

Eingereicht von
Alexander Wolf

Angefertigt am
**Institut für
Wirtschaftsinformatik –
Software Engineering**

Beurteiler / Beurteilerin
Dr. Wolfgang Narzt

Februar 2023

WEBBASIERTE ENTWICKLUNGSUMGEBUNG



Bachelorarbeit

zur Erlangung des akademischen Grades

Bachelor of Science

Im Bachelorstudium

Wirtschaftsinformatik

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Bachelorarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die vorliegende Bachelorarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, 28.02.2023

Unterschrift

Inhaltsverzeichnis

1. Abstract.....	5
2. Einleitung	6
3. Architektur von webbasierten Entwicklungsumgebungen.....	6
3.1. Häufig verwendete Technologien	6
3.2. Auslagerung des Kompilierens und der Ausführung mit Hilfe einer Code Execution Engine	7
3.3. Möglichkeit einer Implementierung mit Hilfe von WebAssembly	8
3.4. Relevante erweiterte Funktionalitäten für Online-IDEs.....	10
3.4.1. Asynchrone Ausgabe	11
3.4.2. Annahme von Input während der Laufzeit.....	11
3.4.3. Annahme von Command Line Argumenten	11
3.4.4. Sicherheit.....	11
3.4.5. Inkludieren von Ressourcen	11
3.4.6. Graphische Ausgabe	12
3.4.7. Einbindung mehrerer Source-Code Dateien	12
3.4.8. Debugger.....	12
3.4.9. Autovervollständigung	12
3.4.10. Unterstützung mehrerer Programmiersprachen.....	13
3.4.11. Automatische statische Code-Überprüfung.....	13
3.4.12. Kollaboration	13
4. Bestehende webbasierte Entwicklungsumgebungen	13
4.1. Liste bestehender Online-IDEs.....	14
4.2. Weitere erwähnenswerte Online-IDEs	14
4.3. Überprüfung der Online-IDEs auf erweiterte Funktionalitäten.....	15
4.3.1. Asynchrone Ausgabe	15
4.3.2. Annahme von Input während der Laufzeit.....	16
4.3.3. Annahme von Command Line Argumenten	17
4.3.4. Sicherheit.....	18
4.3.5. Inkludieren von Ressourcen	22
4.3.6. Graphische Ausgabe	23
4.3.7. Einbindung mehrerer Source-Code Dateien	24
4.3.8. Debugger.....	26
4.3.9. Autovervollständigung	26
4.3.10. Unterstützung mehrerer Programmiersprachen.....	28

4.3.11. Automatische statische Code-Überprüfung.....	28
4.3.12. Kollaboration	29
4.4. Vergleich der Online-IDEs	29
5. Eigene Implementierung einer webbasierten Entwicklungsumgebung.....	30
5.1. Verwendete Technologien.....	31
5.2. Architektur der Entwicklungsumgebung	31
5.2.1. Backend.....	31
5.2.2. Frontend	32
5.3. Schwierigkeiten bei Implementierung.....	33
5.3.1. Klassenkomponenten vs. Funktionskomponenten in React.....	33
5.3.2. Übertragung des Source-Codes.....	34
5.3.3. Asynchrone Ausgabe des Outputs.....	35
6. Offene Fragen.....	35
7. Konklusion	35
8. Appendix	36
8.1. Link zu Code-Samples	36
8.2. Detaillierte Ergebnisse der Technologie-Analyse.....	37
9. Referenzen	38
10. Abbildungsverzeichnis	40
11. Verzeichnis der Code-Snippets	40
12. Tabellenverzeichnis	40
13. Abkürzungsverzeichnis	41

1. Abstract

Webbasierte Entwicklungsumgebungen erleichtern den Einstieg in die Programmierung und können in einer Vielzahl von Bereichen nützlich sein. Die Implementierung einer derartigen Entwicklungsumgebung erweist sich jedoch als nicht triviale Aufgabe, für welche es nur wenige bereits ausgearbeitete Lösungsansätze gibt. Es existiert zudem nur wenige wissenschaftliche Literatur, welche die genauen Funktionsweisen und Konzepte hinter webbasierten Entwicklungsumgebungen beschreibt. Ziel dieser Bachelorarbeit war es, theoretische Konzepte und Architekturen von webbasierten Entwicklungsumgebungen aufzuarbeiten und sich darüber hinaus mit bestehenden Web-IDEs auseinanderzusetzen. Die Analyse von bestehenden Online-IDEs beleuchtet die umfassenden Unterschiede in deren Funktionalität genauer und zeigt mögliche Sicherheitslücken auf. Die theoretische Auseinandersetzung mit möglichen Architekturstilen ergab, dass bei der Implementierung zwischen zwei Architekturen gewählt werden kann. In dieser Arbeit wird ein dritter neuartiger Architekturansatz einer webbasierten Entwicklungsumgebung mit Hilfe von WebAssembly vorgestellt, welcher Sicherheitsrisiken und Rechenlast reduziert. In der Ausarbeitung der eigens implementierten webbasierten Entwicklungsumgebung, welche auf den vorhergehenden theoretischen Erkenntnissen aufbaute, ging hervor, dass nur wenige Frameworks oder Lösungsansätze diesen Prozess erleichtern. Zusätzlich wird auf gängige Funktionsanforderungen und deren Umsetzungsmöglichkeiten eingegangen.

2. Einleitung

Webbasierte Entwicklungsumgebungen bieten den Vorteil, dass keine Installation von etwaigen Compilern und lokal laufenden Entwicklungsumgebungen notwendig ist, um mit dem Programmieren zu starten. Darüber hinaus kann, wenn gewünscht, die Rechenlast mit Hilfe einer webbasierten Entwicklungsumgebung auf externe Server ausgelagert werden.

Es bestehen eine Vielzahl von Services, die eine derartige IDE im Web anbieten. Diese unterscheiden sich jedoch gänzlich in ihrer Schnelligkeit, im Umfang ihrer Funktionen und in deren Sicherheit. In Kapitel 4. möchte ich einige dieser Services näher beleuchten und auf die Unterschiede eingehen, um schlussendlich einen Vergleich zu ermöglichen.

Auch im Kontext des schulischen und universitären Umfelds wurden die Vorteile der Verwendung einer webbasierten IDE bereits bewiesen [1]. Für Student*innen ohne vorherige Programmiererfahrung stellt das erstmalige Einrichten einer Entwicklungsumgebung eine Herausforderung dar. Eine Vereinheitlichung der verwendeten Tools würde eine Erleichterung für eine Vielzahl von Student*innen bedeuten. Darüber hinaus soll das Ziel der einführenden Lehrveranstaltungen mit Programmiercharakter nicht die Auseinandersetzung mit der Installation verschiedener Entwicklungswerkzeuge oder Compilern sein. Genauso wäre es für Lehrende von Vorteil, eine standardisierte webbasierte Entwicklungsumgebung einzusetzen, weil dadurch nahezu kein technischer Support mehr notwendig sein dürfte [2].

Die Forschungsfragen für diese Bachelorarbeit sind demnach wie folgt:

RQ1: Wie sehen die grundsätzlichen Konzepte von webbasierten Entwicklungsumgebungen aus?

RQ2: Existieren bereits Frameworks oder Lösungsansätze die die prototypische Umsetzung von webbasierten Entwicklungsumgebungen erleichtern?

RQ3: Inwiefern unterscheiden sich bestehende webbasierte Entwicklungsumgebungen bezüglich ihrer Funktionalität?

Diese Arbeit wird sich nicht weiter mit diversen Sicherheitskonzepten für Online-Compiler auseinandersetzen. Für Informationen dahingehend verweise ich auf [3] und [4]. In den Kapitel 2.3.4., 3.2.4. und 3.3 wird lediglich veranschaulicht, dass Sicherheit, Sandboxing und das Limitieren von Ressourcen für die Betreiber von webbasierten Entwicklungsumgebungen kritische Rollen spielen.

Die Begriffe *webbasierte Entwicklungsumgebung*, *Online-IDE*, *Web-IDE*, *Online-Compiler*, *Web-Compiler* werden synonym verwendet.

3. Architektur von webbasierten Entwicklungsumgebungen

3.1. Häufig verwendete Technologien

Im Zuge dieses Kapitels wurden mehrere verschiedenen webbasierte Entwicklungsumgebungen untersucht und deren Funktionsweise analysiert. Damit soll ein grundsätzliches Verständnis für die üblicherweise umgesetzte Architektur der Web-IDEs geschaffen werden. Die Liste der analysierten Online-IDEs wird in 4.1. angeführt. Um im ersten Schritt herauszufinden, welche

Technologien diese Services verwenden, wurden die folgenden öffentlich zugänglichen Analyse-Tools verwendet:

- builtWith [5]
- W3Techs [6]
- Wappalyzer [7]

Die Ergebnisse der Tools unterscheiden sich vereinzelt in der Granularität. Beispielsweise wird mit W3Techs bei einem NodeJS-Backend nur erkannt, dass es sich um JavaScript im Backend handelt, nicht aber, um welche konkrete Laufzeitumgebung. Im Folgenden sind die Ergebnisse dieser Analyse zusammengefasst. Die detaillierten Ergebnisse sind im Appendix, Kapitel 8.2., zu finden.

In allen untersuchten Online-IDEs wurde entweder JavaScript oder PHP im Backend verwendet. Auch im Frontend wurde vermehrt ein JavaScript-Framework, wie beispielsweise React, Vue oder Angular verwendet. Vor allem interessant waren die Ergebnisse von Wappalyzer [7], wo unter anderem gezeigt wurde, welche Text-Editoren die Online-IDEs verwenden. 5 von 10 Services verwenden zum Zeitpunkt der Analyse den ACE-Editor [8], während nur 2 von 10 den, auch von Visual Studio Code [9] eingesetzten, Monaco Editor [10] verwenden. Die Mehrheit der Web-IDEs verwenden im Frontend Bibliotheken wie jQuery und Bootstrap.

3.2. Auslagerung des Kompilierens und der Ausführung mit Hilfe einer Code Execution Engine

Judge0 [11] ist die einzige der zehn untersuchten webbasierten Entwicklungsumgebungen, welche open-source ist und damit der Code auf GitHub einsehbar ist. Darüber hinaus verwendet diese Online-IDE für die Kompilierung und Ausführung des im Browser geschriebenen Programms die gleichnamige Judge0 CE API [12]. Damit wird der sicherheitskritische Prozess der Ausführung von willkürlichen Programmen an das Code Execution System ausgelagert. Die API unterstützt die Kompilierung von über 60 verschiedenen Programmiersprachen und verwendet im Hintergrund den in [4] präsentierten *isolate* Sandboxing-Mechanismus.

Nachteil der Verwendung dieser API ist, dass keine asynchrone Rückgabe von Output während der Laufzeit möglich ist. Dies bedeutet, dass nur Programme an die API gesendet werden können, welche mit Sicherheit in der von der API vorgegebenen Zeit zu einem Ergebnis kommen. Die Ausführung von diversen Sortieralgorithmen in Kombination mit einem hohen n , wird somit von der kostenpflichtigen API nicht angeboten. Damit wird nicht nur die Ausführung von Programmen mit langer Laufzeit unterbunden, sondern auch der Output bis zum Zeitpunkt des Timeouts dem Nutzer vorbehalten. In Kapitel 3.4.1. werde ich noch näher auf die Wichtigkeit der Asynchronität eingehen. Es ist anzunehmen, dass diese Funktionalität bewusst nicht angeboten wird, da dadurch die Sicherheit der API selbst und des Benutzers leidet.

Abhilfe schafft der Umstand, dass Judge0 open-source ist und bei Bedarf selbst gehostet und modifiziert werden kann. Nimmt man von dieser Möglichkeit Gebrauch, könnte man die ursprüngliche Zeitlimitierung entfernen. Es besteht damit auch die theoretische Möglichkeit, das Projekt um eine mögliche asynchrone Ausgabe zu erweitern. Dies könnte jedoch zu Einbußen im Thema Sicherheit führen.

Auch dieses Code Execution System verwendet als Fundament verschiedene Command Line Compiler und Interpreter. Zu sehen ist dies in [13]. In diesem System werden die Kommandos zur Kompilierung und Ausführen in einem JSON-artigen Objekt abgespeichert und bei Aufruf der API ausgeführt.

Bei der Verwendung von Code Execution Engines muss zusätzlich bedacht werden, dass bei Ausfall des externen Services praktisch keine Funktionalität im eigenen System mehr angeboten werden kann. Im Idealfall könnte ein Ersatzsystem, obgleich intern oder wiederum extern, Abhilfe schaffen, um mögliche Downtime zu verhindern.

3.3. Möglichkeit einer Implementierung mit Hilfe von WebAssembly

WebAssembly [14] ist ein binäres Instruktionsformat für virtuelle Maschinen und damit geeignet, in allen gängigen Browsern verwendet zu werden. WebAssembly ist als portables Kompilierungsziel konzipiert und bietet somit die Möglichkeit, non-JavaScript Code zu kompilieren und in einer Browser-Umgebung auszuführen. Im Kontext von webbasierten Entwicklungsumgebungen bietet WebAssembly den Vorteil, dass der Betreiber nur den Kompilierprozess übernimmt und das kompilierte Code-Modul an den Benutzer zurücksendet. Dadurch, dass der Code nicht mehr am eigenen bereitgestellten Server ausgeführt wird, sondern in der Browserumgebung des Benutzers, kann auf eine Vielzahl von - normalerweise unbedingt notwendigen - Sicherheitskonzepten verzichtet werden. Dies minimiert wiederum die Entwicklungszeit und den Aufwand des Betreibers. Darüber hinaus wird auch die Rechenlast dem Benutzer auferlegt, was zu einer Reduktion der Kosten auf Betreiberseite führt.

Die klare Überlegenheit von WebAssembly gegenüber JavaScript - bezogen auf die Schnelligkeit - ist in der Literatur noch umstritten. Wang [15] zeigt, dass WebAssembly nur bei Programmen mit kleinem Input signifikant schneller ist als natives JavaScript. Im Test diverser Algorithmen überholt JavaScript WebAssembly im Durchschnitt ab einem Input von $n > 40$. Dies liegt unter anderem an der nicht vorhandenen Gargabe Collection in WebAssembly. [15] zeigt auch, dass im Falle von WebAssembly der benutzte Speicher bei wachsendem Input linear ansteigt, während der benutzte Speicher bei der Verwendung von JavaScript für alle n konstant ist.

Es ist zu bedenken dass, im Gegensatz zu einer gewöhnlichen webbasierten Entwicklungsumgebung, hier der gesamte notwendige Source-Code über das Netzwerk versendet werden muss und nicht nur der Output des Programms in einem Text-Format. Zur Überprüfung des Unterschieds der Response-Größen wurde zwei Szenarien simuliert in dem eine entsprechende Web-IDE als Output „Hello World!“ zurückliefert. Im ersten Szenario gibt es

```

1 const express = require('express');
2 const app = express();
3 const port = 3000;
4
5 app.get('/', (req, res) => {
6   res.send({ output: 'Hello World!' });
7 })
8
9 app.listen(port, () => {

```

Abbildung 1 Beispielhaftes NodeJS/Express Backend

im Backend eine simple REST-Schnittstelle, mit der man den Output eines Programms im JSON-Format abfragen kann. Abbildung 1 zeigt den Beispielcode für dieses Szenario.

Im zweiten Szenario muss der Betreiber den eingegebenen Code des Benutzers etwas adaptieren, um zum gewünschten Ergebnis zu gelangen. Zur Vereinfachung dieses Beispiels wurde auch davon ausgegangen, dass kein Echtzeit-Output notwendig ist. Der Benutzer gibt am Ende des Programms das Ergebnis in Form eines *String Structs* zurück. Abbildung 2 und 3 zeigen den benötigten Code. Natürlich können beide Varianten der Ausführung mit einem Frontend-Framework kombiniert werden.

```

1 use wasm_bindgen::prelude::*;
2
3 #[wasm_bindgen]
4 pub fn user_code() -> String {
5     return "Hello World!".to_string();
6 }

```

Abbildung 3 Rust Code mit WebAssembly Annotation

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8">
5     <title>WebAssembly Example</title>
6     <script type="module">
7         import init, { user_code } from './pkg/dom.js';
8
9         async function start() {
10             await init();
11             const result = user_code();
12             document.getElementById('output').textContent = result;
13         }
14
15         start();
16     </script>
17 </head>
18 <body>
19     <p id="output"></p>
20 </body>
21 </html>

```

Abbildung 2 HTML-Dokument in welchem Rust Code aufgerufen wird

Die Annotation über der Funktion *user_code* ermöglicht die Kompilierung in ein binäres Format, welches von JavaScript aufgerufen werden kann. Der Web-Server (in diesem Szenario wurde ein Python HTTP Server verwendet) liefert das HTML-Dokument samt WebAssembly Code an den Benutzer. Das Ergebnis der *user_code* Funktion wird anschließend in das jeweilige Output Fenster geschrieben.

In Szenario 1 hatte die gesamte JSON-Response eine Größe von 25 Bytes, während in Szenario 2 die gesamte Größe der Antwort 17,80kB betrug. Bei der Verwendung von WebAssembly werden also, im Kontext von Web-IDEs, ungefähr das Tausendfache an Daten versendet im Vergleich zum Versenden des Outputs im JSON-Format. Falls Asynchronität bzw. Echtzeit Anzeige des Outputs gewünscht ist, müsste man noch die durchschnittliche Laufzeit

des Programms und die Intervallzeit, in der der Output vom Backend aus Szenario 1 abgefragt wird, berücksichtigen. Es müsste weiter untersucht werden, inwiefern die Datengröße bei steigender Komplexität und Umfang eines Programms wächst. Möchte man die Netzwerkbelastung geringhalten, ist man besser beraten, die herkömmliche Herangehensweise anzuwenden. Auch im Sinne der Nachhaltigkeit wäre eine serverseitige Kompilierung- und Ausführung zu bevorzugen.

Zurzeit unterstützen alle namhaften Browser-Engines (Mozilla Firefox, Google Chrome, Safari, Microsoft Edge) den Einsatz von WebAssembly. Ein weiteres Problem liegt in der derzeit noch niedrigen Anzahl an kompilierbaren Programmiersprachen. Für beliebte Programmiersprachen wie Java oder Python, gibt es im Moment nur experimentelle Projekte ohne offizielle Unterstützung.

Auch die Unterstützung mehrerer Programmiersprachen könnte in Kombination mit WebAssembly problematisch sein. Beispielweise sind bei Swift zusätzliche Tools notwendig, um die produzierte *.wasm* Binärdatei im Browser ausführen zu können [16]. Geht man davon aus, dass jede Programmiersprache andere Tools benötigt und anders verarbeitet werden muss, um zu einem fertigen WebAssembly Modul zu gelangen, steigt die Komplexität der webbasierten Entwicklungsumgebung rasant. Das Ausführen im Browser stellt hingegen, unabhängig der originären Programmiersprache, kein Problem dar. Die kompilierten Funktionen können immer auf dieselbe Art und Weise, wie es in Abbildung 2 zu sehen ist, von JavaScript importiert und aufgerufen werden.

Ist es Ziel, eine webbasierte Entwicklungsumgebung zu entwickeln, welche nur eine einzige Programmiersprache unterstützt, kann die Verwendung von WebAssembly in Erwägung gezogen werden. Die Reduktion der eingesetzten Sicherheitsmaßnahmen minimiert den Entwicklungsaufwand weitgehend. Auch wenn nur der Kompilierprozess in der Verantwortung des Betreibers liegt, sollten Sicherheitsvorkehrungen nicht vernachlässigt werden. Es besteht nach wie vor eine geringe Chance, dass auch schon während des Kompilierprozesses schadhafter Code ausgeführt wird. Vor allem sollte darauf geachtet werden, dass keine Programmiersprachen angeboten werden, die bereits zur Compile-Zeit Code ausführen oder interpretieren. Auch die zur Verfügung gestellte Zeit im Rahmen der Kompilierung sollte begrenzt werden. Letztendlich bietet es sich auch hier an, das Kompilieren nur in einer Docker Umgebung durchzuführen.

Inklusive WebAssembly gibt es schlussendlich drei Varianten, wie eine webbasierte Entwicklungsumgebung konzipiert sein kann. Grundsätzlich unterscheiden sich die Varianten im Ort der Ausführung des vom Benutzer gesendeten Codes. Die Kompilierung geschieht immer auf dem Server des Betreibers, mit Ausnahme beim Einsatz einer Code Execution Engine, bei welcher der Code wiederum an eine dritte Partei weitergeleitet wird.

3.4. Relevante erweiterte Funktionalitäten für Online-IDEs

Diese Funktionalitäten sind unabhängig von der Programmiersprache für Benutzer nützlich und sollten im Idealfall umgesetzt werden. Wie jedoch später noch dargestellt wird, werden diese Funktionalitäten nur vereinzelt von bereits bestehenden webbasierten Entwicklungsumgebungen umgesetzt. Grund dafür ist oftmals eine schwierige Umsetzung im Rahmen des Browsers. Zum Beispiel werden für graphische Ausgaben fast ausschließlich Betriebssysteme-spezifische

Bibliotheken benutzt, welche nicht in einer Browserumgebung ausgeführt werden können. Auch die Implementierung eines Debuggers mit umfassender Funktionalität, wie beispielsweise *Code Stepping*, würde extensiven Entwicklungsaufwand bedeuten. Die Reihung der folgenden Kapitel ist nicht mit einer Wichtigkeit für Online-IDEs gleichzusetzen. Die Beschreibungen der Funktionalitäten enthalten mehrmals einen Bezug auf das universitäre Umfeld; dies soll nur als Beispiel dienen, denn die Funktionen sind - ungeachtet des Kontexts - hilfreich für alle Benutzer.

3.4.1. Asynchrone Ausgabe

Um eine angenehme UX zu garantieren, muss davon ausgegangen werden, dass ein Programm schon während der Laufzeit wichtigen Output generieren kann. Viele der untersuchten Online-IDEs versuchen jedoch, das eingegebene Programm bis zum Ende laufen zu lassen und danach den gesamten produzierten Output an das Frontend und somit den Benutzer weiterzugeben.

Bei langen Kalkulationszeiten oder bei Programmen, in denen eine regelmäßige Ausgabe für den Benutzer essenziell ist, ist es vorteilhaft, eine asynchrone Ausgabe (während das Programm noch läuft) bereitzustellen. In Kapitel 5. gehe ich darauf ein, wie diese asynchrone Ausgabe implementiert werden kann.

3.4.2. Annahme von Input während der Laufzeit

Für viele Programmieranfänger beinhalten die ersten Programme einen Lese-Mechanismus, in dem der Benutzer beispielsweise eine Zahl während der Laufzeit eingibt und das Programm mit dieser weiterarbeitet. Diese interaktive Art der Kommunikation von Browser zurück zum Terminal wird nur bei wenigen Online-IDEs ermöglicht.

3.4.3. Annahme von Command Line Argumenten

Die meisten Programmiersprachen bieten vielerlei Optionen für ihren Command Line Compiler als auch für ihren Command Line Application Launcher (siehe beispielsweise *javac* [17] und *java* [18]). Die Möglichkeit einer Mitgabe von Flags und Optionen ist für fortgeschrittene Benutzer von Bedeutung und sollte implementiert werden. Dass die meisten bestehenden webbasierten Entwicklungsumgebungen diese Funktionalität bereits umsetzen, ist ein Indiz dafür, dass im Grunde jede Online-IDE auf standardmäßigen Command Line Compilern und Interpretern basiert.

3.4.4. Sicherheit

Im Zuge dieser Arbeit werde ich nur kurz auf diesen wichtigen Aspekt im Kontext von Online-Compilern eingehen. Online-Compiler bieten eine große Angriffsfläche für Benutzer mit böswilligen Absichten. Aus diesem Grund muss während des Kompilierprozesses wie auch während des Ausführens Sandboxing als Schutzmaßnahme eingesetzt werden. Es gibt eine Vielzahl von Arten des Sandboxings, welche etwa auf UNIX System Aufrufen wie *ptrace* [19] oder Konzepten wie *isolate* [4] basieren.

3.4.5. Inkludieren von Ressourcen

Eine gängige Praxis ist das Einlesen von Input aus einer Datei, wie beispielsweise einer CSV-Datei. Diese Funktion bietet dem Benutzer noch mehr Möglichkeiten im Rahmen der webbasierten Entwicklungsumgebung zu arbeiten, ohne auf eine lokale Umgebung wechseln zu müssen.

3.4.6. Graphische Ausgabe

Graphische Programme sind ein fester Bestandteil jedes Coding-Curriculums. Klar ist, dass ausgehend von einer Web-IDE kein nativer graphischer Output, i.e. Output, der nicht im Browser angezeigt wird, an den Benutzer weitergegeben werden kann. Die Frage, ob es möglich ist, vom Backend aus einzelnen Frames an den Browser zu senden, welcher wiederum den Output für den Benutzer darstellt, bleibt offen. Angenommen das Backend der webbasierten Entwicklungsumgebung ist ein headless VPS, so müsste man mit Hilfe von VNC eine virtuelle Oberfläche schaffen, auf welcher etwa Swing-Programme angezeigt werden können.

3.4.7. Einbindung mehrerer Source-Code Dateien

Will man, dass ein vollkommener Umstieg von lokaler Entwicklung auf webbasierte Entwicklung erfolgt, so ist die Möglichkeit einer Einbindung von mehreren Source-Code Dateien ein Muss. Diese Funktionalität korreliert mit der Einbindung diverser Ressourcen, i.e. non Source-Code Dateien. Schlussendlich sollte es im Idealfall möglich sein, gesamte Projekte ausführen zu können. Im Umkehrschluss bedeutet das, dass ein Filesystem im Browser abgebildet werden muss und in Folge manipuliert werden kann.

Um ein möglichst user-freundliches Verhalten zu gewährleisten, muss, unter Umständen, der Code analysiert werden, bevor er kompiliert und ausgeführt werden kann. Im Falle von Java müsste herausgefunden werden, in welcher Klasse sich die *main*-Methode befindet. Wird tatsächlich ein Filesystem im Browser abgebildet, muss auch in der Kompilierung darauf geachtet werden, dass alle Dateien laut *Classpath* gefunden werden können.

3.4.8. Debugger

Die schnellste Art und Weise wie Fehler aus dem Code entfernt werden können, ist die Verwendung eines Debuggers. Während viele Programmierer zu einfachen Print Statements greifen, um Fehler aufzuspüren, ist ein Einblick in den Zustand von etwaigen Variablen nicht möglich. Die Einbindung eines Debuggers, welcher genauso wie die Autovervollständigung, spezifisch für eine einzige Programmiersprache konzipiert ist, ist durchaus mit Aufwand verbunden. Aus [20] geht hervor, dass nur zwei der zehn analysierten Online-Compiler Debugging anbieten. Natürlich wäre hier noch weiter zu unterscheiden, welche Art von Debugging angeboten wird. Es ist nicht evident, ob überhaupt ein umfangreiches Debugging im Rahmen einer Web-IDE möglich ist. Zum Vergleich können hier Tools von JetBrains [21] herangezogen werden, in denen Code zeilenweise ausgeführt werden kann. Ausgenommen aus dieser Einschätzung ist der Cloud9 Service. Bei diesem handelt es sich vielmehr um einen VPS als um eine reine webbasierte Entwicklungsumgebung. Auf diesen wird in Kapitel 4.2. noch genauer eingegangen.

3.4.9. Autovervollständigung

Gegenwärtig ist ein Editor mit umfangreichen Unterstützungen und Autovervollständigung State of the Art für jeden Programmierer. Lediglich in Anfängerkursen wird manchmal auf Editoren mit diesen Funktionen verzichtet, um ein gewisses Verständnis der Grundlagen zu fördern. Im Rahmen von Microsoftprodukten wird auch von *IntelliSense* gesprochen. Um eine Autovervollständigung im Web zu garantieren, muss ein Language Server verwendet werden, mit welchem man gemäß des Language Server Protokolls [22] kommuniziert. Die Implementierung von Language Servern kann unter Umständen einen beachtlichen Teil des Entwicklungsaufwands ausmachen, weshalb oftmals auf diese Art der Autovervollständigung verzichtet wird und anstatt dessen nur lexikalische Vervollständigung angeboten wird. Bei

lexikalischer Vervollständigung werden immer nur Begriffe der gerade offenen Dateien vorgeschlagen. Importiert man zum Beispiel eine außerhalb der Datei liegende Klasse, wird keine Liste der möglichen Methoden angezeigt, nachdem man die Klasse instanziiert hat.

3.4.10. Unterstützung mehrerer Programmiersprachen

Je nach Einsatzkontext müssen unterschiedliche Programmiersprachen in der Online-IDE angeboten werden. Natürlich muss im Backend des Services der jeweilige Compiler der Programmiersprache installiert sein. Dies gilt nur, wenn es sich tatsächlich um eine kompilierte Sprache handelt und nicht um eine interpretierte Sprache, wie beispielsweise Python oder Bash. Dies geht einher mit dem im vorherigen Unterkapitel besprochenen Thema, Autovervollständigung bzw. IntelliSense. Für jede unterstützte Programmiersprache ist der verwendete Language Server um diese Sprache zu erweitern.

3.4.11. Automatische statische Code-Überprüfung

Die Mehrheit von Benutzern erwartet sich, nicht erst beim gewünschten Ausführungszeitpunkt über Fehler im Code informiert zu werden. Ein fehlendes Semikolon wird üblicherweise innerhalb von Sekunden von lokalen IDEs als Fehler bemängelt. Der einfachste Weg dieses Verhalten zu erreichen besteht darin, nach jeder Änderung oder in einem gewissen Zeitintervall, den aktuellen Code im Hintergrund zu kompilieren und etwaige Fehler direkt an das Frontend und damit den Benutzer weiterzuleiten. Das Problem bei dieser Herangehensweise bleibt, dass nur der erste Fehler, bei welchem der Compiler abbricht, angezeigt wird. Will man den gesamten Code statisch analysieren, müssen tiefere Tools herangezogen werden.

3.4.12. Kollaboration

Es kann oftmals praktisch sein, in einem informellen Setting oder in der Berufswelt, einer anderen Person in Echtzeit den aktuellen Code zu zeigen und in Folge mit ihm zu kollaborieren. Natürlich gibt es die Möglichkeit des Screen-Sharings, wobei Änderungen nur durch eine einzige Person vorgenommen werden können. Webbasierte Entwicklungsumgebungen bieten einen komfortablen Weg, wie gemeinsam an Code gearbeitet werden kann. Die Möglichkeit zur Zusammenarbeit beschränkt sich stets auf kleinere Projekte mit wenigen Source-Code-Dateien. Wie schon beschrieben wurde, ist es nicht das Ziel der meisten Web-IDEs, eine umfassende Codebasis zu unterstützen. Ist die Kollaboration in einem professionellen Setting gewünscht, kann auf inoffizielle Plugins diverser lokal installierte Entwicklungsumgebungen zurückgegriffen werden. In der Praxis wird die Kollaboration meistens über Versionsverwaltungssysteme wie *Git* [23] gelöst. Dabei wird zwar auf den Echtzeit-Aspekt verzichtet; dies ermöglicht aber eine genau Rückverfolgung aller Änderungen. In Kombination mit Plattformen wie GitHub [24] wird die Möglichkeit zur Kollaboration noch weiter verbessert.

4. Bestehende webbasierte Entwicklungsumgebungen

In diesem Kapitel möchte ich einige Services auswählen, welche webbasierte Entwicklungsumgebungen anbieten und dann demonstrieren, wie sich diese in Funktionalität und Sicherheit unterscheiden. Dieses Kapitel erweitert die Ergebnisse von Sinanaj *et al.* [20] und wird demnach auf dieselben Services eingehen. Sinanaj *et al.* [20] untersucht grundsätzliche Eigenschaften der verschiedenen Online-Compiler und im speziellen auch die Schnelligkeit in der Ausführung diverser Sortier-Algorithmen, wie beispielweise Bubble-Sort. Für eine kurze Beschreibung der verschiedenen Services sowie die unterschiedlichen Geschwindigkeiten der Tools verweise ich auf die Ausführungen und Ergebnisse in [20]. In [20] werden jedoch wichtige Funktionalitätseigenschaften sowie Sicherheitsaspekte außer Acht gelassen. Daher werde ich

im Folgenden noch auf diese Eigenschaften eingehen und die Services anhand jener vergleichen.

4.1. Liste bestehender Online-IDEs

Die nachfolgenden Online-IDEs werde ich genauer auf erweiterte Funktionalitäten, sofern sie in Kapitel 2.1. angeführt wurden, jedoch noch nicht in [20] genauer untersucht wurden, überprüfen.

- OnlineGDB [25]
- WandBox [26]
- Judge0 [11]
- Tutorialspoint [27]
- GeeksForGeeks [28]
- Ideone [29]
- Replit [30]
- JDoodle [31]
- Paiza.IO [32]
- CodeInterview [33]

4.2. Weitere erwähnenswerte Online-IDEs

Wichtig zu erwähnen sind die beiden proprietären webbasierten IDEs, AWS Cloud9 [34] und die Online-Version von Visual Studio Code [35]. Beiden bieten eine Fülle an Funktionen, unterscheiden sich jedoch in ihrer Architektur. Der kostenpflichtige Cloud9 Service bietet eine benutzerfreundliche Web-Oberfläche für einen darunter liegende EC2 Instanz an, die einen schnellen Start in der Entwicklung gewähren soll. Je nach Interpretation und Use Case trifft dieser Service den Sinn und Zweck einer webbasierten Entwicklungsumgebung entweder vollkommen oder gar nicht. Wer der Meinung ist, eine webbasierte Entwicklungsumgebung sollte einen einfachen und vor allem schnellen Einstieg in die Programmierung bieten, ohne dabei viel Wissen vorauszusetzen, wird sich nicht dem AWS Cloud9 Service anfreunden können. Wer hingegen eine webbasierte Entwicklungsumgebung als wirkliche Alternative zu lokalen Workflows sieht und davon ausgeht, dass die hauptsächliche Zielgruppe erfahrene Softwareentwickler sind, wird den angebotenen Service dankend in Anspruch nehmen. Cloud9 versucht ein lokale Entwicklungsumgebung bestmöglich zu imitieren und bietet dem Nutzer eine breite Palette an Funktionen. Wie gewohnt steigt mit dem Umfang der Funktionen auch die Komplexität des Systems. Programmieranfänger hätten auch hier wieder eine steilere Lernkurve und sollten auf simplere Web-IDEs zurückgreifen. Überraschenderweise verwendet der kostenpflichtige Service keine Language Server um dem Nutzer umfassenden IntelliSense, wie man ihn von lokalen IDEs gewohnt ist, zu bieten. Lediglich eine lexikalische Vervollständigung wird dem Benutzer beim Programmieren angeboten. Der Service verwendet, vermutlich aus markentechnischen Gründen, den ACE-Editor [8] und nicht den moderneren VS Code Editor des Konkurrenzunternehmens Microsoft.

Die Web-Version von Visual Studio Code ist an sich hingegen kostenfrei, greift aber auf das lokale Dateisystem zu. Darüber hinaus muss in einem sogenannten *Codespace* von Github gearbeitet werden, um Programme überhaupt kompilieren und ausführen zu können.

4.3. Überprüfung der Online-IDEs auf erweiterte Funktionalitäten

4.3.1. Asynchrone Ausgabe

Mit Hilfe dieses kurzen Code-Snippets kann überprüft werden, ob ein Online-Compiler eine asynchrone Ausgabe unterstützt oder nicht. Dieser Code wurde verwendet, um die in 3.1. angeführten Online-IDEs auf Unterstützung einer asynchronen Ausgabe zu testen.

```
import java.util.Timer;
import java.util.TimerTask;

public class AsyncResponseTest {

    public static void main(String[] args) {
        Timer timer = new Timer();
        timer.schedule(new TimerTask() {
            private int i = 0;

            @Override
            public void run() {
                System.out.println(++i);
            }

        }, 0, 1000);
    }
}
```

Code-Snippet 1 Testcode Asynchrone Ausgabe

Beim Testen der Web-IDEs auf die asynchrone Ausgabe kam ich zu folgenden Ergebnissen.

Name des Services	Asynchrone Ausgabe unterstützt	Verhalten des Services	Testdatum
OnlineGDB	Ja	Normale Ausführung	14.12.2022
WandBox	Nein	Weder Timeout-Fehlermeldung noch andere Reaktion. Erneutes Laden der Seite notwendig.	14.12.2022
Judge0	Nein	Timeout nach ≈ 20 Sekunden. Ausgabe bis zu Timeoutzeitpunkt wird an Benutzer gesendet.	14.12.2022
Tutorialspoint	Ja	Normale Ausführung	14.12.2022
GeeksForGeeks	Nein	Timeout nach ≈ 15 Sekunden. Keine Ausgabe.	14.12.2022
Ideone	Nein	Timeout nach ≈ 5 Sekunden. Ausgabe bis zu	14.12.2022

		Timeoutzeitpunkt wird an Benutzer gesendet.	
Replit	Ja	Normale Ausführung	14.12.2022
JDoodle	Nein	Timeout nach \approx 5 Sekunden. Ausgabe bis zu Timeoutzeitpunkt wird an Benutzer gesendet.	14.12.2022
Paiza.IO	Nein	Timeout nach \approx 2 Sekunden. Ausgabe bis zu Timeoutzeitpunkt wird an Benutzer gesendet.	14.12.2022
CodeInterview	Ja	Interaktiver Modus muss aktiviert werden. Einstellung ist sehr versteckt und noch in der Beta-Phase.	14.12.2022

Tabelle 1 Ergebnisse Test auf asynchrone Ausgabe

Die Services unterstützen teilweise verschiedene Version des JDK, jedoch dürfte dies keinen Effekt auf die Validität der Ergebnisse haben.

Alle der angeführten Tools bieten Compiler für eine Vielzahl von Programmiersprachen an. Es ist jedoch davon auszugehen, dass sich das Verhalten der Ausgabe bei Wechsel der Programmiersprache nicht ändern würde.

4.3.2. Annahme von Input während der Laufzeit

Mit Hilfe dieses kurzen Code-Snippets kann überprüft werden, ob ein gegebener Online-Compiler die Eingabe von Werten über das Frontend unterstützt. Dieser Code wurde verwendet, um die in 3.1. angeführten Online-IDEs auf Unterstützung dieser Funktionalität zu testen.

```
import java.util.Scanner;

public class InputDuringRuntimeTest {

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter a number:");
        String input = s.nextLine();
        System.out.println("You entered: " + input);
    }
}
```

Code-Snippet 2 Testcode Annahme von Input während der Laufzeit

Beim Testen der Web-IDEs bin ich zu folgenden Ergebnissen gekommen.

Name des Services	Input während der	Verhalten des Services	Testdatum
-------------------	-------------------	------------------------	-----------

	Laufzeit möglich		
OnlineGDB	Ja	Normale Ausführung	14.12.2022
WandBox	Nein	Keine Eingabe möglich. Programm versucht nicht vorhandenen Input zu lesen.	14.12.2022
Judge0	Nein	Keine Eingabe möglich. Programm versucht nicht vorhandenen Input zu lesen.	14.12.2022
Tutorialspoint	Ja	Normale Ausführung	14.12.2022
GeeksForGeeks	Nein	Keine Eingabe während Laufzeit möglich. Einmalige Eingabe vor dem Ausführen des Programms möglich.	14.12.2022
Ideone	Nein	Keine Eingabe während Laufzeit möglich. Einmalige Eingabe vor dem Ausführen des Programms möglich.	14.12.2022
Replit	Ja	Normale Ausführung	14.12.2022
JDoodle	Nein	Keine Eingabe während Laufzeit möglich. Einmalige Eingabe vor dem Ausführen des Programms möglich.	14.12.2022
Paiza.IO	Nein	Keine Eingabe während Laufzeit möglich. Einmalige Eingabe vor dem Ausführen des Programms möglich.	14.12.2022
CodeInterview	Ja	Option zur interaktiven Eingabe ist etwas versteckt und standardmäßig deaktiviert.	14.12.2022

Tabelle 2 Ergebnisse Annahme von Input während der Laufzeit

Eine Vielzahl jener Tools, die keine interaktive Eingabe von Inputs während der Laufzeit erlauben, erlauben es jedoch den *Stdin* vor dem Kompilieren und Ausführen des Programms zu befüllen.

4.3.3. Annahme von Command Line Argumenten

Zum Überprüfen dieser Funktionalität muss kein Code ausgeführt werden, sondern lediglich kontrolliert werden, ob eine Eingabe von Command Line Argumenten in der Web-IDE möglich ist. Eine Überprüfung zeigt nun folgende Ergebnisse.

Name des Services	Input von Command Line Argumenten möglich	Überprüfungsdatum
----------------------	----------------------------------------------------	-------------------

OnlineGDB	Ja	14.12.2022
WandBox	Ja	14.12.2022
Judge0	Ja	14.12.2022
Tutorialspoint	Nein	14.12.2022
GeeksForGeeks	Nein	14.12.2022
Ideone	Nein	14.12.2022
Replit	Ja	14.12.2022
JDoodle	Ja	14.12.2022
Paiza.IO	Nein	14.12.2022
CodeInterview	Nein	14.12.2022

Tabelle 3 Ergebnisse Annahme von Command Line Argumenten

4.3.4. Sicherheit

Für dieses Unterkapitel wurden keine ausführlichen Penetrationstests durchgeführt, da dies nicht der Fokus dieser Arbeit ist. Dennoch spielt die Sicherheit in diesem Themenbereich eine wesentliche Rolle und kann daher nicht vollkommen außer Acht gelassen werden. Mit zwei simplen Tests soll veranschaulicht werden, wie die verschiedenen Online-Compiler mit potenziellen Gefahren umgehen.

Fast jede Programmiersprache bietet die Möglichkeit an, auf die Shell zuzugreifen und beliebige Befehle auszuführen. Dies stellt eine enorme Gefahr für jegliche öffentlich zugänglichen Systeme dar, sodass Sicherheitsmaßnahmen notwendig sind. Im Grunde genommen gibt es drei Möglichkeiten, welche im Idealfall miteinander kombiniert werden sollten, um einen möglichst hohen Grad an Sicherheit zu erlangen.

Die erste Möglichkeit, welche nur von wenigen Online-Compilern umgesetzt wird, ist Bibliotheken oder Pakete der einzelnen Programmiersprachen zu sperren, falls diese ein inhärentes Risiko bergen, missbräuchlich genutzt zu werden. Dies ist jedoch nur ein Workaround, da beispielsweise keine externen Pakete in Java benötigt werden, um einen neuen Prozess mit beliebigen Befehlen zu starten. Dies wird im zweiten Test näher veranschaulicht.

Als zweite Möglichkeit bietet sich das Setzen von sicheren UNIX-Berechtigungen bzw. das Erstellen von sicheren ACLs. Alle der untersuchten webbasierten IDEs erlaubten *read* Access für die UNIX-Benutzergruppe *Other*, ergo dem Benutzer, jedoch keinen *write* Access.

Die dritte Möglichkeit, die sich als Sicherheitsmaßnahme anbietet, ist das Sandboxen der jeweiligen Entwicklungsumgebung bzw. eine Virtualisierung eines Servers. Häufig wird diese Virtualisierung mit Hilfe von Docker [36] umgesetzt. Der Betreiber des Services kann dabei die Ressourcen des virtuellen Servers beschränken. Damit minimiert der Betreiber auch gleichzeitig das Risiko der Ausnutzung der oftmals frei zur Verfügung gestellten Rechenkapazität. Es ist unklar, inwiefern und ob die Rechenleistung von gratis Online-IDEs ausgenutzt werden kann.

Der folgende Code wurde für die Überprüfung der Berechtigungen bei den verschiedenen Services verwendet.

```

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.lang.ProcessBuilder;

public class FileSystemAccessTest {

    public static void main(String[] args) {
        run("ls", "-la", "/");
    }

    private static void run(String... args) {
        try {
            String output;
            Process p = new ProcessBuilder(args).start();
            BufferedReader br = new BufferedReader(
                new InputStreamReader(p.getInputStream()));
            while ((output = br.readLine()) != null)
                System.out.println(output);
            p.waitFor();
            System.out.println("exit: " + p.exitValue());
            p.destroy();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}

```

Code-Snippet 3 Testcode Überprüfen der Berechtigungen

Name des Services	Berechtigung des /dev Ordners im Root Verzeichnis in Oktalnotation	Berechtigungen des /home Ordners im Root Verzeichnis in Oktalnotation	Testdatum
OnlineGDB	755	777	21.12.2022
WandBox	755	755	21.12.2022
Judge0	755	700	21.12.2022
Tutorialspoint	755	755	21.12.2022
GeeksForGeeks	-	-	21.12.2022
Ideone	775	775	21.12.2022
Replit	755	755	21.12.2022
JDoodle	755	755	21.12.2022
Paiza.IO	755	755	21.12.2022
CodeInterview	755	755	21.12.2022

Tabelle 4 Berechtigungen bei den verschiedenen Services

Im Falle der IDE von Judge0 entsprach der Name des Ordners nicht */home*, ist jedoch als äquivalent zu betrachten.

GeeksForGeeks verbietet in gewisser Weise das Ausführen von UNIX-Commands wie *ls*. Somit konnten hier keine Ergebnisse geliefert werden. Bei Ausführung des Code-Snippets 3 wird eine

Fehlermeldung ausgegeben, wonach die UNIX-Kommandos nicht gefunden werden konnten. Auch das UNIX-Kommando *where*, um herauszufinden wo sich ein Skript befindet, konnte nicht ausgeführt werden. Nachdem statt UNIX-Kommandos auf äquivalenten Java-Code umgestiegen wurde, war es möglich, beliebige Pfade und deren Inhalte auszugeben. Im Home-Directory konnten keine Dateien oder Ordner außer einer *.dockerenv* Datei gefunden werden. Die IDE bzw. die Docker-Umgebung verbirgt also standardmäßig alle Dateien und Ordner, bis auf die vom Benutzer selbst erstellten und minimiert somit jedes Risiko von potenziellen Angriffen.

Wie man sieht, werden verschiedene Herangehensweisen für die Berechtigungsvergabe angewandt. Eine Verallgemeinerung, welcher Teil der Oktalnotation für den Zweck dieser Arbeit relevant ist, ist nicht möglich. In manchen Services nimmt man automatisch die Rolle des *root* Benutzers an, während man in anderen in die Kategorie *Sonstige* fallen würde. Im Allgemeinen werden, unabhängig von der Benutzerrolle, einem kaum Rechte für das Erstellen und Ausführen von Dateien, bis auf den eigentlichen Projektordner, eingeräumt. Die wahre Sicherheitsschranke ist und bleibt die Docker-Umgebung, welche jederzeit vom Betreiber terminiert werden kann. Auch wenn die Möglichkeit bestünde etwaige Dateien zu erstellen und auszuführen, kann kein großer Schaden ohne tiefere Kenntnis des Systems angerichtet werden. Es kann nicht vollkommen ausgeschlossen werden, dass es keine Schlupflöcher für etwaige Angriffe gibt.

Eine Überraschung war das Ergebnis der Untersuchung der Tutorialspoint IDE. Die Entwickler jener Web-IDE haben sich gegen den Einsatz von Docker oder ähnlichen Technologien entschieden. Hier bekommt nicht jeder Benutzer seinen eigenen virtuellen Server zur Verfügung gestellt, was jedoch bedeutet, dass durch einfaches *Directory Traversal* der Code anderer Nutzer eingesehen werden kann. Auch wenn hier keine sensiblen Daten im Spiel sind, ist das ein Designfehler, der vermieden werden sollte. Man kann nur vermuten, dass mit dieser Entscheidung Entwicklungsaufwand eingespart werden sollte.

Die Vergabe entsprechender Berechtigungen ist nur ein Teil des Sicherheitskonzepts für Betreiber. Es muss bedacht werden, dass trotz korrekter Berechtigungen Benutzer beliebigen Code auf dem Server ausführen können. Werden die bereitgestellten Rechenressourcen nicht begrenzt, sowohl in Ausführ- und Kompilierzeit als auch in der Rechenleistung, kann es zu einer Ausnutzung durch Benutzer kommen. Darüber hinaus muss bedacht werden, dass auch die Lahmlegung des Services beispielsweise durch Forkbombs, i.e. Programme, die sich immer wieder selbst starten und jeweils einen eigenen Stack erhalten, das Ziel mancher Benutzer ist. Um zu testen, ob die Services gegen Forkbombs gewappnet sind, wurde folgender Java-Code ausgeführt.

```
import java.io.*;
import java.lang.ProcessBuilder;

public class ForkBomb {

    public static void main(String[] args) {
        try (Writer writer = new BufferedWriter(new OutputStreamWriter(
            new FileOutputStream("test.sh"), "utf-8"))) {
            writer.write("#!/usr/bin/env bash\n");
            writer.write("function f() {\n");
            writer.write("f | f&\n");
            writer.write("}\n");
            writer.write("f\n");
        } catch (Exception e) {
            e.printStackTrace();
        }
        run("chmod", "+x", "test.sh");
        run("./test.sh");
    }

    private static void run(String... args) {
        try {
            String output;
            Process p = new ProcessBuilder(args).start();
            BufferedReader br = new BufferedReader(
                new InputStreamReader(p.getInputStream()));
            while ((output = br.readLine()) != null)
                System.out.println(output);
            p.waitFor();
            System.out.println("exit: " + p.exitValue());
            p.destroy();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Code-Snippet 4 Java-Code Forkbomb

Das Verhalten der webbasierten Entwicklungsumgebungen war, mit einer Ausnahme, durchwegs konsistent. Der virtualisierten Umgebung wird nur eine bestimmte Menge an Speicher zur Verfügung gestellt, Swap-Memory ist nicht konfiguriert. Sobald also kein Speicher mehr zur Verfügung steht, muss der Container terminiert werden. Für den eigentlichen Host hat dieser Prozess keine Auswirkungen, da der benutzte Speicher sowieso für die Virtualisierung reserviert war. Will der Benutzer weiter Code ausführen, muss ein neuer Container angefordert werden.

Die Ausnahme dieses Verhaltens war wiederum die IDE von Tutorialspoint. Da kein abgeschotteter Container für jeden einzelnen Benutzer zur Verfügung gestellt wird, brachte die ausgeführte Forkbomb den gesamten Service für ≈ 10 Minuten zum Stillstand. Es wurde dabei nicht überprüft, in welchen Regionen der Service einbrach. Es ist davon auszugehen, dass bei einem Service dieser Größenordnung *Content Delivery Networks* (CDN) eingesetzt werden, um die Serverlast regional zu verteilen und Anfragen schneller beantworten zu können. Der Service würde damit nur in der angegriffenen Region ausfallen. Benutzer in anderen Regionen könnten den Service weiterhin benutzen.

Problematisch bleibt der Umstand, dass theoretisch beliebige Dateien oder Skripte im Betriebssystem des Services von Tutorialspoint gespeichert und ausgeführt werden können. Normalerweise wäre dies kein Problem, da einem Docker-Container für gewöhnlich nur limitierte Ressourcen, sowohl zeitlich als auch leistungstechnisch, zur Verfügung gestellt werden. Es ist unklar, ob es möglich wäre, schadhafte Code dauerhaft im Service zu speichern und automatisch nach Systemneustart wieder auszuführen, um so beispielsweise die Rechenleistung auszunutzen.

4.3.5. Inkludieren von Ressourcen

Zum Testen dieser Funktionalität wurde ein Use Case simuliert, in dem der Benutzer Daten aus einer CSV-Datei einlesen und weiterverarbeiten muss. Die gesamte CSV-Datei kann unter dem Link im Appendix gefunden werden. Auf Seiten der Betreiber der Web-IDEs setzt diese Funktionalität den Upload von Dateien voraus. Folgender Code wurde für diesen Test verwendet.

```
import java.util.*;
import java.io.*;

public class ReadInputTest {

    public static void main(String[] args) {
        try (Scanner scanner = new Scanner(new File("example_data.csv")));
        {
            while (scanner.hasNextLine()) {
                List<String> values =
                getValuesFromLine(scanner.nextLine());
                System.out.println("Date: " + values.get(0));
                System.out.println("Time: " + values.get(1));
                System.out.println("Value: " + values.get(2));
                System.out.println("---");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static List<String> getValuesFromLine(String line) {
        List<String> values = new ArrayList<String>();
        try (Scanner rowScanner = new Scanner(line)) {
            rowScanner.useDelimiter(";");
            while (rowScanner.hasNext()) {
                values.add(rowScanner.next());
            }
        }
        return values;
    }
}
```

Code-Snippet 5 Testcode Einlesen einer CSV-Datei

Folgende Ergebnisse konnte dieser Test liefern.

Name des Services	Einlesen von Daten aus einer separaten Datei möglich	Testdatum
OnlineGDB	Ja	12.2.2022
WandBox	Ja, direkter Upload von Dateien nicht möglich, neue Dateien können jedoch erstellt werden	12.2.2022
Judge0	Nein	12.2.2022
Tutorialspoint	Nein	12.2.2022
GeeksForGeeks	Nein	12.2.2022
Ideone	Nein	13.2.2022
Replit	Ja	13.2.2022
JDoodle	Ja, Datei wird nicht im selben Ordner abgelegt und muss über Shell-Commands im Dateisystem gesucht werden	13.2.2022
Paiza.IO	Ja	13.2.2022
CodeInterview	Nein	13.2.2022

Tabelle 5 Ergebnisse des Test Lesen von Input aus non-Source-Code Dateien

Es gibt eine klare Grenze zwischen Services mit vollkommener Unterstützung mehrerer Dateien, gleichgültig ob Source-Code oder nicht, und Services, welche nur eine einzige Source-Code-Datei unterstützen, zu erkennen. Die GeeksForGeeks IDE bietet zwar den Upload von weiteren Source-Code Dateien an, lässt aber von einer Unterstützung sonstiger Dateitypen ab.

4.3.6. Graphische Ausgabe

Der Versuch ein Fenster mittels Swing und Java zu öffnen, endet für fast alle Web-IDEs in einer *HeadlessException*. Die X11 Display Variable, welche standardmäßig automatisch gesetzt wird, hat in einer virtualisierten Umgebung keinen Wert, wodurch AWT bzw. Swing kein konkretes Ziel für das zu öffnende Fenster hat. Überraschenderweise gelingt der Web-IDE Replit die fehlerfreie Ausführung folgenden Codes.

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class GraphicalOutputTest {

    private static void createAndShowGUI() {
        //Create and set up the window.
        JFrame frame = new JFrame("Graphical Output Test");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel emptyLabel = new JLabel("");
        emptyLabel.setPreferredSize(new Dimension(175, 100));
        frame.getContentPane().add(emptyLabel, BorderLayout.CENTER);
        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}

```

Code-Snippet 6 Testcode Graphischer Output

Replit simuliert dabei ein GUI im Browser und erlaubt darüber hinaus auch die Interaktion mit diesem. Leider ist die Web-IDE nicht open-source, womit die genaue Funktionsweise nicht direkt untersucht werden kann. In der Interaktion mit dem System und über die Firefox Developer Tools ist ersichtlich, dass der Fenstermanager Fluxbox [37] und ein VNC System in Hintergrund verwendet werden.

4.3.7. Einbindung mehrerer Source-Code Dateien

Die untersuchten Web-IDEs sehen ihren Use-Case unterschiedlich und bieten dem Benutzer mehr oder weniger Funktionalität je nach Interpretation und Kontext. Die Möglichkeit weitere Source-Code-Dateien zu erstellen und diese im Projekt zu verwenden, bieten 4 von 10 der webbasierten Entwicklungsumgebungen an. Diese Funktionalität ist nicht gleichzusetzen mit der Option non-Source-Code Dateien in das Projekt einzubinden, wenn auch die Ergebnisse fast deckungsgleich sind. Wie schon in 3.4.7. beschrieben, besteht die Schwierigkeit bei der Ausführung mehrerer Dateien im Finden des Einstiegspunkts des Codes. Natürlich besteht die Möglichkeit den Benutzer angeben zu lassen, in welcher Datei sich der Einstiegspunkt befindet. Die weitaus elegantere Lösung ist jedoch die automatische Analyse des Codes. Inwiefern der Aufbau und Verwendung einer Package Struktur erlaubt ist, wurde in den Web-IDEs nicht untersucht. Manche der Web-IDEs geben explizit den Hinweis, dass keine Packages verwendet werden sollen. Grund dafür dürfte das Speichern aller Source-Code-Dateien in einem einzigen Ordner sein. Falls die IDE den Source-Code auf eine Datei beschränkt, existiert noch immer die Möglichkeit, alle Klassen als innere Klassen zu definieren. Um zu analysieren, ob die Kompilierung und Ausführung mehrerer Dateien möglich ist, wurde folgender Code verwendet. Dabei wurden alle Dateien im *default*-Package gespeichert und keine *import* Statements verwendet.


```
public class CompileMultipleFilesTest {

    public static void main(String[] args) {
        Animal a = new Dog();
        a.makeNoise();
    }

}
```

Code-Snipet 7 Testcode Mainklasse mehrere Source-Code-Dateien

```
public abstract class Animal {
    public abstract void makeNoise();
}
```

Code-Snipet 8 Abstrakte Java-Klasse *Animal*

```
public class Dog extends Animal {

    @Override
    public void makeNoise() {
        System.out.println("Woof woof!");
    }

}
```

Code-Snipet 9 Java-Klasse *Dog*

Folgende Ergebnisse konnte dieser Test liefern.

Name des Services	Erstellen und verwenden von mehreren Source-Code-Dateien möglich	Testdatum
OnlineGDB	Ja	15.2.2022
WandBox	Ja	15.2.2022
Judge0	Nein, IDE wird auf eine einzige Datei beschränkt	15.2.2022
Tutorialspoint	Nein, IDE wird auf eine einzige Datei beschränkt	15.2.2022
GeeksForGeeks	Nein, mehrere Dateien können erstellt werden, stehen jedoch in keinem Zusammenhang zueinander	15.2.2022
Ideone	Nein, IDE wird auf eine einzige Datei beschränkt	15.2.2022
Replit	Ja	15.2.2022
JDoodle	Nein, Source-Code könnte jedoch theoretisch, wie im vorhergehenden Kapitel, über den Upload inkludiert werden. Direkte Funktionalität besteht nicht	15.2.2022
Paiza.IO	Ja	16.2.2022
CodeInterview	Nein, IDE wird auf eine einzige Datei beschränkt	16.2.2022

Tabelle 6 Ergebnis Test mehrere Source-Code-Dateien

4.3.8. Debugger

Um zu überprüfen, ob die ins Auge gefassten Web-IDEs über Debugger verfügen, wurde versucht, ein kurzes Code Snippet Schritt für Schritt auszuführen. In diesem Code-Snippet befindet sich kein wirklicher Bug. Es sollte im Zuge des Tests lediglich überprüft werden, ob eine Schritt-für-Schritt Ausführung und Inspektion der Variablenwerte zu einem gewissen Zeitpunkt möglich ist.

```
public class DebuggerTest {
    public static void main(String[] args) throws InterruptedException {
        int x = 0;
        for(int i = 0; i < 10; i++) {
            x += i*i;
            System.out.println(x);
            Thread.sleep(1000);
        }
    }
}
```

Code-Snippet 10 Testcode Umfang von Debugging-Funktionen

Folgende Ergebnisse konnte dieser Test liefern.

Name des Services	Debugging möglich	Testdatum
OnlineGDB	Nein, Debugger wird angeboten, beim Stoppen des Programms wird jedoch ein Transport Error ausgegeben	20.1.2022
WandBox	Nein	20.1.2022
Judge0	Nein	20.1.2022
Tutorialspoint	Nein	20.1.2022
GeeksForGeeks	Nein	20.1.2022
Ideone	Nein	20.1.2022
Replit	Nein	20.1.2022
JDoodle	Nein	20.1.2022
Paiza.IO	Nein	20.1.2022
CodeInterview	Nein	20.1.2022

Tabelle 7 Ergebnisse Test Debugging möglich

Es war zu erwarten, dass webbasierte Entwicklungsumgebungen ihren lokal laufenden Gegenstücken in dieser Hinsicht unterlegen sind. Das Stoppen des Codes mit anschließender Kommunikation über HTTP mit dem Benutzer stellt sich als eine Herausforderung dar, welcher sich nur OnlineGDB angenommen hat. Leider konnte kein erfolgreicher Testdurchlauf durchgeführt werden, in dem die Debugging Funktion tatsächlich benutzt werden konnte. In meiner Recherche wurden auch keine Frameworks oder Tools gefunden, welche eine Implementierung eines Debuggers in einer Browser Umgebung erleichtern würden.

4.3.9. Autovervollständigung

Autovervollständigung ist nicht gleich Autovervollständigung. Viele der Web-IDEs führen Autovervollständigung als eine ihrer Funktionalitäten an, bleiben im Vergleich aber weit hinter

den gewohnten Annehmlichkeiten lokal laufender IDEs. Folgender Code wurde verwendet, um zu überprüfen, ob es sich um lexikalische Vervollständigung handelt oder ob im Hintergrund zusätzlich andere Source-Code-Dateien nach passenden Methoden oder Keywords durchsucht werden.

```
import java.util.List;
import java.util.ArrayList;

public class AutoCompletionTest {

    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.size();
    }
}
```

Code-Snippet 11 Testcode Autovervollständigung

Im Test wurde überprüft, ob die Methode *size()* der Klasse *ArrayList* bzw. des Interface *List* automatisch vorgeschlagen wird. Um diesen Vorschlag geben zu können, müsste im Hintergrund die Klasse analysiert werden. Wird kein derartiger Vorschlag gemacht, handelt es sich um eine simple lexikalische Vervollständigung, wie sie sowohl beim ACE-Editor als auch beim Monaco-Editor mitgeliefert wird. Die lexikalische Vervollständigung bezieht immer nur Informationen aus jener Datei, an welcher gerade gearbeitet wird. Die Ergebnisse dieses Tests waren folgende.

Name des Services	Vollständiger IntelliSense	Lexikalisch	Keine Vervollständigung	Testdatum
OnlineGDB		X		19.2.2022
WandBox			X	19.2.2022
Judge0		X		19.2.2022
Tutorialspoint			X	19.2.2022
GeeksForGeeks		X		19.2.2022
Ideone			X	19.2.2022
Replit	X			19.2.2022
JDoodle			X	19.2.2022
Paiza.IO		X		19.2.2022
CodeInterview	X			19.2.2022

Tabelle 8 Ergebnisse des Tests auf Autovervollständigung

Für einen vollständigen IntelliSense, welcher eben Klassen im Hintergrund analysieren würde, wäre ein Language Server notwendig, mit welchem der Editor kommunizieren kann. Je mehr Programmiersprachen von einer Web-IDE unterstützt werden, desto größer ist der Arbeitsaufwand für jede einzelne Sprache, einen eignen Language Server bereitzustellen, weswegen die meisten Betreiber von der Implementierung absehen. Replit bietet zwar einen vollständigen IntelliSense an, die Reaktionszeit lässt aber zu wünschen übrig.

4.3.10. Unterstützung mehrerer Programmiersprachen

Alle der untersuchten Web-IDEs unterstützten die Mehrheit der am häufigsten verwendeten Programmiersprachen [38], mit Ausnahme der in der Statistik angeführten Technologien HTML/CSS und SQL, welche nur vereinzelt unterstützt werden. Klassische objektorientierte, imperative und deklarative Programmiersprachen, wie Java, C oder Python werden von allen webbasierten Entwicklungsumgebungen angeboten.

4.3.11. Automatische statische Code-Überprüfung

Im Zuge dieses Kapitels wurden in einem Stück Code mehrere Fehler eingebaut, um zu überprüfen, ob und welche Fehler von den IDEs noch vor der Ausführung durch den Benutzer erkannt werden.

```
public class ErrorDetectionTest extends Test {

    public static void main(String[] args) {
        System.out.println("This is a test")
        System.ot.println("This is also a test");
        square();
    }

    private int square(int x) {
        x = x*x;
    }

    private static String getString() {
        return 1;
    }

}
```

Code-Snippet 12 Testcode statische Code Überprüfung

In Code-Snippet 12 befinden sich eine Reihe an Syntaxfehlern. Um eine bessere Übersicht in der Ergebnistabelle zu erhalten, teilen wir den Fehlern Nummern zu.

1. *extends Test*: Base-Klasse Test existiert nicht
2. *print-Statement-1*: fehlendes Semikolon am Ende der Zeile
3. *print-Statement-2*: fehlendes *u* in *out* Feld
4. *square-Aufruf*: fehlender Parameter
5. *square-Auruf*: Methode ist nicht statisch, sollte also nicht aus *main* aufgerufen werden können
6. *square-Methode*: Methode ist *void*, Signatur zeigt jedoch *int* an
7. *getString-Methode*: Methode gibt einen Integer-Wert zurück und stimmt somit nicht mit Signatur *String* überein

Bei der Eingabe des Code-Snippets 12 in die verschiedenen Web-IDEs werden folgende Fehler erkannt und dem Benutzer als solche angezeigt.

Name des Services	1	2	3	4	5	6	7	Testdatum
OnlineGDB								20.2.2022

WandBox													20.2.2022
Judge0													20.2.2022
Tutorialspoint													20.2.2022
GeeksForGeeks													20.2.2022
Ideone													20.2.2022
Replit	X	X	X	X							X		20.2.2022
JDoodle													20.2.2022
Paiza.IO													20.2.2022
CodeInterview													20.2.2022

Tabelle 9 Ergebnisse statische Code-Überprüfung

Überraschenderweise verwendet nur Replit statische Code-Analyse Tools, um den Benutzer frühzeitig über Fehler zu informieren.

4.3.12. Kollaboration

Kollaboration ist kein Alleinstellungsmerkmal von Web-IDEs. Über die Installation von Third-Party-Plugins in beliebigen IDEs wie Visual Studio Code oder IntelliJ kann ebenso mit anderen Personen an Code kollaboriert werden. Natürlich bedeutet die zusätzliche Implementierung, über die Basisfunktionalität von Web-IDEs hinaus, zu einer gemeinsamen Dokumentbearbeitung, einen erhöhten Entwicklungsaufwand. Um diesen Aspekt zu überprüfen, wurden die Dokumentation wie auch die UIs der Web-IDEs nach passenden Funktionen durchsucht. Nur Replit und CodeInterview bieten die Möglichkeit zur Kollaboration an. Mit Hilfe eines Links können beliebig viele Personen in eine gemeinsame Session eingeladen werden.

4.4. Vergleich der Online-IDEs

Um nun einen Vergleich zwischen den in 3.1. gelisteten IDEs aufstellen zu können, müssen die Ergebnisse der ausgeführten Tests zusammengefasst werden. Alle Tests, ausgenommen die Tests der Kapitel 4.3.4. und Kapitel 4.3.10., haben im Ergebnis einen booleschen oder Integer Wert und können somit in die Wertung mit aufgenommen werden. Der Test aus Kapitel 4.3.11. wird als positiv gezählt, falls überhaupt eine statische Überprüfung durchgeführt wird und somit mindestens ein einziger Fehler erkannt wird. Für Kapitel 4.3.9. wird ein ganzer Punkt für vollständigen IntelliSense und ein halber Punkt für lexikalische Vervollständigung gegeben. Alle sonstigen bestandenen Tests gelten als ein Punkt für die jeweilige Web-IDE. Der Einfachheit halber werden die verschiedenen Tests untereinander nicht anhand ihrer Wichtigkeit gewertet. Um einen noch detaillierteren Vergleich aufstellen zu können, müssten den Tests unterschiedliche Prioritäten gegeben werden. Für eine Prioritätenvergabe fehlt jedoch die notwendige objektive Vergleichbarkeit. Spalten D und J entsprechen den Tests der Kapitel 4.3.4. und 4.3.10..

Name des Services	A	B	C	D	E	F	G	H	I	J	K	L	Summe
OnlineGDB	1	1	1	-	1	0	1	0	0,5	-	0	0	5,5
Wandbox	0	0	1	-	1	0	1	0	0	-	0	0	3
Judge0	0	0	1	-	0	0	0	0	0,5	-	0	0	1,5
Tutorialspoint	1	1	0	-	0	0	0	0	0	-	0	0	2
GeeksForGeeks	0	0	0	-	0	0	0	0	0,5	-	0	0	0,5

Ideone	0	0	0	-	0	0	0	0	0	-	0	0	0
Replit	1	1	1	-	1	1	1	0	1	-	1	1	9
Jdoodle	0	0	1	-	1	0	0	0	0	-	0	0	2
Paiza.io	0	0	0	-	1	0	1	0	0,5	-	0	0	2,5
CodeInterview	1	1	0	-	0	0	0	0	1	-	0	1	4

Tabelle 10 Zusammenfassende Ergebnisse der Online-IDEs

Replit schneidet mit 9/9 erreichbaren Punkten am besten ab und hat einen Abstand von 3,5 Punkte zum Zweitplatzierten OnlineGDB. Es muss bedacht werden, dass diese Ergebnisse nur eine Wertung der Services im Hinblick auf UX und Sicherheit darstellen und den wichtigen Aspekt der Geschwindigkeit außer Acht lassen. In den von Sinanaj *et al.* [20] durchgeführten Tests schneidet Replit stets am schlechtesten ab, während Web-IDEs mit weniger Annehmlichkeiten weitaus schneller Code kompilieren und ausführen. Wie auch bei der Wertung der Tests untereinander, ist es auch hier schwierig, die Ergebnisse der Geschwindigkeitstests mit in das Ranking aufzunehmen. Die Ergebnisse aus meiner Analyse der UX und Sicherheit sollten lediglich die Ergebnisse aus [20] ergänzen und eine breitere Entscheidungsbasis bieten. Verschiedene Use Cases fordern verschiedene Funktionalität, womit eine eindeutige Empfehlung zur Verwendung einer bestimmten Web-IDE nicht möglich ist.

5. Eigene Implementierung einer webbasierten Entwicklungsumgebung

Im praktischen Teil dieser Bachelor-Arbeit wurde eine prototypische webbasierte Entwicklungsumgebung implementiert, mit welcher man eine Reihe von beliebten Programmiersprachen serverseitig kompilieren kann und der Benutzer den Output des gesendeten Codes im Frontend angezeigt erhält. Der Link zum gesamten Projektcode ist im Appendix, Kapitel 8.1., zu finden. Es wurde versucht, viele der bereits aufgezählten essenziellen Features, welche eine Online-Compiler anbieten sollte, zu implementieren. Es wurden keine präventiven Maßnahmen implementiert, um die Sicherheit des Services zu gewährleisten, da dies den Rahmen dieser Bachelorarbeit sprengen würde. Unter normalen Umständen müsste der Sicherheit überaus große Beachtung geschenkt werden. Darüber hinaus beschränkt sich der Prototyp auf die grundlegenden Funktionen, die man von einer webbasierten Entwicklungsumgebung erwarten würde. Der Prototyp ist mit Bedacht auf eine zukünftige mögliche Erweiterung gebaut und kann somit als Basis für kommende weiterführende Projekte verwendet werden.

Folgenden Programmiersprachen werden von meiner webbasierten Entwicklungsumgebung unterstützt:

- Java
- C
- C++
- Python
- Rust
- Go
- Bash
- JavaScript

Zur einfachen Handhabung des Projekts als auch für den potenziellen Einsatz auf einem VPS wird ein Shell-Skript bereitgestellt, welches alle notwendigen Compiler wie auch alle benötigten NPM-Pakete auf dem ausführenden Rechner installiert.

Um ein tieferes Verständnis der grundlegenden Konzepte von webbasierten Entwicklungsumgebungen zu erlangen, habe ich mich gegen die Verwendung von diversen Code Execution Systemen oder einer Implementierung mit Hilfe von WebAssembly entschieden.

5.1. Verwendete Technologien

Aufgrund meiner gewonnenen Erfahrung mit Backend-JavaScript-Technologien zu Beginn des Projekts, entschied ich mich für einen ERN-Stack, bestehend aus Express und Node.JS im Backend und das React Framework im Frontend. Der Stack besteht in diesem Fall nur aus drei Komponenten, da eine Datenbank-Anbindung für die gestellten Anforderungen nicht notwendig ist. Auf Hinweis meines Betreuers, Dr. Narzt, wurde mir React in Kombination mit dem bekannten Monaco-Editor als Grundlage für das Frontend empfohlen. Beide Technologien erwiesen sich als leicht handhab- und erweiterbar. Erwähnenswert sind auch noch die beiden im Frontend verwendeten NPM-Pakete *Axios* [39] und *Bootstrap* [40]. *Axios* ist isomorph, was bedeutet, dass es sowohl im Front- als auch im Backend verwendet werden kann, und bietet die Möglichkeit, mit kurzer und simpler Syntax HTTP-Requests abzuschicken. *Bootstrap* umfasst eine breite Palette an vorgefertigten CSS-Klassen, welche die Entwicklung des UIs vereinfachen.

5.2. Architektur der Entwicklungsumgebung

5.2.1. Backend

Das Backend bietet eine REST-Schnittstelle [41] an und ist in drei separate Komponenten aufgeteilt. *app.js* stellt den Einstiegspunkt für das Backend dar und erstellt notwendige Dateien und Ordner für den Betrieb der API. *helper.js* und *info.js* stellen kleinere Unterstützungsmethoden sowohl für das Frontend als auch das Backend bereit. Die REST-Schnittstelle und alle notwendigen Routen sind in *api.js* zu finden. Die API bietet eine klare Möglichkeit, die Schritte des Code-Uploads, des Kompilierens und der Ausführung zu trennen und lässt den Benutzer der API selbst entscheiden, wann zum nächsten Schritt übergegangen werden soll.

Über folgende Routen kann der Benutzer mit dem Backend kommunizieren:

WebIDE		Find out more ^
POST	<code>/sendCode</code> Upload Code to the server.	∨
POST	<code>/compile/:fileName</code> Compile an already uploaded file.	∨
POST	<code>/run/:fileName</code> Run the compiled file.	∨
POST	<code>/receiveBufferedOutput</code> Get output up until this point.	∨
POST	<code>/stopProgram</code> Stop the program.	∨
GET	<code>/isProgramRunning</code> Get info about program status.	∨

Abbildung 4 Aufbau der REST-API

Der Code an sich als auch der Name der Datei und die entsprechende Programmiersprache werden allesamt im Body des POST-Requests mitgegeben. Zu Beginn der Entwicklung habe ich mit einem File-Upload mit Hilfe eines Input Tags im HTML experimentiert, bin jedoch zum Schluss gekommen, dass vor allem in Kombination mit dem Monaco-Editor diese Herangehensweise umständlich wäre. Zur Speicherung des empfangenen Codes gibt es nun zwei Optionen: entweder man lagert die Source-Code-Dateien in einer Datenbank oder speichert sie selbst im File-System ab. Aufgrund der Projektgröße habe ich mich für das direkte Speichern im File-System entschieden. Es wäre jedoch zu überlegen, die beiden Ansätze möglicherweise zu kombinieren, um eine bessere Skalierbarkeit zu garantieren. Dabei würde der Pfad der abgespeicherten Datei wiederum in einer Datenbank gespeichert werden.

5.2.2. Frontend

Wie schon eingangs erwähnt, wurde zur Umsetzung des Frontends hauptsächlich React in Kombination mit Bootstrap verwendet. Das UI ist in grob in drei Komponenten aufgeteilt. Ganz oben befindet sich eine Menüleiste mit mehreren Auswahlmöglichkeiten, direkt darunter ist der Text-Editor zu sehen. Zu guter Letzt befindet sich im unteren Teil ein Fenster, in welchem der Output des gesendeten Programms angezeigt wird. Bei Start der Single Page Application wird dem Benutzer somit folgende Ansicht präsentiert.

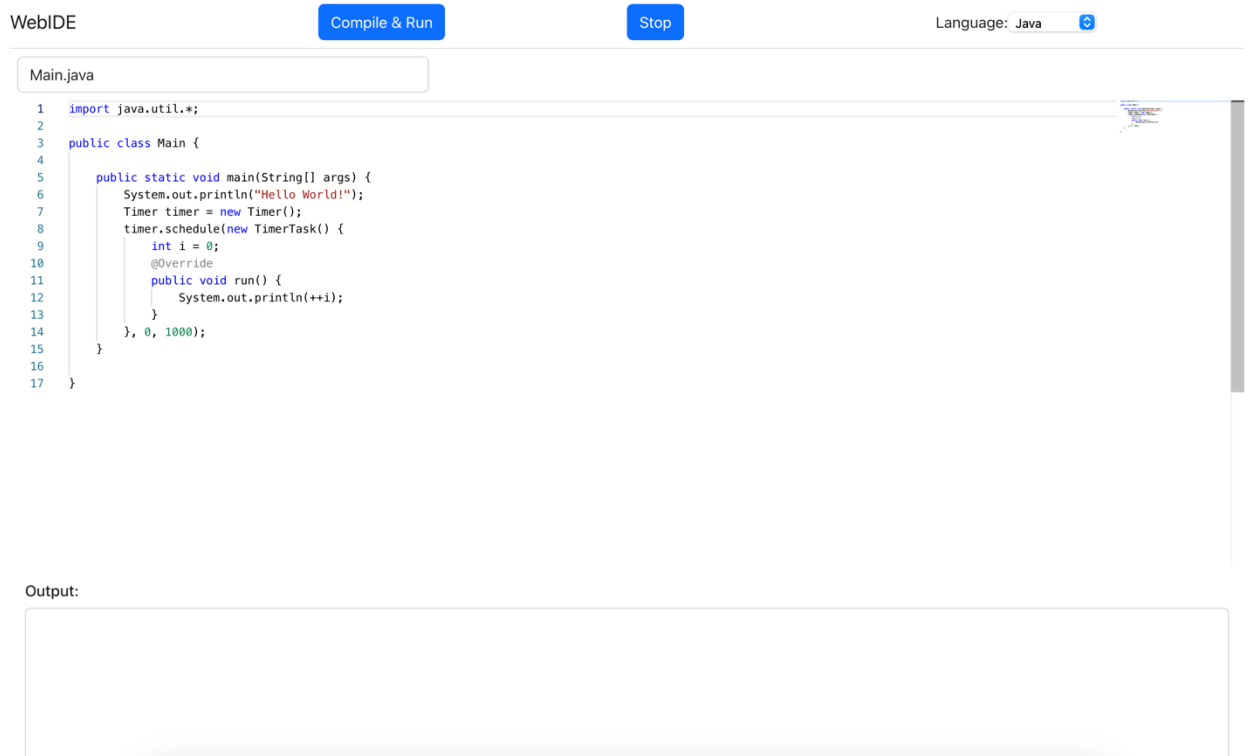


Abbildung 5 Startseite der Web-IDE

Die Auswahl der Sprache kann rechts oben getroffen werden. Beim Ändern der Auswahl wird vom Backend ein Code-Snippet der entsprechenden Sprache angefordert und daraufhin im Monaco-Editor angezeigt. Der Benutzer muss jedoch beachten, dass Änderungen bei einem Sprachwechsel nicht gespeichert werden.

5.3. Schwierigkeiten bei Implementierung

Sowohl in der Verwendung der, mir völlig neuen, Technologie React als auch bei der Konzeptualisierung einzelner Funktionen des Projekts gab es kleinere Probleme.

5.3.1. Klassenkomponenten vs. Funktionskomponenten in React

Der erste Versuch, das gesamte UI mit Hilfe von Klassenkomponenten umzusetzen, erwies sich als etwas umständlich. Diese erste Entscheidung entstand aus meiner Vertrautheit mit klassischen objektorientierten Konzepten, welche ich hier anwenden wollte. Das UI konnte ich auch vollständig mit Klassenkomponenten umsetzen, jedoch benötigen diese Art von Komponenten immer etwas Boilerplate-Code, um den gewünschten Effekt zu erzielen. Bei weiterer Recherche zum Thema React und den aktuellen Entwicklungen stieß ich auf das neuartige Komponentenkonzept der Funktionskomponenten. Diese sollen zukünftig den Platz von Klassenkomponenten in React einnehmen und dementsprechend weiterentwickelt werden. Begründung für diese Umstellung ist unter anderem die mühsame Verhaltensweise von der *this*-Referenz in Javascript. Während bei Klassenkomponenten das *this* für jede einzelne Funktion an die jeweilige Klasse gebunden werden muss, findet *this* in reinen Funktionskomponenten keine Verwendung und kann somit keine Probleme bereiten.

In den Abbildungen 3 und 4 wird der Unterschied der beiden Herangehensweisen ersichtlich. Die beiden Komponenten sind in ihrer Funktion vollkommen äquivalent, jedoch benötigt man für eine Klassenkomponente etwas mehr LoC.

Obwohl eine Funktionskomponente selbst keinen Zustand besitzen kann, ist es möglich, mittels React Hooks, welche nur in Kombination mit Funktionskomponenten verwendet werden können, einen Zustand zu speichern. React selbst bezeichnet Klassenkomponenten bereits als *Legacy-API* und empfiehlt in ihrer Dokumentation stattdessen Funktionskomponenten zu verwenden.

```

1  import React from 'react';
2
3  function FileNameChooser(props) {
4
5      function updateInputValue(e) {
6          props.handleFileNameChange(e.target.value);
7      }
8
9      const fileName = props.fileName ? props.fileName : "";
10     return (
11         <div className="col-4 m-2">
12             <input className="form-control" value={fileName} type="text" onChange={(e) => updateInputValue(e)}>
13             </input>
14         </div>
15     )
16 }
17

```

Abbildung 6 Beispiel einer Funktionskomponente in React

```

1  import React from 'react';
2
3  class FileNameChooser extends React.Component {
4      constructor(props) {
5          super(props);
6          this.updateInputValue = this.updateInputValue.bind(this);
7      }
8
9      updateInputValue(e) {
10         this.props.handleFileNameChange(e.target.value);
11     }
12
13     render() {
14         return (
15             <div className="col-4 m-2">
16                 <input className="form-control" value={this.props.fileName} type="text" onChange={(e) => this.updateInputValue(e)}>
17                 </input>
18             </div>
19         )
20     }
21 }

```

Abbildung 7 Beispiel einer Klassenkomponente in React

5.3.2. Übertragung des Source-Codes

In einer früheren Version des Projekts verwendete ich das NPM-Paket *Multer* [42], um den eingegebenen Source-Code bereits als Datei an das Backend zu senden. Dies stellte sich als Overkill für die Anforderungen heraus und ich empfand es als simpler, den Text als Teil des Request-Bodys zu versenden. Erst im Backend wird der Text in eine Datei geschrieben und für die weiteren Schritte gespeichert. Auch bei einer Erweiterung der Funktionalitäten hin zu einem Multi-Dateien-System wäre eine Mitgabe des Source-Codes über den Request-Body passend. Da es sich im Kontext von webbasierten Entwicklungsumgebung meistens nicht um ausführliche Projekte handelt, stellt die Größe des versendeten Texts kein Problem dar. Darüber hinaus

spezifiziert der HTTP Standard auch keine maximale Größe für POST-Requests [43], lediglich der Server selbst könnte eine maximale Größe definieren.

5.3.3. Asynchrone Ausgabe des Outputs

Bei der Funktionalität der asynchronen Ausgabe des Outputs recherchierte ich zuerst nach einer Möglichkeit, einen einzigen HTTP-Request offen zu halten und nach und nach den Output in diesen Request zu schreiben, während das Frontend somit den Output zeitgleich auslesen kann. Hintergrundgedanke war, die Anzahl der gesendeten Requests so niedrig wie möglich zu halten. Die Recherche brachte jedoch keine Ergebnisse, weshalb ich die Abfrage des Outputs als Teil der REST-Schnittstelle implementierte. Während der Ausführung des Programms wird asynchron der Output in eine Textdatei geschrieben, welche danach über eine Route im Backend abgefragt werden kann. Wichtig ist hier, dass im NodeJS Backend der *spawn Command* anstatt des *exec Commands* des *child_process* Moduls ausgeführt wird. *Exec* verfügt lediglich über die Möglichkeit bei Ende des Subprozesses, Code in Form eines Callbacks auszuführen. *Spawn* hingegen lässt den Benutzer sogenannte Data-Listeners definieren, welche die asynchrone Umleitung des Outputs erlauben. Die Route ist als POST definiert, da bei jedem Aufruf der Route die Textdatei geleert wird.

Im Frontend implementierte ich einen Switch der, auf Basis des Programmzustands, in einem Takt von einer Sekunde den Output von der REST-Schnittstelle abfragt. Erst nach Fertigstellung dieser Komponente wurde mir klar, dass für jede Abfrage eine neue Connection zum Backend aufgebaut werden musste. Nach wiederholter Recherche entdeckte ich die Möglichkeit der *keep-alive* HTTP-Funktion. Diese wird zwar laut Dokumentation in Axios unterstützt, jedoch existieren hier Kompatibilitätsprobleme mit anderen Modulen, weswegen eine Implementierung dieser Funktion nicht möglich war. Dieses konkrete Problem ist auch von zahlreichen anderen Entwicklern in den GitHub Issues bereits vermerkt worden.

6. Offene Fragen

Während des Schreibens dieser Arbeit und in der Diskussion mit Kollegen kamen Fragen bezüglich der möglichen böswilligen Ausnutzung von webbasierten Entwicklungsumgebungen auf. Aufgrund der Tatsache, dass Web-IDEs gratis Rechenleistung zur Verfügung stellen, kann diese zu Gunsten des Benutzers und auf Kosten des Betreibers verwendet werden. Vor allem das Berechnen von Hash-Werten um Crypto zu schürfen stellt hier ein ernsthaftes Problem dar. Natürlich werden die zur Verfügung gestellten Ressourcen in Leistung und Zeit limitiert, aber bei vielen der analysierten Web-Compiler kann ohne Problem sofort eine neue Session gestartet werden. Da die alte Session bzw. das alte Docker-Environment vollständig gelöscht wird, müsste der Zwischenstand im letzten Schritt in einem eigens verwalteten System gespeichert werden. Der erste Schritt in der neuen Session wäre nun das Laden des Zwischenstands, gefolgt von der weiteren Berechnung von Hash-Werten. Hier wäre mehr Forschung bzw. ein Experiment notwendig, um diese theoretische Vorgehensweise in der Praxis zu beweisen.

7. Konklusion

Webbasierte Entwicklungsumgebungen bieten immense Vorteile sowohl für Anfänger als auch für Fortgeschrittene in der Programmierung. Egal ob in einem privaten Kontext oder bei einem Einsatz in öffentlichen Bildungsinstitutionen, können Web-IDEs die Produktivität fördern und den Einstieg in die Softwareentwicklung erleichtern. In meiner Bachelorarbeit habe ich verschiedene

Herangehensweisen an die Implementierung einer webbasierten Entwicklungsumgebung vorgestellt, welche je nach Anforderungen und Kontext Vorteile bieten. Unter anderem stellte ich WebAssembly als mögliche Technologie zur Auslagerung des Ausführprozesses im Kontext von Online-Compilern vor. Dieser Ansatz kann die Rechenlast des Betreibers reduzieren und bietet auch die Möglichkeit auf die normalerweise umfassenden Sicherheitskonzepte zu verzichten. Darüber hinaus analysierte ich zehn verschiedene bestehende webbasierte Entwicklungsumgebungen anhand ihrer UX und Sicherheit. Aus den Ergebnissen dieser Analyse kann man schließen, dass die Funktionalität von Online-IDEs noch weiter hinter der einer traditionellen lokalen Entwicklungsumgebung liegt. Natürlich ist dies eine nicht völlig unbewusste Entwicklung, da der Sinn und Zweck vieler der analysierten IDEs eine Simplifizierung des Kompilier- und Ausführprozesses ist. Funktionale Erweiterungen gehen immer mit einer erhöhten Komplexität des Systems einher und sind somit in diesem Kontext nur begrenzt sinnvoll. Dass weitere Entwicklungen hin zu einer vollständigen Simulierung einer lokal laufenden IDE möglich sind, steht nicht zur Debatte. Nicht zu vergessen ist die Notwendigkeit einer konstanten Netzwerkverbindung bei der Arbeit mit einer webbasierten Entwicklungsumgebung. Je nach Einsatzgebiet muss berücksichtigt werden, dass nicht von überall gearbeitet werden kann.

Im praktischen Teil habe ich eine webbasierte Entwicklungsumgebung selbst implementiert und dabei die Erkenntnisse des theoretischen Teils einfließen lassen. Der Fokus lag dabei auf der asynchronen Ausgabe des Outputs, einer REST-Schnittstelle im Backend, einfacher Erweiterbarkeit im Sinne eines wiederverwendbaren Prototyps und der Unterstützung einer Vielzahl von Programmiersprachen. Weiterführenden Konzepte bzw. Technologien, wie beispielsweise LSP, konnten angesichts einer aufwändigen Implementierung nicht im Rahmen dieser Bachelorarbeit untersucht werden.

Die grundsätzlichen Konzepte von webbasierten Entwicklungsumgebungen wurden in Kapitel 3. ausführlich erläutert. Es wurde sowohl auf grundsätzliche Architekturen als auch in der Theorie wichtige Funktionsweisen eingegangen. Nur wenige Frameworks und Tools erleichtern die prototypische Umsetzung von webbasierten Entwicklungsumgebungen. Der Tatsache geschuldet, dass es sich hier dennoch um einen Nischenbereich handelt, fehlt es an allgemeinen Lösungsansätzen. Hervorheben möchte ich jedoch die Code Execution Engines, die in Kapitel 3.2. genauer beschrieben wurden, welche die Implementierung um ein Vielfaches erleichtern. In Kapitel 4. wurde folglich RQ3 beantwortet und detailliert veranschaulicht, inwiefern sich bestehende webbasierte Entwicklungsumgebungen in ihrer Funktionalität unterscheiden.

8. Appendix

8.1. Link zu Code-Samples

Alle Code-Samples aus dem Kapitel 4. sowie das gesamte Projekt aus Kapitel 5. sind auf GitHub öffentlich zugänglich unter:

<https://github.com/brudi4550/WebIDE>

8.2. Detaillierte Ergebnisse der Technologie-Analyse

So gut wie alle untersuchten Services verwenden Dienste wie Google Analytics oder Google Tag Manager. Diese spielen jedoch für meine Arbeit keine Rolle und wurden demnach nicht in die Tabelle mitaufgenommen.

Ein Minus bedeutet, dass das jeweilige Tool keine konkreten Ergebnisse in der jeweiligen Kategorie liefern konnte.

Die Analyse-Tools lieferten vereinzelt widersprüchliche Ergebnisse, jedoch kann aus den Ergebnissen eine klare Tendenz zur Verwendung eines *JavaScript everywhere* Ansatzes erkannt werden. Wappalyzer [7] lieferte die detailliertesten Ergebnisse, sowie eine Einsicht in die verwendeten Text-Editoren.

Name des Services	Frontend	Backend	Weitere Frameworks/Technologien
OnlineGDB	Bootstrap	NodeJS, Express	-
WandBox	Remix	Remix	-
Judge0*	Semantic UI, jQuery	-	-
Tutorialspoint*	-	-	-
GeeksForGeeks	jQuery	-	-
Ideone	Bootstrap	PHP	-
Replit	NextJS	NextJS	-
JDoodle*	-	-	-
Paiza.IO*	Angular	-	-
CodeInterview	React, Bootstrap, jQuery, Babel	-	-

Tabelle 11 Ergebnisse der Analyse mit builtWith [5]

Name des Services	Frontend	Backend	Weitere Frameworks/Technologien
OnlineGDB	Bootstrap, jQuery	NodeJS	Wordpress
WandBox	-	Javascript	-
Judge0*	Bootstrap, jQuery	-	-
Tutorialspoint*	jQuery	PHP, ASP.NET, Java	-
GeeksForGeeks**	jQuery	NodeJS, PHP	Wordpress, Nginx, RequireJS
Ideone	-	PHP	-
Replit	-	NodeJS	-
JDoodle*	-	-	-
Paiza.IO*	Angular, jQuery, Bootstrap	-	-

CodeInterview	jQuery	PHP	Wordpress
---------------	--------	-----	-----------

Tabelle 12 Ergebnisse der Analyse mit W3Tech [6]

Name des Services	Frontend	Backend	Weitere Frameworks/Technologien
OnlineGDB	Bootstrap, jQuery,	NodeJS, Express, Socket.io	ACE-Editor
WandBox	Remix, React	Remix	-
Judge0*	Semantic UI, jQuery	-	Monaco Editor
Tutorialspoint*	jQuery	-	MathJax
GeeksForGeeks	Bootstrap, jQuery	PHP	ACE-Editor
Ideone	Bootstrap, jQuery	PHP	ACE-Editor
Replit	React, NextJS	NextJS, NodeJS	-
JDoodle*	Vue, jQuery	-	ACE-Editor
Paiza.IO*	Angular, Bootstrap	-	ACE-Editor
CodeInterview	React, jQuery	PHP	Monaco-Editor

Tabelle 13 Ergebnisse der Analyse mit Wappalyzer [7]

Alle Web-IDEs, die mit einem Sternchen markiert sind, befinden sich auf Subdomains, welche vom Analysetool nicht erkannt werden und stattdessen die Hauptdomain analysieren. Die Ergebnisse für diese Services sind möglicherweise nicht akkurat. Dieses Verhalten der Analyse-Tools ist nicht konstant, teilweise werden Subdomains auch als solche erkannt und korrekt analysiert.

Services, welche mit zwei Sternchen markiert wurden, verwenden Wordpress bzw. PHP auf ihrer Hauptdomain, die Subdomain bzw. die jeweilige IDE läuft über NodeJS. Das Tool analysiert in diesem Fall beides, macht im Ergebnis jedoch keine Unterscheidung, auf welcher Domain genau eine gewisse Technologie erkannt wurde.

9. Referenzen

- [1] H. T. Tran, H. H. Dang, K. N. Do, T. D. Tran, and V. Nguyen, "An interactive Web-based IDE towards teaching and learning in programming courses," in *Proceedings of 2013 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)*, Aug. 2013, pp. 439–444. doi: 10.1109/TALE.2013.6654478.
- [2] M. Leisner and P. Brune, "Good-Bye Localhost: A Cloud-Based Web IDE for Teaching Java EE Web Development to Non-Computer Science Majors," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, May 2019, pp. 268–269. doi: 10.1109/ICSE-Companion.2019.00108.
- [3] C. Yi, S. Feng, and Z. Gong, "A Comparison of Sandbox Technologies Used in Online Judge Systems," *Applied Mechanics and Materials*, vol. 490–491, pp. 1201–1204, Apr. 2014, doi: 10.4028/www.scientific.net/AMM.490-491.1201.
- [4] M. Mareš and B. Blackham, "A New Contest Sandbox," vol. 6, Jan. 2012.
- [5] "BuiltWith," *BuiltWith*. <https://builtwith.com/> (accessed Dec. 15, 2022).

- [6] “W3Techs - extensive and reliable web technology surveys.” <https://w3techs.com/> (accessed Dec. 15, 2022).
- [7] “Find out what websites are built with - Wappalyzer.” <https://www.wappalyzer.com/> (accessed Dec. 15, 2022).
- [8] “Ace - The High Performance Code Editor for the Web.” <https://ace.c9.io/> (accessed Dec. 17, 2022).
- [9] “Visual Studio Code - Code Editing. Redefined.” <https://code.visualstudio.com/> (accessed Dec. 17, 2022).
- [10] “Monaco Editor.” <https://microsoft.github.io/monaco-editor/> (accessed Dec. 17, 2022).
- [11] H. Z. Došilović, “Judge0 IDE - Free and open-source online code editor.” <https://ide.judge0.com/> (accessed Dec. 13, 2022).
- [12] H. Z. Došilović and I. Mekterović, “Robust and Scalable Online Code Execution System,” in *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, Sep. 2020, pp. 1627–1632. doi: 10.23919/MIPRO48935.2020.9245310.
- [13] “Language Commands, Judge0 CE.” <https://github.com/judge0/judge0/blob/master/db/languages/active.rb> (accessed Dec. 20, 2022).
- [14] “WebAssembly.” <https://webassembly.org/> (accessed Jan. 23, 2023).
- [15] W. Wang, “Empowering web applications with WebAssembly: are we there yet?,” in *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering*, Melbourne, Australia, Jun. 2022, pp. 1301–1305. doi: 10.1109/ASE51524.2021.9678831.
- [16] “Introduction - Swift and WebAssembly.” <https://book.swiftwasm.org/> (accessed Feb. 17, 2023).
- [17] “javac - Java programming language compiler.” <https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javac.html> (accessed Dec. 20, 2022).
- [18] “java.” <https://docs.oracle.com/javase/7/docs/technotes/tools/windows/java.html> (accessed Dec. 20, 2022).
- [19] M. S. Ul Haq, L. Liao, and M. Lerong, “Design and implementation of sandbox technique for isolated applications,” in *2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference*, May 2016, pp. 557–561. doi: 10.1109/ITNEC.2016.7560422.
- [20] L. Sinanaj, J. Ajdari, M. Hamiti, and X. Zenuni, “A comparison between online compilers: A Case Study,” in *2022 11th Mediterranean Conference on Embedded Computing (MECO)*, Jun. 2022, pp. 1–6. doi: 10.1109/MECO55406.2022.9797096.
- [21] “JetBrains: Essential tools for software developers and teams,” *JetBrains*. <https://www.jetbrains.com/> (accessed Dec. 20, 2022).
- [22] “Language Server Protocol Specification.” <https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/> (accessed Dec. 20, 2022).
- [23] “Git.” <https://git-scm.com/> (accessed Feb. 18, 2023).
- [24] “Build software better, together,” *GitHub*. <https://github.com> (accessed Feb. 18, 2023).
- [25] “GDB online Debugger | Compiler - Code, Compile, Run, Debug online C, C++,” *GDB online Debugger*. <https://www.onlinegdb.com/> (accessed Dec. 13, 2022).
- [26] “Wandbox.” <https://wandbox.org/> (accessed Dec. 13, 2022).
- [27] “Tutorialspoint.” <https://www.tutorialspoint.com/codingground.htm> (accessed Dec. 13, 2022).
- [28] “IDE | GeeksforGeeks | A computer science portal for geeks.” <https://ide.geeksforgeeks.org/> (accessed Dec. 13, 2022).
- [29] “Ideone.com,” *Ideone.com*. <https://ideone.com/> (accessed Dec. 13, 2022).
- [30] replit, “Replit: the collaborative browser based IDE,” *replit*. <https://replit.com/> (accessed Dec. 13, 2022).
- [31] “JDoodle - Online Compiler, Editor for Java, C/C++, etc.” <https://www.jdoodle.com> (accessed Dec. 13, 2022).
- [32] “Paiza.IO.” <https://paiza.io/> (accessed Dec. 13, 2022).
- [33] “CodeInterview - The Online Code Interview Tool & Code Editor,” *Code Interview*. <https://codeinterview.io/> (accessed Dec. 13, 2022).
- [34] “AWS Cloud9 – Amazon Web Services,” *Amazon Web Services, Inc.* <https://aws.amazon.com/de/cloud9/> (accessed Dec. 20, 2022).

- [35]“Visual Studio Code for the Web.” <https://vscode.dev/> (accessed Dec. 20, 2022).
- [36]“Docker: Accelerated, Containerized Application Development,” May 10, 2022. <https://www.docker.com/> (accessed Dec. 21, 2022).
- [37]S. Michel, “Fluxbox.” <http://fluxbox.org/> (accessed Feb. 20, 2023).
- [38]“Most used languages among software developers globally 2022,” *Statista*. <https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/> (accessed Feb. 23, 2023).
- [39]“Axios Documentation.” <https://axios-http.com/docs/intro> (accessed Jan. 24, 2023).
- [40]Mark Otto, Jacob Thornton, and Bootstrap, “Bootstrap.” <https://getbootstrap.com/> (accessed Jan. 24, 2023).
- [41]R. Fielding, “Architectural Styles and the Design of Network-based Software Architectures”; Doctoral dissertation,” 2000. Accessed: Dec. 27, 2022. [Online]. Available: <https://www.semanticscholar.org/paper/Architectural-Styles-and-the-Design-of-Software-Fielding/49fc9782483bc311c2bd1b902dfb32bcd99ff2d3>
- [42]“multer,” *npm*, May 30, 2022. <https://www.npmjs.com/package/multer> (accessed Feb. 15, 2023).
- [43]H. Nielsen *et al.*, “Hypertext Transfer Protocol – HTTP/1.1 - Specification,” Internet Engineering Task Force, Request for Comments RFC 2616, Jun. 1999. doi: 10.17487/RFC2616.

10. Abbildungsverzeichnis

Abbildung 1 Beispielhaftes NodeJS/Express Backend	8
Abbildung 2 HTML-Dokument in welchem Rust Code aufgerufen wird	9
Abbildung 3 Rust Code mit WebAssembly Annotation	9
Abbildung 4 Aufbau der REST-API	32
Abbildung 5 Startseite der Web-IDE	33
Abbildung 7 Beispiel einer Funktionskomponente in React	34
Abbildung 6 Beispiel einer Klassenkomponente in React	34

11. Verzeichnis der Code-Snippets

Code-Snippet 1 Testcode Asynchrone Ausgabe	15
Code-Snippet 2 Testcode Annahme von Input während der Laufzeit.....	16
Code-Snippet 3 Testcode Überprüfen der Berechtigungen	19
Code-Snippet 4 Java-Code Forkbomb.....	21
Code-Snippet 5 Testcode Einlesen einer CSV-Datei.....	22
Code-Snippet 6 Testcode Graphischer Output	24
Code-Snippet 7 Testcode Mainklasse mehrere Source-Code-Dateien	25
Code-Snippet 8 Abstrakte Java-Klasse <i>Animal</i>	25
Code-Snippet 9 Java-Klasse <i>Dog</i>	25
Code-Snippet 10 Testcode Umfang von Debugging-Funktionen.....	26
Code-Snippet 11 Testcode Autovervollständigung	27
Code-Snippet 12 Testcode statische Code Überprüfung.....	28

12. Tabellenverzeichnis

Tabelle 1 Ergebnisse Test auf asynchrone Ausgabe.....	16
-------------------------------------------------------	----

Tabelle 2 Ergebnisse Annahme von Input während der Laufzeit.....	17
Tabelle 3 Ergebnisse Annahme von Command Line Argumenten	18
Tabelle 4 Berechtigungen bei den verschiedenen Services	19
Tabelle 5 Ergebnisse des Test Lesen von Input aus non-Source-Code Dateien	23
Tabelle 6 Ergebnis Test mehrere Source-Code-Dateien	25
Tabelle 7 Ergebnisse Test Debugging möglich.....	26
Tabelle 8 Ergebnisse des Tests auf Autovervollständigung	27
Tabelle 9 Ergebnisse statische Code-Überprüfung	29
Tabelle 10 Zusammenfassende Ergebnisse der Online-IDEs	30
Tabelle 11 Ergebnisse der Analyse mit builtWith [5].....	37
Tabelle 12 Ergebnisse der Analyse mit W3Tech [6]	38
Tabelle 13 Ergebnisse der Analyse mit Wappalyzer [7].....	38

13. Abkürzungsverzeichnis

ACL.....	<i>Access Control List</i>
API.....	<i>Application Programming Interface</i>
CDN.....	<i>Content Delivery Network</i>
CE.....	<i>Code Execution</i>
IDE.....	<i>Integrated Developer Environment</i>
JSON	<i>JavaScript Object Notation</i>
LoC	<i>Lines of Code</i>
LS	<i>list (UNIX Command)</i>
Stdin	<i>Standard Input</i>
SU.....	<i>switch user (UNIX Command)</i>
UNIX.....	<i>UNiplexed Information Computing System</i>
UX.....	<i>User Experience</i>
VNC	<i>Virtual Network Computing</i>
VPS	<i>Virtual Private Server</i>