HAW HAMBURG

BACHELORTHESIS
Rodrigo Antonio Ehlers Terraza

Architektur von Web-IDEs in Hinblick auf Programmierplattformen

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Rodrigo Antonio Ehlers Terraza

Architektur von Web-IDEs in Hinblick auf Programmierplattformen

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. Jens von Pilgrim
Zweitgutachter: Prof. Dr. Axel Schmolitzky

Eingereicht am: 26. November 2021

Rodrigo Antonio Ehlers Terraza

Thema der Arbeit

Architektur von Web-IDEs in Hinblick auf Programmierplattformen

Stichworte

Software Architektur, IDE, Web-IDE, Programmierplattform, Docker, Kubernetes, Eclipse, Eclipse Theia, Eclipse Che, Gitpod

Kurzzusammenfassung

Web-IDEs bieten Entwickelnden kurzfristig einen unkomplizierten Zugang zu einer vollständigen Entwicklungsumgebung innerhalb eines Webbrowsers. Sie können Programmcodeausführung auf anderen Maschinen vereinfachen und eliminieren somit bestehende Einschränkungen lokal zur Verfügung stehender Ressourcen. Mittels einer Web-IDE soll eine Programmierplattform geschaffen werden, die zu Zwecken der Lehre eingesetzt werden soll. Diese soll Lehrende beim Stellen und Studierende beim Lösen von Aufgaben unterstützen. Aus eingangs erarbeiteten Anforderungen an die Programmierplattform, leiten wir in dieser Arbeit Anforderungen an eine Web-IDE und deren Architektur ab. Wir verschaffen uns einen Überblick über bestehende Web-IDEs, erarbeiten den Aufbau ihrer Architekturen und vergleichen diese hinsichtlich ihrer Eignung. Abschließend wählen wir eine Web-IDE aus und empfehlen diese für die Programmierplattform. Um die Eignung der ausgewählten Web-IDE zu verifizieren, entwickeln wir eine Erweiterung der Web-IDE und zeigen somit deren Erweiterbarkeit und Anpassbarkeit.

Rodrigo Antonio Ehlers Terraza

Title of Thesis

Architecture of Web-IDEs in regard to coding platforms

Keywords

Software architecture, IDE, Web-IDE, Programming platform, Docker, Kubernetes, Eclipse, Eclipse Theia, Eclipse Che, Gitpod

Abstract

Web-IDEs provide a quick and uncomplicated access to a fully functional development environment inside of a webbrowser to developers. They can simplify execution of code on a remote machine and therefore eliminate any existing limitations of local resources. Using such a Web-IDE a programming platform shall be created for educational purposes. Its goal is to support lecturers in creation and students in solving assignments. Using initially elaborated requirements for the programming platform, we derive requirements for the Web-IDE and its architecture in this thesis. We create an overview of exisiting Web-IDEs, elaborate their architectures and compare them in regard to their suitability. We are able to choose a Web-IDE and recommend it for use in the programming platform. To further verify the suitability of the chosen Web-IDE, we develop an extension of the Web-IDE and show its extendability and adaptability doing so.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

App Applikation.

CLI Command Line Interface.

CMS Course Management System.

CNCF Cloud Native Computing Foundation.

CPU Central Processing Unit.

CSS Cascading Style Sheets.

DAP Debug Adapter Protocol.

ECOOP European Conference on Object-Oriented Programming.

GUI Graphical User Interface.

GWT Google Web Toolkit.

HAW Hamburg Hochschule für Angewandte Wissenschaften Hamburg.

HMR Hot Module Replacements.

HTML Hypertext Markup Language.

IDE Integrated Development Environment.

IETF Internet Engineering Task Force.

JSON JavaScript Object Notation.

JSON-RPC JavaScript Object Notation Remote Procedure Call.

JSX JavaScript XML.

LIA Language-Independent Assignment.

LMS Learning Management System.

LSA Language-Specific Assignment.

LSP Language Server Protocol.

NCA No-Code Assignment.

OAuth 2.0 Open Authorization 2.0.

OOPSLA Object-Oriented Programming, Systems, Languages, and Applications.

OPPSEE Open Programming Platform for Software Engineering Education.

PoC Proof of Concept.

RAM Random Access Memory.

SSO Single Sign On.

UI User Interface.

UML 2.0 Unified Modelling Language 2.0.

VS Code Visual Studio Code.

WDC Web Developers Conference.

Web-IDE Web-based Integrated Development Environment.

1 Einleitung

Web-IDEs bieten Nutzerinnen und Nutzern schnell und unkompliziert eine vollständige und komplett funktionsfähige, integrierte Entwicklungsumgebung innerhalb eines Webbrowsers. Als Teil von Programmierplattformen können Web-IDEs teilweise erheblich zu reibungslosem Arbeiten sowie effektivem Lernen beitragen.

Mithilfe einer Web-IDE soll eine Programmierplattform entwickelt werden, auf der Studierende, in erster Instanz, Programmieraufgaben bearbeiten und lösen, welche anschließend von Lehrenden bewertet und abgenommen werden können. Langfristig sollen diese angestrebte Programmierplattform und die Web-IDE als Teil dieser im Funktionsumfang erweitert und angepasst werden können.

Dank einer solchen Programmierplattform und einer Web-IDE als Teil dieser, könnte bei der Einrichtung einer Entwicklungsumgebung eines Studierenden, im Vergleich zu einer klassischen Integrated Development Environment (IDE) viel Zeit eingespart werden. Studierende könnten sich deutlich früher auf die zu lösenden Aufgaben konzentrieren und somit ebenso deutlich früher zu Ergebnissen kommen.

Technische Anforderungen an Arbeitsgeräte von Studierenden könnten enorm sinken, da eine solche Programmierplattform in der Lage sein könnte, durch Studierende geschriebenen Programmcode auf einem anderen, ggf. deutlich leistungsstärkeren als dem eigenen Computer auszuführen.

Im Rahmen der Abschlussarbeit meines Bachelor of Science der Angewandten Informatik an der Hochschule für Angewandte Wissenschaften Hamburg werden wir die Architektur bestehender Web-IDEs erarbeiten und auf die Anforderungen des Fallbeispiels einer solchen Programmierplattform ausgerichtet vergleichen. Wir werden einen Überblick aktueller Web-IDEs erzeugen und eine Empfehlung geben, welche dieser Web-IDEs am besten die Anforderungen des Fallbeispiels erfüllt.

1.1 Motivation

Wie gerade bereits kurz angeschnitten, liegen die Vorteile der Nutzung einer Programmierplattform wie der angestrebten, im Vergleich zur Entwicklung mit einer klassischen IDE vor allem darin, Zeit und Ressourcen beim Einrichten von Entwicklungsumgebungen verschiedener Projekte zu sparen und Studierenden dadurch die Bearbeitung ihrer Aufgaben erheblich zu vereinfachen.

Ein weiterer wesentlicher Vorteil der Nutzung einer Web-IDE gegenüber einer klassischen IDE ist die Möglichkeit der zentralen Kontrolle. Lehrenden könnte es möglich sein, gleiche technische Bedingungen für Studierende zu schaffen und so Nachteilen einzelner Studierender, die durch falsche Konfiguration ihrer eigenen Maschinen entstehen können, entgegen zu wirken.

Um eine Web-IDE für solch eine Programmierplattform auszuwählen, ist es nötig, verschiedene, bereits bestehende, Web-IDEs zu untersuchen und zu evaluieren. Hierbei werden wir beim Vergleich den Fokus auf die Architektur der Web-IDEs legen, da diese enorme Auswirkungen auf die Anpassbarkeit, Erweiterbarkeit sowie die Wartung und Instandhaltung der angestrebten Programmierplattform hat. Wir werden Anforderungen und Randbedingungen erarbeiten, die für die Web-IDE und deren Architektur gelten und im Vergleich auf die Einhaltung dieser achten.

Ziel ist es, in dieser Abschlussarbeit eine der bereits bestehenden und auf dem Markt angebotenen Web-IDEs, unter Beachtung der entsprechenden Anforderungen und Randbedingungen, auszuwählen und für eine solche Programmierplattform empfehlen zu können. Mit diesem Ziel geht die Erarbeitung diverser Architekturbeschreibungen und Architekturdiagramme einiger bestehender Web-IDEs einher, da diese für einen wissenschaftlichen Vergleich der Architekturen unabdingbar sind.

Im Anschluss möchten wir dann auf die Anpassbarkeit und Erweiterbarkeit der ausgewählten Web-IDE eingehen und beispielhaft zeigen, wie diese Web-IDE entsprechend den Anforderungen der Programmierplattform erweitert werden kann.

1.2 Forschungsstand

Da es sich bei Web-IDEs und einigen für Web-IDEs benötigten Technologien um eine relativ neue Art der Software handelt, gibt es aktuell kaum wissenschaftliche Arbeiten zu

diesem Themenbereich. Gerade im Bereich der Architektur dieser und deren Ausrichtung auf eine Programmierplattform, war es uns nicht möglich, relevante wissenschaftliche Arbeiten ausfindig zu machen.

Es existiert jedoch das wissenschaftliche Paper *Towards an Online Programming Platform Complementing Software Engineering Education* [64] (im Folgenden: Paper). Die Autoren haben eine bereits bestehende Programmierplattform als Teil eines Projekts an der Hochschule für Angewandte Wissenschaften Hamburg (HAW Hamburg) im Jahre 2019 mit Studierenden genutzt. Sie beschreiben die Nutzung der Programmierplattform als Erfolg, zeigen jedoch auch einige Defizite der Plattform auf und nennen eine starke Nachfrage nach einer Programmierplattform, die genannte Defizite behebt. Um solch eine Programmierplattform zu entwickeln, definieren die Autoren Stakeholder und deren Anforderungen an die Programmierplattform und analysieren anschließend existierende Programmierplattformen auf eben diese Anforderungen hin. Das Paper wird innerhalb dieser Abschlussarbeit maßgeblich die Anforderungen an die angestrebte Programmierplattform und entsprechend auch an die Web-IDE und deren Architektur vorgeben.

Des Weiteren haben es sich bereits einige Unternehmen zur Aufgabe gemacht, Web-IDEs und Programmierplattformen für Entwickelnde zugänglich zu machen. Einige Produkte dieser Unternehmen werden wir in Kapitel 4.1 näher betrachten.

1.3 Vorgehen

Zu Beginn werden wir fachliche Anforderungen an die Programmierplattform aus dem wissenschaftlichen Paper [64] aufarbeiten. Zu diesen fachliche Anforderungen werden wir nach Betrachtung einiger Randbedingungen weitere fachliche Anforderungen hinzufügen. Aus der Gesamtheit der fachlichen Anforderungen, werden wir dann technische Anforderungen an die Web-IDE und deren Architektur ableiten. Diese technischen Anforderungen werden die Basis dafür bilden, die fachlichen Anforderungen abbilden zu können.

Basierend auf den abgeleiteten technischen Anforderungen und weiteren Randbedingungen, wird es uns möglich sein, schrittweise einzelne Web-IDEs aus einer Menge potenzieller Web-IDEs für die angestrebte Programmierplattform auszusortieren.

Besonders Randbedingungen werden zu Beginn eine große Rolle spielen, da diese zwingend einzuhalten sind und eine Web-IDE meist klar und ohne großen Aufwand auf die Einhaltung einer Randbedingung überprüft werden kann. Diese Tatsache ermöglicht es

uns, selbst eine große Menge verschiedener Web-IDEs kurzfristig zu verkleinern und unsere Auswahl so einzuschränken. So können wir aufzeigen, welche der verschiedenen Web-IDEs nicht als Teil der angestrebten Programmierplattform in Frage kommen und diese Web-IDEs entsprechend aussortieren.

Unsere technischen Anforderungen werden erst im Anschluss an das grobe Filtern durch Randbedingungen relevant, um aus der dann verkleinerten Menge an potentiell passenden Web-IDEs, weitere Web-IDEs auszusortieren oder ganz im Gegenteil, die Eignung einer bestimmten Web-IDE aufzuzeigen.

Bevor es jedoch um das tatsächliche, schrittweise Filtern und Aussortieren einzelner Web-IDEs gehen kann, muss eine solche Menge erst als Überblick vorhandener, auf dem Markt angebotener Web-IDEs geschaffen werden. Hierzu müssen wir Web-IDEs finden und sammeln sowie im Anschluss für einzelne dieser Web-IDEs detaillierte Informationen erarbeiten.

Die Suche und das Sammeln wird durch eine Internetrecherche erfolgen.

Das Erarbeiten detaillierter Informationen einzelner Web-IDEs wird dann größtenteils durch Heranziehen entsprechender Ressourcen wie Dokumentationen oder Programmcode dieser einzelnen Web-IDEs erfolgen. Sollte es für einen genaueren Einblick nicht genügen auf theoretischer Ebene zu bleiben und nötig werden, durch das Einrichten einer Web-IDE auf einer echten Maschine, in eine praktische Ebene zu wechseln, werden wir dies ebenfalls tun.

Um einen Vergleich durchführen zu können, reicht es nicht, detaillierte Informationen einzelner Web-IDEs bloß zu erarbeiten. Es ist nötig diese detaillierten Informationen so aufzubereiten, dass wir die entsprechenden Web-IDEs auf einer tatsächlich vergleichbaren Basis betrachten können. Dazu gehört es, bspw. Architekturdiagramme aus verschiedenen, in der Softwarearchitektur relevanten Ansichten zu erzeugen.

Da wir für das Erzeugen solcher Architekturdiagramme enorm viele Informationen über die Architektur einer Web-IDE benötigen und das Erarbeiten dieser Informationen mit großem Aufwand verbunden sein kann, wird es uns erst möglich sein, dies zu tun, nachdem wir die Menge der potentiell passenden Web-IDEs deutlich reduziert haben. Die Architekturdiagramme können dann als Hilfsmittel genutzt werden um eine informierte Entscheidung zu treffen und diese im Anschluss zu bekräftigen.

Zuletzt möchten wir die Anpassbarkeit und die Erweiterbarkeit der ausgewählten Web-IDE zeigen. Hierfür müssen wir definieren, an welcher Stelle eine Anpassung oder Erweiterung entsprechend der Anforderungen an die angestrebte Programmierplattform Sinn ergibt.

1.3.1 Methodik

Zur Anforderungsanalyse werden wir aus fachlichen Anforderungen technische Anforderungen ableiten, da nur solche für die Architektur der gesuchten Web-IDE relevant sind.

Wir werden selbst keine Web-IDE entwickeln und somit in maßgeblichen Teilen deren Architektur keine Entwurfsfreiheit haben, sondern uns ausschließlich auf bereits entwickelte Web-IDEs und entsprechend bereits festgelegte Architekturen beschränken und diese betrachten. Das „Reverse Engineering“ wird als Methodik in unserer Untersuchung bereits festgelegter Architekturen eine wichtige Rolle spielen.

Um Randbedingungen und einzelne, gröbere Anforderungen vergleichen zu können und einen Überblick zu schaffen, eignet sich eine Tabelle der Web-IDEs mit den entsprechenden Randbedingungen und Anforderungen.

Um die in Kapitel 1.3 genannte, vergleichbare Basis aus den erarbeiteten detaillierten Informationen zu schaffen und komplexe Abhängigkeiten innerhalb der Architektur einzelner Web-IDEs darstellen zu können, werden wir die ebenfalls in Kapitel 1.3 genannten, in der Softwarearchitektur relevanten Sichten in Form von Architekturdiagrammen durch die Modellierungssprache Unified Modelling Language 2.0 (UML 2.0) [98] erzeugen und in diese Arbeit mit aufnehmen. Zu relevanten Sichten gehören die Verteilungssicht und die grobe sowie die detaillierte Komponentensicht.

Zuletzt wird das Einrichten der ausgewählten Web-IDE auf einer echten Maschine nötig sein, um unsere Auswahl zu bekräftigen und die Anpassbarkeit und Erweiterbarkeit der ausgewählten Web-IDE zu zeigen.

Reverse Engineering

Reverse Engineering steht für das Erarbeiten bestimmter, teilweise verdeckter, Eigenschaften einer Software durch bloßes Betrachten, Nutzen und dem Ziehen entsprechen-

der Schlussfolgerungen. Hierzu gehören bspw. das Aufteilen einer Software in einzelne, geordnete Komponenten und deren Beziehungen oder das Ableiten oder Übersetzen bestimmter Teile einer Software, wie der Architektur, in abstrakte Formen wie Diagramme [20, S. 15].

Eine laufende Software wird für das Reverse Engineering nicht zwingend benötigt. Es ist durchaus möglich, bereits weit vor der Betrachtung der Software selbst [20, S. 15], bspw. durch Dokumentation dieser Software, einige Schlüsse ziehen zu können, die im Anschluss bei weiterer Betrachtung und Untersuchung nützlich sein können. Hierbei haben die Genauigkeit und der Stand der Dokumentation einen großen Einfluss darauf, was für Informationen durch das Reverse Engineering gewonnen werden können. Eine veraltete Dokumentation, also die Dokumentation einer früheren Version einer bestimmten Software, könnte durchaus sehr detailliert sein und viele Informationen bereitstellen, die jedoch für das Reverse Engineering einer aktuelleren Version dieser Software ggf. nicht von Relevanz sein könnten, falls es zwischen den beiden Versionen bereits gravierende Änderungen der Software gab. Andersherum würde eine sehr aktuelle, jedoch unvollständige oder vage, Dokumentation einer Software ggf. nur wenige relevante Informationen bereitstellen. Die Qualität, gemessen an Aktualität und Genauigkeit einer Dokumentation, spielt beim Reverse Engineering durch Dokumentation also eine große Rolle.

1.3.2 Aufbau

Die Abschlussarbeit wird sich grob in 4 relevante Teile gliedern lassen. Die Anforderungsanalyse (1), die Suche und den Vergleich (2), die Evaluation und die Auswahl (3) und letztendlich die Erweiterung und Anpassung (4).

Die Anforderungsanalyse (1) ist in Kapitel 3 zu finden. Die Suche und der Vergleich (2) sowie die Evaluation und die Auswahl (3) in Kapitel 4 und die Erweiterung und Anpassung (4) in Kapitel 5.

Anforderungsanalyse

In der Anforderungsanalyse werden wir Anforderungen, welche durch die Programmierplattform an die Web-IDE gestellt werden, sammeln und einordnen. Diese werden, wie im Kapitel 1.2 genannt, primär durch das wissenschaftliche Paper [64] vorgegeben. Des

Weiteren werden wir Randbedingungen in Betracht ziehen, die für die angestrebte Programmierplattform und entsprechend für die Web-IDE, durch Stakeholder vorgegeben werden. Diese Anforderungen und Randbedingungen werden wir dann für die folgenden Kapitel aufbereiten, um entsprechend mit Hilfe dieser Anforderungen informierte Entscheidungen treffen zu können.

Suche und Vergleich

Bei der Suche und dem Vergleich geht es darum, bereits existierende, kommerziell oder nicht-kommerziell angebotene Web-IDEs oder ähnliche Software zu suchen und zu sammeln, durch Randbedingungen einzugrenzen und auf einer einheitlichen, tatsächlich vergleichbaren Basis zu vergleichen. Als „einheitliche, vergleichbare Basis“ bezeichnen wir hierbei die umgewandelte Form verfügbarer Informationen einzelner Web-IDEs in die gleiche Sprache. Genauer bedeutet dies, die Möglichkeit zu schaffen, die gleichen Fragen für verschiedene Web-IDEs beantworten zu können.

Evaluation und Auswahl

Nachdem die Anzahl der potentiell passenden Web-IDEs verringert wurde und eine einheitliche, vergleichbare Basis für die übrig gebliebenen Web-IDEs geschaffen wurde, müssen wir entsprechend den Anforderungen aus Kapitel 3 begründet entscheiden, welche der Web-IDEs am besten zur angestrebten Programmierplattform passt. Hierzu werden wir unter anderem versuchen die Web-IDEs auf einer echten Maschine einzurichten und unsere Erfahrung neben weiteren erarbeiteten Informationen in die Evaluation mit aufnehmen.

Erweiterung und Anpassung

Die Erweiterung und Anpassung der ausgewählten Lösung in Richtung der angestrebten Programmierplattform folgt zum Schluss. Welche Art der Erweiterung oder der Anpassung wir in diesem Teil durchführen werden, wird sich im Laufe der folgenden Kapitel ergeben, da die Art dieser Erweiterung oder Anpassung sehr stark von der letztendlich ausgewählten Web-IDE und deren Architektur abhängt.

Ziel dieses Teils ist es eine Art Prototypen oder auch „Proof of Concept“ einer bestimmten Funktion oder eines Bestandteils der angestrebten Programmierplattform zu entwerfen und diesen Entwurf dann umzusetzen. Außerdem eignet sich eine erfolgreiche Umsetzung einer solchen Erweiterung gut, um unsere Entscheidung aus Kapitel 4 zu bekräftigen, da diese die Erweiterbarkeit und Anpassbarkeit der ausgewählten Web-IDE zeigt.

2 Grundlagen

Um das Verständnis der kommenden Kapitel zu vereinfachen, werden wir im Folgenden auf einige Grundlagen des Themenbereichs eingehen. Dazu gehören unter anderem Programmierplattformen, IDEs, die Rolle, die solche IDEs in der Softwareentwicklung und seit kurzem auch im Web spielen, sowie Course Management Systems (CMS) als auch einige weitere relevante Technologien.

2.1 Programmierplattformen

Als eine Programmierplattform definieren wir, wie das wissenschaftliche Paper [64] auch, eine webbasierte Plattform auf der Aufgaben verschiedener Schwierigkeitsgrade gestellt werden, die mittels der Eingabe von Programmcode in verschiedenen unterstützten Sprachen, gelöst werden können.

> „We define an [..] [Online Programming Platform; Anmerk. d. Verf.] as a web-based platform that offers several assignments of varying complexity, a possibility to enter code that solves an assignment, and an automatic feedback mechanism for the proposed solutions.“ [64, S. 27, Kapitel 1]

Programmcode aus Lösungsvorschlägen wird durch einen automatischen Feedbackmechanismus überprüft und validiert. Ergebnisse werden ebenfalls automatisch angezeigt.

2.2 Integrated Development Environments

IDEs sind Anwendungen, die den Nutzen haben, Entwickelnde bei der Entwicklung weiterer Anwendungen zu unterstützen. Primäre Funktionen solcher IDEs sind neben der Eingabe des entsprechenden Programmcodes, unter anderem Programmcodevervollständi-

gung, statische Programmcodeanalyse und daraus resultierende sogenannte Quick-Fixes [83].

Diese Funktionen sind sehr nützlich, so können Quick-Fixes zum Beispiel die Entwicklung einer Anwendung erleichtern oder verbessern, indem sie nach der statischen Programmcodeanalyse sinnvolle Vorschläge geben können, um bestimmte Teile des Programmcodes zu verändern. Diese Veränderungen können Fehler bereits vor der eigentlichen Ausführung des Programmcodes vermeiden oder Performance der Laufzeit steigern. In den meisten Fällen können diese Vorschläge auch direkt von der IDE im entsprechenden Teil des Programmcodes für eine Vielfalt verschiedener Programmiersprachen syntaktisch korrekt umgesetzt werden [83].

Der Nutzen einer IDE liegt demnach offensichtlich darin, zum einen die Produktivität und die Effizienz Entwickelnder, zum anderen aber auch die tatsächliche Qualität der Arbeit dieser zu erhöhen. Heutzutage finden diverse IDEs in dem Berufsfeld der Softwareentwicklung weltweit reichlich Anwendung, erfreuen sich hoher Beliebtheit unter Entwickelnden und können als nicht mehr wegzudenkender Standard der Branche betitelt werden [84].

IDEs können in zwei Gruppen eingeteilt werden: Klassische IDEs und IDEs im Web.

2.2.1 Klassische Integrated Development Environments

Bei klassischen IDEs handelt es sich um Anwendungen, welche heruntergeladen und installiert werden müssen, um genutzt werden zu können. Das Wort „Klassisch “ im Namen, ist kein offizieller Term. Wir nennen diese IDEs lediglich so, da es sich bei diesen um Anwendungen der klassischen Art handelt, die wie beschrieben richtig auf einem Gerät installiert werden müssen.

Wie die klassische IDE auch, müssen weitere Abhängigkeiten für die Softwareentwicklung ebenfalls lokal auf dem Gerät der Nutzenden installiert sein, um von einer klassischen IDE erkannt und genutzt werden zu können. Zu diesen Abhängigkeiten können unter anderem Datenbanken und Programmiersprachen selbst gehören.

Beispiele von IDEs

Einige bekannte IDEs mit Schwerpunkt auf Entwicklung in der Programmiersprache Java sind Eclipse und IntelliJ IDEA, welche von der Eclipse Foundation und JetBrains entwickelt werden. Ein Beispiel einer IDE ohne direkten Fokus auf eine Programmiersprache und mit großem Spektrum an unterstützten Programmiersprachen ist Visual Studio Code, welche Open Source ist und größtenteils von Microsoft entwickelt wird [93].

2.2.2 Integrated Development Environments im Web

Neben solchen klassischen IDEs gibt es mittlerweile auch webbasierte IDEs, die lediglich einen Webbrowser und Zugriff zum Internet zur Nutzung benötigen. Eine solche Web-IDE kann ohne weitere Softwareinstallationen, durch das bloße Aufrufen einer URL innerhalb des Webbrowsers genutzt werden [46].

Webbasierte IDEs erlauben es den Nutzenden nicht nur auf die eigene Maschine, sondern auch von der Ferne aus, auf andere Maschinen zuzugreifen und auf diesen zu entwickeln. Dies heißt konkret, dass der Zugriff auf eine bereits existierende Entwicklungsumgebung ermöglicht wird.

Beispiele von Web-IDEs

Zu den bekanntesten Web-IDEs [18] gehören Amazon Web Services Cloud 9 [10], Repl.it [103] und CodeAnywhere [21], auf die wir in den folgenden Kapiteln auch eingehen werden.

2.3 Course Management Systems

CMS, nicht zu verwechseln mit Learning Management Systems (LMS), sind Anwendungen, die von Lehrenden dazu genutzt werden können, bestimmte Teile eines Kurses oder einer Veranstaltung zu verwalten [1, S. 6]. Diese Kurse und Veranstaltungen finden meist an einer Bildungseinrichtung statt. Dazu gehören das Bereitstellen von Informationen oder Dateien über Downloads sowie das Stellen von Aufgaben, die bspw. direkt in einem webbasierten Graphical User Interface (GUI) des CMS durch Mehrfach- oder Einzelauswahl sowie manchmal auch durch Eingabe von Freitext beantwortet werden können.

Manche CMS bieten auch die Möglichkeit, Programmcode zur Lösung einer gestellten Aufgabe einzugeben, welcher dann teilweise automatisiert durch vorgegebene Tests auf Korrektheit überprüft werden kann [96].

Wie kurz angeschnitten, bieten CMS ein webbasiertes GUI an, durch welche sowohl Studierende als auch Lehrende Zugriff auf diese erhalten. In den allermeisten Fällen wird dafür, ähnlich wie bei webbasierten IDEs, lediglich ein Webbrowser und Zugriff auf das Internet benötigt.

Abgrenzung zu Programmierplattformen

Im Gegensatz zu CMS liegt der Schwerpunkt der angestrebten Programmierplattform nicht auf der Verwaltung von Kursen oder Veranstaltungen in Bildungseinrichtungen, sondern viel mehr direkt auf dem praktischen Lösen von Aufgaben, im Bereich der Informatik, speziell der Softwareentwicklung, durch Eingabe von Programmcode.

Trotz der relativ unterschiedlichen Schwerpunkte von CMS und der angestrebten Programmierplattform soll die Programmierplattform teilweise Funktionen von CMS übernehmen. Hierzu gehört bspw. das Stellen und Lösen von Aufgaben ohne Programmcode, auf die wir in Kapitel 3.3.2 eingehen werden.

2.4 Relevante Technologien

Im Bereich der Softwarevirtualisierung und Softwareorchestrierung, sowie dem Bereich der Webentwicklung gibt es einige Technologien, die für das Verständnis des Aufbaus der Architektur und der Verteilung von Web-IDEs relevant sind. An dieser Stelle werden wir auf fünf dieser Technologien eingehen, die besonders bedeutsam sind.

2.4.1 JavaScript und Node.js

JavaScript ist eine Programmiersprache, die 1995 ursprünglich für den Einsatz in Webbrowsern von Brendan Eich, damals noch unter dem Namen LiveScript, entwickelt wurde. Ein Jahr später wurde sie als Teil des Webbrowsers Netscape 2 durch Netscape veröffentlicht [97]. Im selben Jahr wurde JavaScript Ecma International [57] vorgelegt, welche dann den ECMAScript Standard [58] entwickelte [97].

JavaScript gehört zu den meist genutzten und beliebtesten Programmiersprachen der Welt [19] und ist einer der wichtigsten Bestandteile des Internets wie es heutzutage bekannt ist.

Die Programmiersprache ist schwach typisiert und wird von Webbrowsern zur Laufzeit interpretiert. Entsprechend muss JavaScript im Normalfall nicht durch einen Compiler vor Ausführung kompiliert werden [97].

Im Jahr 2009 wurde die JavaScript Laufzeitumgebung Node.js für den Einsatz außerhalb des Webbrowsers durch Ryan Dahl entwickelt [115]. Durch Node.js wurde die Nutzung von JavaScript außerhalb des Webbrowsers auf lokalen Maschinen und Servern populär.

2.4.2 Docker

Docker ist eine Software, die es auf Basis bereits vorhandener Softwarevirtualisierungsmöglichkeiten einzelner Betriebssysteme ermöglicht, andere Software als sogenannte Docker Images isoliert zu erzeugen. In der Regel beinhalten Docker Images jegliche Softwareabhängigkeiten, wie Laufzeiten, Konfigurationsdateien und Ressourcen, die benötigt werden, um die entwickelte Software laufen lassen zu können [29]. Dies ist ebenso der Fall, wenn die verpackte Software aus mehreren Teilen besteht. Docker erlaubt es all diese Teile als Docker Images zu verpacken. Docker Images können auf weiteren Docker Images aufbauen, sind jedoch nach Erzeugung unveränderbar und dienen lediglich als Vorlage für Docker Container oder Basis für weitere Docker Images.

Um ein Docker Image laufen lassen zu können, wird ein Docker Container benötigt. Docker Container basieren auf Docker Images und können als lauffähige Versionen dieser bezeichnet werden [29].

Standardmäßig sind Docker Container isoliert. Dies bedeutet unter anderem, dass jegliche Kanäle zur Kommunikation nach außen hin, bereits beim Erzeugen des Docker Images auf dem ein Docker Container basiert, jedoch spätestens beim Starten eines Docker Containers, explizit festgelegt werden müssen [29]. Hierbei spielt es keine Rolle, ob es um Kommunikation über das Netzwerk oder lokal zur Maschine, auf welcher der Docker Container läuft, geht [29]. Durch korrekte Einstellung solcher Kanäle zur Kommunikation nach außen hin, können entsprechend auch mehrere Docker Container untereinander

kommunizieren. Es ist demnach nicht möglich, dass Software, die innerhalb eines Containers läuft, während der Laufzeit ihre eigenen Kommunikationswege nach außen erweitert, falls dies nicht explizit beim Erzeugen oder Starten des Containers erlaubt wird.

Das Erzeugen lauffähiger Software mit entsprechender Laufzeitumgebung und allen Abhängigkeiten dieser sowie der Software selbst in Docker Images bringt eine Vielzahl an Vorteilen mit sich. Es ermöglicht einen enorm vereinfachten Transport entsprechender Docker Images zwischen verschiedenen Maschinen über verschiedene Wege wie das Internet oder Festplatten [31]. Es erlaubt außerdem durch genutzte Virtualisierungstechnologien eine starke Annäherung der Laufzeitbedingungen von Docker Containern des gleichen Docker Images zwischen diesen, teils sehr verschiedenen, Maschinen [29].

Um ein Docker Image zu erzeugen, wird in den meisten Fällen, eine spezielle Datei mit dem Dateinamen „Dockerfile“ und ein Kontext benötigt, welcher ein Ordner auf dem lokalen Dateisystem oder eine per URL erreichbare Git Repository sein kann [27].

Docker Registries

Um eben genannten Transport von Docker Images über das Netzwerk zu vereinfachen, bieten sogenannte Docker Registries eine Plattform als zentrale Sammelstelle von Docker Images an. Docker Registries ermöglichen es bereits erzeugte Docker Images unter einem frei wählbaren, jedoch einzigartigen, Namen hochzuladen und zu veröffentlichen, um diese dann für Dritte oder einen selbst verfügbar zu machen [30]. Durch die direkte Integration von Docker Registries in Docker, können Docker Images sehr leicht durch die Angabe des Namens in einem Dockerfile für weitere Docker Images als Basis genutzt werden [32].

Es gibt eine Vielzahl an öffentlichen Docker Registries und auch eine hauseigene Docker Registry namens Docker Hub [28]. Es besteht jedoch ebenfalls die Möglichkeit, selbst eine Docker Registry auf eigener Infrastruktur zu erzeugen. Diese kann nach Belieben öffentlich oder privat sein und entsprechend auf die Bedürfnisse eines Unternehmens oder einer Bildungseinrichtung angepasst und dort genutzt werden [26].

2.4.3 Kubernetes

Kubernetes ist eine ursprünglich von Google entwickelte Open Source Software, welche die Verwaltung von Softwarecontainern vereinfachen soll. Sie automatisiert wichtige Prozesse der Bereitstellung einer Anwendung, die aus mehreren Softwarecontainern besteht.

Dazu gehören das Bereitstellen der Anwendung und entsprechend der Softwarecontainer an sich sowie die Skalierung und die Überwachung einzelner Softwarecontainer [112]. Des Weiteren bietet Kubernetes eine deklarative Schnittstelle zur Konfiguration an, welche sprachunabhängig ist.

Google spendete Kubernetes, nach Gründung der Cloud Native Computing Foundation (CNCF) durch die Linux Foundation und zeitgleich zur Veröffentlichung der Version 1.0 von Kubernetes, an die CNCF [105]. Kubernetes wird oft auch als K8s abgekürzt [112].

Innerhalb Kubernetes bilden sogenannte Pods die kleinstmöglichen von Kubernetes verwaltbaren Einheiten. Pods können aus einem oder mehreren Softwarecontainern bestehen und können entsprechend als kleinstmögliche lauffähige Einheiten gesehen werden. Kubernetes verwaltet demnach keine tatsächlichen Softwarecontainer, sondern lediglich die Pods innerhalb welcher Softwarecontainer ausgeführt werden [110].

Im Kontext von Kubernetes bezeichnet ein Node eine Maschine, also meist einen Server. Für Kubernetes spielt es keine Rolle, ob dieser Server eine echte physische Maschine ist oder lediglich virtualisiert wurde. Nodes bilden die Laufzeitumgebung für Pods. Hierbei können, je nach Kapazität eines Nodes, mehrere Pods auf dem selben Node laufen. Die Kapazität bezeichnet hierbei den Status von Ressourcen, wie bspw. der Central Processing Unit (CPU) und dem Random Access Memory (RAM). Kubernetes stellt sicher, dass nur so viele Pods auf dem selben Node laufen, wie dieser auch in Betracht seiner Kapazitäten und Ressourcen unterstützen kann [109].

Um Pods mitsamt deren Softwarecontainern auf einem Node laufen lassen zu können, muss Kubernetes Virtualisierungs- und Softwarecontainertechnologien unterstützen. Kubernetes stellt bei Installation entsprechende Dienste auf den Nodes zur Verfügung [109].

Kubernetes verteilt Pods automatisch auf verschiedenen Nodes, um die Fehlertoleranz einer Anwendung zu erhöhen. Sollte innerhalb eines Pods oder gar einem Node und entsprechend allen Pods, die auf diesem Node laufen, ein Fehler auftreten, so verringert Kubernetes durch die genannte Verteilung von Pods auf Nodes die Wahrscheinlichkeit eines Totalausfalls [108]. Die Verteilung von Pods auf verschiedenen Nodes wird als horizontale Skalierung bezeichnet.

Da Docker als bekannteste Virtualisierungs- und Softwarecontainerplattform gilt, wird Kubernetes sehr oft zusammen mit Docker genutzt [110]. Jedoch unterstützt Kubernetes auch andere Virtualisierungs- und Softwarecontainerplattformen wie „containerd“ und „CRI-O“. [107].

2.4.4 OAuth 2.0

Open Authorization 2.0 (OAuth 2.0) ist ein Authentifizierungs- und Autorisierungsprotokoll, welches von der Internet Engineering Task Force (IETF) OAuth Working Group entwickelt wird. Es ist sehr weit verbreitet und könnte daher schon als Industriestandard bezeichnet werden [101].

OAuth 2.0 basiert auf der Nutzung von Tokens, welche für verschiedene Zwecke wie bspw. die Verifizierung einer Zugriffsberechtigung auf gesicherte Ressourcen genutzt werden. Diese Tokens können je nach Belieben angepasst werden, um zum Beispiel eine Ablaufzeit eines Tokens zu implementieren [79, S. 35]. Eine solche Ablaufzeit wird vom Protokoll empfohlen [79, S. 35].

Der wesentliche Unterschied zu Authentifizierungsmethoden und Authentifizierungsprotokollen aus Zeiten vor OAuth 2.0 liegt darin, dass Webseiten und Applikationen (Apps), welche Dienste anderer Apps nutzen, durch die Nutzung von OAuth 2.0 nicht mehr dazu gezwungen sind, die Zugangsdaten eines Nutzenden für entsprechende andere Apps abzufragen und gegebenenfalls sogar zu speichern. Stattdessen können diese Apps die Nutzenden zu den Apps weiterleiten, deren Dienste sie nutzen wollen. Dies setzt natürlich voraus, dass sowohl die Apps, die bestimmte Dienste nutzen wollen, als auch die Apps, deren Dienste sie nutzen wollen, OAuth 2.0 komplett unterstützen.

Nach der Weiterleitung können Nutzende sich dann auf den dafür vorgesehenen GUIs entsprechender Apps authentifizieren und den angefragten Zugriff erlauben. Ist dies erfolgreich, so erhält die anfragende App einen Token (Authorization Grant), der die erteilte Erlaubnis bescheinigt [79, S. 7, Abb. 1].

Mittels dieses Tokens, kann die anfragende App mit der App, deren Dienste sie benötigt, kommunizieren um Zugriffstoken (Access Token) für benötige Ressourcen zu erstellen [79, S. 7, Abb. 1]. Diese Zugriffstokens werden dann für den eigentlichen Zugriff auf die benötigen Ressourcen genutzt [79, S. 10, Kapitel 1.4].

Der große Vorteil von OAuth 2.0 liegt auf der Hand: Zugangsdaten einer App müssen nicht mehr mit anderen Apps geteilt werden, um bestimmte Funktionen abbilden zu können. Dies schafft nicht nur Vertrauen, sondern minimiert ebenfalls das Risiko des Bekanntwerdens bestimmter Zugangsdaten durch bspw. Hackerangriffe, da Zugangsdaten auf weniger echten Servern verteilt liegen.

3 Anforderungen

Um Anforderungen an die gesuchte Web-IDE, jedoch primär an deren Architektur, stellen zu können, müssen wir vorerst die Anforderungen an die angestrebte Programmierplattform erarbeiteten. Im Anschluss wird es uns möglich sein, von diesen Anforderungen entsprechende fachliche Anforderungen an die Web-IDE abzuleiten, welche letztendlich die Basis dafür bilden, technische Anforderungen an deren Architektur stellen zu können.

An die angestrebte Programmierplattform gestellte Anforderungen werden im Rahmen dieser Arbeit primär durch das Paper [64] definiert. Da es sich bei den Anforderungen des Papers [64] vor allem um fachliche Anforderungen handelt, müssen technische Anforderungen von diesen abgeleitet werden.

Bevor es jedoch um diese Anforderungen geht, werden wir uns noch mit einigen Randbedingungen befassen.

3.1 Randbedingungen

Als Randbedingungen bezeichnen wir hierbei relativ unspezifische, meist technische Gegebenheiten, die einzelne Web-IDEs erfüllen müssen, um für die angestrebte Programmierplattform in Betracht gezogen werden zu können. Sie bilden Grundvoraussetzungen für die Web-IDEs die wir im Folgenden betrachten werden.

Im Rahmen dieser Arbeit werden wir Randbedingungen berücksichtigen, die vom Open Programming Platform for Software Engineering Education (OPPSEE) Team der Hochschule für Angewandte Wissenschaften Hamburg [104] vorgegeben wurden.

Diese Randbedingungen gelten entsprechend als final festgelegt und können nicht mehr verändert werden. Dies mag im ersten Moment einschränkend wirken, jedoch eignen sich

Randbedingungen bestens, um einen groben Filter zur Erstauswahl der möglichen Web-IDEs zu bilden. Sobald es dann in folgenden Kapiteln detaillierter um die Architektur einzelner Web-IDEs geht, werden wir nicht mehr auf Randbedingungen eingehen müssen.

Es gelten folgende vier Randbedingungen im Bereich der Flexibilität und der Lizenzierung für die Web-IDE, die letztendlich Teil der angestrebten Programmierplattform werden soll.

3.1.1 Flexibilität

Da zum Zeitpunkt der Abschlussarbeit noch nicht vollständig geklärt ist, welchen Umfang die angestrebte Programmierplattform in Zukunft haben soll, sollten wir Wert darauf legen, eine Web-IDE zu finden, bei deren Entwicklung Flexibilität eine wichtige Rolle gespielt hat. Im Folgenden nennen wir zwei Voraussetzungen, die eine gute Basis für Flexibilität bilden.

Open Source

Als Open Source wird laut der Open Source Initiative [100] der Programmcode bestimmter Software bezeichnet, welcher unter anderem öffentlich verfügbar ist und von jeder beliebigen Person benutzt, verändert und geteilt werden kann.

Als Hochschule und Einrichtung der Bildung kommen für die HAW Hamburg nur solche Web-IDEs in Frage, deren Programmcode Open Source ist. Dies erlaubt einen gesamten Einblick in den Programmcode der Web-IDE und damit Transparenz und eine gute Basis, um Rückschlüsse auf die Architektur zu ziehen. Des Weiteren werden beliebte Open Source Projekte meist von einer diversifizierten Gemeinschaft verschiedener Entwickelnder gepflegt, was oft zur Vermeidung von Fehlern eines einzelnen führt.

Lock-In Effekt

Der sogenannte Lock-In Effekt bezeichnet eine Situation in welcher der Aufwand, im Sinne von Zeit und Kosten, eines Anbieterwechsels für Nutzende wirtschaftlich zu hoch sind [113], um in Betracht gezogen zu werden. Eine Situation, die dem Lock-In Effekt unterliegt, kann bspw. durch einen Anbietenden erzeugt werden, in dem bestimmte proprietäre

Standards durch diesen festgelegt werden [113]. Hierdurch haben Anbietende den Vorteil, dass der Wechsel eines Nutzenden zu anderen Anbietenden sehr unwahrscheinlich oder sogar unmöglich ist.

Um Flexibilität zu gewährleisten, müssen wir darauf achten, dass es nicht zu einer Situation mit Lock-In Effekt kommen kann. Kann eine Web-IDE nur in einer bestimmten, proprietären Umgebung laufen, so muss diese frühestmöglich aussortiert werden.

3.1.2 Lizenzierung

Die Web-IDE soll zu Bildungszwecken an der HAW Hamburg, als Teil der angestrebten Programmierplattform, Studierenden zur Verfügung gestellt und von diesen genutzt werden. Eine solche Verteilung und Nutzung muss durch eine entsprechende Lizenzierung erlaubt sein. Des Weiteren wird die Web-IDE noch vor Nutzung mit hoher Wahrscheinlichkeit angepasst werden müssen, was bedeutet, dass der Programmcode verändert werden könnte. Auch dies muss durch eine entsprechende Lizenzierung erlaubt sein.

3.1.3 Technologien

Das OPPSEE Team gibt vor, dass die angestrebte Programmierplattform mittels Docker und Kubernetes bereitgestellt werden soll. Entsprechend müssen wir darauf achten, dass unsere Web-IDE innerhalb eines Docker Containers bereitgestellt werden kann.

3.2 Fachliche Anforderungen

Im Folgenden werden wir die wichtigsten fachlichen Anforderungen, die die Autoren des wissenschaftlichen Papers [64] an die angestrebte Programmierplattform stellen, erarbeiten. Anschließend werden wir von diesen passende technischen Anforderungen an die Web-IDE und deren Architektur ableiten.

Hierbei müssen wir bedenken, dass nur solche Anforderungen im Rahmen dieser Arbeit relevant sind, welche letzten Endes, wenn auch nur teilweise, durch die Architektur der Web-IDE der Programmierplattform abgebildet werden oder auf welche die Architektur dieser Web-IDE einen Einfluss hat.

3.2.1 Stakeholder

Die Autoren nennen im Paper [64] vier verschiedene Stakeholder [64, S. 29 - 30, Kapitel 3, Teil A]: *Students* (Studierende), *Lecturer* (Lehrende), *Developer* (Entwickelnde) und *Administrator* (Administrierende). All diese stellen jeweils verschiedene Anforderungen an die angestrebte Programmierplattform und damit auch an deren Web-IDE.

3.2.2 Anforderungen der Stakeholder

Einzelne Anforderungen der Stakeholder wurden im Paper durch die Autoren mit Kürzeln gekennzeichnet. Die Kürzel bestehen aus drei Buchstaben, einem Minus zur Trennung und zwei Ziffern. Ein Beispiel eines solchen Kürzels ist STD-01. Hierbei stehen die drei Buchstaben „STD “ für eine Abkürzung des Stakeholders zu dem eine Anforderung zugeordnet wird. Die Ziffern hingegen dienen lediglich der Nummerierung. Wir werden diese Kürzel im Folgenden nutzen, sofern wir uns direkt auf Anforderungen des Papers beziehen.

Studierende

Als Studierende werden aus Sicht der Autoren Nutzende der Programmierplattform bezeichnet, die die Programmierplattform hauptsächlich dafür nutzen, auf dieser gestellte Aufgaben zu lösen. Hierbei können die Studierenden laut der Autoren in zwei Gruppen geteilt werden. Zum einen gibt es Studierende, die die Plattform, bspw. zur Erlangung ihres Abschlusses, zwingend nutzen müssen, zum anderen gibt es Studierende, die die Plattform freiwillig nutzen. Erstere können laut der Autoren erneut in zwei Untergruppen geteilt werden.

Zum einen teilen die Autoren die Studierenden, die die Plattform unfreiwillig nutzen, in solche, die hoch motiviert sind und deren Ziel es, neben der Erlangung ihres Abschlusses, ist, neue Fähigkeiten zu lernen und ihre bestehenden Fähigkeiten zu erweitern (Untergruppe 1). Zum anderen in solche, deren Ziel es lediglich ist, die Programmierplattform insofern zu nutzen, wie es zur Erlangung des Abschlusses zwingend notwendig ist (Untergruppe 2) [64, S. 29 - 30, Kapitel 3, Teil A, Punkt 1].

Aus Sicht der Studierenden, werden von den Autoren insgesamt dreizehn Anforderungen and die angestrebte Programmierplattform gestellt.

Die Autoren haben einen starken Fokus auf die Benutzererfahrung der Studierenden gelegt. Sie legen viel Wert auf das GUI und visuelles Feedback, welche Aufgaben attraktiver machen (STD-01 und STD-02) und die Einführung in die Programmierplattform vereinfachen (STD-03) sollen.

Den Autoren ist es außerdem wichtig, dass die Programmierplattform die Möglichkeit bietet, bereits vorhandene Nutzerkonten der Studierenden zur Authentifizierung zu nutzen (STD-04). Dies trägt zu einem geringen Aufwand während der Einrichtung der Programmierplattform seitens der Studierenden bei, was ebenfalls eine Anforderung der Autoren an die Programmierplattform ist (STD-05).

Um Studierende bestmöglich bei der Lösung der Aufgaben zu unterstützen, soll die Programmierplattform Studierende bei der Eingabe von Programmcode insofern unterstützen, als es klassische IDEs auch tun. Hiermit sind Funktionen wie Syntax Highlighting und intelligente Programmcodevervollständigung durch statische Programmcodeanalyse gemeint (STD-05). Dies soll primär Studierende der Untergruppe 1, die gegebenenfalls bereits Erfahrung mit klassischen IDEs gemacht haben, ansprechen.

Durch dynamischen Hilfestellungen während der Bearbeitung einzelner Aufgaben soll Studierenden im Problemfall ein Denkanstoß gegeben werden können (STD-06). Hier geht es insbesondere darum, Studierende der Untergruppe 2 zu unterstützen. Eine Anzeige des Fortschritts der Bearbeitung einer Aufgabe (STD-07) soll Studierenden einen Überblick geben. Mit diesen beiden Anforderungen legen die Autoren erneut auf das GUI Wert.

Den genannten Funktionen klassischer IDEs aus STD-05, die die Programmierplattform ebenfalls unterstützen soll, fügen die Autoren das Leisten von Hilfestellung bei Syntaxfehlern, das Ausführen und die damit einhergehende Evaluation geschriebenen Programmcodes (STD-08 und STD-09) hinzu.

Zusammenarbeit verschiedener Studierender zur Lösung mehrerer oder einzelner Aufgaben (STD-10 und STD-11) sowie die Integration von Software zur Versionierung von Programmcode, wie bspw. Git, soll ebenfalls möglich sein (STD-12). Zuletzt sollen Studierende in der Lage sein, Programmcode ihrer Kommilitonen einzusehen und überprüfen zu können (STD-13).

Alle referenzierten Kürzel sind aus dem Paper [64, S. 30, Kapitel 3, Teil B, Punkt 1].

Lehrende

Als Lehrende bezeichnen die Autoren Nutzende der Programmierplattform, welche die Aufgaben stellen, die letztendlich von Studierenden gelöst werden sollen. Lehrende möchten die Plattform nutzen, um den Umfang ihrer Kurse zu erweitern und deren Qualität zu steigern. Sie möchten Studierenden die Möglichkeit geben, in einem Kurs erlerntes Wissen durch das Bearbeiten und Wiederholen von Aufgaben zu festigen oder sogar zu vertiefen [64, S. 30, Kapitel 3, Teil A, Punkt 2].

Die Autoren stellen zehn Anforderungen an die Programmierplattform aus Sicht eines Lehrenden.

Ähnlich wie auch für Studierende, soll die Nutzung der Plattform für Lehrende ohne viel Aufwand verbunden sein (LCR-01). Der Fokus der Plattform soll auf der Bearbeitung praktischer Programmieraufgaben durch Studierende liegen (LCR-02), jedoch soll die Plattform ebenfalls einige Aufgaben eines CMS übernehmen. Dazu gehören unter anderem das Verwalten von Kursteilnehmenden (Studierenden) und Teams aus Kursteilnehmenden sowie das Verwalten von Aufgaben eines Kurses (LCR-03). Lehrenden soll es ebenfalls möglich sein untereinander Aufgaben zu teilen (LCR-04). Neben der Verwaltung des Kurses soll es Lehrenden ebenfalls möglich sein, Studierenden kursspezifische Informationen zur Verfügung zu stellen, die über Aufgaben hinaus gehen (LCR-05).

Um den Lehrenden mehr Kontrolle über die Bearbeitung von Aufgaben zu geben, sollen diese bestimmte Restriktionen der Bearbeitung festlegen können. Beispiele dafür sind die Reihenfolge, in welcher Studierende Aufgaben bearbeiten müssen, oder aber zeitliche Begrenzungen für die Bearbeitung einzelner Aufgaben (LCR-06).

Es soll drei verschiedene Arten von Aufgaben geben. Aufgaben, die Studierende einzeln bearbeiten, Aufgaben, bei denen Studierende mit einander verglichen werden und in eine Art Wettbewerb kommen und Aufgaben, die von mehreren Studierenden als Gruppe bearbeitet werden können (LCR-07). Diese können jedoch jeweils noch in eine von zwei Unterarten geteilt werden: Language-Independent Assignments (LIAs) und Language-Specific Assignments (LSAs) (LCR-08). Diese Unterarten unterscheiden sich vor allem in der Art der Auswertung ihrer Lösungen. Lehrende sollen hierbei bestimmte Metriken bewerten und gewichten können (LCR-09). Auf diese Unterarten der Aufgaben und ihre Unterschiede werden wir noch genauer eingehen.

Zuletzt soll es Lehrenden noch möglich sein, den Fortschritt einzelner Studierender jederzeit einsehen zu können (LCR-10).

Alle referenzierten Kürzel sind aus dem Paper [64, S. 30 - 31, Kapitel 3, Teil B, Punkt 2].

Entwickelnde

Entwickelnde der Programmierplattform können laut der Autoren in zwei Gruppen geteilt werden: Entwickelnde (Gruppe 1) der Programmierplattform selbst und Entwickelnde der Aufgaben (Gruppe 2). Entwickelnde der Gruppe 1 sind primär damit beschäftigt, die Plattform selbst weiterzuentwickeln. Dazu gehört das Hinzufügen von Funktionen und das Beheben von Fehlern. Entwickelnde der Gruppe 2 beschäftigen sich mit der Implementierung einzelner Aufgaben. Somit stellen Entwickelnde der Gruppe 1 den Rahmen für die Arbeit von Entwickelnden der Gruppe 2 [64, S. 30, Kapitel 3, Teil A, Punkt 3].

Aus Sicht der Entwickelnden stellen die Autoren vier Anforderungen an die Programmierplattform. Diese werden im Paper von DVL-02 bis DVL-05 nummeriert.

Die Autoren möchten, dass die Programmierplattform es möglich macht, abgeschlossene Arbeiten an der Programmierplattform ohne zutun der Nutzenden (Studierende und Lehrende) auszuliefern (DVL-02). Außerdem sollen einzelne Komponenten der Programmierplattform isoliert voneinander ausgeführt und getestet werden können (DVL-03).

Entwickelnden der Gruppe 1 soll es möglich sein, Entwickelnden der Gruppe 2 einen Rahmen für die Entwicklung von Aufgaben innerhalb der Programmierplattform bieten zu können. Entwickelnde der Gruppe 2 sollen in der Lage sein können, Aufgaben für die Programmierplattform zu entwickeln, ohne grundlegende Bestandteile der Programmierplattform verändern zu müssen. Für Interaktionen sollen durch den Rahmen klar definierte Schnittstellen zur Verfügung gestellt werden (DVL-04).

Zuletzt soll es Entwickelnden der Gruppe 2 möglich sein, fertiggestellte Aufgaben Nutzenden jederzeit zur Verfügung stellen zu können, ohne den Programmcode der Programmierplattform bearbeiten zu müssen (DVL-05).

Alle referenzierten Kürzel aus dem Paper [64, S. 31, Kapitel 3, Teil B, Punkt 3].

Administrierende

Administrierende der Programmierplattform sind laut Autoren für die Auslieferung der Programmierplattform verantwortlich. Sie stellen Ressourcen in Form von Maschinen und Netzwerkbandbreite zur Verfügung [64, S. 30, Kapitel 3, Teil A, Punkt 4].

Aus Sicht der Administrierenden stellen die Autoren fünf Anforderungen an die Programmierplattform.

Um die Stabilität der Plattform gewährleisten zu können, soll fremder Programmcode, der ausgeführt werden soll, isoliert in einer Art Sandbox-Umgebung ausgeführt werden können (ADM-01). Fremder Programmcode wäre bspw. die erarbeitete Lösung eines Studierenden in Form von Programmcode. Solch ein Programmcode soll nicht nur isoliert von der Programmierplattform ausgeführt werden, sondern auch isoliert von anderem fremden Programmcode ausgeführt werden (ADM-02). Das heißt, dass bspw. die erarbeiteten Lösungen zweier verschiedener Studierender isoliert von einander ausgeführt werden sollen.

Wird ein fremder Programmcode, also bspw. eine durch Studierende erarbeitete Lösung, in einer Sandbox-Umgebung ausgeführt, so soll eine bestimmte Ausführung nicht von vorherigen Ausführungen abhängen (ADM-03). Die Sandbox-Umgebung muss also nach jeder Ausführung korrekt abgebaut und vor jeder Ausführung erneut unabhängig aufgebaut werden können.

Es soll möglich sein, Ressourcen die einer solchen Sandbox-Umgebung zur Ausführung fremden Programmcodes zur Verfügung stehen, begrenzen zu können. Außerdem soll es jederzeit möglich sein, eine Sandbox-Umgebung abbauen zu können und damit die Ausführung fremden Programmcodes, unabhängig vom Status der Ausführung, stoppen zu können (ADM-04).

Zuletzt soll es möglich sein, fehlerhafte Ausführungen durch bestimmte Maßnahmen zu erkennen und abbrechen zu können oder fehlerhaften Programmcode noch vor dessen Ausführung zu erkennen (ADM-05). Zu solchen Maßnahmen gehört unter anderem die statische Programmcodeanalyse des fremden Programmcodes.

Alle referenzierten Kürzel aus dem Paper [64, S. 31, Kapitel 3, Teil B, Punkt 4].

3.3 Fachliche Anforderungen an die Web-IDE

Nachdem wir nun die Anforderungen der Autoren aus Sicht aller Stakeholder gelesen haben, wird schnell klar, worauf die Autoren des Papers wert legen. Wir möchten im Folgenden diese Anforderungen an die Programmierplattform nun in Anforderungen an die Web-IDE übersetzen und teilweise weitere Anforderungen hinzufügen.

3.3.1 Benutzererfahrung

Zum einen legen die Autoren viel Wert auf die Benutzererfahrung von Studierenden und Lehrenden. Das Einrichten einer klassischen IDE und teilweise mehrerer Programmiersprachen auf der eigenen Maschine, welches oft mit dem Download großer Dateien und manchmal komplexen manuellen Installationsschritten verbunden ist, soll so gut wie möglich umgangen werden. Mittels der Authentifizierung an der Programmierplattform durch bereits bestehende Nutzerkonten (Single Sign On) wird die Barriere zur Nutzung der Programmierplattform ebenfalls gesenkt.

Zum anderen ist es den Autoren wichtig, dass die Einarbeitungszeit in die Programmierplattform gering gehalten wird. Eine klares und intuitives GUI und ein ordentlicher On-Boarding Prozess für Studierende und Lehrende sollen hierbei helfen.

Mithilfe dieser Punkte wird es Studierenden möglich sein, innerhalb von kürzester Zeit mit dem eigentlichen Fokus der Programmierplattform, nämlich dem Erarbeiten von Lösungen für praktische Programmieraufgaben, zu beginnen.

Zu einer positiven Benutzererfahrung der Studierenden, primär der Studierenden der Untergruppe 1, trägt ebenfalls die Unterstützung grundlegender Funktionen klassischer IDEs seitens der Web-IDE bei. Um Studierenden das Gefühl zu geben, sie würden mit einer klassischen IDE arbeiten, ist es wichtig, Funktionen klassischer IDEs, wie Syntax Highlighting, intelligentes Feedback zum Programmcode durch statische Programmcodeanalyse und automatische Vervollständigung des Programmcodes, innerhalb der Web-IDE anbieten zu können.

Durch die Unterstützung visuellen Feedbacks als Teil des GUIs der Programmierplattform, möchten die Autoren es Lehrenden möglich machen, Studierenden Denkanstöße zu geben. Entsprechend muss es möglich sein, innerhalb des GUIs der Web-IDE visuelle Elemente dynamisch hinzuzufügen, zu verändern oder zu entfernen.

Um Studierenden letztendlich die Möglichkeit zu geben, geschriebenen Programmcode zu testen, muss die Web-IDE die Ausführung des Programmcodes auf Anfrage unterstützen.

3.3.2 Umfang

Die Autoren des Papers nennen drei Aufgabenarten, die jeweils noch in zwei Unterarten geteilt werden können. Diese Unterarten sind LIAs und LSAs, die wir in Kapitel 3.2.2 genannt haben. Die Autoren unterscheiden hierbei zwischen Aufgaben, die auf eine bestimmten Programmiersprache beschränkt sind und stark von dieser abhängen, und solchen, die dies nicht tun. Diesen beiden Unterarten von Aufgaben werden wir im Rahmen dieser Arbeit eine weitere Art, No-Code Assignments (NCAs), hinzufügen. Auf alle drei gehen wir im Folgenden ein und erarbeiten erneut resultierende Anforderungen.

Language-Independent Assignments

In anderen bereits bestehenden Programmierplattformen, werden erarbeitete Lösungen von Aufgaben oftmals durch den Eingang und Ausgang serialisierter Daten automatisiert evaluiert. Hierbei ist es für die automatisierte Evaluation nicht von Relevanz, welche Programmiersprache genutzt wird. Die Schnittstelle für die Übertragung von serialisierten Daten zwischen der Laufzeit der erarbeiteten Lösung und der Laufzeit der automatisierten Evaluation wird durch den Standardeingang und Standardausgang realisiert. Es wird klargestellt, welche Datenstruktur die erarbeitete Lösung zur Laufzeit empfangen wird und welche Datenstruktur diese wieder ausgeben soll.

Aufgaben, die in dieser Form gestellt und evaluiert werden können, nennen wir genau wie die Autoren des Papers, LIAs.

Die Verarbeitung serialisierter Daten gehört zu den grundlegenden Aufgaben einer Programmiersprache. Entsprechend steht es Entwickelnden von Programmierplattformen frei, welche Programmiersprache sie für den automatisierten Evaluationsprozess von LIAs nutzen und wie genau sie diesen umsetzen. Diese Freiheit der Auswahl spricht für die Unterstützung von LIAs bei der Entwicklung einer Programmierplattform. Außerdem vereinfacht sie auch die Implementierung der automatischen Evaluierung einzelner LIAs.

Ein weitere Vorteil von LIAs ergibt sich auf Seite der Studierenden, welche sich, soweit von der Programmierplattform unterstützt, ebenfalls für eine beliebige Programmiersprache zur Lösung einer Aufgabe der Unterart LIA aussuchen können. Hierbei muss die ausgewählte Programmiersprache lediglich die Verarbeitung serialisierter Daten unterstützen.

Language-Specific Assignments

Bei den LSAs, handelt es sich anders als bei den LIAs um eine Unterart von Aufgaben, die auf eine bestimmte Programmiersprache beschränkt und stark von dieser abhängig sind. LSAs können nicht wie LIAs mit einer beliebigen Programmiersprache gelöst werden, sondern geben genau vor, welche Programmiersprache zur Lösung genutzt werden muss.

Hierbei kann anders als bei den LIAs neben serialisierten Daten über den Standardeingang und Standardausgang auch auf umfangreichere Möglichkeiten zur automatisierten Evaluation einer erarbeiteten Lösung zurückgegriffen werden. Ein Beispiel solcher Möglichkeiten sind Unit Tests oder Integration Tests. Sowohl Unit als auch Integration Tests müssen jedoch in der gleichen Programmiersprache geschrieben werden wie der Programmcode, den sie testen sollen. Dies bedeutet im Umkehrschluss, dass Entwickelnde der Gruppe 2 sich anders als bei den LIAs, nicht für eine beliebige Programmiersprache entscheiden können, um die automatisierte Evaluation einer Aufgabe der Unterart LSA zu schreiben.

LSAs sind entsprechend schwieriger in einer Programmierplattform zu unterstützen als LIAs. Sie können nicht ohne Weiteres auf alle Programmiersprachen, die innerhalb der Programmierplattform unterstützt werden, erweitert werden, da jede automatisierte Evaluation für jede der unterstützten Programmiersprachen erneut geschrieben werden muss.

Durch die starke Abhängigkeit von einer bestimmten Programmiersprache entsteht zwar ein großer Aufwand bei der Implementierung und der Instandhaltung von LSAs, jedoch kommt dieser Aufwand auch mit einigen Vorteilen. Das vertiefte Verständnis während der automatisierten Evaluation ermöglicht es dieser, bspw. viel genauer auf die Einhaltung vorgegebener Beschränkungen einer Programmiersprache hin zu überprüfen. Lehrenden ist es so möglich, LSAs sehr präzise zu stellen und eingereichte Lösungen ebenso präzise automatisiert evaluieren zu lassen, um diese dann im Anschluss zu bewerten.

Eingereichte Lösungen verschiedener Studierender könnten dann bspw. durch präzise Laufzeitmessungen bewertet werden. Es könnte die Lösung ermittelt werden, durch welche die gestellte Aufgabe am effizientesten gelöst wurde. Diese könnte dann wiederum zur Lehrveranstaltung beigetragen werden und somit allen Studierenden zur Verfügung stehen, was die Qualität der Lehrveranstaltung erhöhen könnte.

Ein weiterer interessanter Vorteil von LSAs ist die Vorgabe oder die Restriktion spezifischer Schnittstellen einer Programmiersprache. Würde in der Programmiersprache Java ein LSA gestellt werden, so könnte hierbei die Nutzung von abstrahierten Schnittstellen wie `List` aus dem `java.util` Paket und Klassen, welche diese implementieren, untersagt werden. Studierende müssten dann entsprechend auf primitivere Datenstrukturen wie einen Array (`[]`) zurückgreifen und könnten somit lernen, wie abstrahierte Schnittstellen implementiert werden können. Ähnliches könnte ebenso auf weitere Programmiersprachen angewandt werden.

Mit der Unterstützung von LIAs und LSAs, würde die Anforderung LCR-08 der Autoren aus dem Paper [64] erfüllt sein.

No-Code Assignments

Für Lehrende kann es ebenfalls interessant sein Aufgaben zu stellen, für deren Lösung es keines Programmcodes bedarf. Entsprechend sollte die Programmierplattform die Möglichkeit bieten, dies zu tun. Diese Unterart der Aufgaben nennen wir NCAs.

NCAs sind Aufgaben, die in Textform gestellt werden. Ein NCA besteht aus mehreren Unteraufgaben, die verschiedene Möglichkeiten der Beantwortung zulassen. Lehrende können Einzel- oder Mehrfachauswahl bereitgestellter Antworten zur Lösung anbieten oder ein Textfeld bereitstellen, in welchem Studierende eine Unteraufgabe frei beantworten können. Die Möglichkeit bestimmte Dateien zur Lösung einer Unteraufgabe hochzuladen könnte in einem weiteren Schritt ebenfalls interessant sein.

NCAs ähneln damit einer Art Aufgaben, die aus CMS bekannt sind. Da wir mit der angestrebte Programmierplattform jedoch auch Teile eines CMS abbilden möchten, ergibt es Sinn NCAs bei der Ausarbeitung der Anforderungen an die Programmierplattform mit aufzunehmen.

Resultierende Anforderungen

Um die drei beschriebenen Arten von Aufgaben, LIAs, LSAs und NCAs in der Programmierplattform unterstützen zu können, müssen wir erarbeiten, ob durch diese weitere Anforderungen an die Web-IDE, speziell der Architektur der Web-IDE, gestellt werden.

Die assistierte Eingabe von Programmcode durch Studierende (STD-05), die Hilfestellung bei Syntaxfehlern (STD-08) und das automatische Evaluieren des eingegebenen Programmcodes (STD-09) sowie dynamisches visuelles Feedback innerhalb des GUIs (STD-02 und STD-06) wurden bereits als Anforderungen gestellt. Diese Anforderungen bilden die Basis für Aufgaben die durch Programmcodeeingabe (LIAs und LSAs) gelöst werden.

Die Unterstützung von Aufgaben der Unterart LIA stellt keine weiteren als die genannten Anforderungen an die Architektur der Web-IDE.

Da LSAs präzisere Kontrolle über den Programmcode einer eingereichten Lösung benötigen, benötigen wir hier die Möglichkeit, solch eine Kontrolle zu erhalten. Diese Anforderung wurde jedoch ebenfalls durch die Möglichkeit der Beschränkung einzelner Programmiersprachen auf bestimmte Bestandteile (LCR-09) bereits genannt. Entsprechend stellt die Unterstützung von Aufgaben der Unterart LSA, genau wie die Unterstützung von Aufgaben der Unterart LIA auch, keine Anforderungen an die Architektur der Web-IDE, die nicht bereits genannt wurden.

Da wir bereits die Möglichkeit der Anzeige neuer visueller Element und Anpassung des GUIs als Anforderungen gestellt haben (STD-02), steht auch der Unterstützung von NCAs nichts im Wege. NCAs erfordern ein spezielles GUI, um die alternativen Möglichkeiten der Eingabe von Lösungen abzubilden.

Jedoch müssen wir berücksichtigen, dass es möglich sein muss, Aufgaben komplett ohne Programmcodeeingabe abbilden zu können. Zwingende Integration mit Software zur Versionierung von Programmcode darf NCAs nicht im Wege stehen.

3.3.3 Anpassbarkeit und Erweiterbarkeit

Zum aktuellen Zeitpunkt gibt es neben den genannten Anforderungen keinen genau festgelegten Funktionsumfang der angestrebten Programmierplattform. Wir möchten den

Funktionsumfang nicht schon verfrüht einschränken und müssen daher darauf achten, dass die Web-IDE anpassbar und erweiterbar ist. Entsprechend sollen uns Architektur und Technologien der Web-IDE bei einer Anpassung oder Erweiterung nicht im Wege stehen.

3.4 Technische Anforderungen

Aus den fachlichen Anforderungen des letzten Kapitel 3.2, werden wir in diesem Kapitel technische Anforderungen erarbeiten. Ziel ist es, möglichst alle fachlichen Anforderungen durch technische Anforderungen abbilden zu können. Erreichen wir dieses Ziel, so werden wir einen guten Überblick bilden können, welche technischen Anforderungen und Voraussetzungen eine Web-IDE erfüllen muss, um für die angestrebte Programmierplattform in Frage zu kommen. Die technischen Anforderungen werden uns also die Möglichkeit geben einen Filter zu bilden.

Neben den technischen Anforderungen, die wir aus den fachlichen Anforderungen erarbeiten werden, müssen wir einige weitere technische Anforderungen an die Programmierplattform und damit auch an die Web-IDE und deren Architektur stellen. Diese weiteren Anforderungen werden im Bereich der Skalierbarkeit, der Abhängigkeiten sowie der Anpassbarkeit und der Erweiterbarkeit liegen und ergeben sich zum einen aus den Randbedingungen aus Kapitel 3.1 und wurden zum anderen vom OPPSEE Team festgelegt.

3.4.1 Benutzererfahrung

Um unsere fachlichen Anforderungen des Bereichs der Benutzererfahrung aus Kapitel 3.3.1 mittels unserer Web-IDE umsetzen zu können, werden wir folgende technische Anforderungen an die Web-IDE und deren Architektur stellen.

- Webbrowserbasiert
- Single Sign On
- Language Server Protocol
- Laufzeitumgebung und Dateisystem

- Debug Adapter Protocol

Wir werden auf jede dieser technischen Anforderungen im Einzelnen eingehen und auch in folgenden Kapiteln auf diese Bezug nehmen.

Webbrowserbasiert

Da eine Web-IDE dem Namen nach bereits zwingend webbrowserbasiert ist, können wir diese technische Voraussetzung bereits als gegeben sehen. Einige unserer fachlichen Anforderungen lassen sich sogar schon durch diesen Fakt abbilden.

Webbrowser eignen sich zur Umsetzung visueller Elemente (STD-02 und STD-06) sehr, da von den meisten Webbrowsern Programmiersprachen, die speziell für die Umsetzung von GUIs konzipiert wurden, unterstützt werden. Zu diesen Programmiersprachen gehören die Hypertext Markup Language (HTML), die Cascading Style Sheets (CSS) und JavaScript. Sie sind für Layout, Aussehen und Funktion von GUIs einer Vielzahl von Anwendungen im Web verantwortlich und können daher als weit verbreiteter Industriestandard bezeichnet werden.

Des Weiteren kann heutzutage davon ausgegangen werden, dass Studierende, gerade Studierende der Fakultät Technik und Informatik, ein Gerät besitzen oder durch ihre Universität oder Hochschule zur Verfügung gestellt bekommen können, welches Zugang zu einem modernen Webbrowser und zum Internet hat. Innerhalb dieses Webbrowsers dann eine URL aufzurufen, um die Plattform zu erreichen, ist wohl einer der trivialsten Wege, welcher mit enorm wenig Aufwand verbunden ist. Dies erfüllt STD-05 und LCR-01.

Single Sign On

Als Single Sign On (SSO) wird die Möglichkeit bezeichnet, Nutzende verschiedener Softwaresysteme an einem zentralen Punkt zu authentifizieren und diese Authentifizierung dann für alle einzelnen dieser Softwaresysteme bei Bedarf nutzen zu können. Im Kapitel 2.4.4 gehen wir auf OAuth 2.0 ein. OAuth 2.0 kann zur Umsetzung eines SSO Systems genutzt werden.

Einige Universitäten und Hochschulen, wie auch die HAW Hamburg [81] oder die Universität Hamburg [114] bieten Studierenden einen internen Nutzeraccount, mit dem sie

sich an diversen, teilweise auch prüfungsrelevanten Verwaltungssystemen anmelden können. Der Zugriff zu E-Mail Konten und dem hauseigenen GitLab [82] wird an der HAW Hamburg ebenfalls über ein SSO geregelt. GitLab ist für diese Arbeit insbesondere von hoher Bedeutung, da GitLab nicht nur die Authentifizierung über SSO nutzt, sondern auch bereitstellen kann. Dazu greift GitLab auf OAuth 2.0 zurück [67].

Unsere Web-IDE sollte im besten Fall also eine entsprechende Unterstützung für OAuth 2.0 anbieten. Ist dies nicht der Fall, so sollte es eine klar erkennbare Möglichkeit geben, solch eine Unterstützung nachzurüsten. So können wir die Anforderung STD-04 erfüllen.

Language Server Protocol

Das Language Server Protocol (LSP) ist ein auf JavaScript Object Notation Remote Procedure Call (JSON-RPC) basierendes [90], von Microsoft entwickeltes Protokoll für die Kommunikation zwischen unabhängiger Programmcodeeditoren oder IDEs und sogenannten Language Servern.

Ziel des Protokolls ist es, die umfangreiche Unterstützung einer Programmiersprache innerhalb von Editoren oder IDEs von diesen zu einem Language Server auszulagern. Dies bringt unter anderem den Vorteil mit sich, dass Entwickelnde eines Editors oder einer IDE die Logik der umfangreichen Unterstützung einer Programmiersprache nicht mehr selbst implementieren müssen. Solange ein Editor oder eine IDE das LSP implementiert, reicht es für die umfangreiche Unterstützung einer Programmiersprache aus, dass ein Language Server für diese existiert. Der Editor oder die IDE können dann mit diesem Language Server kommunizieren und so Funktionen einer umfangreichen Unterstützung der Programmiersprache anbieten.

Zu den Funktionen einer umfangreichen Unterstützung einer Programmiersprache gehören unter anderem Programmcodevervollständigung, Darstellung integrierter Dokumentation und statische Programmcodeanalyse [88].

Das Protokoll definiert eine klare Schnittstelle zwischen Editor oder IDE und Language Server. Implementieren Editoren und IDEs diese Schnittstelle, so können sie mit jeglichen Language Servern kommunizieren. Gleiches gilt auch andersherum [90].

Es wurden bereits mehrfach Language Servers für verschiedene Programmiersprachen implementiert, die das LSP vollständig unterstützen [89].

In der angestrebten Programmierplattform möchten wir eine Web-IDE anbieten, die Funktionen klassischer IDEs unterstützt (STD-05). Eine Möglichkeit solch eine Anforderung zu erfüllen, wäre die Unterstützung des LSP seitens der Web-IDE.

Des Weiteren würde eine Web-IDE, die das LSP implementiert, uns erlauben die angestrebten Programmierplattform in Zukunft mit wenig Aufwand in Form der Unterstützung weiterer Programmiersprachen zu erweitern.

Laufzeitumgebung und Dateisystem

Um Studierenden die Möglichkeit zu geben, zur Lösung einer Aufgabe geschriebenen Programmcode auszuführen, benötigen wir eine Laufzeitumgebung. In klassischen IDEs wird für solch eine Laufzeitumgebung meist auf das Betriebssystem der Maschine und der darauf bereits installierten Abhängigkeiten zurückgegriffen. Da wir in der Web-IDE keinen Zugriff auf das Betriebssystem einer Maschine Studierender haben, muss es uns möglich sein, eine Laufzeitumgebung anders bereitstellen zu können.

Einige Programmiersprachen benötigen neben einer Laufzeitumgebung auch ein Dateisystem, um Programmcode auszuführen. Da wir langfristig auch solche Programmiersprachen unterstützen möchten, wird ein Dateisystem zwingend benötigt. Dieses Dateisystem könnte ebenfalls für das Bearbeiten und Speichern von Programmcodedateien nützlich sein, die für die Lösung einer Aufgabe benötigt werden. Ein Dateisystem wird auch in klassischen IDEs angezeigt und würde als Teil der Web-IDE weiter zur Erfüllung von STD-05 beitragen.

Laufzeitumgebung und Dateisystem müssen über die Ferne bereitgestellt werden. Die Web-IDE muss in der Lage sein, mit einer Laufzeitumgebung und einem Dateisystem zu kommunizieren, welche sich nicht auf der selben Maschine befinden. Sind Laufzeitumgebung und Dateisystem bereitgestellt, so wird es Studierenden möglich sein, ihren Programmcode auszuführen, was STD-09 erfüllt.

Debug Adapter Protocol

Das Debug Adapter Protocol (DAP) ist ein auf JavaScript Object Notation (JSON) basierendes Protokoll zum strukturierten Austausch zwischen Programmcodeeditor oder IDE und einem Debugger über einen sogenannten Debug Adapter [86]. Ähnlich wie das LSP wurde auch das DAP von Microsoft entwickelt.

Debug Adapter fungieren als eine Art Middleware, also eine Art Kommunikationsschicht, zwischen Editoren oder IDEs und Debuggern verschiedener Programmiersprachen. Implementiert ein Editor oder eine IDE das Debug Adapter Protokoll, so kann ein geschriebener Programmcode direkt innerhalb des Editors oder der IDE debugged werden.

Ziel des Protokolls ist es, erneut den Aufwand seitens Entwickelnder von Editoren oder IDEs zu verringern. Durch einmalige Implementierung des DAPs in ihr Produkt kann dieses mit Debug Adaptern kommunizieren. Dies führt dazu, dass das Debugging einer Programmiersprache innerhalb des Editors oder der IDE möglich wird. Voraussetzung dafür ist ein existierender Debug Adapter für die jeweilige Programmiersprache.

Wie in 3.4.1 beschrieben, existieren bereits einige Language Server, die das LSP implementieren. Ebenso existieren bereits einige Debug Adapter für eine Vielfalt von Programmiersprachen, die das DAP implementieren [87].

Implementiert unsere Web-IDE das DAP, so könnten Studierende ihren Programmcode nicht nur ausführen, sondern auch mittels des GUIs der Web-IDE debuggen, was ein großer Vorteil für die erfolgreiche Erarbeitung einer Lösung für eine gestellte Aufgabe wäre.

3.4.2 Umfang

LSAs und NCAs bringen ebenfalls Anforderungen mit sich, die wir technisch abbilden müssen. Dies können wir jedoch bereits mit genannten technischen Anforderungen tun:

- Webbrowserbasiert
- Language Server Protocol

Webbrowserbasiert

Für die Umsetzung von NCAs werden wir komplett eigene GUIs implementieren müssen. Diese werden über die typische Programmcodeeingabe eines Editors oder einer IDE hinausgehen müssen, um spezielle Eingabeelemente für in Kapitel 3.3.2 genannte alternative Lösungsformen abbilden zu können.

In Kapitel 3.4.1 nennen wir die bereits gegebene technische Anforderung unserer Web-IDE, webbrowserbasiert zu sein und die damit einhergehende Unterstützung der Layout-,

Design- und Scriptsprachen: HTML, CSS und JavaScript. Diese Unterstützung erlaubt uns die Implementierung komplett eigener GUIs.

Language Server Protocol

Neben der umfangreichen Unterstützung von Programmiersprachen benötigen wir das LSP auch als technische Anforderung der Umsetzung von LSAs.

Implementiert unsere Web-IDE das LSP, so können wir Language Server von Grund auf selbst implementieren oder bestehende Language Server verändern. Dies erlaubt uns die volle Kontrolle über Unterstützung von Programmiersprachen bis hin zur Einschränkung dieser. Lehrende könnten Programmcode zur Lösung einer Aufgabe insofern beschränken, dass Studierende nur bestimmte Konstrukte einer Programmiersprache nutzen dürften, was einen Teil von LCR-08 erfüllt.

3.4.3 Skalierbarkeit

Die angestrebte Programmierplattform soll innerhalb einer Bildungseinrichtung für mehrere Studierende auf Abruf gleichzeitig zur Verfügung stehen. Um dies abbilden zu können, müssen die Programmierplattform sowie die Web-IDE und deren Architekturen weitere technische Anforderungen erfüllen.

Entsprechend unserer bereits geltenden technischen Anforderungen, muss jede Sitzung eines Studierenden auf der Programmierplattform mindestens zwei Komponenten zur Verfügung stellen können. Zum einen muss wie in Kapitel 3.4.1 eine Laufzeitumgebung für auszuführenden Programmcode zur Verfügung gestellt werden. Zum anderen, ebenfalls in Kapitel 3.4.1 genannt, muss ein Dateisystem zur Verfügung gestellt werden. Um Studierenden die Möglichkeit zu geben, während der Bearbeitung von Aufgaben Pausen zu machen, sollte das Dateisystem über einen persistenten Speicher verfügen.

Hierbei müssen wir darauf achten, dass die Laufzeitumgebung als isolierte Sandbox-Umgebung läuft. Dies würde dann ADM-01 und ADM-02 erfüllen.

Unsere Laufzeitumgebung muss außerdem, um ADM-03 zu erfüllen, nicht für mehrere Ausführungen des Programmcodes zur Verfügung stehen. Es muss für jede Ausführung des Programmcodes eine neue Laufzeitumgebung erzeugt und nach jeder Ausführung abgebaut werden.

Docker und Kubernetes

Die Nutzung von Docker und Kubernetes ist eine durch das OPPSEE Team vorgegebene Randbedingung. Diese ist sinnvoll, da Docker es uns erlaubt, die genannten Komponenten gekapselt und nach Belieben zur Verfügung stellen zu können. Hierbei muss nur vorausgesetzt sein, dass wir genügend Infrastruktur in Form von tatsächlichen Maschinen mit ausreichenden Ressourcen zur Verfügung haben.

Docker Images und deren Container können, wie in Kapitel 2.4.2 genannt, isoliert ausführt werden, was ADM-01 und ADM-02 erfüllt.

Wird ein Docker Container vor jedem Start erneut von dessen Docker Image erzeugt, so kann sichergestellt werden, dass die Ausführung eines Docker Containers niemals von einer vorherigen abhängt. Dies erfüllt ADM-03. Docker Container können außerdem jederzeit gestoppt werden, was ADM-04 erfüllt.

Die entstehenden Docker Container bündeln wir mit Kubernetes als Pods. Kubernetes erlaubt es uns durch das Starten von bestimmten Pods, einzelne Komponenten für die Sitzung eines Studierenden bereitzustellen. Durch das Stoppen einzelner Pods, erlaubt uns Kubernetes ebenso den Abbau einzelner Sitzungen, die nicht mehr benötigt werden.

3.4.4 Externe Abhängigkeiten

Einige Web-IDEs könnten externe Abhängigkeiten haben, derer wir uns bei der Auswahl bewusst sein müssen. Ein Beispiel einer technischen externen Abhängigkeit wäre eine zwingende Git Integration. Hierbei könnte eine Web-IDE eine Git Repository auf einem Server benötigen, um ein Projekt anlegen zu können. Diese Abhängigkeit mag bei LIAs und LSAs weniger problematisch erscheinen, ist jedoch bei der Umsetzung von NCAs eine unnötige Einschränkung.

Es gilt also zu prüfen, ob eine Web-IDE bestimmte externe technische Abhängigkeiten hat und ob diese uns bei der Umsetzung der angestrebten Programmierplattform einschränken. Ist dies der Fall, muss die Web-IDE entweder aussortiert oder ein Weg gefunden und aufgezeigt werden, wie diese externe Abhängigkeit umgangen werden kann.

3.4.5 Anpassbarkeit und Erweiterbarkeit

Wie in Kapitel 3.3.3 erwähnt, gibt es aktuell noch keinen festgelegten Funktionsumfang der angestrebten Programmierplattform. Entsprechend möchten wir eine Web-IDE auswählen, die möglichst anpassbar und erweiterbar ist und uns in Zukunft nicht im Wege steht.

Wie anpassbar oder erweiterbar etwas ist, lässt sich oft nur schwer definieren. Es gibt jedoch einige Hinweise darauf, dass eine Web-IDE anpassbar und erweiterbar ist.

Zu diesen gehört unter anderem eine ausgeprägte Unterstützung von Plugins und Erweiterungen über eine programmatische Schnittstelle, die eine Web-IDE anbieten könnte. Eine klar aufgeteilte Architektur der Web-IDE mit einer sinnvollen Aufgabenteilung einzelner Bestandteile ist ebenso ein guter Hinweis.

Die Unterstützung von Protokollen wie dem LSP und dem DAP sind ebenfalls ein Anzeichen für gute Anpassbarkeit und Erweiterbarkeit, wie wir in Kapitel 3.4.1 und 3.4.1 bereits durch vorgeschlagene Erweiterungen erwähnt haben.

4 Suche, Vergleich und Auswahl

Bevor eine Web-IDE für die angestrebte Programmierplattform ausgewählt werden kann, müssen wir uns einen Überblick über die verschiedenen angebotenen Web-IDEs verschaffen. Es müssen Quellen gefunden und ausgewählt werden, um diese im Anschluss nach Web-IDEs zu durchsuchen. Jegliche Web-IDEs, die auf den ersten Blick potentiell passen könnten, werden wir aufnehmen.

Mithilfe der in Kapitel 3.1 genannten Randbedingungen werden wir aus den verschiedenen, potentiell passenden Web-IDEs der Übersicht, eine engere Auswahl treffen. Diese wird die Basis dafür bilden einzelne Web-IDEs näher zu betrachten und entsprechend der in Kapitel 3.1 genannten Anforderungen zu bewerten.

Für eine solche Bewertung werden wir die verschiedenen Web-IDEs der engeren Auswahl erst hinsichtlich ihres Funktionsumfang, vor allem jedoch ihrer Architektur, vergleichbar machen. Hierfür bieten sich in der Softwareentwicklung etablierte Standards wie die Modelliersprache UML 2.0 an. Mithilfe von UML 2.0 werden wir den Aufbau und die Architektur der einzelnen Web-IDEs gleichmäßig darstellen können. Die entstehenden Diagramme können dann für einen Vergleich und eine Bewertung genutzt werden.

Um Architekturdiagramme mittels UML 2.0 zu erzeugen, benötigen wir viele Informationen über die einzelnen Architekturen. Diese Informationen müssen nicht nur aus verschiedensten Quellen gesammelt werden, sondern auch sortiert und verarbeitet werden. Es liegt entsprechend auf der Hand, dass wir nicht für alle verschiedenen, potentiell passenden Web-IDEs der engeren Auswahl Architekturdiagramme mittels UML 2.0 erstellen können. Daher müssen wir abwägen, für welche der potentiell passenden Web-IDEs der engeren Auswahl es Sinn ergibt, solche Architekturdiagramme zu erstellen. Außerdem müssen wir uns Gedanken darüber machen, welche Art von Architekturdiagrammen für einen Vergleich von Bedeutung sind und wie diese dem Aufwand und Nutzen entsprechend zu priorisieren sind.

Letzten Endes werden wir eine der verschiedenen, potentiell passenden Web-IDEs der engeren Auswahl begründet auswählen. Im Anschluss werden wir diese gegebenenfalls noch etwas detaillierter betrachten, sollte es nötig sein, unsere Entscheidung weiter zu kräftigen. Auf diesem Wege wird es uns möglich sein, eine Empfehlung für die angestrebte Programmierplattform auszusprechen und die Basis für das Kapitel 5 zu schaffen, in dem wir unsere ausgewählte Web-IDE erweitern werden.

4.1 Suche

Um einen ersten Überblick zu erzeugen, eignet sich eine Marktanalyse mittels einer einfachen Internetrecherche durch verbreitete Suchmaschinen wie DuckDuckGo [35], Ecosia [59] oder Google [85]. Des Weiteren kann auf Webseiten verschiedener wissenschaftlicher Konferenzen wie bspw. der European Conference on Object-Oriented Programming (ECOOP) [2] oder der Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) [11], aber auch wirtschaftlicher Konferenzen wie der Web Developers Conference (WDC) [36], nach passenden Ergebnissen gesucht werden.

4.1.1 Vorgehen und Durchführung

Zu Beginn können wir durch Nutzung der Suchmaschinen DuckDuckGo, Ecosia und Google Ergebnisse sammeln.

Die Webseite der ECOOP aus dem Jahre 2020 [8] bietet eine Suchfunktion an [9], mit welcher wir nach Vorträgen und anderen Beiträgen über Web-IDEs innerhalb der Konferenz suchen können. Gleiches gilt ebenfalls für die Jahre 2015 bis 2019 [3] [4] [5] [6] [7].

Ebenso wie die ECOOP bietet auch die OOPSLA aus den Jahren 2015 bis 2020 Suchfunktionen auf den entsprechenden Webseiten an [12] [13] [14] [15] [16] [17]. Diese können wir ebenfalls nutzen, um nach Vorträgen und anderen Beiträgen der Konferenz zum Thema Web-IDEs zu suchen.

Geeignete Suchbegriffe sind unter anderem „web ide“, „web based ide“, „online ide“ und „ode“. Mit diesen Begriffen durchsuchen wir die genannten Suchmaschinen und sammeln jegliche Web-IDEs die wir finden.

4.1.2 Ergebnisse

Mit Hilfe der genannten Suchmaschinen und Suchfunktionen der wissenschaftlichen und wirtschaftlichen Konferenzen, konnten wir folgende Begriffe sammeln:

- Gitpod [69]
- Eclipse Theia [44]
- CodeEnvy [102]
- Eclipse Che [41]
- CodeAnywhere [21]
- Coder [23]
- Remix IDE [60]
- Cloud 9 [10]
- Repl.it [103]
- CodePen [22]
- CodeSanbox [24]
- CodeTasty [25]

Bei all diesen Begriffen handelt es sich um Namen verschiedener Web-IDEs. Die meisten dieser Web-IDEs werden auf eigenen Webseiten vorgestellt, auf denen einige Informationen zum Umfang und teilweise auch zum Aufbau zu finden sind.

4.1.3 Erste Auswahl

Um eine erste engere Auswahl treffen zu können, werden wir überprüfen, welche der gesammelten Ergebnisse aus Kapitel 4.1.2 die Randbedingungen aus Kapitel 3.1 erfüllen.

Zu diesen Randbedingungen gehören zum einen Flexibilität durch einen Open Source Status und die Möglichkeit des Hostings der Web-IDE auf eigener Infrastruktur sowie zum anderen die richtige Lizenzierung für ein bestimmtes Vorhaben, welches in unserem Fall die Nutzung als Teil der angestrebten Programmierplattform ist.

Open Source Status

Wie in Kapitel 3.1 genannt, ist ein Open Source Status eine zwingend notwendige Randbedingung. Es muss also die offizielle Möglichkeit der Einsicht in den Programmcode der gesuchten Web-IDE geben.

Viele der Web-IDEs aus den Ergebnissen in Kapitel 4.1.2 haben aktuell keinen Open Source Status. Zu diesen gehören CodeEnvy, CodeAnywhere, Coder, Cloud 9, Repl.it, CodeTasty, CodePen und CodeSanbox. (Stand: 21. April 2021)

Der Open Source Status beliebiger Software und entsprechend auch einer Web-IDE kann sich jederzeit ändern. Dies können wir am Beispiel der Web-IDE Gitpod zeigen, die bis zum 25. August 2020 noch keinen Open Source Status hatte [71].

Da für uns jedoch nur die Web-IDEs relevant sind, die zum aktuellen Zeitpunkt einen Open Source Status haben, können wir im Rahmen dieser Arbeit die genannten acht Web-IDEs ohne Open Source Status nicht weiter für die angestrebte Programmierplattform in Betracht ziehen. Somit bleiben uns aus den ehemals zwölf Web-IDEs, die nach unserer Internetrecherche zur Auswahl standen, nun noch vier übrig.

Hosting auf eigener Infrastruktur

Die offiziell unterstützte Möglichkeit des Hostings einer Web-IDE auf eigener Infrastruktur wird ebenso wie der Open Source Status in Kapitel 3.1 genannt und ist somit ebenfalls eine zwingend notwendige Randbedingung.

Da im Unterpunkt Open Source Status bereits acht Web-IDEs ausgeschlossen wurden, ist es für uns nur noch relevant, die Möglichkeit des Hostings auf eigener Infrastruktur für die verbleibenden vier Web-IDEs Gitpod, Eclipse Theia, Eclipse Che und Remix IDE zu überprüfen.

Alle vier Web-IDEs unterstützen das Hosting auf eigener Infrastruktur.

Gitpods offizielle Dokumentation enthält ein gesamtes Kapitel [72] mit Anleitungen für das Einrichten innerhalb der Google Cloud Platform, Amazon Web Services und Kubernetes. Letzteres ist für uns besonders interessant.

Teil der offiziellen Dokumentation von Eclipse Che ist eine Anleitung [51] zur Einrichtung innerhalb verschiedener Umgebungen, wie Google Cloud Platform, Amazon Web

Services, Microsoft Azure, aber auch Kubespray. Letzteres ist ein Werkzeug für die Bereitstellung von Kubernetes [106] und entsprechend ebenfalls für uns interessant.

Für Eclipse Theia und Remix IDE wird jeweils ein offizielles Docker Image in Docker Hub angeboten [33] [34]. Diese können auf jeder Maschine laufen, die Docker installiert hat und sind somit ebenfalls mit Kubernetes kompatibel.

Anpassbarkeit und Erweiterbarkeit

Über die Anpassbarkeit oder die Erweiterbarkeit einer Web-IDE können wir durch eine oberflächliche Betrachtung kaum Aussagen treffen. Jedoch wird bei Remix IDE relativ schnell klar, dass diese Web-IDE im Gegensatz zu den anderen Web-IDEs eine klare, deutlich kleinere Zielgruppe hat.

Remix IDE ist eine Web-IDE, die den Fokus auf der Entwicklung von Verträgen im Ethereum Netzwerk legt. Dazu wird die Programmiersprache Solidity genutzt, deren Compiler direkt in Remix IDE integriert ist. Dieser Fokus müsste erst Rückgängig gemacht werden, was den anderen Web-IDEs im direkten Vergleich einen Vorteil verschafft. Die anderen Web-IDEs zielen auf Entwickelnde unabhängig der letztendlich genutzten Programmiersprache ab. Somit eignet sich Remix IDE im Vergleich mit den anderen Web-IDEs nicht für die angestrebte Programmierplattform.

Für den detaillierteren Vergleich bleiben uns dementsprechend nur noch Eclipse Theia, Gitpod und Eclipse Che.

Zusammenfassung

Die Tabelle 4.1 gibt einen Überblick über die beschriebenen Web-IDEs. Die Zeilen der Tabelle stehen für die einzelnen Web-IDEs, die Spalten für folgendes:

- **Name**: Name der Web-IDE
- **Open Source**: Ob die Web-IDE einen Open Source Status hat
- **Hosting**: Ob die Web-IDE die Möglichkeit des Hostings auf eigener Infrastruktur bietet
- **Anpassbarkeit**: Ob die Web-IDE anpassbar ist

Name	Open Source	Hosting	Anpassbarkeit
Gitpod	Ja	Ja	Ja
Eclipse Theia	Ja	Ja	Ja
CodeEnvy	Nein	-	-
Eclipse Che	Ja	Ja	Ja
CodeAnywhere	Nein	-	-
Coder	Nein	-	-
Remix IDE	Ja	Ja	Nein
Cloud 9	Nein	-	-
Repl.it	Nein	-	-
CodeTasty	Nein	-	-
CodePen	Nein	-	-
CodeSanbox	Nein	-	-

Tabelle 4.1: Übersicht der möglichen Web-IDEs

Aus anfangs zwölf möglichen Web-IDEs und Komplettlösungen sind nach näherer Betrachtung nur drei übrig geblieben, die sich für die angestrebte Programmierplattform eigenen könnten:

- Eclipse Theia
- Gitpod
- Eclipse Che

Dies lässt sich gut an der Tabelle 4.1 ablesen. Rote Felder markieren den Punkt an dem eine Web-IDE für die angestrebte Programmierplattform nicht mehr in Frage kommt. Grüne Felder zeigen, dass eine Web-IDE alle Randbedingungen erfüllt und sich somit möglicherweise für die angestrebte Programmierplattform eignen könnte.

4.2 Vergleich

Nun möchten wir unsere verbliebenen Web-IDEs etwas näher betrachten und vergleichen. Wir werden diese kurz vorstellen und im Folgenden erneut auf Randbedingungen aus Kapitel 3.1 sowie auf technische Anforderungen aus Kapitel 3.4 eingehen. Dazu werden wir die Architekturen der einzelnen Web-IDEs näher betrachten.

4.2.1 Vorstellung

Eclipse Theia

Eclipse Theia ist eine Web- und Desktop-IDE, die Teil der Software der Eclipse Foundation ist. Eclipse Theia unterstützt mehrere Programmiersprachen, ist erweiterbar und nutzt aktuelle Web-Technologien [44]. Des Weiteren unterstützt Eclipse Theia Visual Studio Code (VS Code) Plugins [47] und kann innerhalb von Docker und Kubernetes laufen [33].

Eclipse Theia ist Open Source und wird unter der Eclipse Public License - v 2.0 [43] vertrieben [66], welche eine Nutzung als Teil der angestrebten Programmierplattform erlaubt.

Eclipse Theia erfüllt somit alle unsere Randbedingungen und eignet sich dementsprechend sehr gut als mögliche Web-IDE für die angestrebte Programmierplattform.

Gitpod

Gitpod ist eine auf Eclipse Theia basierende Web-IDE. Neben zahlreichen Funktionen wie der kollaborativen Arbeit in Echtzeit, bietet Gitpod auch eine OAuth Integration für die Authentifizierung über OAuth an [74].

Gitpod kann statt der Möglichkeit auf extern verwaltetes Hosting durch die Gitpod GmbH zurückzugreifen, auch auf eigener Infrastruktur bereitgestellt werden. Dies ermöglicht es Gitpod auch in Unternehmens- und Universitätsnetzwerken intern zu nutzen und ist unter bestimmten Voraussetzungen kostenlos [73].

Seit dem 25. August 2020 ist der Programmcode von Gitpod Open Source [71].

Gitpod erfüllt somit unsere Randbedingungen und eignet sich daher auf den ersten Blick bestens als mögliche Web-IDE für die angestrebte Programmierplattform.

Eclipse Che

Eclipse Che ist eine weitere Web-IDE der Eclipse Foundation, die jedoch etwas umfangreicher als Eclipse Theia ist. Eclipse Che bietet zusätzlich zur Web-IDE bspw. Laufzeitumgebungen an, um geschriebenen Programmcode direkt auszuführen [49].

Der Web-IDE Teil von Eclipse Che war im Vergleich zu Eclipse Theia lange Zeit veraltet und weniger performant. Dies ist auch der Grund, warum Eclipse Che in der Version 7.0 auf Eclipse Theia für den Web-IDE Teil statt auf das Google Web Toolkit (GWT) setzt. Somit ist Eclipse Theia ein Bestandteil von Eclipse Che.

Wie Eclipse Theia auch, wird Eclipse Che unter der Eclipse Public License - v 2.0 [43] vertrieben [65].

Eclipse Che eignet sich, wie Gitpod und Eclipse Theia auch, auf den ersten Blick gut für die angestrebte Programmierplattform.

4.2.2 Verteilungsdiagramme

Schauen wir uns nun die Architektur der drei verbleibenden Web-IDEs genau an. Hierzu werden wir diese durch UML 2.0 Diagramme darstellen, um eine vergleichbare Basis zu schaffen.

Zu Beginn werden wir Verteilungsdiagramme der einzelnen Web-IDEs erstellen und diskutieren. Verteilungsdiagramme geben uns einen Überblick darüber, wie eine bestimmte Software während der Bereitstellung verteilt wird oder im Einsatz bereits verteilt ist.

Eclipse Theia

Abbildung 4.1 zeigt die Verteilungssicht von Eclipse Theia. Diese ist in zwei Bereiche aufgeteilt Eclipse Theia selbst und externe Services zu denen Language Server und Debug Server gehören.

Beginnen wir im Eclipse Theia Bereich, so sehen wir, dass sich dieser auf zwei Maschinen aufteilt. Zum einen gibt es den Server, auf dem das Backend von Eclipse Theia läuft, zum anderes die Maschine des Nutzenden, auf welcher das Frontend läuft. Das Backend läuft innerhalb von der Node.js Laufzeit, welche wiederum innerhalb eines Docker Containers auf der Docker Engine läuft.

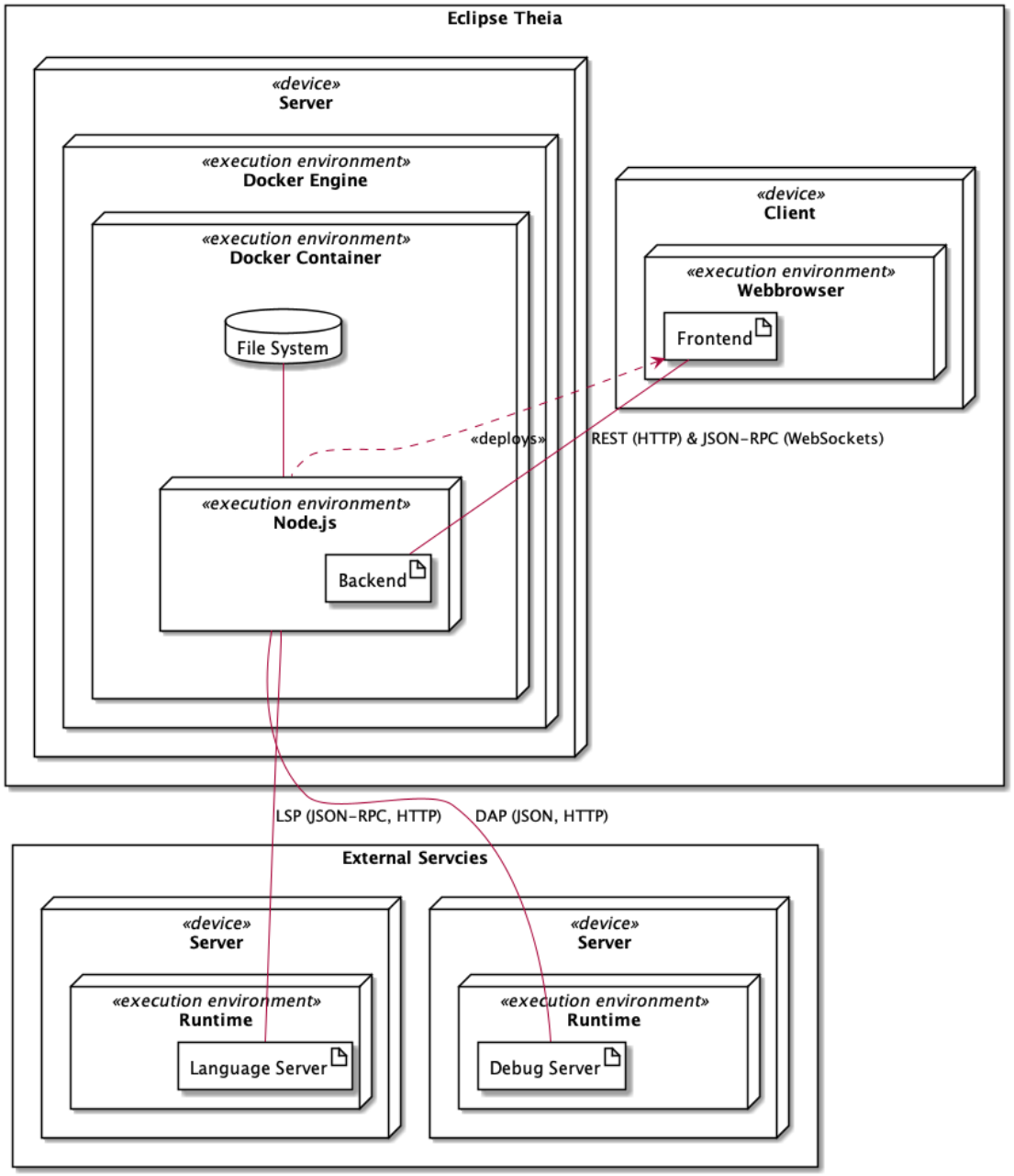

Abbildung 4.1: Verteilungssicht von Eclipse Theia

Das Backend stellt Nutzenden durch Aufruf über einen Webbrowser das Frontend bereit. Dieses läuft dann innerhalb des Webbrowsers und kommuniziert mit dem Backend über REST (HTTP) und JSON-RPC (WebSockets) [45].

Außerdem hat das Backend Zugriff auf den Docker Container, in dem es läuft. In der Abbildung 4.1 haben wir dafür das Dateisystem des Docker Container abgebildet. Jedoch hat das Backend ebenso Zugriff auf Laufzeiten, falls solche im Docker Container verfügbar sind.

Der Language Server sowie der Debug Server sind hier auf verschiedenen Maschinen zu sehen. Es wäre jedoch genau so möglich, sowohl Language Server als auch Debug Server auf der selben Maschine oder sogar im selben Docker Container laufen zu lassen, in dem auch das Backend läuft. Die Kommunikation des Backends mit dem Language Server entspricht dem LSP, die Kommunikation mit dem Debug Server, dem DAP.

Vorteil der Nutzung des LSPs und des DAPs sind an dieser Stelle die Unabhängigkeit der Implementierung der Server sowie die Möglichkeit, die Server jederzeit auszutauschen.

Abbildung 4.1 zeigt eine Instanz von Eclipse Theia. Entsprechend kann nur eine Person diese Instanz nutzen. Würden sich mehrere Nutzende über ihre Webbrowser mit der Instanz verbinden, so würden sie die selben Dateien sehen und bearbeiten können, da sie mit der selben Instanz verbunden wären. Für die angestrebte Programmierplattform müsste entsprechend jeweils eine Eclipse Theia Instanz für jeden Studierenden starten.

Gitpod

Die Verteilungssicht von Gitpod haben wir in mehrere übersichtliche Diagramme aufgeteilt. Abbildung 4.2, Abbildung 4.3, Abbildung 4.4 und Abbildung 4.5 zeigen jeweils partiell die Verteilungssicht von Gitpod. Da Eclipse Theia in Gitpod als Web-IDE fungiert, können wir ebenfalls Abbildung 4.1 erneut heranziehen.

Wir beginnen in Abbildung 4.2. Diese zeigt die Verteilungssicht des Frontends (dashboard) von Gitpod. Das Backend haben wir hier sehr reduziert. Wie zeigen nur den Teil des Backends, welcher für das Frontend relevant ist. Dazu gehört ein Proxy (proxy) und ein Server (server) [116, 3:30]. Das Proxy leitet die Anfragen, die an die App gestellt werden, an den richtigen Empfangenden weiter [116, 3:40]. Der Rest des Backends ist hier als Blackbox (Additional Pods) dargestellt.

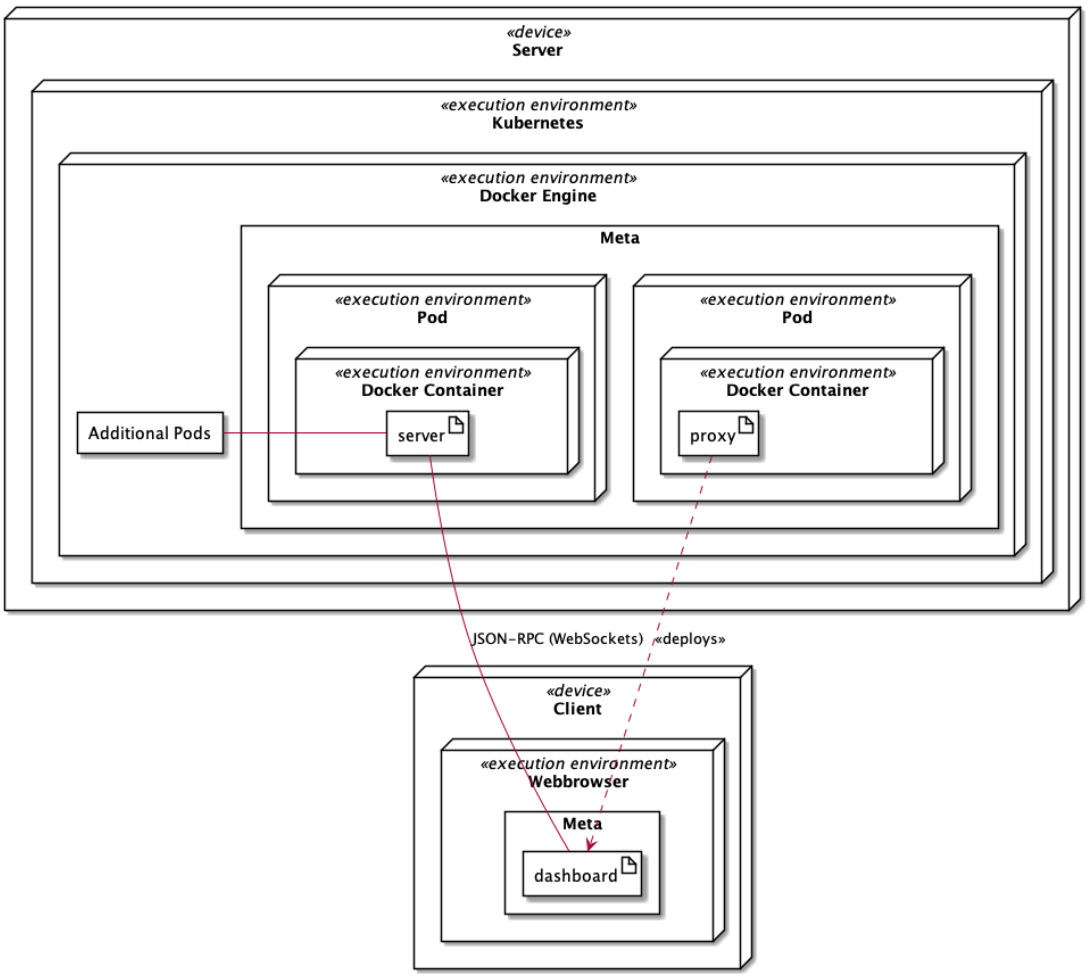

Abbildung 4.2: Partielle Verteilungssicht des Gitpod Bereichs „Meta“

Nutzende können das Frontend über einen Webbrowser anfragen. Hierzu stellt das Proxy das Frontend bereit. Anfragen des Frontends selbst bearbeitet der Server. In Abbildung 4.2 ist hier direkt ein Kommunikationspfad vom Frontend zum Server eingezeichnet. Dies ist nur der Einfachheit halber, da Anfragen des Frontends genau genommen ebenfalls über das Proxy empfangen und an den Server weitergeleitet werden [116, 3:40]. Die Kommunikation zwischen Frontend und Server läuft über JSON-RPC mittels WebSockets [116, 3:30].

Gitpods Architektur wird in zwei Teile eingeteilt; Meta und Workspace [116, 1:40]. Ersteres kümmert sich um periphere Bereiche, wie das Frontend zum Verwalten von Entwick-

lungsumgebungen [116, 2:05], während sich letzteres um die Entwicklungsumgebungen selbst [116, 2:10] kümmert. Der Server, das Proxy und das Frontend sind Teil von Meta.

Gitpod nutzt Eclipse Theia als Web-IDE. Abbildung 4.3 zeigt die Verteilung von Eclipse Theia als Teil von Gitpod. Die in Abbildung 4.1 bereits dargestellte Verteilungssicht von Eclipse Theia lässt sich gut mit Abbildung 4.3 vergleichen.

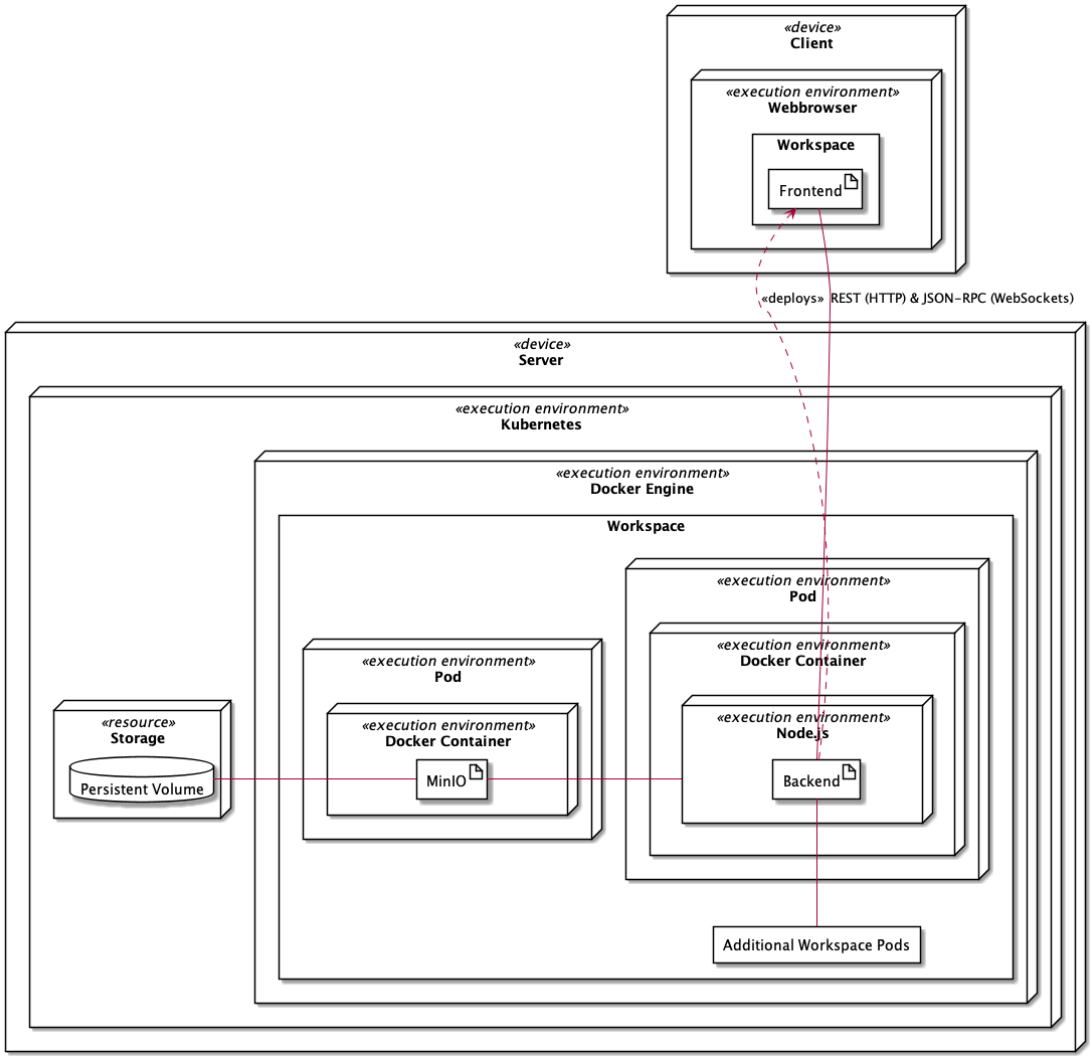

Abbildung 4.3: Partielle Verteilungssicht von Eclipse Theia als Teil von Gitpod im Bereich „Workspace“

Im Vergleich zu Abbildung 4.1, ist das Eclipse Theia Backend hier innerhalb eines durch Kubernetes verwalteten Pods dargestellt. Das in 4.1 dargestellte Dateisystem (File System) des Docker Containers wurde hier durch ein Kubernetes Persistent Volume aus-

getauscht, welches über MinIO angesprochen wird [70]. Außerdem wurden sowohl der Language Server als auch der Debug Server hier ausgelassen. Diese könnten jedoch, wie auch in 4.1 beschrieben, verteilt werden. Dies wäre dann hier auch in Kubernetes Pods möglich. Eclipse Theia ist innerhalb von Gitpod ein Teil von Workspace.

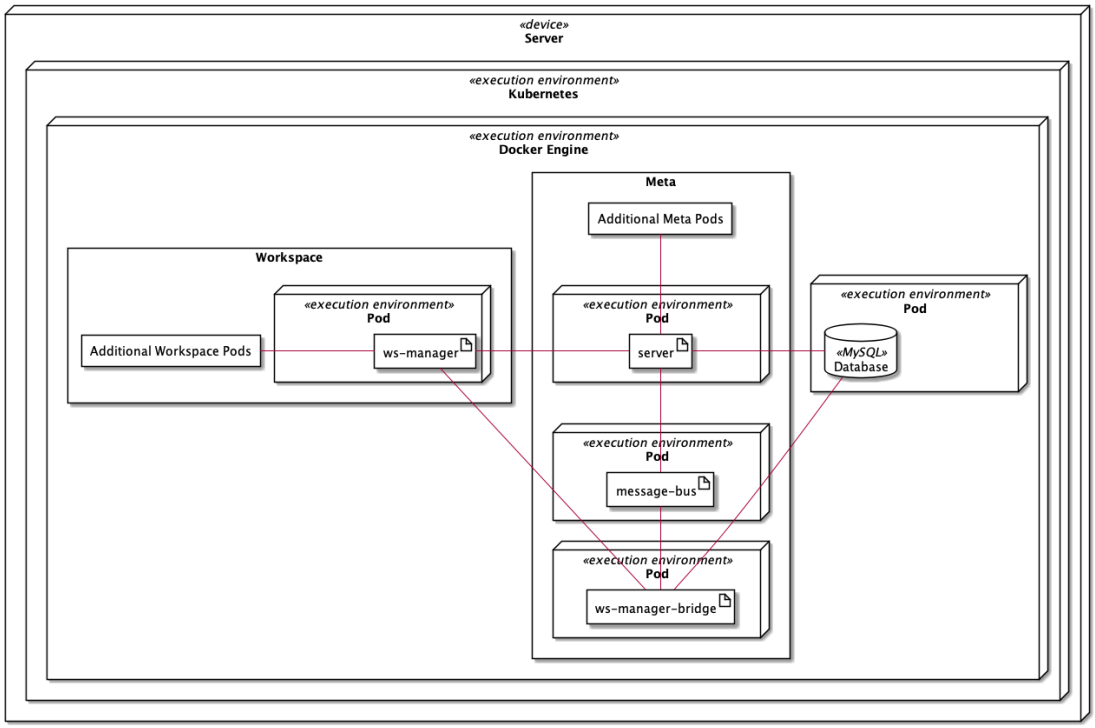

Abbildung 4.4: Partielle Verteilungssicht von Gitpod

Abbildung 4.4 zeigt nun die Verteilungssicht von Gitpod ohne Eclipse Theia. Es wird dargestellt, wie Meta und Workspace verteilt werden und wie die Kommunikation zwischen den Teilen ist.

Aus dem Workspace kommuniziert nur der Workspace Manager (ws-manager) mit der Workspace Manager Bridge (ws-manager-bridge) und dem Server, welche Teile von Meta sind. Außerdem greifen sowohl die Workspace Manager Bridge als auch der Server auf eine Datenbank (MySQL Database) zu [116, 4:02 und 12:45].

Abbildung 4.5 zeigt die Verteilungssicht des Workspace von Gitpod. Teile dieses wurden in 4.4 als Blackbox (Additional Workspace Pods) dargestellt. Es wurde lediglich der Workspace Manager (ws-manager) in 4.4 dargestellt.

In Abbildung 4.5 steht die Blackbox (Additional Workspace Pods) für die Verteilungssicht aus Abbildung 4.3. Das Proxy (ws-proxy) leitet direkte Anfragen an die Workspaces an diese weiter [116, 13:38] und der Workspace Daemon kümmert sich unter anderem um die Verwaltung der Eclipse Theia Instanzen [116, 10:34].

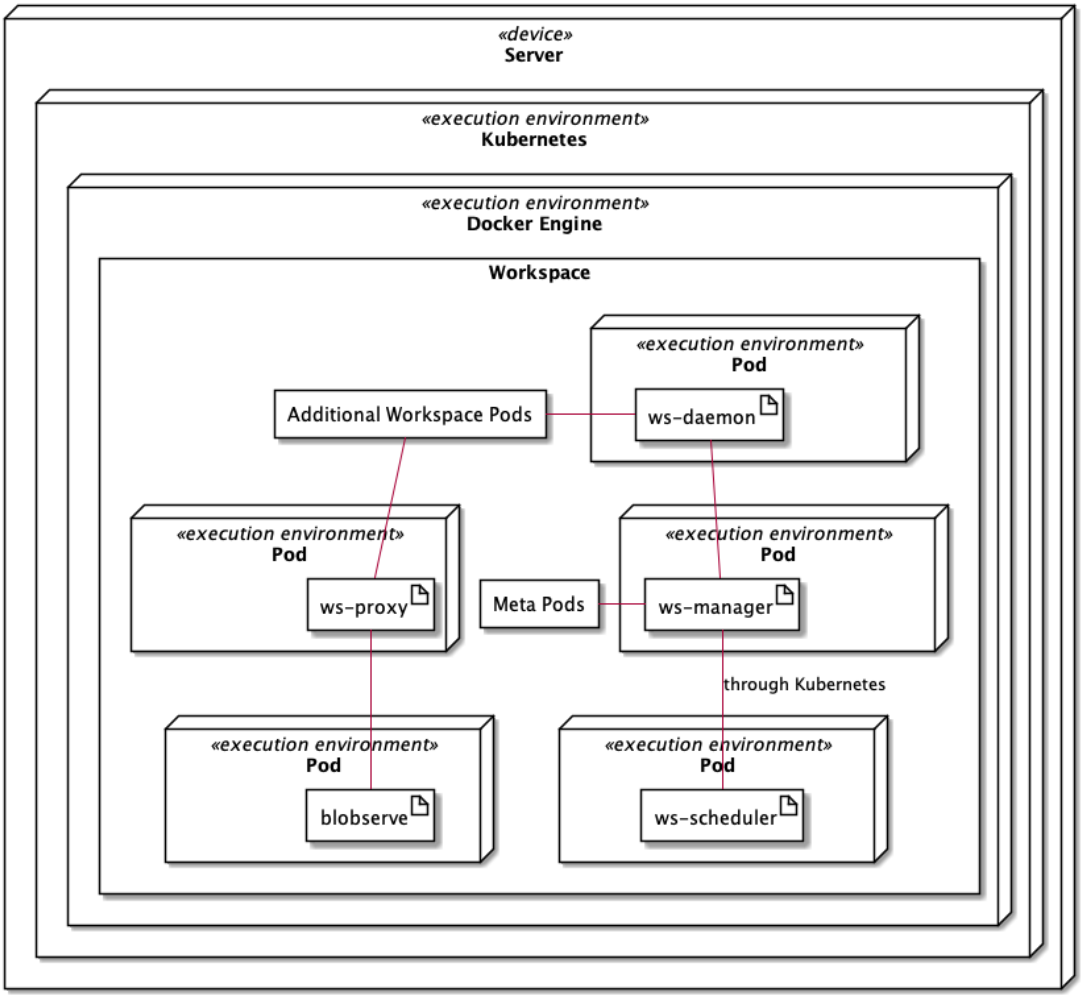

Abbildung 4.5: Partielle Verteilungssicht des Gitpod Bereichs „Workspace“

Eclipse Che

Da die Verteilungssicht von Eclipse Che wie auch die Verteilungssicht von Gitpod deutlich komplexer als die Verteilungssicht von Eclipse Theia ist, haben wir diese ebenfalls in mehrere Diagramme unterteilt. Eclipse Che lässt sich selbst in zwei Teile aufteilen. Den Workspace Controller (1) und die Workspaces (2) selbst.

Der Workspace Controller ist für die Verwaltung von Eclipse Che Workspaces verantwortlich. Dazu gehört die Bereitstellung eines GUIs, aber auch das Bereitstellen sowie der Abbau eines Eclipse Che Workspaces selbst, je nach dem ob und wann ein solcher benötigt wird [37]. Hierzu nutzt der Workspace Controller nebst eigenen Komponenten die Kubernetes API [55], durch die die Verwaltung einzelner Pods ermöglicht wird [111].

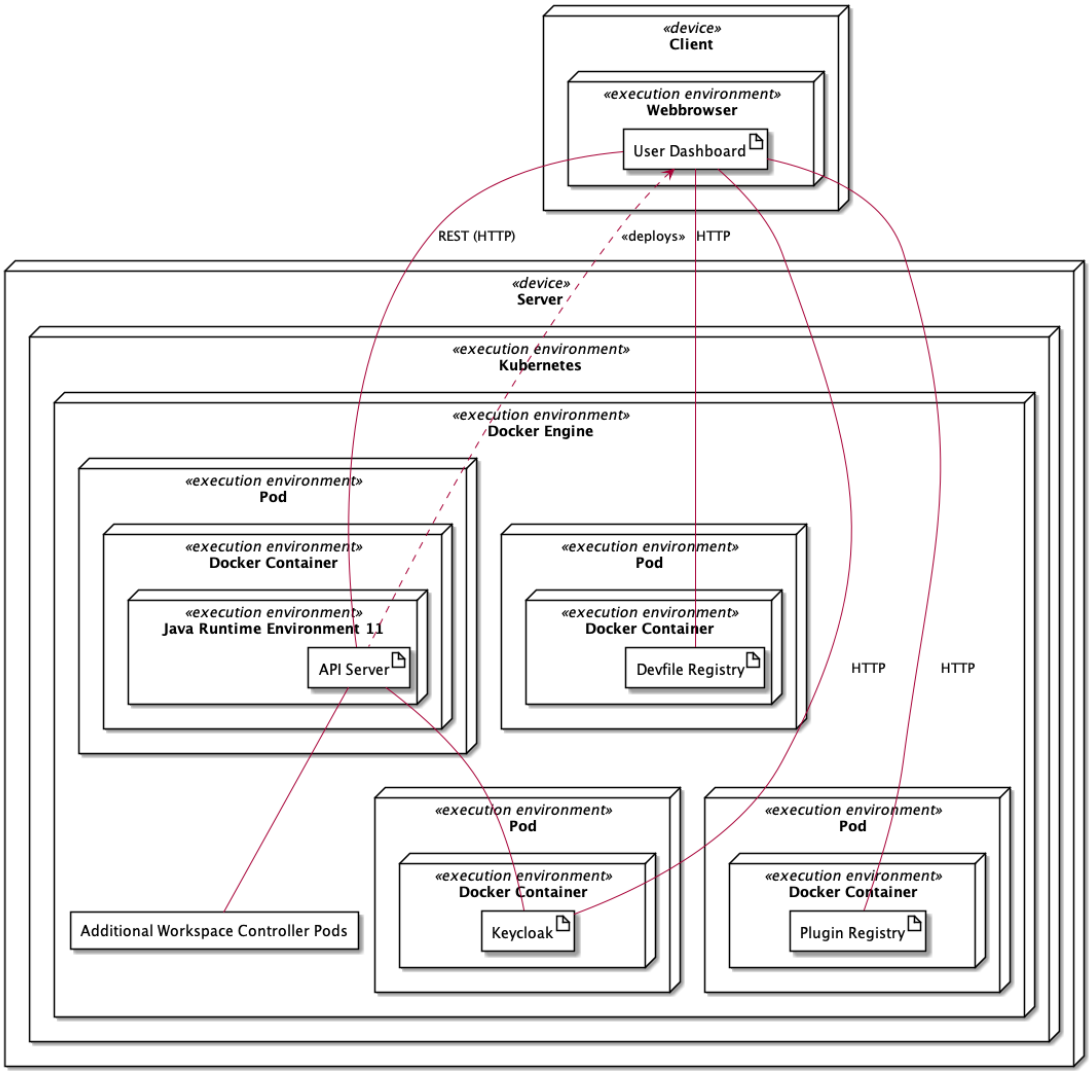

Abbildung 4.6: Partielle Verteilungssicht des Eclipse Che Workspace Controller Frontends

Abbildung 4.6 zeigt eine partielle Verteilungssicht des Workspace Controllers mit Fokus auf dem Frontend Bereich. Einstiegspunkt bildet das GUI (User Dashboard) des Workspace Controllers, welches innerhalb eines Webbrowsers des Nutzenden läuft. Das GUI ist eine Angular [77] App für das Web [42]. Es wird durch den Hauptserver (API

Server) statisch bereitgestellt und kommuniziert neben dem Hauptserver auch mit weiteren Komponenten des Workspace Controllers. Zu diesen weiteren Komponenten gehören zum einen Keycloak, und zum anderen Devfile Registry sowie Plugin Registry [55].

Der Hauptserver ist eine Java App die innerhalb der Java Laufzeitumgebung (Java Runtime Environment) in der Version 11 läuft [38]. Über den Hauptserver kann das GUI Eclipse Che Workspaces verwalten. Des Weiteren verwaltet der Hauptserver mit Hilfe von Keycloak Nutzende.

Keycloak agiert im Workspace Controller als Authentifizierungseinheit für Nutzende des GUIs. Administrierenden ist möglich bei einer Installation von Eclipse Theia zu entscheiden ob eine neue Keycloak Instanz innerhalb eines Kubernetes Pods initiiert werden soll oder ob Eclipse Che eine bereits bestehende Keycloak Instanz nutzen soll. Keycloak arbeitet mit dem OpenID Connect Protokoll [55], welches auf OAuth 2.0 basiert [99].

Die Devfile Registry und Plugin Registry dienen dem Bereitstellen statischer Dateien, die zur Laufzeit eines Eclipse Che Workspaces gebraucht werden [40]. Die Devfile Registry stellt sogenannte Devfile Dateien bereit, die als Initialisierungsdateien für Workspaces dienen. Die Plugin Registry stellt Plugins für Eclipse Che Workspaces bereit. Dazu gehören Che Editor Plugins und Che Plugin Plugins.

Bei einer normalen Installation von Eclipse Che werden die Devfile Registry und die Plugin Registry auf verschiedenen Kubernetes Pods installiert [40]. Hierbei ist zu beachten, dass die Kommunikation mit beliebig vielen Instanzen der Devfile Registry und der Plugin Registry durch das GUI unterstützt wird. Es ist ebenfalls nicht relevant auf welcher Maschine oder unter welchen Bedingungen diese Instanzen laufen. Die Instanzen müssen lediglich durch das GUI über das Internet erreichbar sein und sich an das Kommunikationsprotokoll halten. Die Plugin Registry muss nicht nur für das GUI sondern auch für den Hauptserver erreichbar sein [42]. In Abbildung 4.6 wurden Devfile Registry und Plugin Registry der Übersichtlichkeit halber nur als einzelne Instanzen dargestellt.

Die Kommunikation des GUIs mit dem Hauptserver läuft über eine vom Hauptserver angebotene REST Schnittstelle über HTTP [55]. Die Kommunikation mit Keycloak, der Devfile Registry und der Plugin Registry läuft ebenfalls über HTTP.

Abbildung 4.7 zeigt den Teil des Workspace Controllers, der in Abbildung 4.6 als Blackbox „Additional Workspace Controller Pods“ zu sehen ist. Dieser Teil besteht erneut aus dem Hauptserver (API Server), einer Datenbank (PostgresSQL), der Kubernetes API und

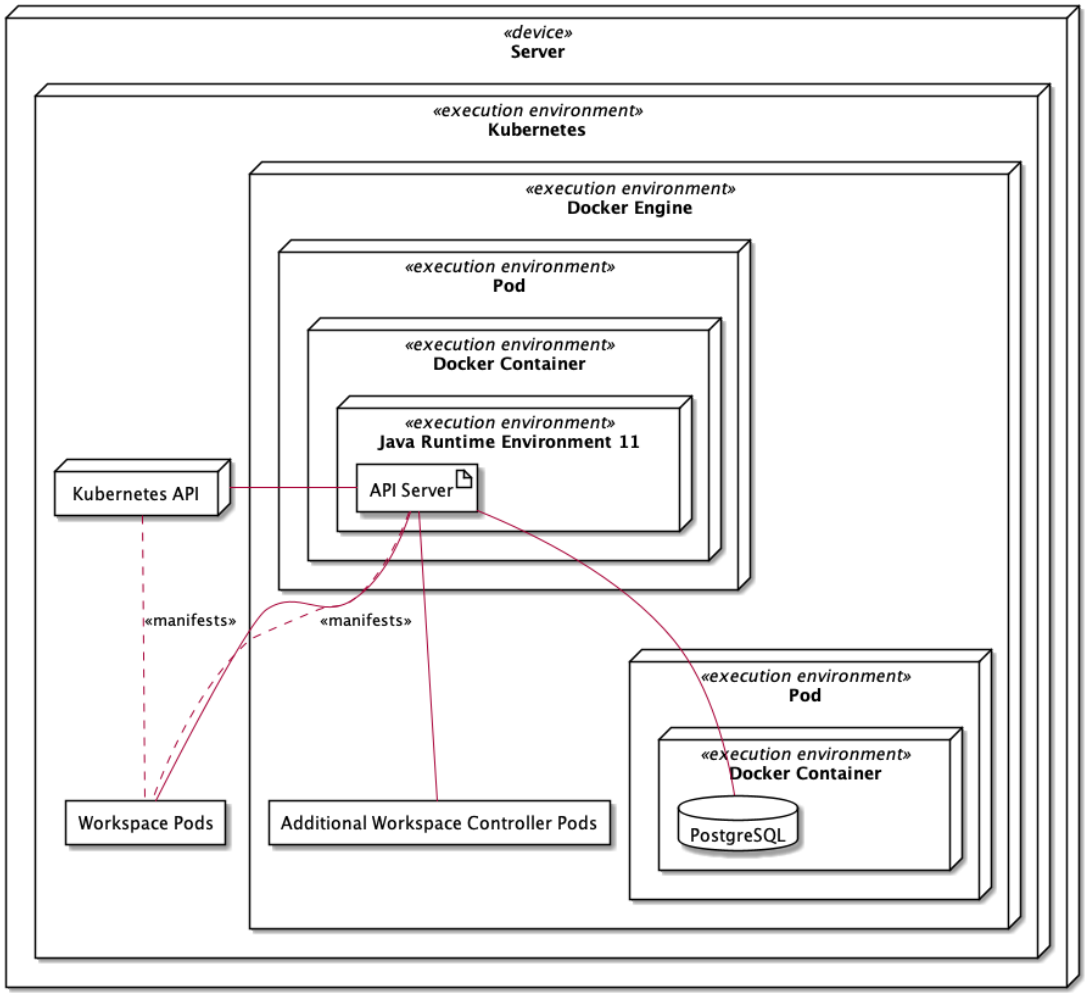

Abbildung 4.7: Partielle Verteilungssicht des Eclipse Che Workspace Controller Backends

einer weiteren Blackbox „Workspace Pods“, auf die wir in Abbildung 4.9 weiter eingehen. Die Datenbank wird vom Hauptserver direkt und von Keycloak (aus Abbildung 4.6) indirekt als Persistenzschicht genutzt. Um einzelne Pods für Eclipse Che Workspaces verwalten zu können, nutzt der Hauptserver die Kubernetes API, die von Kubernetes selbst bereitgestellt wird [111].

Anfragen des GUIs aus Abbildung 4.6 werden also über den Hauptserver verarbeitet und an die Kubernetes API oder an einzelne Eclipse Che Workspaces direkt weitergeleitet oder in der Datenbank gesichert.

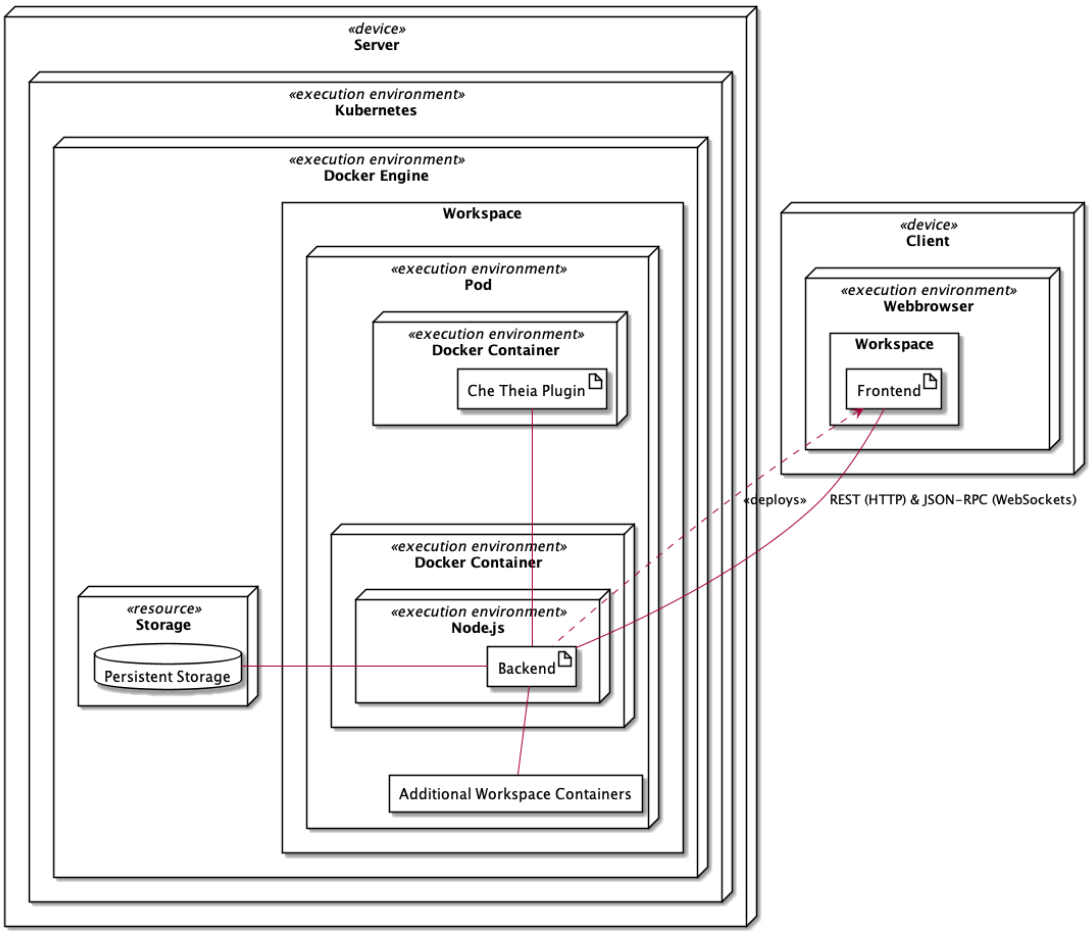

Abbildung 4.8: Partielle Verteilungssicht von Che-Theia als Teil von Eclipse Che im Bereich „Workspace“

In Abbildung 4.8 ist eine partielle Verteilungssicht eines Eclipse Che Workspaces zu sehen, die sich auf den Einsatz von Eclipse Theia innerhalb von Eclipse Che Workspaces fokussiert. In Eclipse Che wird Eclipse Theia als abgewandelte Form mit dem Namen Che-Theia genutzt [39]. Che-Theia ist ein Eclipse Che Workspace Editor Plugin [39], welches standardmäßig zum Einsatz kommt. Da Che-Theia eine nur leicht abgewandelte, erweiterte Form von Eclipse Theia ist, sind Parallelen zwischen den Verteilungssichten in Abbildung 4.1 und in Abbildung 4.8 zu erkennen.

Wie auch in Abbildung 4.1 zeigt Abbildung 4.8 einen Server und einen Client. Das Frontend wird durch das Backend bereitgestellt und kommuniziert mit diesem über REST (HTTP) und JSON-RPC (WebSockets). Die in 4.1 gezeigten „External Services“, werden in 4.8 in den „Che-Theia Plugins“ dargestellt. Zugriff zu einem persistenten Speicher hat

das Che-Theia Backend in Abbildung 4.8 genau so wie das Eclipse Theia Backend in Abbildung 4.1.

Parallelen zwischen dem Einsatz von Che-Theia in Eclipse Che und dem Einsatz von Eclipse Theia in Gitpod sind ebenso erkennbar. Die Diagramme sind sich sehr ähnlich. Jedoch wird der Zugriff auf einen persistenten Speicher bei Gitpod in Abbildung 4.3 über „MinIO“ realisiert, was bei Eclipse Che mit Che-Theia in Abbildung 4.8 nicht der Fall ist.

Abbildung 4.9 ergänzt 4.8 um weitere Bestandteile eines Eclipse Che Workspaces. Zu sehen sind ein JWT Proxy, eine Laufzeit (User Runtime), ein Eclipse Che Plugin (Che Plugin), ein Plugin Broker und zwei persistente Speicher. Die Blackbox „Additional Workspace Containers“ stellt hier all das dar, was in Abbildung 4.8 zu sehen ist.

Das JWT Proxy bildet eine Kommunikationsbrücke zwischen den einzelnen Komponenten. Die Aufgabe des JWT Proxys ist es Anfragen von Workspace Komponenten an andere Workspace Komponenten, zwecks Authentifizierung und Autorisierung, zu signieren [56]. So kann Integrität in der Kommunikation zwischen Workspace Komponenten sichergestellt werden [52].

Mit der Laufzeit ist in Abbildung 4.9 ein Docker Container gemeint, in dem eine Laufzeit für Programmcode verfügbar ist. Beispiele für eine solche Laufzeit sind Node.js und Java. Programmcode, der innerhalb Eclipse Che geschrieben wird, kann in einem solchen Container ausgeführt werden. Die Laufzeit ist hier der Übersichtlichkeit halber als einzelne Instanz dargestellt. Es ist jedoch möglich in Eclipse Che Workspaces mehrere Laufzeiten zu nutzen, um bspw. Teile des Programmcodes in der einen und andere Teile des Programmcodes in der anderen Laufzeit laufen zu lassen.

Der Plugin Broker wird vom Hauptserver aus Abbildung 4.7 gestartet und bereitet Plugins für die Nutzung in Eclipse Che vor. Dies passiert auf Basis von Plugin Metadaten, die dem Plugin Broker vom Hauptserver übergeben werden. Plugin Broker können für die zwei unterstützten Plugin Arten Eclipse Che Editor Plugin und Eclipse Che Plugin Plugin gestartet werden. Sind die Vorbereitungen eines Plugins abgeschlossen, so werden eventuell benötigte Dateien auf einen persistenten Speicher (Plugins Storage) gelegt, auf den auch die Workspaces Zugriff haben.

Abschließend ist noch ein weiterer persistenter Speicher zu sehen, der für Projektdateien gedacht ist. Programmcodedateien und andere für ein Projekt relevante Dateien werden

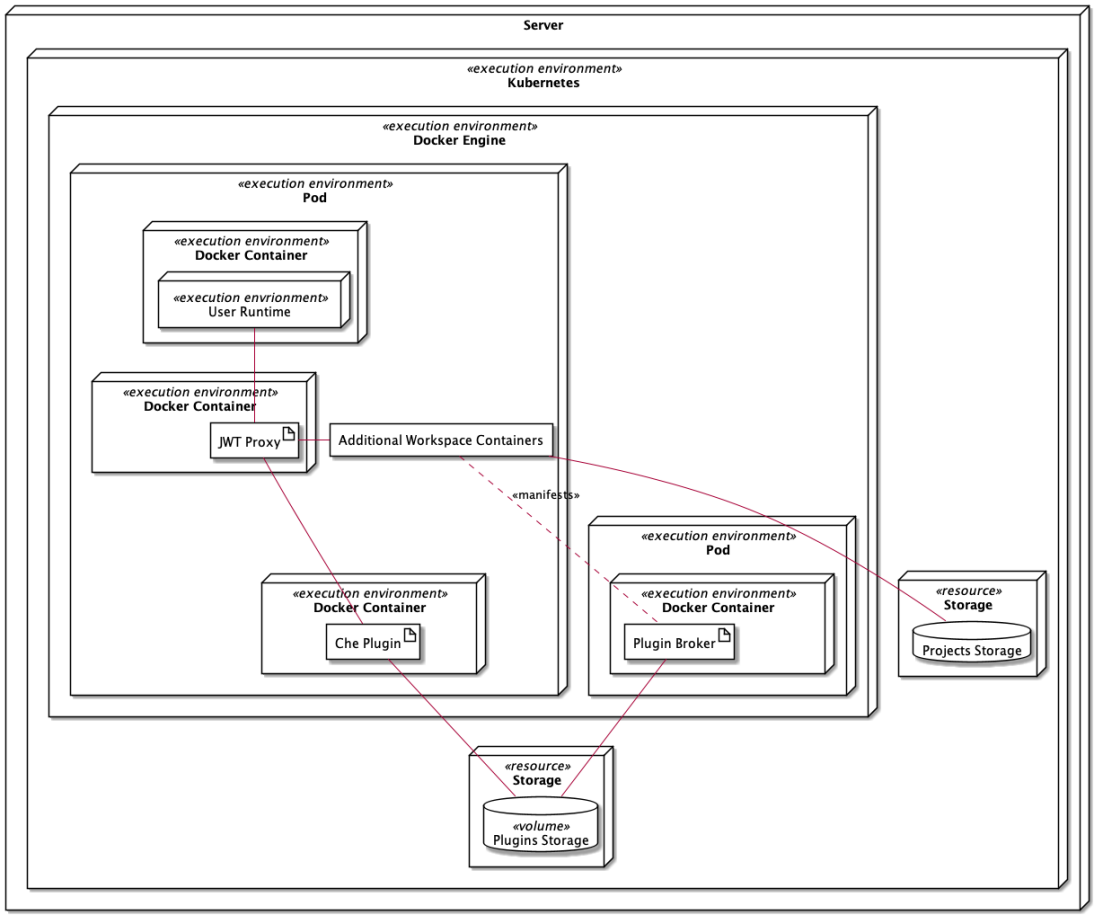

Abbildung 4.9: Partielle Verteilungssicht des Eclipse Che Bereichs „Workspace“

hier abgespeichert und können so über den Lebenszyklus eines Workspaces Pods hinweg persistiert werden.

Es ist schnell zu erkennen, dass sich Eclipse Theia nur schlecht direkt mit Gitpod und Eclipse Che vergleichen lässt, da Eclipse Theia eine andere, weniger umfangreiche Art Software ist. Eclipse Theia versucht eine Web-IDE für einzelne Entwicklungsumgebungen zu schaffen, während Gitpod und Eclipse Che versuchen, breitere Gesamtlösungen für den Einsatz mit mehreren Entwicklungsumgebungen darzustellen. Eine Web-IDE, wie Eclipse Theia eine ist, ist nur ein Teil von Gitpod und Eclipse Che.

Die angestrebte Programmierplattform zielt klar darauf ab, mehreren Studierenden und Lehrenden eine untereinander unabhängige, parallele Nutzung anbieten zu können. Die Möglichkeit, direkt mehrere Entwicklungsumgebungen unabhängig von einander anbie-

ten zu können, macht Gitpod und Eclipse Che deutlich attraktiver gegenüber Eclipse Theia. Damit scheidet Eclipse Theia hier für uns aus, da der Aufwand der Planung und Umsetzung einer Gesamtlösung um Eclipse Theia zu hoch wäre und Gitpod und Eclipse Che bereits solche Gesamtlösungen sind.

Wir werden im Folgenden also ausschließlich Gitpod und Eclipse Che betrachten.

4.2.3 Lokale Einrichtung

Zuerst werden wir die beiden Web-IDEs lokal auf unserer Maschine einrichten, um etwas mehr über den Aufbau zu erfahren. Gelingt uns dies, werden wir für einen detaillierteren Vergleich Komponentendiagramme erstellen.

Vorbereitung

Für die lokale Einrichtung der beiden Web-IDEs nutzen wir eine Maschine auf der das Betriebssystem macOS in der Version 11.5.2 läuft.

Um der Bereitstellung der angestrebten Programmierplattform nahe zu kommen, möchten wir Docker und Kubernetes lokal auf unserer Maschine installieren. Dafür nutzen wir die von Docker direkt angebotene Software Docker Desktop. Diese richtet auf unserer Maschine Docker ein und erlaubt die Konfiguration von Kubernetes über die Einstellungen.

Zwischen den Einrichtungen der beiden Web-IDEs werden wir Docker und Kubernetes komplett auf unserer Maschine zurücksetzen, sodass von neuem begonnen werden kann.

Gitpod

Die Dokumentation von Gitpod zählt eine Reihe von Voraussetzungen für die lokale Einrichtung einer Gitpod Instanz auf [75].

Dazu gehören unter anderem:

- Die Einrichtung von lokalen, variablen DNS Einträgen
- Die Einrichtung von SSL-Zertifikaten für die lokalen DNS Einträge

Da es sich bei uns um die Einrichtung von Gitpod auf einer lokalen Kubernetes Installation handelt, stellen beide dieser Punkte bereits Hürden dar. Ein Hostname kann mittels der `hosts`-Datei einer bestimmten IP-Adresse zwar lokal zugewiesen werden, jedoch unterstützt diese Datei keine variablen Einträge. Um variable DNS Einträge abbilden zu können, müssten wir also bereits weitere Schritte ergreifen, die nicht in der Dokumentation von Gitpod zu finden sind.

Gitpod unterstützt außerdem keine selbst signierten SSL-Zertifikate für die DNS Einträge [76]. Eine Anleitung zur Signierung durch eine dritte Instanz wird zwar in der Dokumentation genannt [68], besser wäre jedoch die Unterstützung selbst signierter Zertifikate.

Folgen wir nun trotz diesen beiden Hürden der Anleitung zur Installation von Gitpod auf Kubernetes, gilt es eine Datei `values.custom.yaml` zu erzeugen. In dieser werden Zugangsdaten zu zwei Technologien festgelegt, die von Gitpods Komponenten genutzt werden. RabbitMQ wird für den in Abbildung 4.5 gezeigten Message Bus (message-bus) benötigt [116, 5:08]. MinIO hingegen wird für die Kommunikation zwischen Eclipse Theia und des Kubernetes Persistent Volumes benötigt. Dies ist in Abbildung 4.3 zu sehen.

Gitpod nutzt Helm für die Bereitstellung in Kubernetes [75]. Helm hilft bei der Verwaltung von Software für Kubernetes [80].

Haben wir Helm installiert und führen wir dann die genannten Befehle der Anleitung der Reihe nach aus, so werden die einzelnen Komponenten von Gitpod in unserer Kubernetes Installation bereitgestellt.

Sowohl das in Abbildung 4.2 zu sehende Proxy (proxy) als auch das in 4.5 zu sehende Workspace Proxy (ws-proxy) können nicht korrekt gestartet werden. Als Fehlermeldung geben beide `secret "https-certificates" not found` aus. Dies hängt damit zusammen, dass wir nicht in der Lage waren variable DNS Einträge zu erzeugen und entsprechend auch keine SSL-Zertifikate für diese Erzeugt haben.

An dieser Stelle können wir unsere lokale Einrichtung von Gitpod nicht fortführen.

Eclipse Che

Für Eclipse Che wird ein Command Line Interface (CLI) `chectl` angeboten, welches die Einrichtung von Eclipse Che auf verschiedenen Plattformen vereinfachen soll. Zu diesen Plattformen gehört auch unsere lokale Installation von Docker Desktop.

Mittels des Befehls `chectl server:deploy -p docker-desktop` lässt sich durch das CLI die lokale Einrichtung starten. Jegliche benötige Ressourcen werden vollkommen automatisch heruntergeladen, installiert und entsprechend konfiguriert.

Manuell muss dann nur noch ein SSL-Zertifikat auf der lokalen Maschine als vertrauenswürdig markiert werden. Hierzu gibt das CLI den Pfad des SSL-Zertifikats auf der lokalen Maschine und eine URL mit einer Anleitung aus [50].

Nachdem wir der Anleitung gefolgt sind und das SSL-Zertifikat auf der lokalen Maschine als vertrauenswürdig markiert haben, können wir Eclipse Che mittels unseres Webbrowser aufrufen. Dazu nutzen wir die Adresse, die durch das CLI ausgegeben wurde und geben diese in die Adresszeile des Webbrowsers ein:

`https://che-eclipse-che.192.168.0.40.nip.io/`

Nach Aufruf sehen wir eine Maske zum Login in das Eclipse Che User Dashboard. Die Daten für das Login wurden während der Einrichtung durch das CLI ebenfalls ausgegeben. Nach dem Login können wir direkt einen Workspace erzeugen und nutzen. Dazu können wir der Einfachheit halber einen der Beispiel Workspaces auswählen, die Eclipse Che anbietet. Wir wählen dazu den Beispiel Workspace „NodeJS Express Web Application“ aus. Der Workspace wird nach einigen Minuten bereitgestellt und die Beispiel Anwendung kann gestartet und ausprobiert werden.

Somit wurde Eclipse Che voll funktionsfähig auf unserer lokalen Maschine installiert. Eclipse Che läuft hierbei innerhalb unserer lokalen Instanz von Kubernetes, die wir durch Docker Desktop verwalten. Die Verfügbarkeit der lokalen Installation von Eclipse Che ist ein großer Vorteil, auf den wir in Kapitel 5 zurückgreifen können, falls wir für Eclipse Che eine Erweiterung entwickeln.

Die Tatsache, dass die lokale Installation reibungslos und größtenteils vollkommen automatisiert funktioniert hat, spricht für Eclipse Che und ist ein Indiz dafür, dass es sich bei Eclipse Che offenbar um ausgereifte Software handelt.

Da es uns nicht möglich war Gitpod lokal einzurichten, werden wir keine Komponentendiagramme für Gitpod und Eclipse Che erstellen.

4.3 Auswahl

Wie bereits in Kapitel 4.2.2 erwähnt, haben wir Eclipse Theia aussortiert. Um Eclipse Theia in Betracht für die angestrebte Programmierplattform ziehen zu können, müssten wir eine Gesamtlösung um Eclipse Theia herum, planen und umsetzen. Eine solche Planung und Umsetzung bringt im Vergleich zu Gitpod und Eclipse Che zwar deutlich mehr Flexibilität im Ausbau von Funktionen mit sich, jedoch auch einen enormen Aufwand. Dieser Aufwand ist vor dem Hintergrund, dass Gitpod und Eclipse Che bereits Gesamtlösungen um Eclipse Theia herum sind, für uns zu hoch und nicht rechtfertigbar.

Die lokale Einrichtung von Gitpod auf unserer Maschine war uns, wie in Kapitel 4.2.3 beschrieben, letztendlich nicht möglich. Dies lag, neben unvollständiger Dokumentation, vor allem an lediglich partieller Unterstützung der Einrichtung auf einer lokalen Installation von Kubernetes seitens Gitpod.

Gitpod und dessen Dokumentation scheinen im Vergleich zu Eclipse Che und dessen Dokumentation noch nicht ausgereift genug zu sein. Um Gitpod gegenüber Eclipse Che als Basis für die angestrebte Programmierplattform auszuwählen, hätte dies jedoch der Fall sein müssen. Eine solide und ausgereifte Dokumentation ist langfristig ein wichtiges Werkzeug zur Einrichtung der Programmierplattform und später auch zur Erweiterung und Anpassung dieser.

Wir entscheiden uns daher Gitpod an dieser Stelle aus dem Vergleich zu nehmen. Gitpod kommt somit nicht für die angestrebte Programmierplattform in Frage.

Eclipse Che ist die Web-IDE, für die wir uns entscheiden. Dies tun wir nicht nur aus dem Grund, dass sowohl Eclipse Theia und Gitpod sowie alle anderen in Betracht gezogenen Web-IDEs ausgeschieden sind, sondern da es sich bei Eclipse Che um gut dokumentierte, ausgereifte Software zu handeln scheint, deren Aufbau einer klaren Struktur folgt. Eclipse Che ist seit vielen Jahren in kontinuierlicher Entwicklung und konnte sich bereits mehrfach in mittlerweile sieben verschiedenen Versionen bewähren. Außerdem unterstützt Eclipse Che mehrere verschiedene Arten von Erweiterungen, die den Ausbau in Richtung der angestrebten Programmierplattform beschleunigen könnten.

Wie in Kapitel 4.2.3 beschrieben, war die lokale Einrichtung auf unserer Maschine mittels des angebotenen CLIs `chectl` reibungslos möglich. Dies bildet die Basis für die Umsetzung unserer geplanten Erweiterung in Kapitel 5.

5 Erweiterung und Anpassung

Um mit unserer ausgewählten Web-IDE der angestrebten Programmierplattform etwas näher zu kommen, möchten wir Eclipse Che entsprechend in Richtung dieser erweitern und anpassen. Dies soll, sofern erfolgreich, als Beweis dafür gelten, dass unsere Web-IDE Auswahl aus Kapitel 4 angemessen ist.

5.1 Definition und Anforderungen

Für die Erweiterung und Anpassung bieten sich einige Bestandteile verschiedener Bereiche der angestrebten Programmierplattform an. Hierzu gehören bspw. die Bereiche rund um die Aufgabenstellung an Studierende, die automatisierte Bewertung bearbeiteter Aufgaben oder auch etwas weniger relevante Bereiche der Verwaltung einer Veranstaltung, wie bspw. das Erstellen und Ändern von Veranstaltungen oder die Teilnehmerverwaltung einer Veranstaltung. Aus all diesen Bereichen werden wir uns in diesem Kapitel konkret auf einen bestimmten Teilbereich der Aufgabenstellung an Studierende, nämlich den Teilbereich der NCAs, konzentrieren.

5.1.1 Definition

In Kapitel 3.3.2 sind wir auf drei Arten von Aufgaben eingegangen, die innerhalb der angestrebten Programmierplattform an Studierende gestellt und von diesen bearbeitet werden sollen. Von diesen drei Arten von Aufgaben sind sowohl die LIAs als auch die LSAs, weniger für eine Erweiterung interessant, da beide Arten von Aufgaben durch Programmcodeeingabe gelöst werden können und Programmcodeeingabe bereits ein wesentlicher Bestandteil unserer ausgewählten Web-IDE Eclipse Che ist. Deutlich interessanter wird es bei den NCAs, da Aufgaben dieser Aufgabenart sich insofern von LIAs und LSAs unterscheiden, dass eben keine Programmcodeeingabe zur Lösung durch Studierende erforderlich ist. NCAs sollen durch eine andere Form der Eingabe bearbeitet und gelöst

werden. Genau diese andere Form der Eingabe, speziell für NCAs, bildet den Teilbereich der angestrebten Programmierplattform, um den wir unsere ausgewählte Web-IDE Eclipse Che in diesem Kapitel erweitern wollen.

Bei unserer geplanten Erweiterung im Rahmen der Aufgabenstellung an Studierende, speziell der Aufgabenstellung von NCAs, wird es sich um einen Proof of Concept (PoC) handeln. Somit wird es sich bei der in diesem Kapitel erarbeiteten Umsetzung der Erweiterung keinesfalls um eine vollständig funktionsfähige Implementierung handeln. Es wird also nicht möglich sein die Erweiterung direkt in die angestrebte Programmierplattform zu übernehmen. Viel mehr geht es darum, einen der vielen möglichen Wege der Implementierung einer solche Erweiterung aufzuzeigen und umzusetzen. Wir möchten uns hierbei speziell darauf konzentrieren, den Aufwand unserer Umsetzung einer solchen Erweiterung so gering wie möglich zu halten. So kann unsere Umsetzung als übersichtliches Beispiel mit Vorbildfunktion agieren. Dies könnte die Wahrscheinlichkeit einer möglichen, wenn auch nur teilweisen, Übernahme unserer Umsetzung der Erweiterung in die angestrebten Programmierplattform erhöhen.

No-Code Assignments

Die eben genannte andere Form der Eingabe durch Studierende zur Lösung von NCAs, haben wir in Kapitel 3.3.2 unter dem Punkt „No-Code Assignments“ näher beschrieben. Kurz zusammengefasst, geht es bei dieser Form der Eingabe um eine Programmcode-unabhängige Form der Eingabe. Es wird statt einer Programmcodeeingabe lediglich eine Mehrfach- oder Einzelauswahl vorgefertigter Antwortmöglichkeiten oder auch eine durch Studierende formulierte Texteingabe erwartet. NCAs können damit eine intuitive Form der grundlegenden, teilweise aber auch spezifischen Wissensabfrage, wie es auch erfahrungsgemäß in Abschlussarbeiten der Fall ist, abbilden. Entsprechend geht es bei unserer Erweiterung speziell darum, die Programmcodeeingabe in Form eines Texteditors durch ein User Interface (UI) auszutauschen, welche eben genannte Mehrfach- oder Einzelauswahl sowie Texteingabe unterstützt.

5.1.2 Anforderungen

Durch klar definierte Anforderungen möchten wir Richtlinien für die Planung und anschließende Umsetzung der Erweiterung sowie realistische Erwartungen an diese schaffen.

Entsprechend werden wir im Folgenden Anforderungen in Form von erwarteten Funktionen der Erweiterung definieren. Diese erwarteten Funktionen, die wir implementieren möchten, sollen zeigen, dass eine Erweiterung in Richtung der angestrebten Programmierplattform möglich ist. Auch hier berücksichtigen wir erneut, dass es sich bei der Umsetzung unserer Erweiterung lediglich um ein PoC handelt.

Erwartete Funktionen der geplanten Erweiterung im Rahmen dieses Kapitels:

- Einheitliches Datenformat für Aufgaben verschiedener Unteraufgabentypen von NCAs
  - Single Choice Aufgaben
  - Multiple Choice Aufgaben
  - Freitextaufgaben
- Schnittstelle für den Eingang von gestellten Aufgaben
- Schnittstelle für den Ausgang von eingereichten Lösungen
- GUI zur Bearbeitung von Aufgaben durch Studierende für alle genannten Unteraufgabentypen von NCAs
- Möglichkeit des intuitiven Aufrufs des GUIs innerhalb von Eclipse Che

Zur ordentlichen Planung eines PoC, gehört es auch zu definieren welche ggf. offensichtlichen oder weiterführenden Funktionen nicht zu unserem PoC gehören und entsprechend nicht als Teil dieses Kapitels umgesetzt werden.

Funktionen die nicht im Rahmen dieses Kapitels umgesetzt werden:

- GUI zum Erzeugen von Aufgaben
- Jegliche (automatisierte) Bewertung von bearbeiteten Aufgaben
- Persistenz von gestellten und bearbeiteten Aufgaben über das lokale Dateisystem hinaus
- Kommunikationsschicht für Kommunikation mit externen Maschinen wie bspw. Servern

5.2 Vorgehen

Um uns die anschließende Implementierung zu vereinfachen, möchten wir an dieser Stelle grob unser Vorgehen planen. Dazu gehören die Einordnung der Erweiterung in Eclipse Che und eine Beschreibung der Entwicklungsumgebung sowie Technologien, die wir zur Entwicklung der Erweiterung nutzen werden.

5.2.1 Einordnung

Da Eclipse Theia innerhalb von Eclipse Che als Web-IDE agiert, profitieren wir von der bereits ausgeprägten Unterstützung von Erweiterungen in Form von sogenannten Eclipse Theia Plugins. Es bietet sich demnach an, unsere prototypische Erweiterung in Form eines Eclipse Theia Plugins umzusetzen.

Entwicklungsumgebung

Die von Eclipse Theia angebotene Unterstützung von Erweiterungen in Form von Eclipse Theia Plugins umfasst ebenfalls eine Unterstützung von Erweiterungen für VS Code (VS Code Extensions). Eclipse Theia unterstützt also von Haus aus Erweiterungen die für VS Code implementiert wurden [47]. An dieser Stelle müssen wir kurz klarstellen, dass Eclipse Theia Plugins und VS Code Extensions sehr ähnlich sind, Eclipse Theia Extensions jedoch, trotz gleichem Wortlaut, etwas ganz anderes sind [48].

Durch die Unterstützung von VS Code Extensions Eclipse Theias entsteht uns ein großer Vorteil. Wir können uns bei der Entwicklung unserer Erweiterung an der ausgeprägten Dokumentation zur Entwicklung von VS Code Extensions orientieren [92]. Da VS Code in genannter Dokumentation selbst als Entwicklungsumgebung für VS Code Extensions dienen kann, können wir an dieser Stelle viel Zeit beim Einrichten einer Entwicklungsumgebung sparen, wenn wir auf VS Code zurückgreifen [95].

Entsprechend werden wir unsere Erweiterung für Eclipse Che erst innerhalb von VS Code als Entwicklungsumgebung entwickeln und diese dann im Anschluss in unsere lokale Eclipse Che Plugin Registry hochladen. Dies wird es uns ermöglichen, die Erweiterung in den Eclipse Che Workspaces unserer lokalen Eclipse Che Installation nutzen zu können.

5.2.2 Technologien

VS Code Extensions werden primär in JavaScript geschrieben. JavaScript wird im Rahmen von VS Code Extensions primär innerhalb der Laufzeitumgebung Node.js ausgeführt [95]. Es kann jedoch durch sogenannte WebViews auch innerhalb einer Laufzeitumgebung eines Webbrowsers ausgeführt werden [94]. Dies kann bspw. dazu genutzt werden mit JavaScript GUIs in ihren Funktionsweisen zu erweitern, wie es andere Anwendungen für üblichen Webbrowser auch tun.

Wie in Kapitel 2.4.1 beschrieben, ist JavaScript keine typisierte Sprache. Da wir unseren Programmcode vorzugsweise typisiert schreiben möchten, werden wir mittels Nutzung von TypeScript eine Typisierung hinzufügen. TypeScript ist eine syntaktische Obermenge von JavaScript [53], die der Sprache eine Typisierung verleiht. Da TypeScript eine Obermenge von JavaScript ist, ist jeder JavaScript Programmcode auch valider TypeScript Programmcode. Da dies nicht andersherum gilt, muss TypeScript Programmcode meist erst zu JavaScript Programmcode kompiliert werden, um von einer JavaScript Laufzeitumgebung interpretiert werden zu können. Als Teil dieser Kompilierung prüft der Compiler die Typisierung des TypeScript Programmcodes und macht auf Probleme oder offensichtliche Fehler aufmerksam [91]. Dies führt dazu, dass eine bestimmte Art von Fehlern bereits vor dem Moment des letztendlichen Ausführens in einer Laufzeitumgebung erkannt werden kann. TypeScript vereint somit einige Vorteile einer typisierten Sprache mit der Flexibilität von JavaScript. Eine Alternative zu TypeScript ist bspw. Eclipse N4JS [53].

Um das GUI unserer Erweiterung, welches der Eingabe von Lösungen durch Studierende dienen soll, zu implementieren, werden wir auf eine verbreitete UI Library namens React zurückgreifen [78]. Diese unterstützt neben JavaScript auch TypeScript [63], was für uns einen Vorteil darstellt. Neben dem TypeScript Programmcode, den wir mit Hilfe von React schreiben, bringt React eine weitere erweiterte Form von JavaScript ins Spiel. Diese Form nennt sich JavaScript XML (JSX) und vereinfacht das Deklarieren von HTML Elementen durch JavaScript und React [61]. Da JSX genau wie TypeScript nicht direkt von einer JavaScript Laufzeitumgebung interpretiert werden kann, muss JSX erst zu JavaScript übersetzt werden. Es handelt sich hierbei ebenfalls um eine Kompilierung durch einen Compiler [62].

Um nun unseren TypeScript und JSX Programmcode als Teile von React zu JavaScript kompilieren zu können, benötigen wir ein Werkzeug, welches dies erledigt. Hierzu grei-

fen wir auf Vite zurück, welches neben der Unterstützung für TypeScript und JSX, eine angenehme Entwicklung für Entwickelnde bspw. durch Nutzung eines lokalen Entwicklungsservers und Hot Module Replacements (HMR) bietet [117].

Neben den bereits genannten Technologien werden noch einige weitere genutzt werden. Da wir jedoch nicht auf jede dieser Technologien detailliert eingehen können, werden wir sie an dieser Stelle bloß benennen:

- npm - Node Package Manager
- Prettier
- ESLint

5.3 Umsetzung

Wir beginnen mit der Definition unseres einheitlichen Datenformats für Aufgaben verschiedener Typen. Es gilt sowohl das Format von Daten, die in Zukunft von einem Server kommen könnten (Aufgaben), als auch das Format von Daten, die in Zukunft an einen Server geschickt werden könnten (Lösungen), zu definieren.

Hierbei berücksichtigen wir, dass wir sowohl Dateneingang (Aufgaben) als auch Datenausgang (Lösungen) im Rahmen dieses Kapitels lediglich auf lokaler Dateisystemebene abbilden und keine Kommunikation mit einer anderen Maschine in Form eines Servers umsetzen werden.

5.3.1 Datenformat

Wie gerade beschrieben, kann das Datenformat in zwei Bereiche aufgeteilt werden. Zum einen gibt es Daten, die von einer externen Quelle in unsere Extension kommen (Dateneingang), und solche, die von unserer Extension an eine externe Quelle gehen (Datenausgang). Wie bereits genannt, wird diese externe Quelle in unserer Planung und Umsetzung das lokale Dateisystem sein. Trotzdem könnte das folgende Datenformat in Zukunft genau so gut die Kommunikation mit einem Server auf einer anderen Maschine definieren.

Für beide der genannten Datenbereiche (Datenein- und ausgang) werden wir ausschließlich das Format JSON nutzen.

Dateneingang

Als Dateneingang wird unsere Extension Aufgaben erwarten. Dabei wird eine Aufgabe (`Assignment`) aus einer ID (`id`), einem Titel (`title`) und einer Reihe an Unteraufgaben (`tasks`) bestehen. Beispiel 5.1 zeig den strukturellen Aufbau einer Aufgabe.

```
{
    "id": "assignment-1"
    "title": "Aufgabe 1",
    "tasks": [...]
}
```

Beispiel 5.1: Struktureller Aufbau einer Aufgabe

Einzelne Unteraufgaben (`tasks`) werden in eine von drei Unteraufgabentypen eingeteilt, wobei sich alle drei bestimmte Attribute teilen, andere Attribute jedoch nur bei einzelnen Typen auftreten.

- Single Choice (*single-choice*)
- Multiple Choice (*multiple-choice*)
- Freitext (*text*)

Alle Unteraufgaben werden unabhängig von ihrem Typen, aus einer ID (`id`), einem Bezeichner des Typs (`type`), einem Titel (`title`) und einer optionalen Beschreibung (`desc`) bestehen.

Die Unteraufgaben vom Typ Single Choice und vom Typ Multiple Choice werden neben den genannten Attributen zusätzlich aus Antwortmöglichkeiten (`options`) bestehen. Beispiel 5.2 zeigt den strukturellen Aufbau einer Unteraufgabe vom Typ Single Choice.

```
{
    "id": "task-1"
    "type": "single-choice",
    "title": "Task 1",
    "desc": "Select the valid statement."
    "options": [...]
}
```

Beispiel 5.2: Struktureller Aufbau einer Unteraufgabe vom Typ Single Choice

Beispiel 5.3 zeigt den strukturellen Aufbau einer Unteraufgabe vom Typ Multiple Choice. Es ist zu erkennen, dass der einzige strukturelle Unterschied im Vergleich zu Auflistung 5.2 der Typ der Unteraufgabe ist ist.

```
{
    "id": "task-2"
    "type": "multiple-choice",
    "title": "Task 2",
    "desc": "Select all valid statements."
    "options": [...]
}
```

Beispiel 5.3: Struktureller Aufbau einer Unteraufgabe vom Typ Multiple Choice

Unteraufgaben vom Typ Freitext haben neben den Attributen, die sich alle Unteraufgaben teilen, keine weiteren. Ein struktureller Aufbau solch einer Aufgabe ist in Beispiel 5.4 zu sehen.

```
{
    "id": "task-3"
    "type": "text",
    "title": "Task 3",
    "desc": "Describe the solution."
}
```

Beispiel 5.4: Struktureller Aufbau einer Unteraufgabe vom Typ Freitext

Dateien die Aufgaben in dieser Struktur als JSON beinhalten, werden im lokalen Dateisystem durch einen Dateinamen im Format *x.assignment.json* gekennzeichnet, wobei x

eine beliebige Zeichenfolge darstellt. Die Extension wird entsprechend nur Dateien, die diesem Format folgen, als Aufgaben erkennen und nur aus solchen ein GUI darstellen.

Datenausgang

Für jeden Dateneingang wird es maximal einen Datenausgang geben. Wurde eine Aufgabe bearbeitet, gibt es einen Datenausgang; wurde eine Aufgabe noch nicht bearbeitet, so gibt es keinen.

Die Struktur des Datenausgangs wird dem des Dateneingangs sehr ähneln. Im Datenausgang werden IDs (`id`) des dazugehörigen Dateneingangs referenziert. Dies genügt, da wir davon ausgehen, dass ein Empfangender eines Datenausgangs auch Zugriff auf den Dateneingang hat, zu dem der Datenausgang gehört. Somit wird ein Empfangender die referenzierten IDs zuordnen können.

In unserem Fall ist der Empfangende auch der Absendende, nämlich das lokale Dateisystem. Im Falle eines zentralen Servers zur Aufgabenverwaltung würden diesem ebenfalls sowohl Dateneingänge als auch Datenausgänge zur Verfügung stehen. Dieser könnte somit einen Sinn aus einem Dateneingang voller referenzierter IDs ziehen können.

```
{
    "id": "assignment-1"
    "tasks": []
}
```

Beispiel 5.5: Struktureller Aufbau einer Lösung zur Aufgabe `assignment-1`

Beispiel 5.5 zeigt den strukturellen Aufbau einer Lösung zu einer Aufgabe mit der ID `assignment-1`. Eine Lösung mit solch einem Aufbau würde also dann erzeugt werden, wenn die Aufgabe mit der ID `assignment-1` bearbeitet wurde.

Wie auch beim Dateneingang werden einzelne Unteraufgaben (`tasks`) gesammelt. Lösungen einzelner Unteraufgaben bestehen aus einer ID (`id`), welche entsprechend die gelöste Unteraufgabe referenziert.

Des Weiteren bestehen Lösungen einzelner Unteraufgaben vom Typ Single Choice und Multiple Choice aus einer Liste an Werten (`values`), welche IDs der angebotenen Optionen referenzieren. Hierbei handelt es sich um die von den Studierenden ausgewählten

Optionen. Wichtig ist, dass die Liste an Werten bei Lösungen des Unteraufgabentyps Single Choice eine Länge von genau 1 hat. Bei Lösungen des Unteraufgabentyps Multiple Choice hat diese Liste eine Länge von mindestens 1.

```
{
    "id": "task-1"
    "values": ["option-1"]
}
```

Beispiel 5.6: Struktureller Aufbau der Lösung einer Unteraufgabe vom Typ Single Choice oder Multiple Choice

Beispiel 5.6 zeigt den strukturellen Aufbau einer Lösung für Aufgaben des Unteraufgabentyps Single Choice und Multiple Choice. Es ist eine Liste an Werten (`values`) der Länge 1 zu sehen. Diese wäre als Lösung sowohl für Unteraufgaben vom Typ Single Choice als auch Multiple Choice gültig. Bei Lösungen für Unteraufgaben vom Typ Multiple Choice kann diese Liste jedoch, wie beschrieben, auch länger sein.

Lösungen der Unteraufgaben vom Typ Freitext bestehen neben der ID aus einem Textwert (`value`), welcher die vom Studierenden formulierte Eingabe darstellt. Beispiel 5.7 zeigt den strukturellen Aufbau der Lösung einer Unteraufgabe vom Typ Freitext.

```
{
    "id": "task-1"
    "value": "This is my answer."
}
```

Beispiel 5.7: Struktureller Aufbau der Lösung einer Unteraufgabe vom Typ Freitext

Dateien, die Lösungen in dieser Struktur als JSON beinhalten, werden im lokalen Dateisystem wie auch Dateien, die Aufgaben enthalten, durch einen bestimmten Dateinamen gekennzeichnet. Das Format ähnelt hier dem Format der Dateien, die Aufgaben enthalten. Es wird durch ein Wort erweitert; `y.assignment.response.json` wobei `y` wie auch `x` bei Dateien, die Aufgaben enthalten, für eine beliebige Zeichenfolge steht.

Um die erzeugten Lösungen den gestellten Aufgaben zuordnen zu können, müssen `y` und `x` übereinstimmen. Gibt es eine Datei mit einer Aufgabe namens `1.assignment.json`, so würde die Datei mit der Lösung den Namen `1.assignment.response.json` tragen.

5.3.2 Struktur

Wie in Kapitel 5.2.2 beschrieben, gibt es für JavaScript in VS Code Extensions zwei verschiedene Laufzeitumgebungen: Node.js und die Laufzeitumgebung innerhalb eines WebViews. Wir werden unsere Extension und entsprechend auch den Programmcode dieser in zwei Teile aufteilen.

Den ersten Teil bildet jeglicher Programmcode, welcher sich um die Integration in VS Code und später auch in Eclipse Che kümmert sowie die Kommunikation des Dateneingangs und Datenausgangs übernimmt. Diesen Teil nennen wir im Folgenden „Extension Teil“. Den zweiten Teil bildet der Programmcode, welcher primär die Darstellung des GUIs und die Interaktion mit den Studierenden abbildet. Dieser Teil wird im Folgenden „App Teil“ genannt. Der Extension Teil wird innerhalb der Node.js Laufzeitumgebung ausgeführt, während der App Teil innerhalb der Laufzeitumgebung des WebViews ausgeführt wird.

Zu den Aufgaben des Extension Teils gehören bspw. das Registrieren eines Befehls zum Aufruf des GUIs (App Teil), das Erkennen und Parsen von Aufgaben in Form von JSON-Dateien, der eigentliche Aufruf des GUIs (App Teil) sowie das Erzeugen einer JSON-Datei der eingereichten Lösung zu einer Aufgabe.

Zur Logik des App Teils gehört bspw. das Darstellen wichtiger Informationen einer Aufgabe wie Titel, Beschreibung und deren Unteraufgaben sowie das Darstellen eines Formulars mit Eingabeelementen zur Interaktion und Eingabe von Lösungen durch Studierende. Zu diesen Eingabeelementen gehören sogenannte Radio-Buttons und Checkboxen jedoch auch Bereiche zur Texteingabe. Des Weiteren gehört zur Logik des App Teils die Möglichkeit ein ausgefülltes Formular abzuschicken, was in unserem Fall ein Versand an den Extension Teil zur weiteren Verarbeitung bedeutet.

Der Aufbau eines Kommunikationsweges zwischen Extension Teil und App Teil sowie die Kommunikation an sich gehört zu den Aufgaben beider Teile.

5.3.3 Applikationsfluss

Zu Beginn muss die Extension in der Web-IDE installiert werden. Dies kann je nach Anwendungsfall entweder durch den Nutzenden initiiert werden oder aber auch durch die Betreibenden bereits vor Bereitstellung der Programmierplattform und der Web-IDE geschehen.

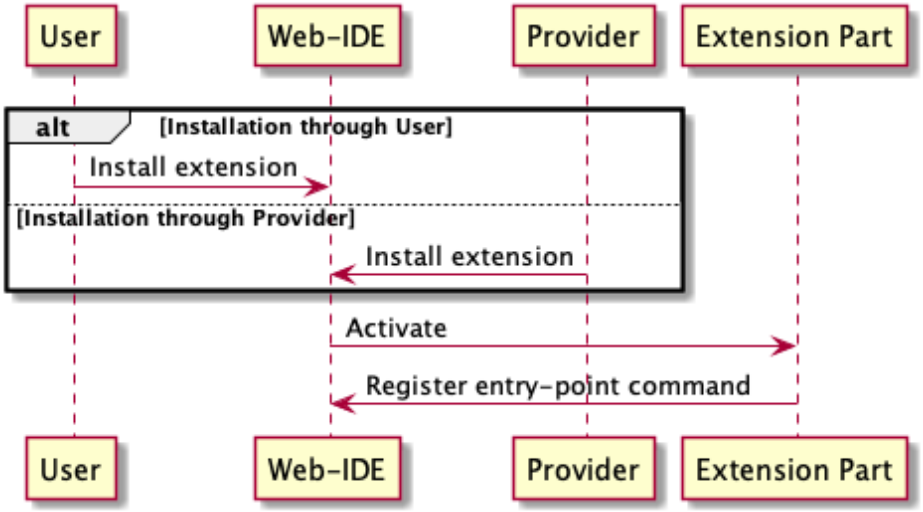

Abbildung 5.1: Installationsfluss der Extension

Nach erfolgter Installation aktiviert die Web-IDE die Extension, was dazu führt, dass die Extension innerhalb der Web-IDE einen neuen Befehl hinzufügt, mit dem in Zukunft durch Nutzende das UI angefragt werden kann. Dieser Befehl bildet den Eintrittspunkt in die Extension und deren Aufgabe. Abbildung 5.1 zeigt den Prozess der Installation, Aktivierung sowie der Registrierung des relevanten Befehls.

Nach erfolgreicher Registrierung des Befehls steht es Nutzenden frei diesen jederzeit auszuführen. Führen Nutzende den Befehl aus, so führt dies dazu, dass die Web-IDE, welche die Befehlsanfrage erhält, den Teil der Extension aufruft, den diese bei ursprünglichen Registrierung des Befehls als Eintrittspunkt hinterlegt hat. Abbildung 5.2 zeigt den Ablauf einer Befehlsanfrage bis hin zur Anzeige des UIs für die Eingabe der Lösungen einer Aufgabe.

Wurde der Befehl ausgeführt und ist nun ein UI sichtbar, so kann der Nutzende durch bereitgestellte Eingabefelder Lösungen für die Aufgabe und deren einzelne Teilaufgaben eingeben. Nach vollständiger Eingabe können Nutzende die Lösungen durch die Betätigung eines Knopfes abgeben. Dies führt in unserem Fall dazu, dass die Lösungen als JSON serialisiert auf das Dateisystem geschrieben werden. Hierbei hält sich die Struktur des JSON an die in Kapitel 5.3.1 beschriebene Struktur. Dieser gesamte Prozess von der Eingabe der Lösungen bis zum Schreiben auf das Dateisystem ist in Abbildung 5.3 dargestellt.

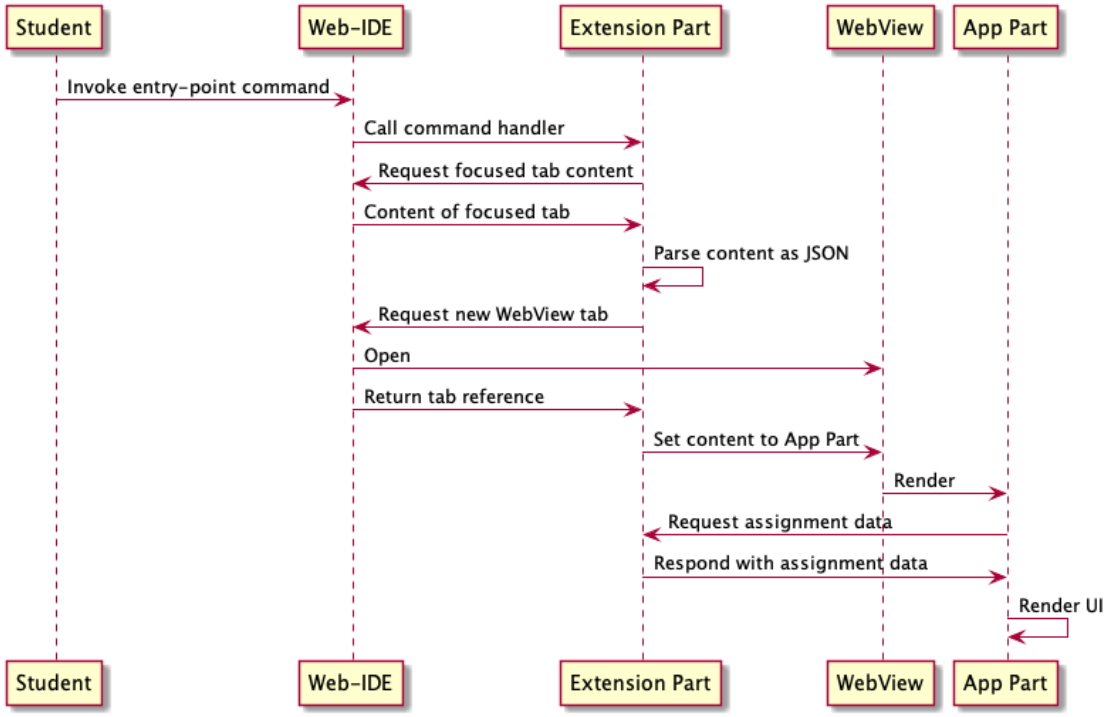

Abbildung 5.2: Aufruf des relevanten Befehls durch Nutzende

5.3.4 Programmcode

Der Programmcode der fertiggestellten Erweiterung kann der beigefügten CD-ROM entnommen werden. Eine kompilierte Version der Erweiterung (`.vsix`-Datei) kann ebenfalls der CD-ROM entnommen werden.

5.4 Installation

Nun möchten wir die fertiggestellte Erweiterung in Eclipse Che einbinden. Dazu können wir auf unsere lokale Einrichtung von Eclipse Che aus Kapitel 4.2.3 zurückgreifen. Um die Erweiterung innerhalb eines Workspaces zur Verfügung zu stellen, müssen wir diese in der Devfile Datei des Workspaces (`devfile.yaml`) definieren. Hierzu muss die Erweiterung jedoch entweder in der Eclipse Che Plugin Registry unserer lokalen Installation von Eclipse Che registriert sein oder durch diese Eclipse Che mittels Internet über eine URL abrufbar sein.

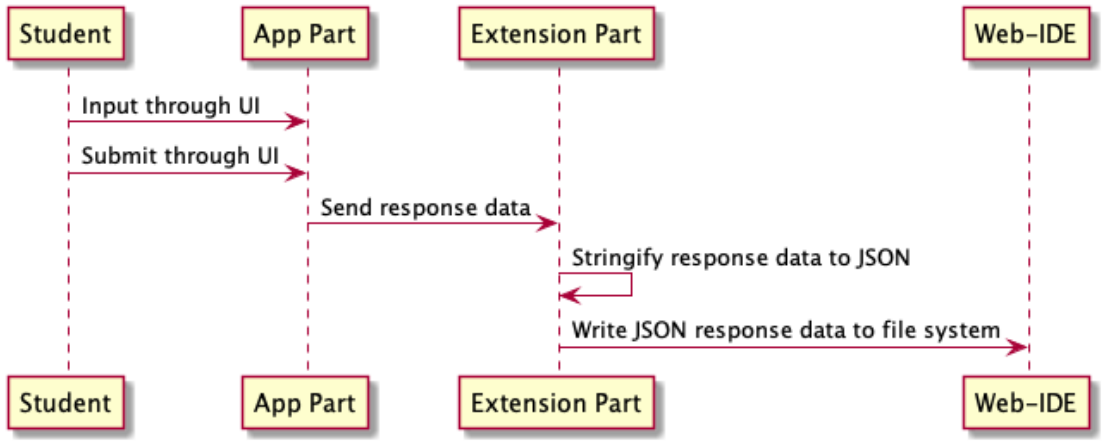

Abbildung 5.3: Eingabe und Abgabe von Lösungen durch Nutzende

Die Dokumentation von Eclipse Che beschreibt beide Wege [54]. Wir haben uns in unserem Fall für den Weg mittels GitHub Gist entschieden.

Nachdem wir den Anweisungen gefolgt sind, um die `.vsix`-Datei und die `plugin.yaml` Datei unserer Erweiterung mittels GitHub Gist öffentlich im Internet über eine URL aufrufbar zu machen, können wir uns in das Eclipse Che User Dashboard einloggen und einen eigenen Workspace mittels einer Devfile Datei erzeugen. Dazu kopieren wir die Devfile Datei des in Kapitel 4.2.3 genutzten Beispiel Workspaces „NodeJS Express Web Application“ und erweitern sie um die Registrierung unserer Erweiterung. Dazu fügen wir wie in der Auflistung 5.8 zu sehen, eine weitere Komponente hinzu.

```
(...)
components:
  - type: chePlugin
    reference: "OEFFENTLICHE_URL_ZUR_PLUGIN_YAML_DATEI"
(...)
```

Beispiel 5.8: Beispielhafte Registrierung der Erweiterung innerhalb der Devfile Datei eines Workspaces

Die Zeichenfolge `OEFFENTLICHE_URL_ZUR_PLUGIN_YAML_DATEI` ersetzen wir dabei mit der URL zur `plugin.yaml` Datei innerhalb unseres GitHub Gists. Jegliche andere durch Eclipse Che erreichbare URL zur `plugin.yaml` Datei unserer Erweiterung würde

hier ebenfalls funktionieren, solange diese ebenfalls eine erreichbare URL zur `.vsix`-Datei enthält.

Starten wir dann unseren selbst konfigurierten Workspace, so erscheint im Befehlsmenü der Web-IDE der Befehl zum öffnen einer Datei im Assignment View. Dies können wir testen indem wir eine Assignment Datei, wie sie in Kapitel 5.3.1 beschrieben ist, anlegen, fokussieren und den Befehl ausführen. Es öffnet sich das GUI unserer Erweiterung. Die Installation war damit erfolgreich und die Erweiterung ist bereit für den Einsatz.

Dank Eclipse Ches umfangreicher Unterstützung von VS Code Extensions, war es uns in kurzer Zeit möglich, einen funktionsfähigen Prototypen einer Erweiterung in Richtung der angestrebten Programmierplattform hin zu entwickeln.

Durch die Unterstützung weiterer Erweiterungen wie den Eclipse Che Plugin Plugins und den Eclipse Che Editor Plugins, sollten weitere ggf. erhebliche Anpassungen ebenso möglich sein. Dass Che-Theia selbst ein Eclipse Che Editor Plugin ist, zeigt wie umfangreich das Plugin System von Eclipse Che ist.

Diese Tatsachen zeigen, dass Eclipse Che enorm erweiterbar und anpassbar ist und stärkt unsere Entscheidung aus Kapitel 4, Eclipse Che für die angestrebte Programmierplattform auszuwählen.

6 Fazit

Ziel dieser Arbeit ist die Auswahl einer Web-IDE für eine Programmierplattform zum Einsatz in der Lehre. Fachliche Anforderungen an die Programmierplattform und die Web-IDE selbst konnten wir zum Teil aus dem Paper [64] erarbeiten und auch von einigen vorgegebenen Randbedingungen ableiten. Diese Anforderungen haben wir diskutiert und mit weiteren Randbedingungen als Basis für unsere Auswahl festgelegt.

Einen Überblick über angebotene Web-IDEs konnten wir uns durch eine breite Internetrecherche verschaffen. Nur wenige der gefundenen Web-IDEs erfüllten alle unsere Randbedingungen. Eclipse Theia, Gitpod und Eclipse Che haben wir für die angestrebte Programmierplattform in Betracht gezogen und diese detaillierter verglichen.

Nach Erzeugung diverser Verteilungsdiagramme der drei Web-IDEs war zu erkennen, dass Eclipse Theia im Vergleich zu Gitpod und Eclipse Che nicht umfangreich genug für unser Vorhaben, die angestrebte Programmierplattform umzusetzen, ist.

Durch die Einrichtung beider verbleibender Web-IDEs auf einer lokalen Kubernetes Installation, haben wir versucht die Web-IDEs noch detaillierter kennenzulernen und eine Basis für eine mögliche Erweiterung und Anpassung dieser zu schaffen.

Leider war es uns aus verschiedenen Gründen nicht möglich Gitpod auf unserer lokalen Kubernetes Installation einzurichten. Eclipse Che hingegen konnte problemlos lokal eingerichtet werden und ließ sich überraschenderweise schon nach kurzer Zeit vollkommen funktionsfähig nutzen. Da wir über Gitpod ohne eine lokal eingerichtete funktionale Version nur schwer mehr herausfinden konnten, haben wir uns entschieden, Gitpod nicht weiter für die angestrebte Programmierplattform in Betracht zu ziehen.

Die Dokumentation von Eclipse Che ist detailliert. Der Programmcode ist seit einigen Jahren Open Source und befindet sich in kontinuierlicher Weiterentwicklung. Dank der Unterstützung für VS Code Extensions konnten wir mit wenig Aufwand eine Erweiterung

für Eclipse Che entwerfen und umsetzen. Diese hat den wichtigen Bereich der Aufgabenstellung von NCAs an Studierende innerhalb der angestrebten Programmierplattform in Form eines PoCs abgebildet. Sie ließ sich einwandfrei in unsere lokale Installation von Eclipse Che einbinden und konnte in einem von uns angelegten Workspace genutzt werden.

Offensichtlich scheint es sich bei Eclipse Che um sehr ausgereifte Software zu handeln, die für eine Programmierplattform, wie der angestrebten, einen gute Basis bilden könnte. Die Tatsache, dass die Umsetzung der Erweiterung von Eclipse Che, dank der ausgeprägten Unterstützung für verschiedene Formen von Plugins, mit relativ wenig Aufwand möglich war, lässt darauf schließen, dass Eclipse Che gut erweiterbar und anpassbar ist. Dies verstärkt die Eignung als Basis für die angestrebte Programmierplattform weiter und führt dazu, dass wir Eclipse Che für die angestrebte Programmierplattform auswählen und dessen Einsatz empfehlen können.

Literaturverzeichnis

[1] An Argument for Clarity: What are Learning Management Systems, What are They Not, and What Should They Become? In: *TechTrends* 51 (2007), Nr. 2, S. 28–34. – URL https://doi.org/10.1007/s11528-007-0023-y. ISBN 1559-7075

[2] AITO e.V.: *European Conference on Object-Oriented Programming*. https://ecoop.org. – Letzter Zugriff: 08. April 2021

[3] AITO e.V.: *European Conference on Object-Oriented Programming 2015 - Suche*. https://2015.ecoop.org/search/. – Letzter Zugriff: 08. April 2021

[4] AITO e.V.: *European Conference on Object-Oriented Programming 2016 - Suche*. https://2016.ecoop.org/search/. – Letzter Zugriff: 08. April 2021

[5] AITO e.V.: *European Conference on Object-Oriented Programming 2017 - Suche*. https://2017.ecoop.org/search/. – Letzter Zugriff: 08. April 2021

[6] AITO e.V.: *European Conference on Object-Oriented Programming 2018 - Suche*. https://2018.ecoop.org/search/. – Letzter Zugriff: 08. April 2021

[7] AITO e.V.: *European Conference on Object-Oriented Programming 2019 - Suche*. https://2019.ecoop.org/search/. – Letzter Zugriff: 08. April 2021

[8] AITO e.V.: *European Conference on Object-Oriented Programming 2020*. https://2020.ecoop.org. – Letzter Zugriff: 08. April 2021

[9] AITO e.V.: *European Conference on Object-Oriented Programming 2020 - Suche*. https://2020.ecoop.org/search/. – Letzter Zugriff: 08. April 2021

[10] Amazon Web Services, Inc.: *AWS Cloud9*. https://aws.amazon.com/cloud9/. – Letzter Zugriff: 09. November 2020

[11] Association for Computing Machinery: *Object-Oriented Programming, Systems, Languages, and Applications - History*. http://oopsla.org/oopsla-history/. – Letzter Zugriff: 08. April 2021

[12] Association for Computing Machinery: *Object-Oriented Programming, Systems, Languages, and Applications 2015 - Suche*. https://2015.splashcon.org/search/. – Letzter Zugriff: 08. April 2021

[13] Association for Computing Machinery: *Object-Oriented Programming, Systems, Languages, and Applications 2016 - Suche*. https://2016.splashcon.org/search/. – Letzter Zugriff: 08. April 2021

[14] Association for Computing Machinery: *Object-Oriented Programming, Systems, Languages, and Applications 2017 - Suche*. https://2017.splashcon.org/search/. – Letzter Zugriff: 08. April 2021

[15] Association for Computing Machinery: *Object-Oriented Programming, Systems, Languages, and Applications 2018 - Suche*. https://2018.splashcon.org/search/. – Letzter Zugriff: 08. April 2021

[16] Association for Computing Machinery: *Object-Oriented Programming, Systems, Languages, and Applications 2019 - Suche*. https://2019.splashcon.org/search/. – Letzter Zugriff: 08. April 2021

[17] Association for Computing Machinery: *Object-Oriented Programming, Systems, Languages, and Applications 2020 - Suche*. https://2020.splashcon.org/search/. – Letzter Zugriff: 08. April 2021

[18] Carbonnelle, Pierre: *Top ODE Index*. https://pypl.github.io/ODE.html. – Letzter Zugriff: 31. Januar 2021

[19] Cass, Stephen: *Top Programming Languages 2021*. https://spectrum.ieee.org/top-programming-languages-2021. – Letzter Zugriff: 16. November 2021

[20] Chikofsky, E. J. ; Cross, J. H.: Reverse engineering and design recovery: a taxonomy. In: *IEEE Software* 7 (1990), Nr. 1, S. 13–17

[21] Codeanywhere, Inc.: *CodeAnywhere*. https://codeanywhere.com. – Letzter Zugriff: 09. November 2020

[22] CodePen: *CodePen*. https://codepen.io. – Letzter Zugriff: 09. November 2020

[23] Coder Technologies Inc.: *Coder*. https://coder.com. – Letzter Zugriff: 09. November 2020

[24] CodeSandbox BV: *CodeSandbox*. https://codesandbox.io. – Letzter Zugriff: 19. April 2021

[25] CodeTasty: *CodeTasty*. https://codetasty.com. – Letzter Zugriff: 09. November 2020

[26] Docker Inc.: *Deploy a registry server*. https://docs.docker.com/registry/deploying/. – Letzter Zugriff: 27. April 2021

[27] Docker Inc.: *Docker Build*. https://docs.docker.com/engine/reference/commandline/build/. – Letzter Zugriff: 27. April 2021

[28] Docker Inc.: *Docker Hub*. https://hub.docker.com. – Letzter Zugriff: 27. April 2021

[29] Docker Inc.: *Docker Overview*. https://docs.docker.com/get-started/overview/. – Letzter Zugriff: 26. April 2021

[30] Docker Inc.: *Docker Registries*. https://docs.docker.com/registry/. – Letzter Zugriff: 26. April 2021

[31] Docker Inc.: *Docker Save*. https://docs.docker.com/engine/reference/commandline/save/. – Letzter Zugriff: 27. April 2021

[32] Docker Inc.: *Dockerfile reference*. https://docs.docker.com/engine/reference/builder/. – Letzter Zugriff: 27. April 2021

[33] Docker Inc., Eclipse Foundation: *theiaide/theia*. https://hub.docker.com/r/theiaide/theia. – Letzter Zugriff: 08. September 2021

[34] Docker Inc., Ethereum: *remixproject/remix-ide*. https://remix-project.org. – Letzter Zugriff: 09. November 2020

[35] Duck Duck Go Inc.: *DuckDuckGo*. https://duckduckgo.com. – Letzter Zugriff: 08. April 2021

[36] Ebner Media Group GmbH & Co. KG: *Web Developer Conference*. https://www.web-developer-conference.de. – Letzter Zugriff: 08. April 2021

[37] Eclipse Foundation: *Che architecture overview*. https://www.eclipse.org/che/docs/che-7/administration-guide/che-architecture-overview/. – Letzter Zugriff: 14. November 2021

[38] Eclipse Foundation: *Che server*. https://github.com/eclipse-che/che-server. – Letzter Zugriff: 16. November 2021

[39] Eclipse Foundation: *Che-Theia*. https://github.com/eclipse-che/che-theia. – Letzter Zugriff: 16. November 2021

[40] Eclipse Foundation: *Customizing the registries*. https://www.eclipse.org/che/docs/che-7/administration-guide/customizing-the-registries/. – Letzter Zugriff: 16. November 2021

[41] Eclipse Foundation: *Eclipse Che*. https://www.eclipse.org/che/. – Letzter Zugriff: 09. November 2020

[42] Eclipse Foundation: *Eclipse Che Dokumentation - Workspace Controller Overview*. https://www.eclipse.org/che/docs/che-7/administration-guide/che-workspace-controller/. – Letzter Zugriff: 19. November 2020

[43] Eclipse Foundation: *Eclipse Public License - v 2.0*. https://www.eclipse.org/legal/epl-2.0/. – Letzter Zugriff: 10. September 2021

[44] Eclipse Foundation: *Eclipse Theia*. https://theia-ide.org. – Letzter Zugriff: 09. November 2020

[45] Eclipse Foundation: *Eclipse Theia - Architecture Overview*. https://theia-ide.org/docs/architecture/. – Letzter Zugriff: 11. September 2021

[46] Eclipse Foundation: *Eclipse Theia - Composing Applications*. https://theia-ide.org/docs/composing_applications/. – Letzter Zugriff: 31. Januar 2021

[47] Eclipse Foundation: *Eclipse Theia - Extensions and Plugins*. https://theia-ide.org/docs/extensions/#vs-code-extensions. – Letzter Zugriff: 30. August 2021

[48] Eclipse Foundation: *Eclipse Theia Documentation - Extensions and Plugins*. https://theia-ide.org/docs/extensions/#extensions-and-plugins. – Letzter Zugriff: 30. August 2021

[49] Eclipse Foundation: *Features*. https://www.eclipse.org/che/features/. – Letzter Zugriff: 10. September 2021

[50] Eclipse Foundation: *Importing certificates to browsers*. https://www.eclipse.org/che/docs/che-7/end-user-guide/importing-certificates-to-browsers/. – Letzter Zugriff: 12. September 2021

[51] Eclipse Foundation: *Installing Che in cloud*. https://www.eclipse.org/che/docs/che-7/installation-guide/installing-che-in-cloud/. – Letzter Zugriff: 08. September 2021

[52] Eclipse Foundation: *JWT Proxy*. https://github.com/eclipse/che-jwtproxy. – Letzter Zugriff: 25. November 2021

[53] Eclipse Foundation: *N4JS and TypeScript*. https://www.eclipse.org/n4js/faq/comparison-typescript.html. – Letzter Zugriff: 25. November 2021

[54] Eclipse Foundation: *Testing a Visual Studio Code extension in Che*. https://www.eclipse.org/che/docs/che-7/end-user-guide/testing-a-visual-studio-code-extension-in-che/. – Letzter Zugriff: 12. September 2021

[55] Eclipse Foundation: *Understanding Che workspace controller*. https://www.eclipse.org/che/docs/che-7/administration-guide/che-workspace-controller/. – Letzter Zugriff: 14. November 2021

[56] Eclipse Foundation: *Understanding Che workspaces architecture*. https://www.eclipse.org/che/docs/che-7/administration-guide/che-workspaces-architecture/. – Letzter Zugriff: 14. November 2021

[57] Ecma International: *Ecma International*. https://www.ecma-international.org. – Letzter Zugriff: 16. November 2021

[58] Ecma International: *TC39 - ECMAScript*. https://www.ecma-international.org/technical-committees/tc39/. – Letzter Zugriff: 16. November 2021

[59] Ecosia GmbH: *Ecosia*. https://www.google.com. – Letzter Zugriff: 07. September 2021

[60] Ethereum: *Remix*. https://remix-project.org. – Letzter Zugriff: 09. November 2020

[61] Facebook Open Source: *React - Introducing JSX*. https://reactjs.org/docs/introducing-jsx.html. – Letzter Zugriff: 25. November 2021

[62] Facebook Open Source: *React - JSX Represents Objects*. https://reactjs.org/docs/introducing-jsx.html#jsx-represents-objects. – Letzter Zugriff: 25. November 2021

[63] Facebook Open Source: *React - TypeScript*. https://reactjs.org/docs/static-type-checking.html#typescript. – Letzter Zugriff: 25. November 2021

[64] Gandrass, Niels ; Hinrichs, Torge ; Schmolitzky, Axel: Towards an Online Programming Platform Complementing Software Engineering Education / Hamburg University of Applied Sciences - Hamburg, Germany. 2020. – Forschungsbericht

[65] Github Inc., Eclipse Foundation: *Eclipse Che - LICENSE*. https://github.com/eclipse/che/blob/main/LICENSE. – Letzter Zugriff: 10. September 2021

[66] Github Inc., Eclipse Foundation: *Eclipse Theia - LICENSE*. https://hub.docker.com/r/theiaide/theia. – Letzter Zugriff: 08. September 2021

[67] GitLab Inc.: *GitLab as an OAuth 2.0 provider*. https://docs.gitlab.com/ee/api/oauth2.html. – Letzter Zugriff: 05. September 2021

[68] Gitpod GmbH: *Configure the ingress to your Gitpod installation - Using Let's Encrypt to generate HTTPS certificates*. https://www.gitpod.io/docs/self-hosted/latest/configuration/ingress#using-let\T1\textquoterights-encrypt-to-generate-https-certificates. – Letzter Zugriff: 12. September 2021

[69] Gitpod GmbH: *Gitpod*. https://www.gitpod.io. – Letzter Zugriff: 09. November 2020

[70] Gitpod GmbH: *Gitpod - Configure the storage used by your Gitpod installation*. https://www.gitpod.io/docs/self-hosted/latest/configuration/storage. – Letzter Zugriff: 11. September 2021

[71] Gitpod GmbH: *Gitpod Open Source*. https://www.gitpod.io/blog/opensource/. – Letzter Zugriff: 09. November 2020

[72] Gitpod GmbH: *Gitpod Self-Hosted*. https://www.gitpod.io/docs/self-hosted/latest. – Letzter Zugriff: 12. September 2021

[73] Gitpod GmbH: *Gitpod Self Hosted*. https://www.gitpod.io/self-hosted/. – Letzter Zugriff: 09. November 2020

[74] Gitpod GmbH: *How does it work?* https://github.com/gitpod-io/website/blob/d7912988f3e508fd3434107e41ab78dd3b475321/src/routes/blog/gitpod-self-hosted-0.4.0.md#how-does-it-work. – Letzter Zugriff: 10. September 2021

[75] Gitpod GmbH: *Installation requirements for Gitpod Self-Hosted*. https://www.gitpod.io/docs/self-hosted/latest/requirements. – Letzter Zugriff: 12. September 2021

[76] Gitpod GmbH: *Installation requirements for Gitpod Self-Hosted - SSL*. https://www.gitpod.io/docs/self-hosted/latest/requirements#ssl. – Letzter Zugriff: 12. September 2021

[77] Google: *Angular*. https://angular.io. – Letzter Zugriff: 19. November 2020

[78] Greif, Sacha ; Benitte, Raphaël: *State of JavaScript 2020 - Front-end Frameworks*. https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/. – Letzter Zugriff: 25. November 2021

[79] Hardt, Ed. D.: *The OAuth 2.0 Authorization Framework*. https://tools.ietf.org/html/rfc6749. – Letzter Zugriff: 26. April 2021

[80] Helm Authors: *Helm*. https://helm.sh. – Letzter Zugriff: 12. September 2021

[81] Hochschule für Angewandte Wissenschaften Hamburg: *HAW-Account und Passwort*. https://www.haw-hamburg.de/haw-account/. – Letzter Zugriff: 05. September 2021

[82] Hochschule für Angewandte Wissenschaften Hamburg: *Single Sign On der hauseigenen GitLab Instanz*. https://git.haw-hamburg.de/users/sign_in. – Letzter Zugriff: 05. September 2021

[83] JetBrains r.s.o: *JetBrains Intellij IDEA Features*. https://www.jetbrains.com/de-de/idea/features/. – Letzter Zugriff: 23. Januar 2021

[84] JetBrains r.s.o: *The State of Developer Ecosystem 2019*. https://www.jetbrains.com/lp/devecosystem-2019/. – Letzter Zugriff: 27. Januar 2021

[85] Limited, Google Ireland: *Google*. https://www.ecosia.org. – Letzter Zugriff: 08. April 2021

[86] Microsoft: *Debug Adapter Protocol*. https://microsoft.github.io/debug-adapter-protocol/. – Letzter Zugriff: 03. Februar 2021

[87] Microsoft: *Debug Adapter Protocol - Implementations*. https://microsoft.github.io/debug-adapter-protocol/implementors/adapters/. – Letzter Zugriff: 03. Februar 2021

[88] Microsoft: *Language Server Protocol*. https://microsoft.github.io/language-server-protocol/. – Letzter Zugriff: 03. Februar 2021

[89] Microsoft: *Language Server Protocol - Implementations*. https://microsoft.github.io/language-server-protocol/implementors/servers/. – Letzter Zugriff: 03. Februar 2021

[90] Microsoft: *Language Server Protocol - Overview*. https://microsoft.github.io/language-server-protocol/overviews/lsp/overview/. – Letzter Zugriff: 03. Februar 2021

[91] Microsoft: *TypeScript*. https://www.typescriptlang.org. – Letzter Zugriff: 25. November 2021

[92] Microsoft: *Visual Studio Code - API*. https://code.visualstudio.com/api. – Letzter Zugriff: 30. August 2021

[93] Microsoft: *Visual Studio Code - GitHub*. https://github.com/Microsoft/vscode. – Letzter Zugriff: 27. Januar 2021

[94] Microsoft: *Visual Studio Code - Webview API*. https://code.visualstudio.com/api/extension-guides/webview. – Letzter Zugriff: 25. November 2021

[95] Microsoft: *Visual Studio Code - Your First Extension*. https://code.visualstudio.com/api/get-started/your-first-extension. – Letzter Zugriff: 30. August 2021

[96] Moodle community: *Zusatzplugin CodeRunner*. https://docs.moodle.org/310/de/Fragetyp_Coderunner. – Letzter Zugriff: 24. April 2021

[97] Mozilla and individual contributors: *A re-introduction to JavaScript (JS tutorial)*. https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript. – Letzter Zugriff: 16. November 2021

[98] Object Management Group: *UML Spezifikationen*. https://www.omg.org/spec/UML/. – Letzter Zugriff: 22. April 2021

[99] Okta: *Worin unterscheiden sich OAuth 2.0, OpenID Connect und SAML?* https://www.okta.com/de/identity-101/whats-the-difference-between-oauth-openid-connect-and-saml/. – Letzter Zugriff: 16. November 2021

[100] Open Source Initiative: *Open Source Initiaitve FAQ - What is Open Source Software?* https://opensource.org/faq#osd. – Letzter Zugriff: 29. August 2021

[101] Parecki, Aaron: *OAuth 2.0*. https://oauth.net/2/. – Letzter Zugriff: 17. Januar 2021

[102] Red Hat: *CodeEnvy*. https://codenvy.com. – Letzter Zugriff: 09. November 2020

[103] Replit, Inc.: *Repl.it*. https://repl.it. – Letzter Zugriff: 09. November 2020

[104] Schmolitzky, Axel: *Open Programming Platform for Software Engineering Education - OPPSEE*. https://oppsee.informatik.haw-hamburg.de/. – Letzter Zugriff: 29. August 2021

[105] The Linux Foundation: *Cloud Native Computing Foundation - News*. https://www.cncf.io/news/2015/07/21/techcrunch-as-kubernetes-hits-1-0-google-donates-technology-to-newly-formed-cloud-native-computing-foundation/. – Letzter Zugriff: 18. Januar 2021

[106] The Linux Foundation, The Kubernetes Authors: *Installing Kubernetes with Kubespray*. https://kubernetes.io/docs/setup/production-environment/tools/kubespray/. – Letzter Zugriff: 08. September 2021

[107] The Linux Foundation, The Kubernetes Authors: *Kubernetes - Container runtimes*. https://kubernetes.io/docs/setup/production-environment/container-runtimes/. – Letzter Zugriff: 18. Januar 2021

[108] The Linux Foundation, The Kubernetes Authors: *Kubernetes - Disruptions*. https://kubernetes.io/docs/concepts/workloads/pods/disruptions/. – Letzter Zugriff: 29. April 2021

[109] The Linux Foundation, The Kubernetes Authors: *Kubernetes - Nodes*. https://kubernetes.io/de/docs/concepts/architecture/nodes/. – Letzter Zugriff: 28. April 2021

[110] The Linux Foundation, The Kubernetes Authors: *Kubernetes - Pods*. https://kubernetes.io/de/docs/concepts/workloads/pods/. – Letzter Zugriff: 28. April 2021

[111] The Linux Foundation, The Kubernetes Authors: *The Kubernetes API*. https://kubernetes.io/docs/concepts/overview/kubernetes-api/. – Letzter Zugriff: 08. September 2021

[112] The Linux Foundation, The Kubernetes Authors: *Was ist Kubernetes?* https://kubernetes.io/de/docs/concepts/overview/what-is-kubernetes/. – Letzter Zugriff: 18. Januar 2021

[113] The Linux Information Project: *Vendor Lock-In*. http://www.linfo.org/vendor_lockin.html. 04 2006. – Letzter Zugriff: 16. November 2020

[114] Universität Hamburg: *Benutzerverwaltung des Regionalen Rechenzentrums der Univeristät Hamburg*. https://bv.uni-hamburg.de. – Letzter Zugriff: 05. September 2021

[115] Unknown: *About Node.js, and why you should add Node.js to your skill set?* http://blog.training.com/2016/09/about-nodejs-and-why-you-should-add.html. – Letzter Zugriff: 16. November 2021

[116] Weichel, Christian: *Gitpod Architecture - A look inside the kube*. https://youtu.be/svV-uE0Cdjk. 04 2021. – Letzter Zugriff: 11. September 2021

[117] You, Evan ; Vite Contributors: *Vite*. https://vitejs.dev/. – Letzter Zugriff: 25. November 2021

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Architektur von Web-IDEs in Hinblick auf Programmierplattformen

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original