher wie bereits in diesem Kapitel beschrieben in abgewandelter Form, mit Hauptaugenmerk auf den individuellen Wissenserwerb in den EZWEB-Editor einfließen.

Der hohe Verbreitungsgrad und die lange Historie von graphischen / visuellen Programmiersprachen (vgl. 2.3) macht diese zu Kandidaten, mit denen sich die Idee des einfachen Erlernens von Webseitenentwicklung erfolgreich umsetzen lässt. 2.3 hat gezeigt, dass weder Scratch, noch Snap! (BYOB) die wesentlichen Grundvoraussetzungen für die Abbildbarkeit auf HTML bieten, aber Google Blockly über ausreichende Funktionalität verfügt, um Blöcke flexibel auf HTML-Strukturen abzubilden. Der Vergleich aktueller Webseiten-Editoren in 2.4 hat gezeigt, dass es zum aktuellen Zeitpunkt keine Werkzeuge gibt, die maximale Freiheit bei der Erstellung von Webseiten bieten, visuell unterstützen und gleichzeitig Kenntnisse über das Programmieren von Webseiten an Nichtprogrammierer vermitteln. Motiviert durch diese Erkenntnisse, soll der EZWEB-Editor PfNP mit dem abgewandelten Ansatz von PBL unterstützen und mit Hilfe des Frameworks Google Blockly umgesetzt werden. Mit Hilfe dieses EZWEB-Editor sollen Nutzer Singlepage-Internetauftritt als Steckbrief über ein Hobby oder bestimmtes Thema erstellen lernen. Im nachfolgenden Kapitel 3 soll diese Zielsetzung in einer systematischen Anforderungsanalyse konkretisiert werden.

3 Anforderungsanalyse

In der Anforderungsanalyse sollen die funktionalen in 3.1 und nicht-funktionalen Anforderungen vgl. 3.2 für den Entwurf des EZWEB-Editor zur Erstellung von hierarchisch strukturierten Webseiten festgehalten werden. Die Zielgruppe für die Benutzung dieses EZWEB-Editors sollen Nichtprogrammierer und Anfänger mit geringen Vorkenntnissen in der Erstellung von Webseiten sein. Daher sollen die folgenden Anforderungen, auch unter Berücksichtigung dieser Nutzergruppen, definiert werden.

3.1 Funktionale Anforderungen

Funktionale Anforderungen sind spezifisch für das beschriebene Produkt. Sie beschreiben dessen Umsetzung direkt und zeigen die konkrete Zweckbestimmung dafür auf.

Die funktionalen Anforderungen des EZWEB-Editors können in sechs große Bereiche unterteilt werden. Zuerst wird der graphische Editor aus der Toolbox 3.1.1, welche die HTML-Tags in Blöcken mit Ihren Attributen bereitstellt, und dem Workspace 3.1.1, der den Ablagebereich für die Blöcke mit der Möglichkeit der Komposition dieser Blöcke darbietet, zusammengesetzt, beschrieben. Danach wird auf die Persistenz 3.1.2 der Elemente auf dem Workspace eingegangen. Im Folgenden werden Import und Export in 3.1.3 von Webseiten aus den graphisch erstellten Elementen spezifiziert. Als weiterer Punkt sind eine Code-Preview des aus den Blöcken erstellten HTML-Codes und eine HTML-Preview in 3.1.4 verzeichnet. Zuletzt wird die Spezifikation der Tutorials 3.1.5, welche nach der Abwandlung von PBL 2.2, die dort definierte tutorielle Betreuung ersetzten sollen, vorgenommen.

3.1.1 Graphischer Editor

Um die Anforderungen besser zu verstehen soll hier kurz beschrieben werden, wie mit dem EZWEB-Editor gearbeitet werden kann. Im graphischen Editor können HTML Tags aus der Toolbox via Drag & Drop auf dem Workspace abgelegt werden. Innerhalb der Toolbox sind HTML-Tags mit deren Attributen in Blöcken verpackt und farblich nach Zusammengehörigkeit zu Head, Body oder Sonstiges unterteilt, abgelegt. Je ein Block entspricht hierbei einem

HTML-Tag mit seinen Attributen. Die im Workspace abgelegten Blöcke können nun anhand Ihrer Verbindungen zusammengesteckt werden. Hier gibt es zwei unterschiedliche Arten von Konnektoren, an denen die Blöcke miteinander verbunden werden können. Diese ermöglichen entweder das hierarchische oder sequenzielle Verbinden dieser Blöcke.

Weitere Funktionen dieses Editors sollen in einer darüberliegenden Navigationsleiste bereitgestellt werden. Zu dessen Funktionen gehören das Laden, Speichern, Importieren und Exportieren von Webseiten. Weiterhin werden grundlegende Funktionen wie das Rückgängigmachen und Wiederholen von Aktionen, sowie das Umschalten zwischen den verschiedenen Sichten zu denen die Bearbeitungsansicht, die Code-Preview und die HTML-Preview gehören, beschrieben.

3.1.1.1 Allgemeines

An dieser Stelle sollen zunächst die allgemeingültigen Randbedingungen an den Editor gestellt werden.

3.1.1.1.1 Editor Aufbau

Der Aufbau des Editors soll dem eines gewöhnlichen Code-Editors wie Notepad++[21] oder Sublime Text[22] entsprechen, um für Code-Editoren keinen großen Umgewöhnungseffekt zu erzeugen. Hierfür sollen die folgenden Editor-Funktionen innerhalb des Navigationsmenüs in den Überpunkten „File“, „Edit“ und „Views“ untergebracht werden.

3.1.1.1.2 Displayauflösung

Der Editor soll für einen Arbeitsplatzrechner mit einer Auflösung von mindestens 720p (HD) ausgelegt sein, empfohlen werden jedoch 1080p (Full-HD).

3.1.1.1.3 HTML-Abbildung

Die Blöcke aus Blockly, welche hier die Tags mit Ihren Attributen repräsentieren, sollen die Hierarchie von HTML abbilden können. Dazu werden umschließende Tags mit einen Input-Konnektor versehen, sofern ein Inhalt neben dem Text erforderlich sein kann. Für die sequenzielle Abbildung werden Blöcke mit Next- sowie Previous-Konnektoren versehen. Eine Auflistung der Konnektoren aus Blockly kann in der Abbildung 2.3 nachgelesen werden.

Bei der Abbildung ist wichtig, dass ein Block genau einen Tag mit seinen Attributen repräsentiert.

3.1.1.1.4 HTML-Code Generierung / Webseiten Erstellung

Der Editor soll in der Lage sein, aus visuellen Elementen, die hierarchisch und sequenziell den Aufbau einer Webseite skizzieren, den dafür erforderlichen HTML-Code zu generieren.

3.1.1.1.5 Fehlerbehandlung

Der Editor soll typische Probleme wie Flüchtigkeitsfehler, wie zum Beispiel das Vergessen der schließenden Tags sowie der Anführungszeichen bei den Attributen dieser, gegenüber den Code-Editoren, welche im Allgemeinen maximale Freiheit beim Erstellen von Webseiten bilden, verhindern. Auch entsteht oft unübersichtlicher Quellcode, begründet durch fehlende oder falsche Einrückung, welche hier kompensiert werden soll. Die Attribute eines Tags werden durch Textfelder repräsentiert und stellen bei der Code-Generierung den korrekten Code mit vollständiger Syntax bereit. Tags und Attribute sollen korrekt geschlossen werden. Die Hierarchie der Blöcke als Baum soll bei der Ausgabe des HTML-Codes auf die HTML-Baumstruktur korrekt abgebildet werden. In diesem Zusammenhang soll auch die automatische Einrückung der Elemente im Code geschehen.

3.1.1.2 Toolbox

Auf der linken Seite im graphischen Editor soll eine Toolbox bereitgestellt werden. Die Toolbox enthält Blöcke für die verschiedenen HTML-Artefakte. Es gibt leere und gefüllte Blöcke sowie zusammengesteckte vorgefertigte Blöcke, welche beispielsweise ein Kontaktformular repräsentieren können. Um diese Toolbox genauer zu spezifizieren sollen hier die einzelnen Anforderungen aufgeführt werden:

a) Ein Block soll genau ein HTML-Tag mit seinen Attributen repräsentieren

b) Blöcke sollen farblich gekennzeichnet werden, sodass die Zusammengehörigkeit zu den Kategorien und imaginären Restriktionen klar erkennbar ist

c) Blöcke sollen in sinnvollen Kategorien wie Head, Body und Sonstiges unterteilt werden. Weiterhin soll innerhalb der Body-Kategorie noch weiter unterteilt werden

d) In der Umsetzung sollen keine Restriktionen innerhalb der Blöcke hinsichtlich der Umsetzung von HTML vorliegen. Jedoch müssen Constraints, die von der visuellen Programmiersprache gebildet werden, hinsichtlich der Verbindungen eingehalten werden (vgl. Abbildung 2.3). Hierzu zählen sequenzielle, sowie hierarchische Constraints. Jedoch unterliegen diese Verbindungen keiner Belegungspflicht

e) Blöcke besitzen Konnektoren, welche zur hierarchischen und sequenziellen Abbildung von HTML-Code benötigt werden. Dazu werden umschließende Tags mit einen Input-Konnektor versehen, sofern ein Inhalt neben dem Text erforderlich sein kann. Für die sequenzielle Abbildungen werden Blöcke mit Next- sowie Previous-Konnektoren versehen

f) Farben sollen Hinweise auf Restriktion und Kombinierbarkeit geben. Vier grundlegende Klassifikationen können hier vorgenommen und mit je einer Farbe versehen werden. Umschließendes HTML zusammen mit dem DOCTYPE, weiterhin Head, Body und gesonderte Tags. Diese Farben sollen sich auch in der Toolbox widerspiegeln

g) Blöcke sollen per Drag & Drop aus der Toolbox herausgenommen und auf dem Workspace platziert werden können

h) Es soll einen Block für protected Code geben

i) Der Editor soll innerhalb der Toolbox auch vorgefertigte, bereits zusammengesteckte Blöcke bieten. Sinnvolle Beispiele sind hier die Grundstruktur, der Head mit Title und META-Daten. Tabellen, Listen und ein Kontaktformular

j) In der Toolbox in der Kategorie Tabellen → pre-built soll ein Button vorhanden sein, der ein Popup öffnet. Dieses Popup zeigt zwei Eingabefelder, eines für die Anzahl der Spalten und eines für die Anzahl der Zeilen, sowie einem Button, der nach der Eingabe eine Tabelle nach diesen Angaben auf dem Workspace erzeugt. Falsche Eingaben sollen geprüft und ggf. mit aussagekräftigen Fehlermeldungen gekennzeichnet werden

3.1.1.3 Workspace

Im Zentrum des EZWEB-Editors soll sich der Workspace befinden. Auf diesem können Blöcke niedergelassen, verschoben, gelöscht und verbunden werden. Weiterhin sollen sich auf diesem diverse Funktionen wie Zoom und ein Papierkorb befinden. Auf dem Workspace sollen für die Blöcke Rechtsklickfunktionen bereitgestellt werden.

3.1.1.3.1 Papierkorb

Auf dem Workspace soll sich unten rechts ein Papierkorb befinden. In diesen können Blöcke zum Löschen abgelegt werden.

3.1.1.3.2 Zoom

Innerhalb des Workspace soll mit Hilfe des Mausrades, sowie eigenständigen Buttons ein- und ausgezoomt werden können.

3.1.1.3.3 Kontextmenü der Blöcke

Die Blöcke sollen durch das Klicken der rechten Maustaste ein Kontextmenü mit folgenden Funktionen bereitstellen:

a) Ein- und Ausklappen des Blocks, sowie seiner hierarchisch darunterliegenden Blöcke

b) Duplizieren des Blocks, sowie seiner hierarchisch darunterliegenden Blöcke

c) Löschen des Blocks, sowie seiner hierarchisch darunterliegenden Blöcke

d) Hinzufügen eines Kommentars an diesen Block

3.1.1.3.4 Neue Datei erstellen

Es soll möglich sein, eine neue, leere Datei auf dem Workspace zu erstellen.

3.1.1.3.5 Rückgängig

Innerhalb der Umgebung soll es mit Hilfe eines Buttons sowie der Tastenkombination STRG + Z möglich sein, Schritte rückgängig zu machen.

3.1.1.3.6 Wiederholen

Innerhalb der Editorumgebung soll es mit Hilfe eines Buttons sowie der Tastenkombination STRG + SHIFT + Z möglich sein, den letzten rückgängig gemachten Schritt zu wiederholen.

3.1.1.3.7 Alles Löschen

Diese Funktion soll die Möglichkeit geben, alle Blöcke innerhalb des Workspaces zu löschen.

3.1.2 Persistenz

Die interne Abbildung der Webseiten soll innerhalb des EZWEB-Editors „xml“ sein. Um allerdings nicht für Einsteiger zu suggerieren, dass aus jeder „.xml“ Datei eine Webseite entstehen kann, soll im Folgenden „.xmlbly“ verwendet werden. Um zu garantieren, dass

dieses neu geschaffene Dateiformat noch nicht verwendet wird, wurde es mit Hilfe von dafür vorgesehenen Seiten geprüft. Vgl. [3, 4, 2]

3.1.2.1 Datei Laden

Es soll die Möglichkeit bestehen, eine Datei im Blockly internen Dateiformat „.xmlbly“ in den Workspace zu laden.

3.1.2.2 Datei Speichern

Es soll die Möglichkeit bestehen, eine Datei im Blockly internen Dateiformat „.xmlbly“ zu speichern.

3.1.3 Import / Export von Webseiten

Der EZWEB-Editor soll final Reverse-Engineering unterstützten. Dazu soll zunächst ein Workaround entstehen. Mit Hilfe einer Import- / Exportfunktion sollen Webseiten von Extern importiert und aus dem Workspace heraus exportiert werden können. Diese Funktionen bilden die Grundlage um später Änderungen aus der Code-Preview editieren zu können und dessen Änderungen in Echtzeit in der Bearbeitungsansicht zu übertragen sowie Fremdcode einzulesen zu können.

3.1.3.1 Datei Importieren

„.html“ Dateien sollen importiert werden können. Um bei Fremdcode eine Fehlerbehandlung vorzunehmen, sollen beim Einlesen Meldungen bei fehlerhaftem Code erscheinen. Nicht implementierter Code soll in sog. „Protected Areas“ in diesem Fall speziellen Hinweisblöcken verpackt und an der äquivalenten Stelle in der Baumstruktur als Hinweisblock abgelegt werden.

3.1.3.2. Datei Exportieren

Eine in „.xmlbly“ erstellte Webseite soll nach „.html“ exportiert werden können. Hierbei soll richtige Code-Einrückung gegeben sein. Dafür soll jeder Block beim Erstellen des Codes seine Tiefe kennen und angeben.

3.1.4 Code- / HTML-Preview

Neben dem graphischen Editor, der in der Bearbeitungsansicht gezeigt wird, sollen zwei weitere Ansichten entstehen. Diese Ansichten zeigen den Code der Blöcke in anderer Darstellung

und werden näher als Code-Preview und HTML-Preview beschrieben. In der Bearbeitungsansicht, innerhalb der Tutorials, wird neben dem graphischen Editor, parallel dazu, die aktuelle Aufgabenstellung angezeigt.

3.1.4.1 Code-Preview

Die Code-Preview zeigt den aus den zusammengesteckten Blöcken übersetzten Code als HTML-Code. Der Code soll an dieser Stelle eine hervorgehobene Syntax aufweisen und über korrekte Einrückung verfügen. Zunächst soll diese Ansicht readonly sein, aber später im Zusammenhang mit Reverse-Engineering 3.1.3 auch eine Bearbeitung des Codes bereitstellen.

3.1.4.2 HTML-Preview

Die HTML-Preview soll als Übersetzung den HTML-Code als Internetauftritt im EZWEB-Editor darstellen. Hierzu kann innerhalb eines iFrames das Attribut „srcdoc“ verwendet werden.

3.1.5 Tutorials

Damit der EZWEB-Editor die Lehrmethode PBL implementieren kann, müssen Tutorials gegeben sein. Dafür soll neben einem Tutorial für die Grundlagen des EZWEB-Editors ansich, welche eine Hilfeseite ersetzten soll auch noch sieben Anfängertutorials für die HTML-Basics erstellt werden.

3.1.5.1 Tutorial 0 - Editor Grundlagen

In Tutorial 0 - Editor Grundlagen sollen, soweit möglich, alle Kernfunktionen des EZWEB-Editor erklärt werden. Dieses Tutorial soll für das Erste die Hilfeseite stellen. Später soll es eine eigenständige Hilfeseite geben.

3.1.5.2 Tutorial 1 - HTML Grundlagen

In dem Tutorial 1 - HTML Text soll dem Nutzer das Wissen über den grundlegenden Aufbau von Webseiten vermittelt werden. Dazu gehören neben der Grundstruktur, bestehend aus Initiation mit DOCTYPE, Head und Body auch der Titel und ein Text im Body.

3.1.5.3 Tutorial 2 - HTML Textbetonung

Tutorial 2 - HTML Textbetonung soll einfache Textbetonungsmöglichkeiten lehren, dazu gehören z.B. fettgedruckt, kursiv, unterstrichen und diverse andere die laut dem in 3.2 definierten

HTML5 Standard unterstützt werden. Hierzu soll eine nicht vollständige, aber hinreichende Auswahl getroffen werden.

3.1.5.4 Tutorial 3 - HTML Textstyle

In Tutorial 3 - HTML Textstyle soll der Nutzer das Wissen über Farben, Paragraphen, Zeilenumbrüche sowie horizontale Linien erhalten. Auch sollen Tags wie und näher gebracht werden.

3.1.5.5 Tutorial 4 - HTML Listen

In dem Tutorial 4 - HTML Listen sei zu lehren, wie Listen in Webseiten funktionieren. Dazu gehört der Aufbau, das Hinzufügen von Elementen, geschachtelte Listen und sortierte sowie unsortierte Listen.

3.1.5.6 Tutorial 5 - HTML Medien

Tutorial 5 - HTML Medien soll den Umgang mit Bildern lehren. Das Ziel hier ist es, ein Bild-Einbindungs-Tag mit seinen Attributen kennenzulernen und anwenden zu können.

3.1.5.7 Tutorial 6 - HTML Tabellen

In Tutorial 6 - HTML Tabellen soll der grundlegende Aufbau von Tabellen erklärt werden. Als Beispiel soll hier ein Datenbankeintrag mit Nummer, Vorname, Nachname und Geburtsdatum herangezogen werden.

3.1.5.8 Tutorial 7 - HTML Kontaktformular

Das Tutorial 7 - HTML Kontaktformular soll den grundlegenden Aufbau von Formularen in HTML darstellen und exemplarisch aufzeigen, wie ein funktionsloses Kontaktformular gestaltet werden kann. Dieses soll Vorname, Nachname, E-Mail-Adresse, Telefonnummer, Betreff und einen Freitext enthalten. Weiterhin soll ein Button zum Absenden eingebunden werden.

3.2 Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen sind in der Literatur nicht einheitlich definiert. Die Gemeinsamkeiten dieser Definitionen sind allerdings, dass die nicht-funktionalen Anforderungen über die der funktionalen Anforderungen hinausgehen. Im Allgemeinen können diese als

Qualitätseigenschaften und Randbedingungen verstanden werden. Sprich, wie gut das System die Leistung erbringen soll.
Für den EZWEB-Editor sollen folgende nicht-funktionale Anforderungen nach „ISO 25010“ gelten:

Übertragbarkeit: Der EZWEB-Editor soll online verfügbar sein und daher Plattformunabhängigkeit gewährleisten. Weiterhin soll er CBK aufweisen und Skalierbarkeit für gängige Displaygrößen ab 720p bieten.

Zuverlässigkeit: Die Importfunktion soll mit fehlerhaftem Fremdcode umgehen können. Dieser soll nicht zum Abbruch des Importierens führen.

Effizienz: Da der EZWEB-Editor online im Browser läuft, darf kein hoher Ressourcenbedarf vorausgesetzt werden und Antwortzeiten sollen den Nutzer befriedigen. Eine akzeptable Zeit kann hier als kleiner 350ms angesehen werden.

Benutzbarkeit: Wie schon beschrieben, soll das Aussehen und die Handhabung einsteigerfreundlich intuitiv sein und zugleich auch den späteren Umstieg auf einen Code-Editor erleichtern. Dazu soll der Aufbau des EZWEB-Editors grundlegend dem eines Code-Editors wie Notepad++ oder sublime ähneln.

Wartbarkeit: Kategorien in der Toolbox, sowie Tags und Attribute in Blöcken und Tutorials sollen erweiterbar sowie bearbeitbar sein. Weiterhin sollen Computersprachen im EZWEB-Editor bearbeitet werden können, um ggf. auf neue Standards reagieren zu können. Ebenfalls soll für den EZWEB-Editor Internationalisierung möglich sein. Dazu gehört auch die Unterbringung eines Buttons auf der Editoroberfläche zum Ändern der aktuellen Spracheinstellungen. Sinnvollerweise kann hierfür eine Übersetzungsdatei, die alle Texte enthält, verwendet werden.

Flexibilität: Unterstützung von Standards: Es soll die HTML5-Spezifikation des W3C[1] vom 28. Oktober 2014 unterstützt werden. Weiterhin sollen CSS, JavaScript und php nach folgenden Spezifikationen eingebunden werden: CSS soll nach der CSS Level 3-Spezifikation W3C[1] von 2014 unterstützt werden. Unter Berücksichtigung der Einbindung von CSS, sollen ebenfalls die CSS-Bibliotheken Bootstrap[2] bis Version 3.3.7 und Materialize[3] bis

[1] https://www.w3.org
[2] https://getbootstrap.com/docs/3.3
[3] http://materializecss.com

Version 0.100.2 im EZWEB-Editor verankert und unter den genannten Standards unterstützt werden. JavaScript soll nach Version 1.8.1 + ECMAScript 5 Compliance vom Juli 2010, sowie den Ergänzungen bis 2016 unterstützt werden. Ebenfalls soll php der Version 7.1 von Dezember 2016 unterstützt werden.

Die in Kapitel 3 vorgestellten Anforderungen sollen, unter Berücksichtigung der NFAs und mit Bezug auf die prinzipiellen erarbeiteten Konzepte aus Kapitel 2 für den Entwurf in 4 ausschlaggebend sein.

4 Entwurf

In diesem Kapitel werden die Anforderungen aus 3 und die Erkenntnisse aus 2 in einen Softwareentwurf überführt. Kapitel 4.1 führt anhand von Komponentendiagrammen die wesentlichen Module der Lösungen ein und veranschaulicht gleichfalls die Schnittstellen zu anderen Komponenten. Die Abbildung des Blocklymodelles wird in Abschnitt 4.2 näher betrachtet. In 4.3 soll der EZWEB-Editor prototypisch implementiert werden und somit für die spätere Untersuchung bereitgestellt werden. Weiterhin sollen innerhalb der prototypischen Implementierung der aktuelle Stand des EZWEB-Editors, die Tutorials, die Erweiterbarkeit und Probleme die bei der Umsetzung entstanden sind, näher erläutert werden.

4.1 Komponentendiagramm

Die zwei nachfolgenden Diagramme zeigen in Abbildung 4.1 zunächst die übergeordneten großen Komponenten. Darauffolgend soll in der Abbildung 4.2 der detaillierte Aufbau der jeweiligen Komponenten gezeigt werden. Diese Diagramme sollen zusammen mit der folgenden Beschreibung die Interaktionen der Komponenten untereinander aufzeigen und zudem beschreiben, an welchen Stellen auf das Framework von Google Blockly zurückgegriffen wurde. Um die gleichen Komponenten im Detail- sowie Übersichtsdiagramm zu kennzeichnen, wurden identische Farben für die gleichen Komponenten verwendet. Innerhalb der Grafik wurden die Komponenten, welche im wesentlichen auf dem Code vom Google Blockly Framework beruhen, schraffiert gekennzeichnet.

Das Übersichtskomponentendiagramm (vgl. 4.1) beschreibt die übergeordneten Komponenten und zeigt auf, dass die Dateien in der Import- / Laden-Komponente sowie die Datenbestand-Komponente für die Blockly-Komponente bereitgestellt werden. Dort werden die Daten verarbeitet und in der Anzeige-Komponente visualisiert. Zum Speichern und Exportierten wird die Export- / Speichern-Komponente verwendet. Diese übergeordneten Komponenten sollen im Folgenden detailliert beschrieben werden.

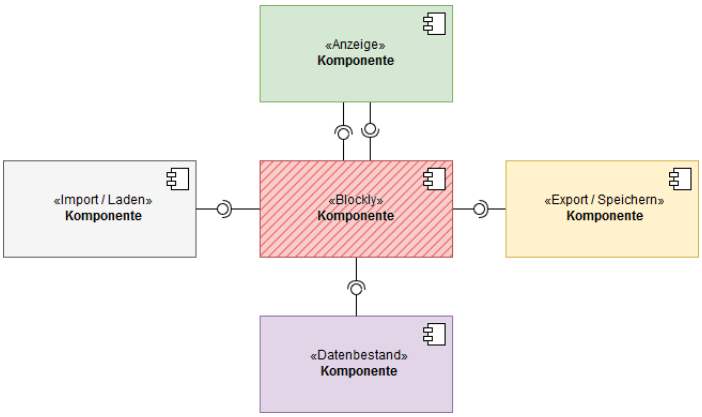


Abbildung 4.1: EZWEB-Editor - Komponentendiagramm - grob Quelle: eigenes Werk.

Im Mittelpunkt der Anwendung steht der graphische Editor, welcher zusammen mit der Navigationsleiste im Komponentendiagramm (vgl. 4.2) in der Bearbeitungsansicht-Komponente zu finden ist. Dieser graphische Editor hat immer genau einen Workspace und eine Toolbox. Der Workspace wird hierbei aus der Framework-Komponente von Blockly bereitgestellt. Die Toolbox kommt aus der Datenbestand-Komponente und greift neben der Toolbox-Tree-Komponente, welche die Baumstruktur der Kategorien für die Blöcke in der Toolbox bereitstellt, auch auf die Block-Komponente zurück, aus der sie die Blöcke erhält. Blöcke können aus der Toolbox auf dem Workspace abgelegt und miteinander verbunden werden.

Wenn ein Tutorial ausgewählt wurde, besteht die Bearbeitungsansicht in diesem Fall aus dem eben genannten graphischen Editor, der parallel hierzu noch die jeweilige Aufgabenstellung des aktuellen Tutorials anzeigt. Dabei kann mitunter der Umfang der bereitgestellten Toolbox eingeschränkt und auf das Tutorial abgestimmt sein. Weiterhin wird hierbei wieder auf die Navigationsleiste zurückgegriffen.

Die HTML-Preview aus der Anzeige-Komponente ist eine andere Darstellung der Bearbeitungsansicht und zeigt den aktuellen Code von den komponierten Blöcken als Webseite an.

Die Code-Preview aus der Anzeige-Komponente ist ebenfalls eine andere Darstellung der Bearbeitungsansicht und zeigt den aktuellen Code von den komponierten Blöcken als richtigen Code an, welcher entsprechend der Syntax farblich hervorgehoben wird.

Das Importieren von Fremdcode und Laden bereits erstellter Webseiten findet innerhalb der Import- / Laden-Komponente statt. Dazu wird der Code je nach Funktion eingelesen, mit Hilfe der Blockly-Komponenten verarbeitet und in der Anzeige-Komponente ausgegeben.

Das Exportieren der Webseiten als „.html“ Datei und das Speichern dieser als „.xmlbly“ Datei finden in der Export- / Speichern-Komponente statt. Dazu wird der in der Persistenzschicht gehaltene Code in das jeweilige Dateiformat gewandelt und in diesem zum Herunterladen angeboten.

Die Blockly-Komponente setzt sich aus der Persistenz-Komponente und dem Blockly-Framework zusammen. Diese Funktionen werden von Google Blockly bereitgestellt. Hier findet die grundlegende Verarbeitung statt.

Die Datenbestand-Komponente hat zwei Unterkomponenten, die jeweilig nur Daten für das Google Blockly Framework bereitstellen. Zu diesen gehört die Toolbox-Tree-Komponente, welche wie bereits oben beschrieben die Baumstruktur der Kategorien für die Blöcke in der Toolbox bereitstellt. Die interne Repräsentation findet hierbei als „.xml“ Datei statt. Die zweite Unterkomponente ist die Block-Komponente. In dieser befinden sich alle für den Editor erstellen Blöcke mit Ihren Funktionalitäten.

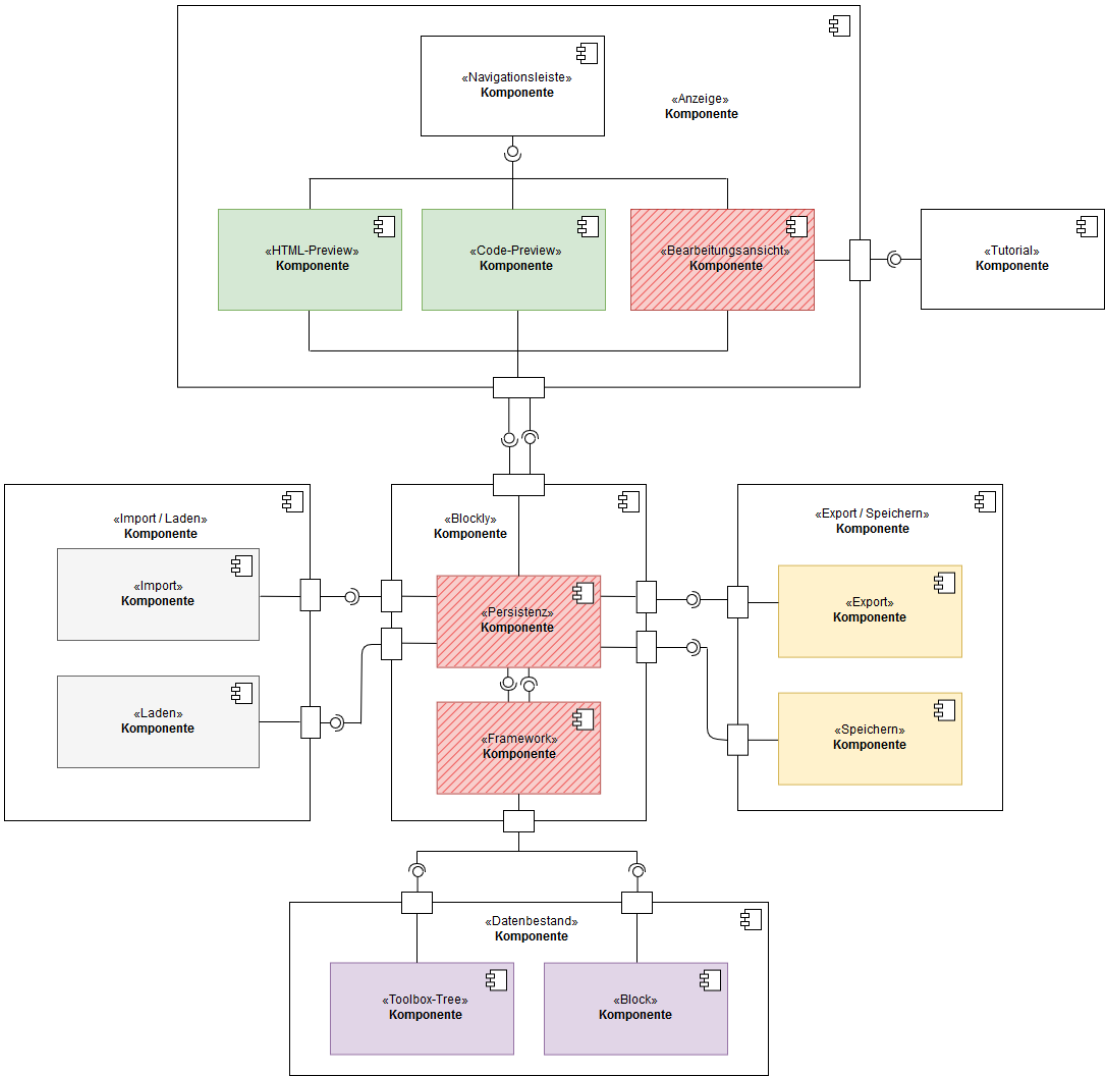


Abbildung 4.2: EZWEB-Editor - Komponentendiagramm - fein Quelle: eigenes Werk.

4.2 Abbildung des Blocklymodelles auf HTML

An dieser Stelle soll mit Hilfe der Grafik 4.3 erklärt werden, wie die konkrete Abbildung des Blocklymodelles auf HTML in dem EZWEB-Editor umgesetzt wurde. Aufzuzeigen sei hierbei die sequenzielle und hierarchische Abbildung von HTML mit Hilfe der externen Blockansicht, sowie der interne Aufbau der Blöcke zur Umsetzung diverser Funktionen wie z.B. der Import- / Exportfunktion.

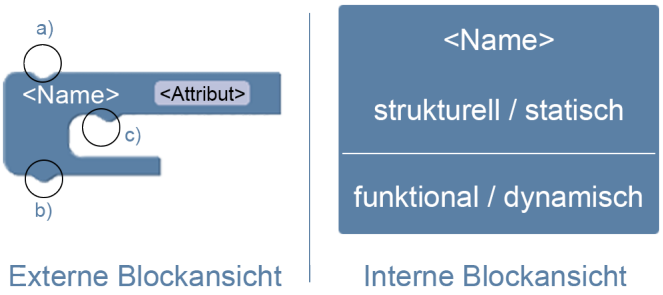


Abbildung 4.3: Abbildung des Blocklymodelles auf HTML. Interne und Externe Blockansicht. Quelle: eigenes Werk.

HTML unterliegt einer Baumstruktur. Das bedeutet, sog. Tags werden hierarchisch und sequenziell als Baum abgebildet. Ein HTML-Tag kann mehrere Attribute besitzen, welche hier innerhalb der Blöcke hintereinander als Inputfield angeordnet werden. Mit Hilfe der Verbindungselemente a) und b) aus der Grafik, 4.3 (Externe Blockansicht) welche in Blockly „Previous- und Next-Connector“ genannt werden, können HTML-Tags als Blöcke sequenziell abgebildet werden, dazu müssen diese lediglich miteinander verbunden werden. Der „Input-Connector“ aus dieser Grafik, näher gekennzeichnet als c), ermöglicht die hierarchische Abbildung von Tags innerhalb der Baumstruktur. Um Blöcken Verbindungselemente zuzuweisen müssen diese im strukturellen / statischen Teils des Blocks definiert werden (vgl. 4.3) (Interne Blockansicht). Neben der Zuweisung der Verbindungselemente wird innerhalb des strukturellen / statischen Bereichs die Farbe der Blöcke, der Name, die Attribute und der Tooltip angegeben. Innerhalb des funktionalen / dynamischen Teils, befinden sich eigens erstellte JavaScript-Funktionen für die Codeausgabe und die Importfunktion. Die strukturellen / statischen Befehle werden dabei von Google Blockly bereitgestellt.

4.3 Prototypische Implementierung

In dem Kapitel prototypische Implementierung soll aufgezeigt werden, wie die Anforderungen und die Designentscheidungen des EZWEB-Editor in einem ersten Prototypen implementiert worden sind. Dieser Prototyp soll die Grundlage für die weitere Forschung und die Umfrage zur Validation des EZWEB-Editor stellen. Dazu werden zuerst notwendige Bibliotheken in 4.3.1 und der funktionale Umfang in 4.3.2 beschrieben. Anschließend wird in 4.3.3 der graphische Editor erläutert. In 4.3.4 wird auf die Tutorialumsetzung eingegangen und unter 4.3.5 die Erweiterbarkeit der prototypischen Implementierung des EZWEB-Editors erläutert. Weiterhin werden noch in 4.3.6 die Probleme, die bei der Umsetzung des Prototypen aufgetreten sind,

aufgezeigt. Abschließend soll in Tabelle 4.1 im Kapitel Anforderungsumsetzung 4.3.7 aufgezeigt werden, welche Punkte aus der Anforderungsanalyse in der prototypischen Implementation umgesetzt worden sind.

4.3.1 Bibliotheken

Um einen onlinebasierten Editor bereitzustellen, der das Komponieren von HTML-Seiten ermöglicht, soll Google Blockly als Entwicklungsumgebung / Framework verwendet werden.

Die für den EZWEB-Editor wichtigen Eigenschaften CBK und Plattformunabhängigkeit sollen durch die CSS-Bibliothek Bootstrap[2] in Kombination mit Blockly vereint in einem Internetauftritt für die Erfüllung dieser Eigenschaften sorgen und freie Skalierbarkeit ab einer Displayauflösung von 720p ermöglichen.

4.3.2 Funktionaler Umfang

Um den EZWEB-Editor nicht nur für eine natürliche Sprache wie Deutsch zugänglich zu machen, sollen alle Texte und Wörter des EZWEB-Editor in eine separate Datei gelegt werden. Somit besteht die Möglichkeit, für diverse Sprachen Übersetzungen anzulegen. Jedoch soll die prototypische Implementierung zunächst nur in Englisch verfügbar sein und auf dieses Feature verzichten.

Der EZWEB-Editor soll neben HTML auch die Sprachen CSS, JavaScript und php verarbeiten können und Lernmaterial dafür anbieten. In der prototypischen Implementation soll zunächst nur HTML angeboten werden. Hierbei soll mit Hilfe von Blöcken, sowie deren hierarchischen und sequenziellen Contraints die HTML-Struktur abgebildet werden. Um CSS trotzdem verwenden zu können, sollen in der prototypischen Implementierung viele Elemente mit einem Style-Attribut angeboten werden. Für die Einbringung von JavaScript- und php-Code soll vorerst ein custom-HTML-Tag als Block dienen und mit Hilfe dieses Blocks ausgeführt werden können.

4.3.3 Graphischer Editor

Der zu realisierende EZWEB-Editor wurde mit Hilfe des blockbasierten Google Blockly umgesetzt und in einen mit *Bootstrap* gestalteten Internetauftritt eingebunden, um kostenlos für Jedermann zur Verfügung gestellt werden zu können. Anhand der Grafik 4.4 sollen zunächst die Funktionen und der grundlegende Aufbau des graphischen Editors erläutert werden. Dazu wurde die Grafik mit den Markierungen a) für die Toolbox 4.3.3, b) für den Workspace 4.3.3, c) für das Navigationsmenü 4.3.3 sowie d) für die Tutorials 4.3.4 versehen.

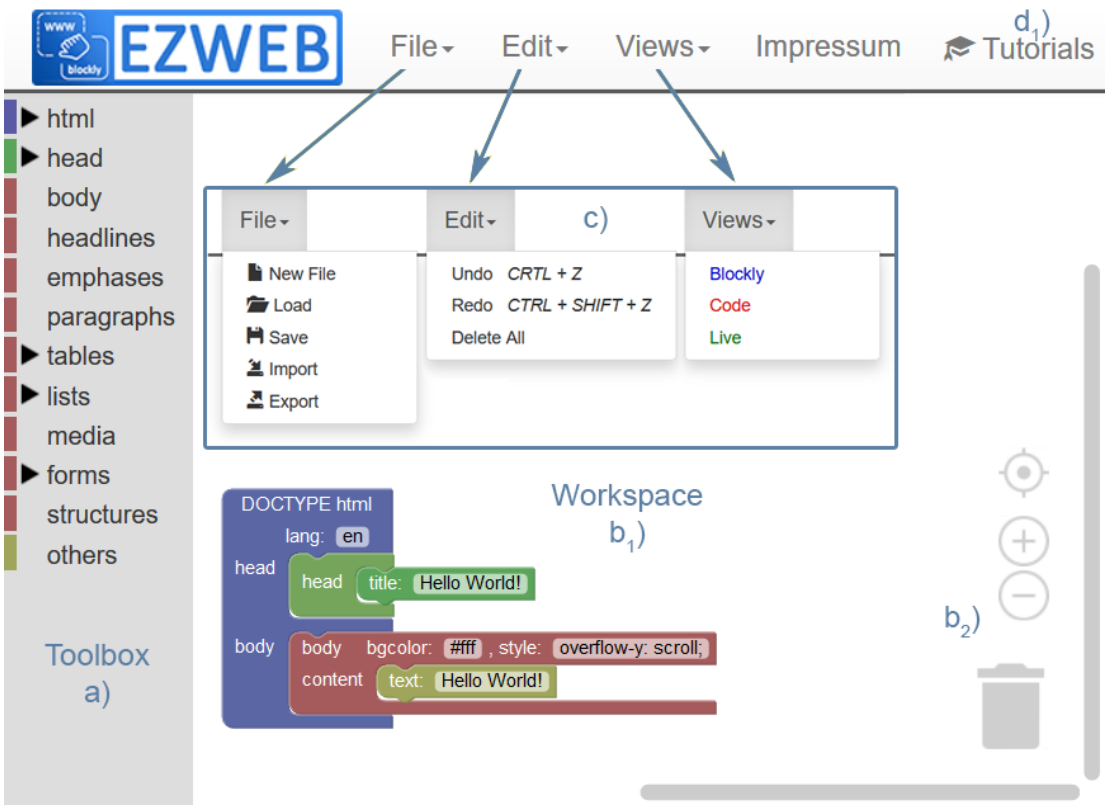


Abbildung 4.4: Aufbau der prototypischen Implementation des Editors. Quelle: eigenes Werk.

Toolbox Unter a) (vgl. Grafik 4.4) ist die vollständige Toolbox zu sehen, welche Blöcke innerhalb von Kategorien enthält. Für eine bessere Übersicht der Toolbox wurden zusammengehörige Kategorien inklusive untergeordneter Tags gleichfarbig gekennzeichnet und somit von anderen Kategorien abgegrenzt. Grundsätzlich besitzt die Toolbox nur leere Elemente, zu denen Attribute und Inhalten gehören. Die leeren Elemente werden erst beim Initialisieren gefüllt und beherbergen zumeist auch sinnvolle Hilfestellungen für die Nutzer. Weiterhin enthält die Toolbox in den Unterkategorien *pre-built* diverse vorgefertigte Blöcke, um häufig auftretende Kompositionen von Blöcken bereitzustellen und einen zügigen Arbeitsfluss zu gewähren. Diese enthalten z.B. ein komplettes Kontaktformularlayout und bieten hierbei gerade für Einsteiger einen Mehrwert. In Abbildung 4.5 ist der Unterschied der einzelnen Elemente im Gegensatz zu einem *pre-built* Beispiel zu sehen. Nicht zu vergessen befindet sich der Menüpunkt für den Custom-Table-Generator auch innerhalb der Toolbox in der Kategorie *tables*. Dieser soll die zunächst durchaus komplex wirkende Struktur von Tabellen in HTML einsteigerfreundlicher gestalten.

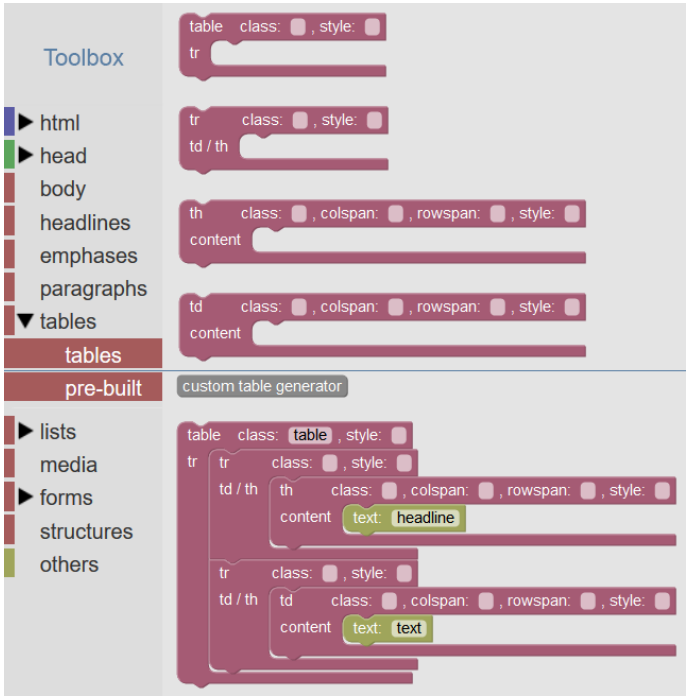


Abbildung 4.5: Aufbau Toolbox normal und pre-built. Quelle: eigenes Werk.

Workspace Unter b_1) (vgl. 4.4) ist der Workspace in Form der Bearbeitungsansicht zu sehen. Unter b_2) sind der Papierkorb und die Zoom-Optionen des Workspaces zu sehen. Zoomen kann hierbei ebenfalls über das Scrollrad der Maus erfolgen. Auf der gleichen Grafik ist ein zusammengesetzter Block abgebildet, der eine triviale Webseite mit dem Inhalt „Hello World!“ zeigt. Um in diesem Editor eine Webseite erstellen zu können, werden Blöcke mit Hilfe der linken Maustaste, also per Drag & Drop, aus der Toolbox in den Workspace gezogen und abgelegt. Weiter können diese Blöcke mit der Maus nun auf gleichem Wege an die Konnektoren der anderen auf dem Workspace liegenden Blöcke angebunden werden. Viele Sonderfunktionen stellen Blöcke durch die von Google Blockly gegebene Rechtsklick-Funktion bereit. Die Sonderfunktionen werden durch die Grafik 4.6 aufgezeigt.

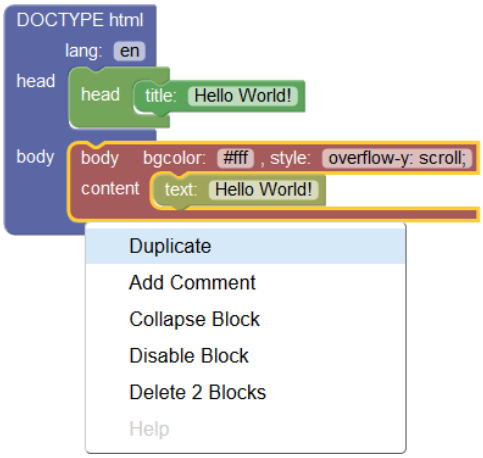


Abbildung 4.6: Rechtsklick-Funktionen der Blöcke. Quelle: eigenes Werk.

Navigationsmenü und Funktionen Auf der Oberfläche des EZWEB-Editors befindet sich in der Abbildung 4.4 unter c) das Navigationsmenü. Dieses ist zu besseren Übersicht ausgeklappt in einem Kasten ebenfalls auf dieser Grafik zu sehen. Dadurch lassen die Unterpunkte zu *File*, *Edit* und *Views* erkennen.

Wie erwähnt sollen eine Import-, sowie eine Exportfunktion in diesem Editor, zu finden unter dem Menüpunkt *File*, Einzug halten. Die Exportfunktion zum Exportieren der Blöcke einer „.html“-Datei greift hierbei auf die Textausgaben der in den Blöcken definierten Code-Zeichenketten zurück und gibt diese entlang des hierarchischen und sequenziellen aufgebauten Baumes aus. Die Zeichenketten, die anhand der Hierarchie des Baumes erstellt werden, erhalten beim Entlanggehen dieses Baumes Tabulatoren zur korrekten Einrückung. Die Ausgabe findet als „.html“-Datei statt und vernachlässigt bei der Anzeige nicht ausgefüllte Attribute um Speicherplatz zu sparen. Für die Code-Preview wird die gleiche interne Funktion verwendet, mit dem Unterschied der Ausgabe, die hier in Kombination mit der eingebunden Bibliothek codemirror[1] zusammen als Textausgabe erfolgt.

Deutlich aufwendiger ist die Implementation der Importfunktion. Die zu importierende „.html“-Datei wird zunächst mit Hilfe einer selbstgeschriebenen MinifyHTML-Funktion in eine einheitliche Form gebracht. MinifyHTML erfüllt dabei folgende Aufgaben:

- Konvertieren des Doctype (convertDoctype)
- Entfernen aller Tabulatoren (removeAllTabs)
- Entfernen aller multiplen Leerzeichen (removeMultipleWhitespaces)

- Konvertieren der Kommentare in HTML (convertHtmlComment)
- Entfernen aller Leerzeichen zwischen Tags (removeWhitespacesBetweenTags)
- Entfernen aller Zeilenvorschübe (removeAllNewlines)

Danach zerlegt ein Parser den String in die HTML-Baumstruktur, ausgeschlossen dem Doctype. Dieser Baum wird nun rekursiv durchgegangen und für die gefundenen Tags Blöcke erstellt. Die hierbei enthaltenen Attribute werden an der äquivalenten Stelle in den Blöcken, welche die Tags repräsentieren, eingelesen. Das bedeutet, dass die Blöcke ebenfalls Felder für Attribute aufweisen. Aktuell werden beim Importieren von Fremdcode nicht implementierte Sprachen und Tags gekennzeichnet oder in einem Custom-Code-Feld angezeigt. Sofern Funktionalitäten nicht unterstützt sind wird also eine Fehlerbehandlung durchgeführt. JavaScript, php und CSS werden insofern sinnvoll als custom-Tag eingebracht oder, falls dieser Teil definitiv nicht unterstützt wird, anstattdessen ein Fehlermeldungsblock eingebracht. Die Funktion der Delegation und Blockerzeugung verwaltet jeder Block für sich.

Möchte ein Nutzer eine exkludierte Sprache oder Funktionalität einbringen steht ihm das Custom-Code-Feld zur Verfügung, zu finden in der Toolbox unter *others*.

Weiterhin gibt es drei Ansichten die mit Hilfe des Navigationspunktes *Views* aufgerufen werden können: Blockly Bearbeitungsansicht, Code-Preview und HTML-Preview. Angefangen bei der HTML-Preview ist hier als Ausgabe das Endresultat zu sehen, welches dann auch live im Internet bei einem Webseitenaufruf zu sehen seien würde. Realisiert wird diese Ansicht mit Hilfe eines iFrames, wobei der aktuelle Webseitencode aus der Code-Preview in das Attribut „srcdoc“ eingefügt wird.

Die Bearbeitungsansicht zeigt neben der Toolbox auch den aktuellen Prozessstand der Webseite abgebildet als Blöcke. Vgl. Abbildung 4.4

Ferner zeigt die Code-Preview parallel zu den Blöcken den Code, den diese als Ausgabe haben. Um dem Nutzer in der Code-Preview eine bessere Visualisierung zu bieten wurde mit Hilfe der Bibliothek *codemirror*[1] eine farbliche Kennzeichnung von Tags und Attributen eingeführt. Aktuell ist, wie in der Anforderungsanalyse beschrieben, diese Ansicht zunächst readonly. Dieses soll sich dann im späteren Verlauf mit dem Einbringen der vollständigen Reverse-Engineering Funktion ändern. Weiterhin werden in der Code-Preview nicht verwendete, also leere Attribute, aus der Bearbeitungsansicht ausgeblendet. Dieses sorgt für eine bessere Übersicht und zeigt daher auch nur den erforderlichen Code. In der Grafik 4.7 ist als Gegenüberstellung der Unterschied zwischen Bearbeitungsansicht und Code-Preview zu sehen.

[1] https://codemirror.net

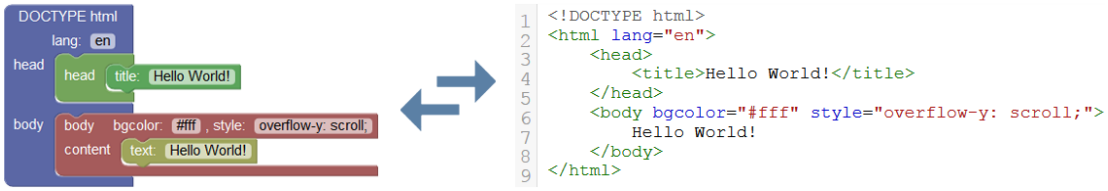


Abbildung 4.7: Blockly View Gegenüberstellung zur Code View. Quelle: eigenes Werk.

Besonderheiten und Einschränkungen Da sich im Allgemeinen gegen Restriktionen der Blöcke entschieden wurde sollen farbliche Kennzeichnungen hier ebenfalls einen Hinweis auf kompatiblen Code geben.

Blockly verwendet als Dateistruktur „xml“ innerhalb der Toolbox zur Repräsentation. Um in der Import- und Export-, sowie Load- und Save-Funktion nicht für zu viel Verwirrung zu sorgen wurde sich dafür entschieden, „.xmlbly“ als Dateiformat zu verwenden. Dies hat den Hintergrund, dass so nicht fälschlicherweise vermutet werden kann, dass aus jedem „.xml“ File eine Website erstellt werden kann. Um zu garantieren, dass dieses neu geschaffene Dateiformat noch nicht verwendet wird, wurde es mit Hilfe von dafür vorgesehenen Seiten geprüft. Vgl. [3, 4, 2]

Der EZWEB-Editor sowie dessen Funktionen und Tutorials stehen zunächst ausschließlich in Englisch zur Verfügung. Auch Quellcode und Kommentare sind auf Englisch erstellt worden. Eine Erweiterung für andere Sprachen ist zu einem späteren Zeitpunkt geplant. Vgl. 6.1

Nach einigen Versuchen wurde sich aus praktikablen Gründen gegen größen-veränderbare Fenster entschieden. Dabei haben sich bisher verglichene Bibliotheken als inkompatibel herausgestellt. Aktuell würde nur eine Eigenentwicklung eines dreigeteilten Rahmensystems mit darin liegenden größen-veränderbaren Flächen, basierend auf JavaScript, den gewünschten Effekt liefern. Dazu müsste sichergestellt werden, dass die Ansichten dabei zur Laufzeit ohne Rendering-Fehler abgebildet werden können. Die Eigenentwicklung eines solchen Systems würde allerdings den zeitlichen Rahmen dieser Arbeit überschreiten.

4.3.4 Tutorials

Die Grafik 4.8 zeigt den Editor mit einem geöffneten Tutorial (Tutorial 1 - HTML Grundlagen). d_1) innerhalb dieser der Grafik zeigt den eigenständigen Menüpunkt des EZWEB-Editors zum Erreichen der Tutorials und d_2) zeigt hier die Beschreibung für die jeweilige Aufgabe, in diesem Fall *Tutorial 1 - HTML Grundlagen*. Diese Sonderansicht (Beschreibung) ist im normalen Editor-Modus nicht zu sehen und taucht nur innerhalb der Tutorialumgebung auf. Innerhalb der Tutorials ist die Toolbox, abhängig vom jeweiligen Tutorial, eingeschränkt. Am Ende

eines jeden Tutorials gibt es die Möglichkeit, durch einen *Next-Tutorial-Button* zum nächsten Tutorials zu springen.

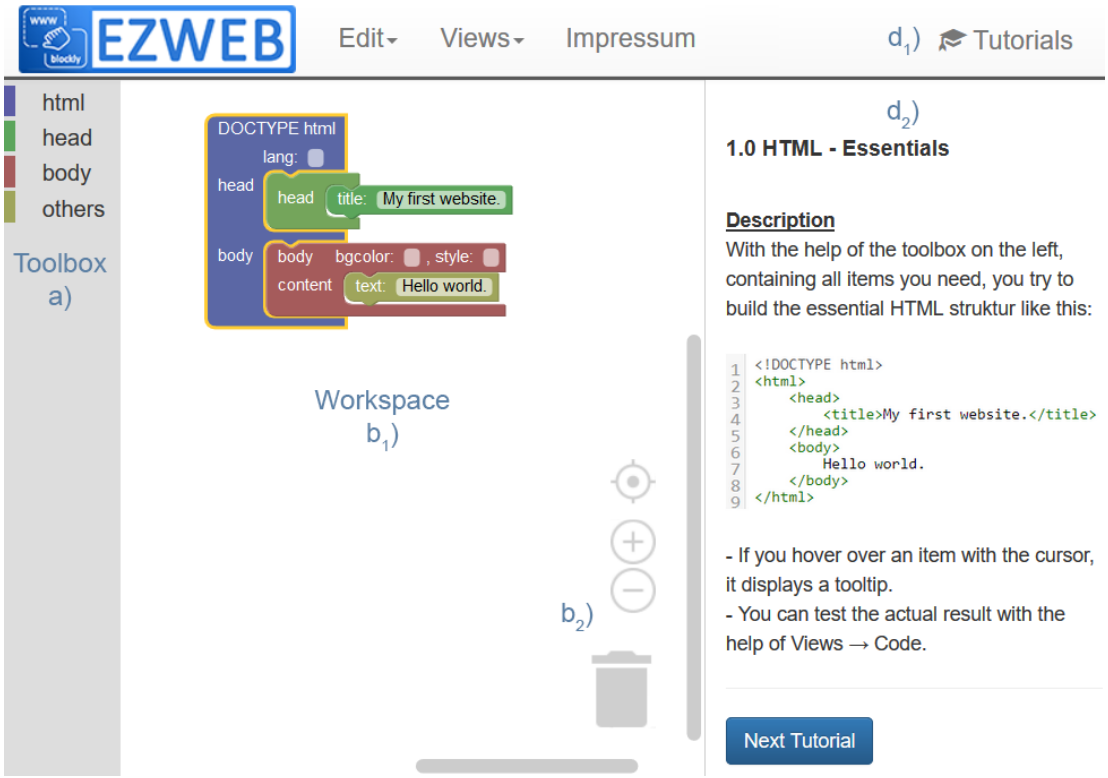


Abbildung 4.8: Aufbau der prototypischen Implementation des Editors mit einem Tutorial. Quelle: eigenes Werk.

Die Tutorials im EZWEB-Editor sollen zunächst nur den einfachen Sachverhalt von HTML erklären. Dabei sollen neben einem Tutorial zum Bedienen des Editors als Bedienungsanleitung auch sieben weitere Tutorials implementiert werden, welche, anfangend bei der Grundstruktur, auch das Manipulieren von Schrift lehren. Wichtige Lerninhalte sind weiterhin Listen, Tabellen, Formulare, Textformatierungen, sowie der Umgang mit Medien. Die Überpunkte der Tutorials können der Übersichtsseite (vgl. 4.9) entnommen werden. Die konkrete Aufgabenstellung der einzelnen Tutorials ist in 3.1.5 beschrieben.

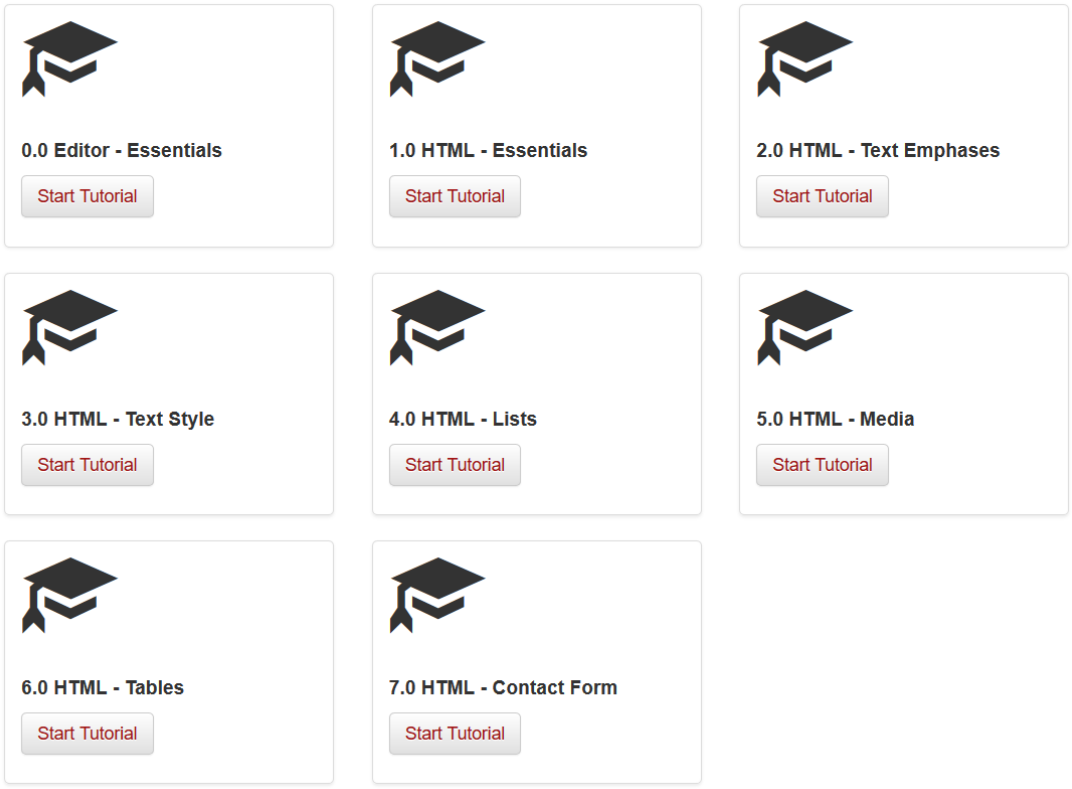


Abbildung 4.9: Übersicht der Tutorials. Quelle: eigenes Werk.

Welcher Lernerfolg kann von den Tutorials erwartet werden? Neueinsteiger sollen nach Abschluss der Tutorials in der Lage sein, eine Singlepage-Website zu erstellen, die sie als Steckbrief über Hobbys oder ein bestimmtes Thema verwenden können. Diese Zielsetzung wurde bereits in Kapitel 2.1 beschrieben. Der Lernerfolg der Aufgaben in den Tutorials soll mit Hilfe der Lernmethode PBL, beschrieben in 2.2, erreicht werden. Die Aufgaben in den Tutorials haben keine automatische Lösungsüberprüfung. Hierfür wurde sich entschieden, da es mitunter mehrere Lösungen geben kann und eine automatisierte Kontrolle, die alle Variationen der Aufgabenlösungen überprüfen würde, den Rahmen dieser Arbeit überschreiten würde. Ein solches Konzept kann im Anschluss der Arbeit erarbeitet und in Ausblick gestellt werden.

Wie in der Untersuchung aufgezeigt, war eines der Ergebnisse aus der in 5.3 beschriebenen Umfrage die Integration einer Funktion, die es ermöglicht zwischen den Tutorials direkt zu wechseln, ohne das Menü erneut aufzurufen. Dieses fördere einen durchgehenden Arbeitsfluss

und verhindere Störungen im Lernablauf und wurde später durch einen „Next-Tutorial-Button“ (siehe oben) nachgeliefert.

4.3.5 Erweiterbarkeit

An dieser Stelle soll die Erweiterbarkeit des EZWEB-Editors betrachtet werden. Schematisch wird in diesem Abschnitt die Erstellung neuer Blöcke und deren Integration in die Kategorien beschrieben. Weiterhin auch die Erstellung neuer Kategorien und Tutorials.

Erstellung neuer Blöcke: Im Folgenden soll zunächst schematisch erklärt werden, wie neue Blöcke erstellt und in den EZWEB-Editor integriert werden können. Um einen Block zu erstellen, muss dieser innerhalb einer dafür geeigneten Kategorie im Ordnerpfad „javascript/toolbox/[Kategorie].js“ erstellt werden. Als Vorlage hierfür können bereits erstellte Blöcke genommen werden. Als wichtig ist hierbei zu erachten, dass für den Block die richtigen Verbindungen / Konnektoren herausgesucht werden und die korrekte Importfunktion gewählt wird. An dieser Stelle sei noch einmal betont, dass ein Block genau einen Tag mit seinen Attributen repräsentiert. Konnektoren ermöglichen dem Tag die sequenzielle und hierarchische Abbildung und unterliegen keiner Belegungs- / Verbindungspflicht. Wie kann die Art des Blockes hinsichtlich seiner potentiellen Konnektoren sowie die korrekte Importfunktion gewählt werden? Für gewöhnlich werden alle umschließenden Tags, als Beispiel das -Tag, in dem sich weiterer Inhalt befinden kann, als Block mit einem Inputkonnektor definiert. Als dazugehörige Importmethode muss die „importHtmlChildrenThreeParameter(...)-Funktion verwendet werden. Sofern selbst-schließende Tags wie z.B. das -Tag eingebracht werden sollen, wird auf den Inputkonnektor verzichtet und die importHtmlChildrenParameter(...)-Funktion verwendet. Bei beiden Blöcken werden zusätzlich neben den Next- und Previous-Konnektoren die Textfelder der Attribute definiert. Beispiele zur Einbringung befinden sich ebenfalls in den oben beschrieben Dateien. Um den neu erstellten Block anschließend im graphischen Editor verwenden und anzeigen zu können, muss dieser auch in die „javascript/toolbox.js“ an richtiger, kategorisch sinnvoller Stelle eingetragen werden.

Erstellung neuer Kategorien: Die innerhalb der im Verzeichnis „javascript“ liegenden toolbox-Dateien können um neue Kategorien ergänzt werden, indem neue xml-Befehle zum Erstellen von Kategorien eingetragen werden. Diese sind gekennzeichnet durch den Namen „category“ und verfügen über die Attribute „Name und Farbe“. Als Beispiel können die bereits dort verankerten Kategorie verwendet werden.

Erstellung neuer Tutorials: An dieser Stelle soll die Erstellung und Integration von neuen Tutorials für den EZWEB-Editor aufgezeigt werden. Im Hauptverzeichnis des EZWEB-Editors muss eine neue „tutorial[Nr.].html"-Datei erschaffen und in der „index.html" verlinkt werden. Aktuell muss die Aufgabenstellung, welche noch prototypische Redundanzen enthält, für die Beschreibung in jeder der einzelnen Ansichten definiert werden. Später soll diese durch einen einmaligen Eintrag vereinfacht werden, um dauerhaft eingeblendet werden zu können. Für die einzelnen Tutorials wurden unterschiedliche Toolbox-Dateien erstellt, die sich im gleichen Verzeichnis mit der gleichen Nummer befinden und mit „toolbox-tutorial[Nr.].js" gekennzeichnet worden sind. Dieses wurde vorgenommen, damit die Toolbox innerhalb der Tutorials hinsichtlich ihres Leistungsumfangs eingeschränkt werden kann. Innerhalb der „tutorial[Nr.].html"-Datei muss, sofern ein anderes Tutorials als Beispiel herangezogen wird, auch die Nummer der Toolbox geändert werden.

Ergänzung von natürlichen Sprachen: Bei der prototypischen Implementierung wurde zunächst auf die Erweiterung für natürliche Sprachen verzichtet. Um dieses später realisieren zu können, sollte der gesamte Text in Variablen ausgelagert und je nach Spracheinstellungen die richtige Sprache eingefügt werden.

Ergänzung von Programmiersprachen: Für die Integration weiterer Sprachen wie CSS, JS und php muss im Weiteren noch ein Konzept zur sinnvollen Einbindung gefunden werden. Hierzu gehört ebenfalls das sich-Abheben dieser Sprachen von HTML durch Code-Highlighting innerhalb der Code-Preview und die Kenntlichmachung in der Bearbeitungsansicht.

4.3.6 Probleme

Probleme bereitete die Textarea, welche aktuell für den Nutzer keinen Umfang bietet, der dem Anspruch einer vollständigen Textarea gerecht wird. Google Blockly hat bei aktuellem Entwicklungsstand vom 24.10.2016 keine vollfunktionale Textarea integriert. Somit dient hier ein einzeiliges, daher bei langen Texten unübersichtliches, Inputfield als Zwischenlösung. Google Blockly hat aber in Aussicht gestellt, eine eigenständige Textarea in die kommenden Versionen einzupflegen.

Ein weiteres Problem trat beim Testen auf, als das JavaScript-Tag „<script>...</script>" vorzeitig durch ein anderes schließendes JavaScript-Tag aus der Testvariablen beendet wurde, denn auch die Funktionen waren in JavaScript geschrieben. Dieser Umstand konnte durch Auslagerung der Testvariablen in eine externe Datei gelöst werden, wie es auch später im regulären Programmbetrieb der Fall wäre.

4.3.7 Anforderungsumsetzung

Um abschließend zu klären, welche der in 3 gegebenen Anforderungen in der prototypischen Implementierung erfüllt worden sind, soll die Tabelle 4.1 vollständig umgesetzte Anforderungen und nicht vollständig bzw. ausstehende Anforderungen aufzeigen:

Anforderung	Vollständig	Teilvollständig	Ausstehend
4.1.1.1.1 Editor Aufbau	✓		
4.1.1.1.2 Displayauflösung	✓		
4.1.1.1.3 HTML-Abbildung	✓		
4.1.1.1.4 HTML-Code Gen. / Web.	✓		
4.1.1.1.5 Fehlerbehandlung	✓		
4.1.1.2 Toolbox	✓		
4.1.1.3 Workspace	✓		
4.1.1.3.1 Papierkorb	✓		
4.1.1.3.2 Zoom	✓		
4.1.1.3.3 Kontextmenü der Blöcke	✓		
4.1.1.3.4 Neue Datei erstellen	✓		
4.1.1.3.5 Rückgängig	✓		
4.1.1.3.6 Wiederholen	✓		
4.1.1.3.7 Alles Löschen	✓		
4.1.2 Persistenz	✓		
4.1.2.1 Datei Laden	✓		
4.1.2.2 Datei Speichern	✓		
4.1.3 Import / Export von Webseiten	✓		
4.1.3.1 Datei Importieren	✓		
4.1.3.2 Datei Exportieren	✓		
4.1.4 Code- / HTML-Preview	✓		
4.1.4.1 Code-Preview	✓		
4.1.4.2 HTML-Preview	✓		
4.1.5 Tutorials	✓		
4.1.5.1 Tutorial 0 - Editor Grundlagen	✓		
4.1.5.2 Tutorial 1 - HTML Grundl.	✓		
4.1.5.3 Tutorial 2 - HTML Textbet.	✓		
4.1.5.4 Tutorial 3 - HTML Textstyle	✓		
4.1.5.5 Tutorial 4 - HTML Listen	✓		
4.1.5.6 Tutorial 5 - HTML Medien	✓		
4.1.5.7 Tutorial 6 - HTML Tabellen	✓		
4.1.5.8 Tutorial 7 - HTML Kontaktf.	✓		
4.2. NFA - Übertragbarkeit	✓		
4.2. NFA - Zuverlässigkeit	✓		
4.2. NFA - Effizienz			✓
4.2. NFA - Benutzbarkeit	✓		
4.2. NFA - Wartbarkeit		✓	
4.2. NFA - Flexibilität		✓	

Tabelle 4.1: Umsetzung der Anforderungen in der prototypischen Implementierung

5 Untersuchung

Im Rahmen dieser Arbeit ergab es Sinn, den Entwicklungsstand und die Qualität des EZWEB-Editors als Gesamtkonzept zu validieren. Um neue Erkenntnisse zu gewinnen und einen Eindruck für die Weiterentwicklung zu erhalten, wurde eine Umfrage durchgeführt, die darin bestärkte, den EZWEB-Editor weiter zu entwickeln. Die Probanden haben den EZWEB-Editor im Allgemeinen positiv angenommen und beschrieben ihn als ausbaufähiges Konzept. Auch konstruktive Kritik wurde geäußert und lieferte dienliche Hinweise auf die Weiterentwicklung. Als Umfragetool wurde Typeform[1] verwendet um die Umfrage unabhängig und anonym durchführen zu können. Innerhalb dieser Arbeit wurde die Umfrage in die Umfragebeschreibung in 5.1, Auswertung (vgl. 5.2) und das Fazit der Auswertung siehe 5.3 unterteilt.

5.1 Umfragebeschreibung

In dieser Untersuchung soll die Frage geklärt werden, ob ein visueller Drag & Drop-Webseiten-Editor unter dem abgewandelten Ansatz des PBL und PfNP Potential aufweist, Nutzern das Lernen der Auszeichnungssprache HTML zu erleichtern. Um möglichst viele potentielle Probanden erreichen zu können wurden etwa 30 Freunde und Bekannte persönlich kontaktiert, sowie der Verteiler der HAW Hamburg in den Studiengängen angewandte-, technische- und Wirtschaftsinformatik angeschrieben und in Vorlesungen dafür geworben. Hier kamen 26 Umfragebeteiligungen zu Stande, welche zwar nicht repräsentativ sind, aber einen Hinweis auf den Stand der Entwicklung und des Potentials lieferten. Die Durchführung belief sich auf die vorausgehende Auseinandersetzung mit dem Prototypen. Beschrieben ist dieser in 4.3. Eine anschließende online-Umfrage, zurückgreifend auf das unabhängige Umfragetool Typeform, in welcher neben den Fragen zur Altersgruppe, Geschlecht, Auseinandersetzungszeit auch Vorkenntnisse im Bereich Webseitenerstellung erfragt wurden, wurde durchgeführt. Erfragt wurde ebenfalls, wie intuitiv der EZWEB-Editor wahrgenommen wurde, ob die Tutorials geholfen haben, welche Art von Editor bevorzugt wird und ob das Konzept überzeugend ist. Durch einen Freitextfeld am Ende konnten sinnvolle Hinweise und Anregungen, sowie

[1] https://www.typeform.com

Verbesserungsvorschläge gesammelt werden.

Umfrage Einleitung

Probanden wurden informell in einem Appell aufgefordert, den EZWEB-Editor zu testen und zum Handling, dem look-and-feel und dem Funktionsumfang Feedback zu geben. Dabei wurden zusätzlich Hardwareinformationen gegeben. Die Umfrage wurde unter dem Hintergrund geführt, ob der EZWEB-Editor dem Nutzer einen Mehrwert bieten kann. Die genaue Fragestellung der Umfrage kann im Anhang unter 6.1 eingesehen werden.

5.2 Auswertung

Die von Typeform autogenerierten Grafiken zeigen unbearbeitet das Ergebnis der Umfrage.

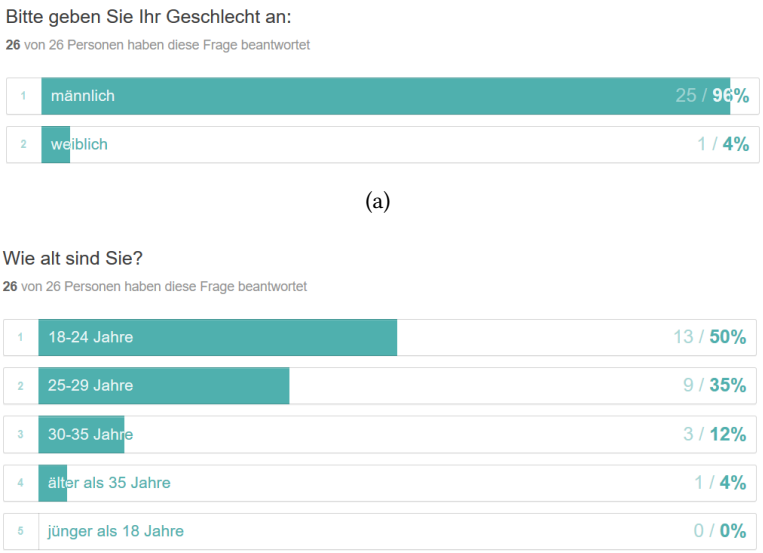


Abbildung 5.1: Typeform - Umfrage Ergebnisse der Fragen 1-2. Quelle: Autogeneriert durch Typeform.

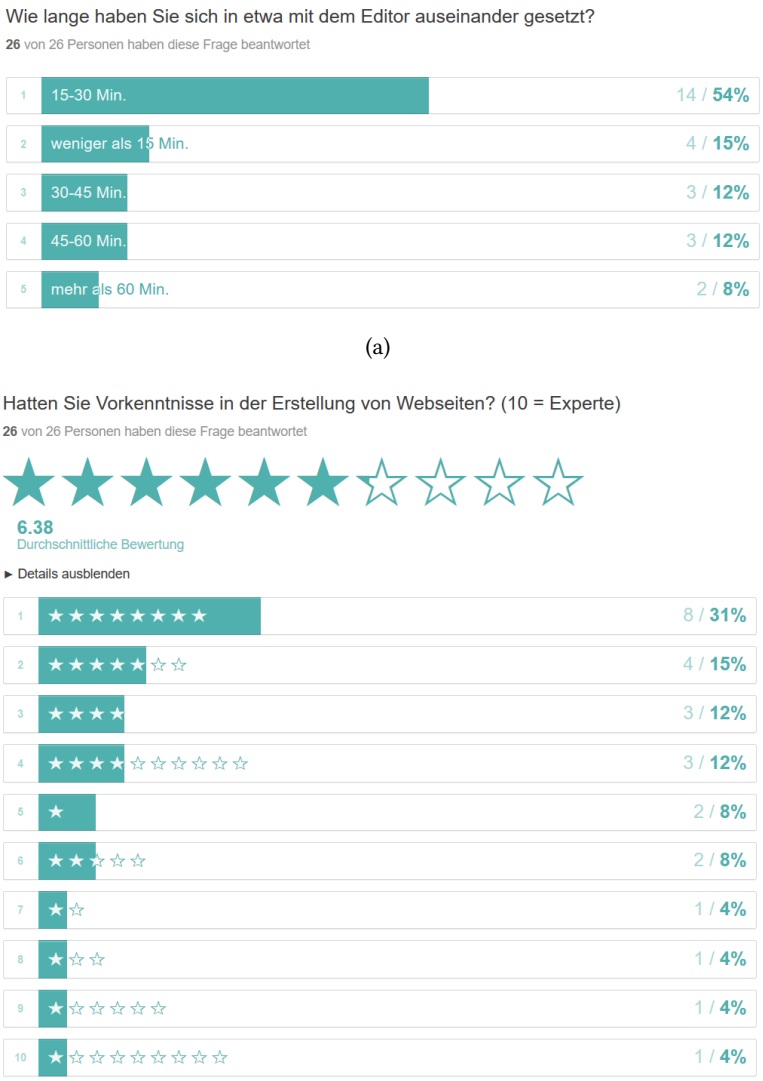
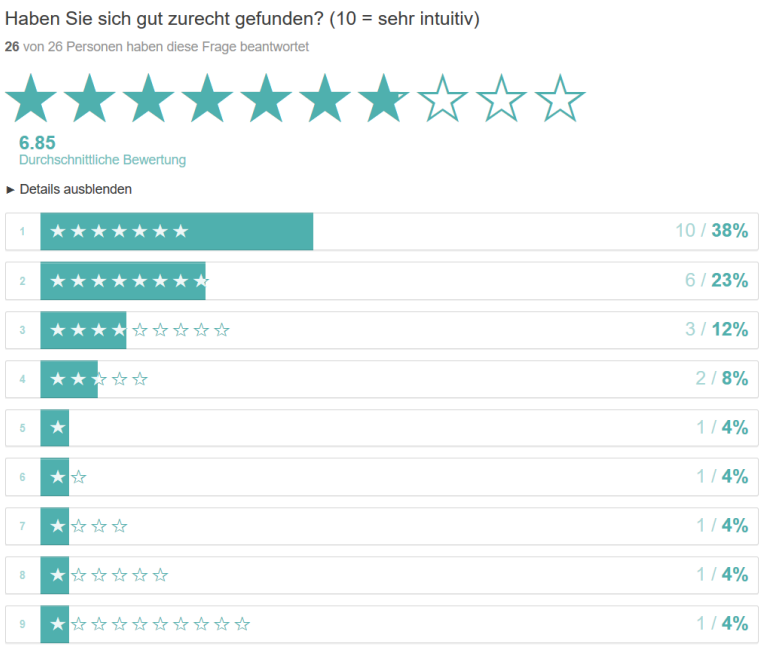
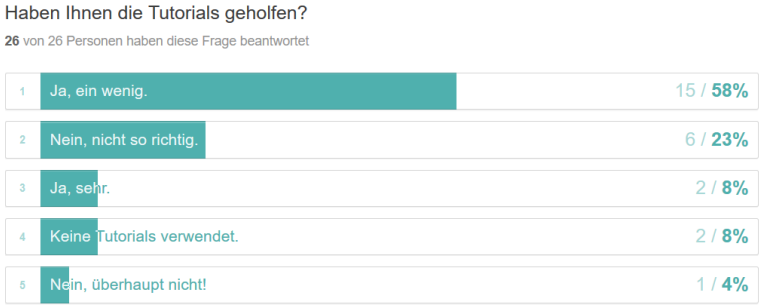


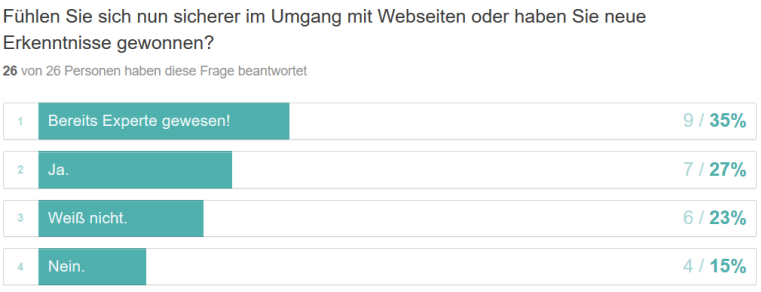
Abbildung 5.2: Typeform - Umfrage Ergebnisse der Fragen 3-4. Quelle: Autogeneriert durch Typeform.



(a)

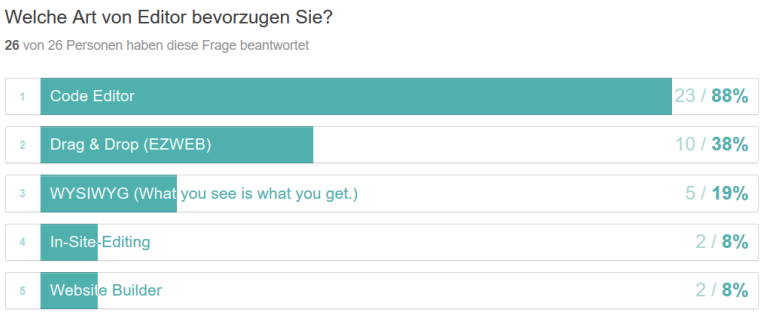


(b)

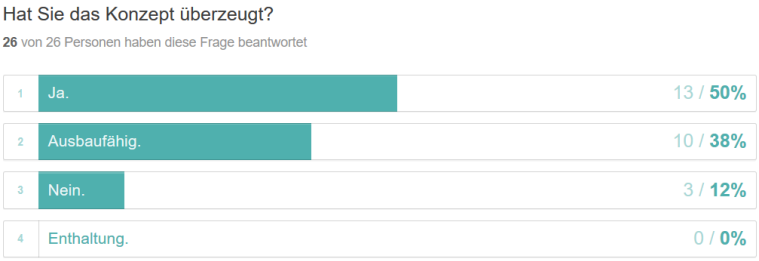


(c)

Abbildung 5.3: Typeform - Umfrage Ergebnisse der Fragen 5-7. Quelle: Autogeneriert durch Typeform.



(a)



(b)

Abbildung 5.4: Typeform - Umfrage Ergebnisse der Fragen 8-9. Quelle: Autogeneriert durch Typeform.

5.3 Fazit Auswertung

An der Umfrage haben von den etwa 3000 angeschriebenen Probanden 26 teilgenommen. Wie die Auswertung in 5.2 zeigt, wurde die prototypische Implementierung des EZWEB-Editors bereits mit einer Bewertung von 6,85/10 Punkten als intuitiv beschrieben. Weiterhin ergab die Umfrage, zu sehen in der Auswertung 5.2, dass der EZWEB-Editor den Platz Zwei der bevorzugten Editoren erreichte. Auf Platz eins landete der Code-Editor als Konzept. Diese Wahl der Plätze kann als positiv wahrgenommen werden, denn zu einem zeigt sie, dass der EZWEB-Editor einen Mehrwert für Einsteiger gegenüber des WYSIWYG-Editors, In-Site-Editing und dem Website Builder, bietet. Zum Anderen motiviert es, den EZWEB-Editor weiter zu entwickeln, um später den Nutzer in den beliebteren Ansatz des Code-Editors zu überführen. Dabei muss bei diesem Zusammenhang auch erwähnt werden, dass in etwa ein Drittel der Probanden keine bis wenig Vorerfahrungen im Bereich der Webseiten-Entwicklung hatten. Aus diesem Resultat geht bereits hervor, dass für die in Kapitel 3 definierte Lösung ein erster Erfolg zu verzeichnen ist, der die Anforderungen an eine solche weitestgehend erfüllt und aussagt, dass eine Weiterentwicklung anhand dieser Grundlage durchaus Sinn ergibt.

Aus den Einzelergebnissen der Umfrage geht ebenfalls hervor, dass den Tutorials bereits ein erster Lernerfolg zuzuweisen ist, denn Probanden mit wenigen bis keinen Vorkenntnissen haben diese Tutorials ein wenig bis sehr viel geholfen.

Besonders auffällig war der Fakt, dass Probanden mit wenig HTML-Vorkenntnissen den EZWEB-Editor deutlich besser bewertet haben. Diese Tatsache, die aus der Einzelauswertung der Ergebnisse hervorkam, spricht ebenfalls für die Weiterentwicklung, da der EZWEB-Editor ja maßgeblich für diese Zielgruppe konzipiert wurde.

Aus Frage Nr. 9 (siehe 5.2) ging als Ergebnis hervor, dass mehr als 88% der Probanden den EZWEB-Editor positiv wahrgenommen haben und als ausbaufähig erachten. Aus den Freitexten der Umfrage gingen diverse konstruktive Hinweise und Ideen zur Weiterentwicklung des EZWEB-Editors hervor, beispielsweise für neue Features sowie zur Verbesserung des Editors. Diese sollen im Folgenden kurz beschrieben und in den Ausblick aufgenommen werden.

Verlinkungen zwischen Tutorials: Besonders hob sich hier der Wunsch nach Verlinkungen zwischen Tutorials hervor. Anstatt mehrmaligen Aufrufens des Navigationsmenüs soll zukünftig ein Next-Tutorial-Button in jedem Tutorial am Ende der Aufgabenstellung dafür sorgen, dass der Lern- und Arbeitsfluss nicht unterbrochen wird. Ein solcher ist noch innerhalb dieser Arbeit in den EZWEB-Editor eingeflossen.

Lösungshilfe beim ersten Tutorial: Probanden äußerten, dass zumindest bei dem ersten Tutorial (HTML - Essentials) neben der Aufgabenstellung als Code auch die Lösung als Blockkomposition dargestellt werden sollte, um so den Zusammenhang zwischen Code und Blöcken verstehen zu können.

Tastenkombinationen: Eine andere Anmerkung bezog sich auf das Einbringen von weiteren Tastenkombinationen. Hier wurde besonders der Wunsch nach Tastenkombinationen zum Umschalten der Ansichten hervorgehoben.

Reverse-Engineering: Weiterhin kam die Forderung nach Reserve-Engineering auf. Dazu solle die Code-Preview nicht nur readonly, sondern auch editierbar sein. Da diese Funktion bereits in der Anforderung in 3.1.3 angedacht war, wird diese nachfolgend mit hoher Priorität gekennzeichnet.

Attribute untereinander anordnen: Der Anreiz, Attribute von Tags untereinander anzuordnen, kam auf und soll für eine bessere Ansicht sorgen. Dieser Anreiz kann im Anschluss an

die Arbeit untersucht werden. Dafür können z.B. diverse häufig auftretende Kompositionen erstellt und miteinander hinsichtlich der Ansicht verglichen werden.

Tooltip Erweiterung: Weiterhin sollen die Tooltips für Einsteiger weiter ausgeführt werden, so z.B. für Styles und Farben. In den Tutorials soll erklärt werden, wie Schriftarten eingebracht werden können und wie mit Farbcodes umgegangen werden kann.

Erfolgs- und Fehlermeldungen: Der Punkt, dass Erfolgs- und Fehlermeldungen innerhalb der Tutorials für ein besseres Lernerlebnis sorgen würden, wurde ebenfalls angeregt. Hierfür kam der Vorschlag einer automatisierten Überprüfung des aktuellen Fortschritts auf.

Motivation: Um den Nutzer des EZWEB-Editors zu motivieren, die Tutorials fortzuführen, wurde die Idee von Belohnungen in Form von Punkten oder Motivationsvideos genannt.

Zwischenspeicherung der Ergebnisse: Eine Anregungen zum Zwischenspeichern der Blöcke wurde ebenfalls angeregt, um bei einer neuen Seite nicht den bisherigen Code zu verlieren oder diesen auch innerhalb der Tutorials mitnehmen zu können. Hierfür könnte eine Lösung sein, den Workspace in einer Session abzulegen.

Abschließend kann neben den Anregungen die aus der Umfrage hervorgingen und den Ergebnissen, welche eine gute Tendenz hinsichtlich der Weiterentwicklung des EZWEB-Editors lieferten, gesagt werden, dass der EZWEB-Editor einen Mehrwert für die Zielgruppe der Nichtprogrammierer bietet. Diese Umfrage kann aufgrund der Teilnehmerzahl von lediglich nur 26 Probanden nicht als repräsentativ beschrieben werden, jedoch liefert sie einen Hinweis auf den aktuellen Entwicklungsstand und die zukünftige Weiterentwicklung. Es bietet sich daher an, nach der Einbringung neuer Features und Erweiterungen, eine neue Umfrage durchzuführen, die jedoch nicht mehr Teil dieser Arbeit sein soll.

Die detaillierten Umfrageergebnisse können beim Ersteller dieser Arbeit nach Bedarf eingesehen werden.

6 Zusammenfassung

In dieser Arbeit wurde die Umsetzung eines Drag & Drop-Webseiten-Editors, näher beschrieben als EZWEB-Editor, mit Hilfe der blockbasierten Programmiersprache Google Blockly in Hinblick auf den Einstieg der Erstellung von Webseiten realisiert. Mit steigendem Bedarf nach Internetauftritten von Unternehmen und Privatpersonen steigt auch die Nachfrage nach der Fähigkeit der Erstellung dieser. Bisherige Technologien werden oft als wenig einsteigerfreundlich empfunden oder liefern kein Wissen über den Code dahinter, daher ergibt sich die Notwendigkeit nach einem einfachen Einstieg in die Erstellung von Webseiten. Als Grundlage für das Lernen mit dem EZWEB-Editor fungiert die Idee des Lernens durch Untertitel sowie eine Abwandlung des Lernansatz PBL mit Tutorials. Als geeigneten Rahmen wurde die blockbasierte Sprache Google Blockly in ergänzter Form verwendet und in einen Internetauftritt eingebunden. Als Ergebnis der Evaluierung zu dem EZWEB-Editor ging hervor, dass dieser Editor positiv aufgenommen und ein Potential zu Weiterverfolgung aufweist.

PfNP ist das Keyword für das Lehren von Programmierkenntnissen an die Zielgruppe der Nichtprogrammierer. In den Grundlagen in Kapitel 2.1 wurden Konzepte für diese Zielgruppe näher beschrieben um Erkenntnisse für die Entwicklung des EZWEB-Editors zu sammeln. Hierbei ging hervor, dass grundlegende Eigenschaften, wie ein einfacher Einstieg mit steiler Lernkurve, intuitive Bedienelemente, eine geeignete Lernumgebung und weiterführende Informationen über das darüberhinausgehende Lernen für Nichtprogrammierer essentiell sind und in den EZWEB-Editor mit einzubringen waren.

Mit Hilfe des in PBL 2.2 abgewandelten Ansatzes, der die kollaborative Umsetzung des EZWEB-Editors im Sinne des Lernens in Kleingruppen auf den Wissenserwerb im alleinigen Eigenstudium verlegt, wurde die Grundlage der Integration des Lehrkonzeptes für die Eigenschaften aus PfNP gebildet. Weiterhin wurde die tutorielle Unterstützung mit Hilfe von authentischen Problemstellungen in Form von Tutorials eingebracht. Des Weiteren ging aus PBL hervor, dass eine umfassende Auswertung diverser Studien zu den Effekten von PBL einen positiven Lerneffekt gegenüber dem des traditionellen Lernens belegen konnte.

In 2.3 wurden bereits existierende graphische Programmiersprachen und deren Ausprägungen hinsichtlich ihrer Eigenschaften verglichen. Als Erkenntnis aus diesem Kapitel ging hervor,

dass Google Blockly bereits über einen ausreichenden Leistungsumfang verfügt, um durch Ergänzung neuer Funktionen den EZWEB-Editor mit seinen Anforderungen umsetzten zu können.

Der Abschnitt 2.4 lieferte die Erkenntnis, dass aktuelle Entwicklungswerkzeuge für Webseiten nicht den gewünschten Leistungsumfang bieten, der die notwendigen Eigenschaften, die an den EZWEB-Editor gestellt wurden, hinreichend erfüllen. Zu den elementaren Eigenschaften zählen: maximale Freiheit bei der Erstellung von Webseiten gepaart mit visueller Unterstützung sowie die Vermittlung von Programmierkenntnissen an Nichtprogrammierer für webseitenspezifische Sprachen.

Um den EZWEB-Editor genauer zu spezifizieren, wurden in Kapitel 3 die funktionalen und nicht-funktionalen Anforderungen detailliert niedergeschrieben. Diese Anforderungen bildeten die Basis für den Entwurf und die prototypische Implementation.

In Kapitel 4 wurde die Anforderungsanalyse zusammen mit den Erkenntnissen aus den Grundlagen in einen Softwareentwurf überführt. Dabei werden das Komponentendiagramm, die Abbildung des Blocklymodelles auf HTML, die prototypische Implementation, die Bibliotheken, sowie der Aufbau und die Funktionsweise des graphischen Editors und der Tutorials beschrieben. Weiterhin wurde die Erweiterbarkeit in Form einer Anleitung und die aufgetretenen Probleme aufgezeigt. Zuletzt wurde tabellarisch dokumentiert, welche der Anforderungen in der prototypischen Implementation umgesetzt wurden.

Abschließen wird in Kapitel 5 die Evaluation des EZWEB-Editors beschrieben. Die Durchführung wurde hierbei durch einen informellen Appell an die Probanden eingeleitet. Die Probanden sollten den EZWEB-Editor, bereitgestellt als Internetauftritt, vor dem Hintergrund eines Mehrwertes für Nutzer testen und Fragen zu dem Handling, dem look-and-feel sowie dem Funktionsumfang Feedback beantworten. Die Auswertung hat ergeben, dass 88% der Probanden den EZWEB-Editor als positiv wahrgenommen haben. Hervorzuheben sei der Fakt, dass der EZWEB-Editor besonders von Probanden mit keinen bis wenigen Vorkenntnissen in der Erstellung von Webseiten besser bewertet wurde. Da die Nichtprogrammierer die Hauptzielgruppe waren, soll dieses Umfrageergebnis darin bestärken, den EZWEB-Editor auch nach Abschluss dieser Arbeit weiterzuentwickeln.

6.1 Ausblick

Für die weitere Arbeit mit dem EZWEB-Editor stehen diverse Verbesserungen zur Realisierung offen. Daher sollen im folgenden potentielle Erweiterungen aufgezeigt werden.

Einbau eines Dateibrowsers: Denkbar ist der Einbau eines Dateibrowsers zur besseren Ansicht, ggf. zurückgreifend auf eine Projektdatei. Hierzu wäre ein anderer Ansatz die Verwendung von Sessions. Eine für den Dateibrowser sinnvolle Ergänzung wäre die Integration von Tabs, sodass bei mehreren geöffneten Dateien ein leichtes Umschalten zwischen diesen, ggf. auch durch Tastenkombinationen unterstützt, erfolgen kann. Hierbei kann der Hinweis auf ungespeicherte Dateien durch ein Modal beim Schließen dieser als sinnvoll erachtet werden. Zur Realisierung kann der aktuell offene Workspace mit einem Flag in die Session eingetragen werden. Generell kann überlegt werden, den oder die aktuellen Workspace/s in einer Session abzulegen.

Integration von CSS-Bibliotheken: Um Bootstrap und Materialize als CSS-Bibliotheken in den EZWEB-Editor einzubinden, ist es notwendig hierfür ein geeignetes Konzept zu erarbeiten. Denn es handelt sich bei den o.g. Bibliotheken nur um Sammlungen von Klassen, die über das class-Tag eingebunden werden. Denkbar wäre in diesem Fall die Integration einer Autovervollständigung. Die Neuerschaffung einer eigenen Kategorie sowie einer eigenen Blockfarbe wurde aus Gründen der Redundanz ausgeschlossen.

Textarea: Ein weiteres ausstehendes, sehr wichtiges Feature ist die Erweiterung der Textarea, die aktuell nur aus einer Zeile besteht, einem Textfield. Die Entwickler von Google Blockly stellten jedoch in Ausblick, eine reguläre Textarea als grundlegendes Feature für deren Bibliothek bereitzustellen. So kann bei Erscheinen dieses Updates leicht ausgetauscht werden. Im Allgemeinen führt dieses zu einer besseren Übersicht bei langen Texten.

Tooltips: Als Ergänzung für die Tooltips der einzelnen Tags können Beispiele sowie ein Beschreibungstext hinzugefügt werden. Auch können hilfreiche Verlinkungen zu Seiten wie „https://www.w3schools.com/html/default.asp“ eingebunden werden. Diese Idee kam aus der Umfrage hervor.

Integration von php, JS und CSS: Weiterhin soll ein praktikables benutzerfreundliches Konzept erarbeitet werden, die verbleibenden webbasierten Sprachen, wie php, JS, und CSS eigenständig in den EZWEB-Editor zu integrieren und auch mit hierfür passenden Code-Hervorhebungen einzubringen, statt wie bisher die Custom-Tags dafür zu verwenden.

Kodierung: Normalerweise bietet ein Webseiten-Editor eine Möglichkeit, die Kodierungen eine Dokuments zu ändern. Aktuell steht diese Option nicht zur Verfügung, soll aber in späteren Versionen unter dem Reiter „edit“ eingebracht werden.

Tutorials Erweiterung: Der bisherige Stand der Tutorials soll neben den entstandenen Tutorials für Anfänger (Grundlagen) auch für Fortgeschrittene erweitert werden. Ebenfalls sollen auch in diesem Zusammenhang weitere pre-built Blocks angedacht werden. Eine Kategorisierung für die einzelnen Sprachen (HTML, php, JavaScript und CSS) als Hauptübersicht soll ebenfalls eingebracht werden. Weiterhin wurde durch die Umfrage die Anregung gegeben, in den Tutorials Kenntnisse über Farben und die Farbcodetabelle, also auch den Unterschied zwischen Farbname und RGB-Code zu erläutern. Diskutiert werden kann außerdem die Idee Belohnungen für das Durcharbeiten von Tutorials einzubauen. Neben den kleinen Lernerfolgen soll so auch das körpereigene Belohnungssystem weiter bestärkt werden. Dieses kann beispielsweise durch ein Punktesystem oder durch Motivationsvideos geschehen.

Autokorrektur: Wie in der Motivation beschrieben, soll neben den gewöhnlichen Anfängerfehlern auch eine erweiterte Validationsfunktion zur automatischen Überprüfung der Korrektheit des Codes eingebracht werden. Diese soll dem Nutzer ebenfalls Hinweise zur Codeverbesserung aufzeigen.

Größenveränderbare Fenster: Über eine Einbringung von resizeable Windows, sog. größenveränderbaren Fenstern zum parallelen Betrachten von der Bearbeitungsansicht, HTML-Preview und der Code-Preview muss noch entschieden werden. Aktuell überzeugen die Ideen hinsichtlich einer Verbesserung der Usability noch nicht.

Performance Verbesserungen: Verbesserungswürdig kann mitunter die Performance der Importfunktion sein, diese weist aus manuellen Tests ein Steigerungspotential hinsichtlich der Laufzeit auf. Weiterhin weist der Custom-Table-Generator ab 15 Zeilen und 15 Spalten eine messbare Verzögerung auf. Hier sollte überprüft werden, wie sinnvoll eine Verbesserung, auch hinsichtlich der realitätsnahen Anwendung, ist.

Redundanter Code: Generell kann versucht werden, redundanten Code zusammenzufassen oder ggf. sogar mit AngularJS auf ein Minimum zu beschränken, sowie Fehler zu identifizieren und korrigieren.

Eigenständige Hilfeseite: Denkbar wäre neben Tutorial 0 der Einbau einer eigenen Hilfeseite zum Editor, um unerfahrenen Nutzern den Einstieg in den EZWEB-Editor zu erleichtern.

Automatiserte Lösungsüberprüfungen der Tutorialaufgaben: Eine automatisierte Lösungsüberprüfung der Tutorialaufgaben würde das Benutzererlebnis verbessern. Hierzu kann ein geeignetes Konzept entwickelt werden.

Korrekturen: Das Parapgraph-Tag „<p>“ sollte ein umschließendes Tag sein, da innerhalb dieses Tag weitere Tags wie fett oder kursiv hinein gepackt werden können, dies ging als Anregungen von außerhalb hervor.

Anhang

Umfrage - Konkrete Fragestellungen

1.) Bitte geben Sie Ihr Geschlecht an:

- weiblich
- männlich

2.) Wie Alt sind Sie?

- jünger als 18 Jahre
- 18-24 Jahre
- 25-29 Jahre
- 30-35 Jahre
- älter als 35 Jahre

3.) Wie lange haben Sie sich in etwa mit dem Editor auseinander gesetzt?

- weniger als 15 Min.
- 15-30 Min.
- 30-45 Min.
- mehr als 60 Min.

4.) Hatten Sie Vorkenntnisse in der Erstellung von Webseiten? (10 = Experte)
1-10

5.) Haben Sie sich gut zurecht gefunden? (10 = sehr intuitiv)
1-10

6.) Haben Ihnen die Tutorials geholfen?

- Ja, sehr.
- Ja, ein wenig.
- Nein, nicht so richtig.
- Nein, überhaupt nicht!
- Keine Tutorials verwendet.

7.) Fühlen Sie sich nun sicherer im Umgang oder haben neue Kenntnisse gewonnen?

- Ja.
- Weiß nicht.
- Nein.
- Bereits Experte gewesen!

8.) Welche Art von Editor bevorzugen Sie? (Mehrere Auswahlmöglichkeiten)

- Code-Editor
- Drag & Drop (EZWEB)
- WYSIWYG (What you see is what you get.)
- Website Builder
- In-Site-Editing

9.) Hat Sie das Konzept überzeugt?

- Ja.
- Nein.
- Ausbaufähig.
- Enthaltung.

10.) Ich freue mich hier über Verbesserungsvorschläge und Kommentare. (max. 990 Zeichen)
→ Antwort als Text.

Glossar

CBK : Cross-Browser-Kompatibilität bezeichnet die Fähigkeit von Inhalten und Designkonzepten in Webseiten, unabhängig vom jeweiligen Betriebssystem und Browsertyp immer als identische Ausgabe angezeigt zu werden, Seite 15

CSS : Cascading Style Sheets, Seite 3

HTML : Hypertext Markup Language, Seite 2

iFrame : Ein iFrame oder auch Inlineframe ist ein zur Strukturierung verwendetes HTML-Element. Es wird eingesetzt, um andere Webinhalte als selbständige Dokumente in einem definierten Bereich des Browsers anzuzeigen, Seite 25

JS : JavaScript, Seite 43

NFA : Nicht-funktionale Anforderungen, Seite 28

PBL : Problembasiertes Lernen, Seite 11

PfNP : Programming for Non-Programmers, Seite 6

php : Hypertext Preprocessor, Seite 9

WYSIWYG : What You See Is What You Get, Seite 3

Literaturverzeichnis

[1] : *Bundesregierung | Bulletin | Rede von Bundeskanzlerin Dr. Angela Merkel zur Eröffnung der 30. Medientage am 25. Oktober 2016 in München:*. – URL https://www.bundesregierung.de/Content/DE/Bulletin/2016/10/124-1-bk-medientage.html. – Zugriffsdatum: 2017-10-27

[2] : *Dateiendung: Von 001 bis ZIP*. – URL http://www.chip.de/artikel/Dateiendung-Von-001-bis-ZIP_42524854.html. – Zugriffsdatum: 2017-09-08

[3] : *Dateiendung.com - Wie öffnet man Dateien?*. – URL http://www.dateiendung.com/. – Zugriffsdatum: 2017-09-08

[4] : *Endungen.de - Weisst du wie's endet? - Das Dateiendungen-Nachschlagewerk im Internet*. – URL http://www.endungen.de/. – Zugriffsdatum: 2017-09-08

[5] : *Flow-based programming*. – URL https://en.wikipedia.org/wiki/Flow-based_programming. – Zugriffsdatum: 2017-10-26

[6] : *Leaders and trend-setters all agree on one thing*. – URL https://code.org/quotes. – Zugriffsdatum: 2017-10-27

[7] : *Logo (Programmiersprache)*. – URL https://de.wikipedia.org/wiki/Logo_(Programmiersprache). – Zugriffsdatum: 2017-10-01

[8] : *Programming*. – URL http://stevejobsdailyquote.com/2013/09/24/programming/. – Zugriffsdatum: 2017-10-27

[9] : *Skandinavier sprechen sehr gut English: warum?*. – URL https://polyglotclub.com/help/language-learning-tips/scandinavians-good-english/translate-german. – Zugriffsdatum: 2017-10-11

[10] : *Smartphones - Prognose Absatz bis 2021 | Statistik*. – URL https://de.statista.com/statistik/daten/studie/12865/umfrage/prognose-zum-absatz-von-smartphones-weltweit/. – Zugriffsdatum: 2017-10-05

[11] : *Smartphones - Prognose zu den weltweiten Anschlüssen bis 2020 | Statistik*. – URL https://de.statista.com/statistik/daten/studie/312258/umfrage/weltweiter-bestand-an-smartphones/. – Zugriffsdatum: 2017-10-05

[12] : *Snap! (BYOB)*. – URL https://de.wikipedia.org/wiki/Snap!_(BYOB). – Zugriffsdatum: 2017-10-03

[13] : *Webseiten - Anzahl weltweit 2015 | Statistik*. – URL https://de.statista.com/statistik/daten/studie/290274/umfrage/anzahl-der-webseiten-weltweit/. – Zugriffsdatum: 2017-10-06

[14] Heckendorf, Katharina: Programmieren: Ist das die Zukunft? | ZEIT Campus. . – URL http://www.zeit.de/campus/2017/02/programmieren-digitalisierung-arbeitsmarkt-berufschancen. – Zugriffsdatum: 2017-10-01. – ISSN 0044-2070

[15] http://helllicht.com, helllicht medien GmbH : *Statistiken - Responsive Webdesign*. – URL http://be-responsive.de/statistiken/. – Zugriffsdatum: 2017-10-10

[16] Jatzlau, Sven ; Romeike, Ralf: Herausforderungen durch neue Programmierkonzepte in blockbasierten Programmiersprachen. In: *Lecture Notes in Informatics (LNI)* 11 (2017), S. 1

[17] Koh, Gerald Choon-Huat ; Khoo, Hoon E. ; Wong, Mee L. ; Koh, David: The effects of problem-based learning during medical school on physician competency: a systematic review. 178 (2008), Nr. 1, S. 34–41. – ISSN 0820-3946, 1488-2329

[18] Krampen, Günter ; Zayer, Hermann: *Didaktik und Evaluation in der Psychologie*. Hogrefe Verlag, 2006. – 245 S. – ISBN 978-3-8409-1984-8

[19] Marron, Assaf ; Weiss, Gera ; Wiener, Guy: A Decentralized Approach for Programming Interactive Applications with JavaScript and Blockly. In: *Proceedings of the 2Nd Edition on*

Programming Systems, Languages and Applications Based on Actors, Agents, and Decentralized Control Abstractions. New York, NY, USA : ACM, 2012 (AGERE! 2012), S. 59–70. – URL http://doi.acm.org/10.1145/2414639.2414648. – ISBN 978-1-4503-1630-9

[20] ONLINE, heise: *Mobile Internetnutzung weiter auf dem Vormarsch*. – URL https://www.heise.de/ho/meldung/Mobile-Internetnutzung-weiter-auf-dem-Vormarsch-3455624.html. – Zugriffsdatum: 2017-10-10

[21] ONLINE, Statista: *Smartphones - Prognose Absatz bis 2021 | Statistik*. – URL https://de.statista.com/statistik/daten/studie/12865/umfrage/prognose-zum-absatz-von-smartphones-weltweit/. – Zugriffsdatum: 2017-10-10

[22] REUSSER, Kurt: Problemorientiertes Lernen.- Tiefenstruktur, Gestaltungsformen, Wirkung. In: *Beiträge zur Lehrerinnen- und Lehrerbildung* 23 (2005), Nr. 2

[23] SCHIFFER, Stefan: *Visuelle Programmierung - Grundlagen und Einsatzmöglichkeiten*. Addison Wesley Verlag, 2001. – 12–13 S. – ISBN 978-3827312716

Abbildungsverzeichnis

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 1. Dezember 2017 Henning Kahl