



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Moritz Stückler

**Entwurf und prototypische Implementierung eines
webbasierten Classroom-Response-Systems für die
Programmierlehre**

*Fakultät Technik und Informatik
Studiendepartment Informatik*

*Faculty of Engineering and Computer Science
Department of Computer Science*

Moritz Stückler

**Entwurf und prototypische Implementierung eines
webbasierten Classroom-Response-Systems für die
Programmierlehre**

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

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

Betreuender Prüfer: Prof. Dr. Axel Schmolitzky
Zweitgutachter: Prof. Dr. Martin Becke

Eingereicht am: 9. Mai 2019

Moritz Stückler

Thema der Arbeit

Entwurf und prototypische Implementierung eines webbasierten Classroom-Response-Systems für die Programmierlehre

Stichworte

Audience-Response-Systeme, Classroom-Response-Systeme, StuReSy, Webentwicklung, React, Redux, WebRTC, JavaScript

Kurzzusammenfassung

Sogenannte Classroom-Response-Systeme (CRS) werden inzwischen an vielen Universitäten eingesetzt. Studierende können damit während einer Veranstaltung mit ihrem Smartphone oder anderen internetfähigen Geräten Fragen zum Inhalt der Veranstaltung beantworten. Im Kontext der Programmierlehre sind bisherige CRS-Lösungen allerdings häufig nicht geeignet. Im Rahmen dieser Arbeit sollen bestehende CRS auf ihre Eignung für den Einsatz in der Programmierlehre bewertet werden. Anschließend wird eine Software konzipiert und als Prototyp implementiert, die sich von bestehenden CRS in vier Kern-Aspekten unterscheidet:

Sie ist vollständig **webbasiert** und läuft als JavaScript-Anwendung im Browser. Der Download von Software oder die Installation von Plugins ist nicht erforderlich.

Die Verbindung zwischen Studenten und Dozent wird **direkt zwischen den beteiligten Browsern** hergestellt. Ein dediziertes Server-System ist nicht notwendig.

Die Anwendung ist optimiert auf die **Darstellung und Einbettung von Java-Quelltexten** (zum Beispiel durch Monospace-Formatierung und Syntax-Highlighting).

Außerdem können Java-Quelltexte, die in die Fragestellungen eingebettet werden, unmittelbar **im Browser ausgeführt werden**. Dazu wird eine Java-Virtual-Machine (implementiert in JavaScript) im Browser ausgeführt.

Moritz Stückler

Title of Thesis

Design and prototypical Implementation of a web based Classroom Response System for Computer Science Education

Keywords

Audience Response Systems, Classroom Response Systems, StuReSy, Web development, React, Redux, WebRTC, JavaScript

Abstract

So-called classroom response systems (CRS) are being used in many universities. Students can use their smartphone or other connected devices to answer questions about the class' contents. However many CRS are not ideally suited for computer science education. Within this thesis, existing CRS will be compared in regards to their compatibility with programming education. After that, a new software will be devised and implemented as a prototype, which sets itself apart from other CRS in four main aspects:

It is completely **web based** and runs as a JavaScript application within the browser. Downloading software or installing plugins is not necessary.

The connection between the students and the instructor will be created **directly between the browsers**. A dedicated server system will not be used.

The application is optimised to display and embed Java source code by using **syntax highlighting** and monospace fonts.

Additionally, Java source code which is embedded in the content of a question will be able to run right in the browser. To achieve this, a Java Virtual Machine (implemented in JavaScript) will be run in the browser.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Was sind Classroom-Response-Systeme?	1
1.2	Motivation: Vom Software-Download zur Web-Applikation	3
2	Anforderungen an ein modernes CRS für die Programmierlehre	5
2.1	Vollständige Umsetzung als Web-Applikation	5
2.2	Peer-to-Peer-Verbindungen zwischen den Nutzern	5
2.3	Formatierungsmöglichkeiten für Quelltext	6
2.4	Quelltext-Ausführung im Browser	6
3	Bewertung bestehender CRS-Lösungen	7
3.1	StuReSy	7
3.2	Pingo	11
3.3	Tabellarische Gegenüberstellung	13
4	Entwurf und Implementierung eines modernen CRS für die Programmierlehre	14
4.1	Implementierung als Single-Page-Application mit dem React-Framework	15
4.1.1	Komponenten, Eigenschaften und Zustand	16
4.1.2	Globale Zustandsverwaltung mit Redux	20
4.1.3	Konkrete Umsetzung am Beispiel des Fragen-Editors	24
4.2	Peer-to-Peer-WebRTC-Verbindungen mit PeerJS	28
4.3	Text- und Code-Formatierung mit Quill und CodeMirror	33
4.4	Code-Ausführung im Browser via DoppioJVM	37
5	Fazit und Ausblick	43
5.1	Diskussion der Ergebnisse	43
5.2	Ausblick	46

Abbildungsverzeichnis

3.1	Hauptmenü der StuReSy-Editor-Komponente.	8
3.2	Probleme bei der Darstellung von Quelltexten in StuReSy (1)	9
3.3	Probleme bei der Darstellung von Quelltexten in StuReSy (2)	10
3.4	Fragen-Editor in Pingo ohne Formatierungsmöglichkeiten	12
3.5	Darstellung von Quelltexten in Pingo	12
4.1	Redux Dataflow	21
4.2	Datenfluss einer typischen Redux-Anwendung.	23
4.3	Komponenten-Hierarchie des Fragen-Editors	24
4.4	Fertiggestellter Fragen-Editor in Weclare	25
4.5	Start einer neuen Weclare-Sitzung	28
4.6	StuReSy-Fragen-Editor	34
4.7	Darstellung eines Code-Fragments in StuReSy	34
4.8	Quill-Editor von Weclare	35
4.9	Darstellung eines Code-Fragments in Weclare	35
4.10	Darstellung eines ausführbaren Java-Quelltexts in Weclare	36
4.11	Ausführungsmöglichkeiten eines Java-Quelltexts im Browser.	37
4.12	Ausführung einer Java-Klasse in Weclare mittels DoppioJVM.	42

Quelltextverzeichnis

4.1	Einfache React-Komponente ohne JSX-Syntax.	16
4.2	Einfache React-Komponente mit JSX-Syntax.	17
4.3	Komponenten erhalten Daten über ihre Eigenschaften (props).	17
4.4	Eigenschaften werden wie normale HTML-Attribute verwendet.	17
4.5	Jede Komponente kann über einen modifizierbaren Zustand verfügen. . . .	19
4.6	„Lifting state up“: Mehrere Komponenten greifen auf die gleichen Daten zu.	20
4.7	Über den connect-Aufruf beim Exportieren der Komponente wird sie mit dem Store verbunden. (aus: src/client/components/ClientHeaderContainer.js)	22
4.8	Ein Action-Objekt ist die Beschreibung einer Änderungsoperation und wird in einem ActionCreator erzeugt.	22
4.9	In einem Reducer werden die Änderungsoperationen eines Stores als pure Funktion implementiert.	23
4.10	Auszug aus dem Reducer für den Fragen-Editor (aus: src/server/reducers/questions.js)	26
4.11	Zustand des Fragen-Editors innerhalb des Redux-Store mit einer einzigen Frage.	27
4.12	Verbindungsaufbau mit der PeerJS-Bibliothek auf der Server-Seite. (aus: src/server/actions/server.js)	31
4.13	Verbindungsaufbau mit der PeerJS-Bibliothek auf der Client-Seite. (aus: src/client/actions/client.js)	31
4.14	ActionCreator zum Versenden von Antworten vom Client zum Server. (aus: src/client/actions/client.js)	32
4.15	Instanziierung der DoppioJVM (aus: src/server/actions/doppio.js).	40
4.16	Java-Loader, der programmatisch Quelltext kompiliert und ausführt. (aus: public/doppio/Loader.java)	41

Abkürzungsverzeichnis

HAW	Hochschule für Angewandte Wissenschaften
CRS	Classroom-Response-System
ARS	Audience-Response-System
StuReSy	Student Response System
SPA	Single-Page-Application
MPA	Multi-Page-Application
JVM	Java Virtual Machine
JRE	Java Runtime Environment
Weclare	Web Classroom Response (System)
WASM	WebAssembly
AJAX	Asynchronous JavaScript and XML
JSX	JavaScript Syntax Extension

1 Einleitung

1.1 Was sind Classroom-Response-Systeme?

Interaktivität ist ein Schlüsselfaktor für gute Lehrveranstaltungen (vgl. [28, S. 1]). Der klassische Frontalunterricht gilt nicht mehr als besonders effektiv. Gleichzeitig ist es für Lehrende immer schwieriger, besagte Interaktivität herzustellen, je größer die Anzahl der Teilnehmer einer Veranstaltung ist. In kleinen Gruppen kann durch mündliche Kommunikation meistens noch ein gewisser Grad an Interaktivität garantiert werden. Im Lehrbetrieb von großen Hochschulen und Universitäten ist das aber aufgrund der hohen Teilnehmerzahlen kaum noch realisierbar. (vgl. [14, S. 389])

Eine Möglichkeit, die Interaktivität zu steigern, ist der Einsatz sogenannter Audience-Response-Systeme (ARS) (vgl. [28, S. 5]). Im akademischen und schulischen Umfeld werden diese Systeme auch noch spezifischer als Classroom-Response-Systeme (CRS) bezeichnet. Dabei handelt es sich um Hardware- oder Software-Lösungen, die es einem Dozenten ermöglichen, während einer Veranstaltung beliebig viele Fragen nacheinander an das Publikum zu stellen. Diese können entweder mithilfe dedizierter Hardware-Geräte (sogenannte Clicker) oder mit einem Smartphone, Laptop oder Tablet-PC beantwortet werden. Üblicherweise werden dabei verschiedene Fragetypen unterstützt, zum Beispiel Simple- und Multiple-Choice-Varianten.

Ein Merkmal dieser Systeme ist die Anonymität der Antworten, denn nur so können die Teilnehmer ehrlich antworten und lernen, ihren Wissensstand tatsächlich besser einzuschätzen (vgl. [6, S. 106]). CRS sind explizit keine Werkzeuge, um digitale Tests oder Prüfungen durchzuführen.

Die Nutzung solcher Systeme im akademischen Umfeld ist bereits relativ weit verbreitet (vgl. [14, S. 392]). Weltweit werden diese Lösungen an vielen Hochschulen eingesetzt (vgl.

[4]). Studien und Umfragen attestieren diesen Systemen eine Steigerung der Interaktivität, eine Verbesserung der Selbsteinschätzung und eine Steigerung der Aufmerksamkeit in Lehrveranstaltungen, zum Beispiel an der University of Wisconsin-Milwaukee [13, S. 5]:

„Students similarly reported that the use of clickers increased their engagement, involvement, and interaction, and help students pay attention in class.“

Oder auch nach zehnjährigem Einsatz an der Harvard University [3, S. 6]:

„We find that, upon first implementing Peer Instruction, our students’ scores on the Force Concept Inventory and the Mechanics Baseline Test improved dramatically, and their performance on traditional quantitative problems improved as well.“

Eine Bewertung des didaktischen Konzepts von CRS ist daher nicht Bestandteil dieser Arbeit. Die Wirksamkeit und Relevanz wird als gegeben angenommen.

Auch an der Hochschule für Angewandte Wissenschaften (HAW) Hamburg werden CRS eingesetzt. Eine der verwendeten Software-Lösungen, die dort im Department Informatik zum Einsatz kommt, ist das Student Response System (StuReSy) – eine Software, die im Rahmen einer Bachelorarbeit an der Universität Hamburg entwickelt wurde [27]. Wegen der positiven Erfahrungen, die der Autor und Herr Axel Schmolitzky, einer der Lehrenden an der HAW Hamburg, mit der Software gemacht haben, wird StuReSy im Lauf dieser Arbeit als funktionales Vorbild für die Entwicklung eines neuen CRS dienen, welches das System konzeptionell verbessert, bestehende Probleme behebt und das Anwendungsgebiet stärker auf den Spezialfall der Programmierlehre zuspitzt.

1.2 Motivation: Vom Software-Download zur Web-Applikation

Neben den spezifischen, fachlichen Problemen, die StuReSy mit sich bringt und die im weiteren Verlauf dieser Arbeit noch genauer benannt werden, steckt auch eine idealistische Motivation in dieser Arbeit.

Seit wenigen Jahren etabliert sich eine moderne Variante der Software-Distribution: die Web-Applikation. Der Internet-Browser, der ursprünglich nur dazu gedacht war, statische Texte und Bilder darzustellen, wurde immer leistungsfähiger und hat sich zu einer beliebten Anwendungsplattform entwickelt (vgl. [16, S. 1]). Unter anderem sind dank der Skriptsprache JavaScript Webbrowser heute leistungsfähige und umfangreiche, clientseitige Laufzeitumgebungen, wie die folgenden Beispiele zeigen.

Gerade Consumer-Software wie Officeanwendungen (Google Docs¹, Microsoft Office 365²), Grafikprogramme (Figma³, Draw.io⁴), Musik- und Videoplayer (Spotify⁵, Netflix⁶) oder Messenger (Telegram Web⁷, WhatsApp Web⁸) sind Anwendungen, die heute üblicherweise im Browser ausgeführt werden, ohne vorher (sichtbar) heruntergeladen und installiert werden zu müssen. Die Liste dieser Anwendungen wächst stetig und mit WebAssembly steht eine neue Technologie in den Startlöchern, die dafür sorgen könnte, dass zukünftig auch besonders leistungshungrige und komplexe Anwendungen wie etwa 3D- oder CAD-Software, Videobearbeitung und Spiele im Browser ausgeführt werden.

Webbrowser stellen also bereits heute eine Abstraktionsebene dar, die es ermöglicht, plattformunabhängige Software zu entwickeln. Damit nehmen sie immer öfter eine Funktion ein, die bisher zum Beispiel von virtuellen Maschinen (etwa der Java Virtual Machine) oder Browser-Plugins (Adobe Flash, Java-Applets, ActiveX) erledigt wurde.

Im Gegensatz zu diesen älteren Technologien ist ein Browser aber auf den meisten Rechnern bereits installiert, oder wird gar mit dem Betriebssystem ausgeliefert (zum Beispiel Microsoft

¹<https://docs.google.com>

²<https://www.office.com>

³<https://www.figma.com>

⁴<https://www.draw.io>

⁵<https://www.spotify.com>

⁶<https://www.netflix.com>

⁷<https://web.telegram.org>

⁸<https://web.whatsapp.com>

Edge bei Windows 10 oder Safari bei Apple Mac OS X). Die gesamte, sogenannte User Experience eines Browsers ist besser als die der bisherigen Lösungen. Die Benutzung und das Funktionsprinzip von Adobe Flash oder der Java Virtual Machine sind für Laien schwer nachzuvollziehen. Der Gebrauch eines Browsers hingegen ist heute vielen Menschen vertraut.

Ein weiterer Vorteil der Webbrowser ist, dass sie offene Web-Standards interpretieren und es sich nicht um die proprietäre Technologie eines einzigen Herstellers handelt. Damit haben Browser und Web-Technologien gute Voraussetzungen, um zukünftig eine immer größere Rolle bei der Entwicklung plattformunabhängiger Software zu spielen.

Die konzeptionellen Vorteile von Web-Anwendungen für die Nutzer sind offensichtlich: Das Auffinden der Software ist gleichbedeutend mit dem Merken einer Internetadresse. Das Herunterladen der Software wird vor dem Nutzer versteckt (sichtbar ist nur das Aufrufen einer Webseite, auch wenn im Hintergrund weiterhin Code heruntergeladen wird) und eine Installation entfällt. Software-Updates werden ebenfalls vor dem Nutzer verborgen (bei jedem Aufruf der Applikation wird automatisch die neuste Programmversion geladen).

Aber natürlich gibt es auch Nachteile: Web-Applikationen benötigen meistens eine dauerhafte und schnelle Internetverbindung, je nachdem welcher Anteil der Anwendung beim Client im Browser ausgeführt wird, und wie viel der Server vorab berechnet. Außerdem ist die Leistungsfähigkeit von Web-Applikationen gegenüber nativen Anwendungen an einigen Stellen noch unterlegen, zum Beispiel durch den eingeschränkten Zugriff auf Betriebssystem-Schnittstellen.

Ob und wie ein CRS sinnvoll als moderne Web-Applikation realisiert und weiterentwickelt werden kann, wird in den folgenden Kapiteln überprüft.

2 Anforderungen an ein modernes CRS für die Programmierlehre

Im Gespräch mit Herrn Schmolitzky konnten unter Berücksichtigung bisheriger Erfahrungen beim Einsatz von StuReSy sowohl auf Seiten der Studenten als auch auf Seiten der Dozenten, vier Anforderungen für den Einsatz von CRS in der Programmierlehre aufgestellt werden.

2.1 Vollständige Umsetzung als Web-Applikation

Die Zugänglichkeit einer Web-Applikation, die vollständig im Browser und ohne (vom Nutzer betätigte) Downloads verwendet werden kann, ist gerade für den Einsatz in der Universität wichtig. Der Download oder die Installation zusätzlicher Software stellt eine unnötige Hürde für den Einsatz der Software dar. Während andere CRS durch spezielle Anforderungen (z.B. die Unterstützung von Hardware-Clickern) dazu gezwungen sind, als native Anwendung zu laufen, gibt es im vorliegenden Fall keine solchen Gründe. Die Unterstützung von dedizierten Hardware-Geräten erscheint wegen der Allgegenwärtigkeit von vernetzten Computern heute nicht mehr zeitgemäß. Die Umsetzung als Web-Applikation ermöglicht außerdem die plattformübergreifende Nutzung auf verschiedenen Geräten. Ein modernes CRS im akademischen Einsatz soll daher vollständig webbasiert sein.

2.2 Peer-to-Peer-Verbindungen zwischen den Nutzern

Um die Langlebigkeit eines CRS zu erhöhen, soll der Wartungsaufwand einer solchen Software so gering wie möglich ausfallen. Außerdem muss die Hürde zum Einsatz gerade

gegenüber den Dozenten möglichst weit gesenkt werden. Der Betrieb eines eigenen, dedizierten Anwendungs-Servers mit individuellen Anforderungen (z.B. vorhandene Interpreter oder Datenbanksysteme) widerspricht diesem Prinzip. Daher soll ein modernes CRS ohne dedizierten Anwendungs-Server funktionieren, und stattdessen auf Direktverbindungen unter den Teilnehmern setzen. Da es sich um eine Web-Applikation handelt, wird natürlich weiterhin ein Webserver benötigt, der die Anwendung ausliefert.

2.3 Formatierungsmöglichkeiten für Quelltext

Im Kontext der Programmierlehre ist die Darstellung von Quelltexten in Fragestellungen unerlässlich. Um die Lesbarkeit von kurzen und langen Quelltext-Ausschnitten zu gewährleisten, ist eine optische Hervorhebung solcher Ausschnitte notwendig. Das beinhaltet sowohl die Unterstützung von simplen Formatierungsmöglichkeiten (z.B. das Einfügen von Absätzen oder den Einsatz von Monospace-Schriftarten) um Code von Fließtext abzuheben, als auch die Verwendung von Syntax-Highlighting um längere Abschnitte übersichtlich darzustellen.

2.4 Quelltext-Ausführung im Browser

Um ein CRS noch spezifischer auf das Einsatzgebiet der Programmierlehre zuzuschneiden, wird außerdem evaluiert, ob sich die Abhängigkeit von weiteren Programmen wie etwa Entwicklungsumgebungen reduzieren lässt, indem die Ausführung von Java-Quelltexten direkt im CRS ermöglicht wird. Der Dozent soll keine weiteren Anwendungen neben dem CRS benötigen. Ein Parallelbetrieb von CRS und Entwicklungsumgebung, um zwischen Code-Ausführung und Fragestellung hin- und herzuschalten, ist unübersichtlich und verhindert den administrativen Einsatz eines CRS auf einem fremden Computer.

3 Bewertung bestehender CRS-Lösungen

Zwei bestehende CRS aus dem akademischen Bereich werden nun in Bezug auf die ermittelten Anforderungen bewertet und miteinander verglichen: Einerseits die bisher an der HAW Hamburg eingesetzte Lösung StuReSy und zum Vergleich eine populäre, professionellere Lösung namens Pingo.

3.1 StuReSy

StuReSy ist eine freie Software, die im Rahmen der Bachelorarbeit von Herrn Wolf Posdorfer im Jahr 2012 an der Universität Hamburg entstanden ist [27]. Der Name StuReSy ist ein Akronym für „**S**tudent **R**esponse **S**ystem“. StuReSy wurde erfolgreich und viele Jahre an der Universität Hamburg und der HAW Hamburg eingesetzt.

StuReSy besteht aus zwei Komponenten:

- **Server (in PHP geschrieben)**: Stellt die Client-Benutzeroberfläche für die Abstimmungs-Teilnehmer und eine Administrations-Oberfläche bereit. Benötigt eine relationale SQL-Datenbank (vgl. [10]).
- **Editor (in Java geschrieben)**: Verbindet sich mit dem Server, um Fragen zu erstellen, zu bearbeiten und Umfragen zu starten (vgl. [9]).

Der StuReSy-Editor verfügt über folgende Hauptfunktionen bzw. Programmteile:

- **Abstimmung**: Durchführung einer Umfrage
- **Fragen-Editor**: Erstellung und Bearbeitung von Fragesätzen.
- **Abstimmungs-Analyse**: Auswertung von Abstimmungs-Ergebnissen im Nachhinein.

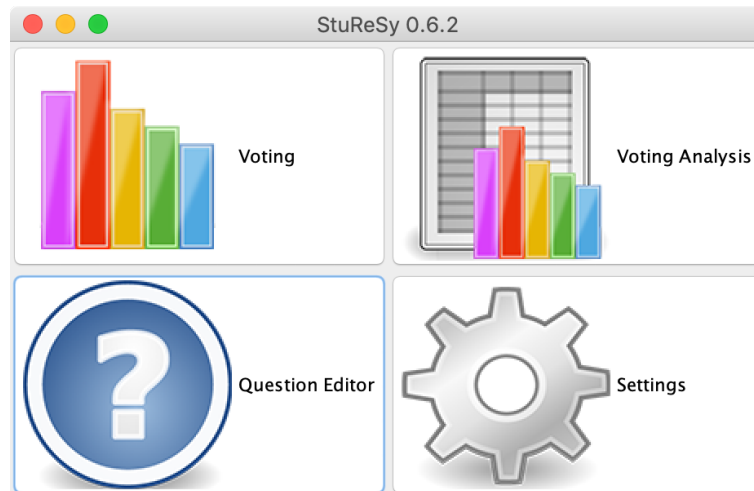


Abbildung 3.1: Hauptmenü der StuReSy-Editor-Komponente.

Auch wenn StuReSy erfolgreich im Hochschulbetrieb verwendet wurde, so gibt es dennoch einige konzeptionelle Nachteile:

- **Software-Download und Java notwendig:** Um StuReSy administrativ einsetzen zu können, muss zwangsläufig Software heruntergeladen werden und eine Java-Installation muss auf dem jeweiligen System vorhanden sein. Administrativer Zugang vom Tablet oder Smartphone aus ist damit kaum möglich, da die Ausführung von Java-Anwendungen auf mobilen Betriebssystemen wie Google Android und Apple iOS nicht ohne Weiteres möglich ist.
- **Anwendungs-Server notwendig:** Um StuReSy betreiben zu können, wird zwingend eine Instanz des StuReSy-Servers benötigt. Diese muss von der jeweiligen Institution oder einem Dozenten aufgesetzt und gewartet werden.
- **Kompliziertes System von Tokens und Lecture-IDs:** Um eine Abstimmung in StuReSy durchzuführen, muss zunächst eine Lecture-ID in der Server-Admin-Oberfläche eingerichtet werden. Anschließend muss ein generierter Token in den Java-Client übertragen werden. Diese Vorgehensweise erscheint unnötig kompliziert und sorgt dafür, dass nur der Administrator der Server-Komponente neue Lecture-IDs einrichten kann. Ein niedrighschwelliger Einsatz (zum Beispiel für Studenten untereinander oder für Dozenten zum Ausprobieren) wird damit erschwert.

3 Bewertung bestehender CRS-Lösungen

Neben den konzeptionellen Problemen gibt es auch Probleme mit der Implementierung von StuReSy, die sich gerade im Bereich der Programmierlehre bemerkbar machen, denn dort beinhalten die Fragetexte oft Quelltext-Ausschnitte:

Die Darstellung von Quelltexten in StuReSy ist aufwendig. Obwohl die Formatierung von Fragen mittels HTML unterstützt wird, ist es komplex, optisch ansprechende und einheitliche Ergebnisse zu erzielen. Nach dem Einfügen von Quelltexten aus der Zwischenablage müssen sowohl Zeilenumbrüche und Einrückungen als auch die optische Abhebung des Quelltexts (z.B. durch eine andere Schriftart, Größe oder Farbe) manuell vorgenommen werden. Einen Button oder eine Voreinstellung für Code-Formatierung gibt es nicht. Die Ausrichtung der Quelltext-Blöcke ist anspruchsvoll, denn der Quelltext selbst muss aufgrund der Lesbarkeit linksbündig ausgerichtet sein – gleichzeitig soll ein ganzer Quelltext-Block zentriert unter der Fragestellung auftauchen und nicht am linken Bildrand festhängen. Um solche Formatierungen zu erzielen kann es auch notwendig sein, den HTML-Code der Fragestellung manuell bearbeiten zu müssen. Beispiele dieser Probleme sind in den Abbildungen 3.2 und 3.3 dargestellt.

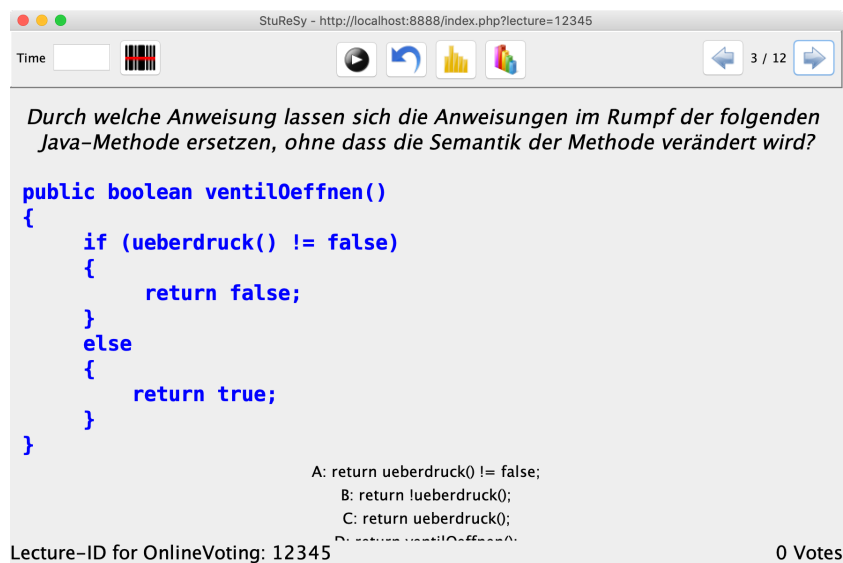


Abbildung 3.2: Probleme bei der Darstellung von Quelltexten: Im Gegensatz zur Fragestellung wirkt der linksbündige Quelltext deplatziert. Eine Antwort-Möglichkeit wird vom zu kleinen Fenster verdeckt. Syntax-Highlighting ist nicht vorhanden.

3 Bewertung bestehender CRS-Lösungen

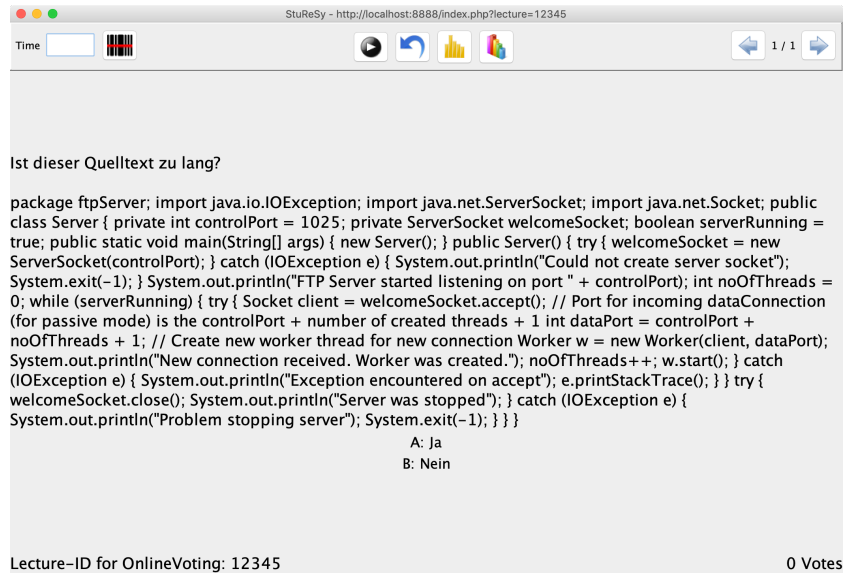


Abbildung 3.3: Beim Einfügen von Quelltexten aus der Zwischenablage gehen sämtliche Absätze und Einrückungen verloren und müssen manuell wieder eingefügt werden.

3.2 Pingo

Pingo ist eine Software-Lösung, die bereits seit dem Jahr 2011 an der Universität Paderborn entwickelt wird. Der Name ist ebenfalls ein Akronym und steht für „**P**eer **I**nstruction for **V**ery Large **G**roups“. Im Gegensatz zu StuReSy ist Pingo bereits weiter verbreitet und wird an vielen deutschen Hochschulen eingesetzt – im September 2018 gab es 22.000 angemeldete Nutzer (vgl. [1]). Dahinter stand außerdem ein ganzes Team akademischer Mitarbeiter (vgl. [31]). Seit 2019 wird Pingo von der universitätsnahen Coactum GmbH betrieben und weiterentwickelt (vgl. [2]). Das Projekt ist damit professioneller ausgerichtet als StuReSy.

Im Gegensatz zu StuReSy ist Pingo eine reine Web-Applikation, die öffentlich unter `http://trypingo.com/` auffindbar ist und kostenlos genutzt werden kann. Sowohl Administratoren als auch Teilnehmer können alle Arbeiten im Browser erledigen, ein Software-Download ist nicht notwendig. Für die administrative Nutzung muss jedoch ein Benutzerkonto erstellt werden.

Eine Direktverbindung zwischen Umfrage-Teilnehmern wird nicht eingesetzt – stattdessen wird ein Pingo-Server benötigt. Der Betreiber stellt einen öffentlichen Pingo-Server zur Verfügung. Pingo steht zudem unter einer Open-Source-Lizenz, so dass Nutzer auch eine eigene Instanz betreiben können. Pingo ist in der Programmiersprache Ruby und mithilfe des Web-Frameworks „Ruby on Rails“ implementiert (vgl. [8]). Entsprechend dazu muss ein potenzieller Server auch über einen Ruby-Interpreter und über eine NoSQL-Datenbank verfügen, um Pingo ausführen zu können.

In ihren Kernfunktionen sind sich Pingo und StuReSy sehr ähnlich. Trotzdem fehlt eine kritische Funktion für den Einsatz in der Programmierlehre:

Fragen innerhalb der Pingo-Plattform können nicht formatiert werden. Damit können selbst simple Formatierungen wie Fettschreibungen, Unterstreichungen oder Zeilenumbrüche nicht verwendet werden (erkennbar in Abbildung 3.5). Dementsprechend ist auch die übersichtliche Darstellung von Quelltext nicht möglich und Pingo für den Einsatz in der Programmierlehre ungeeignet.

Edit question

Name*

Durch welche Anweisung lassen sich die Anweisungen im Rumpf der folgenden Java-Methode ersetzen, oh

Tags

code x Tags

add from your tags:

Public

Abbildung 3.4: Der Fragen-Editor von Pingo verfügt über keine Formatierungsmöglichkeiten.

Test Session

Durch welche Anweisung lassen sich die Anweisungen im Rumpf der folgenden Java-Methode ersetzen, ohne dass die Semantik der Methode verändert wird? public boolean ventilOeffnen()
{ if (ueberdruck() != false) { return false; } else { return true; } }

Choose an option:

return ueberdruck() != false;

return !ueberdruck();

return ueberdruck();

return ventilOeffnen();

Abbildung 3.5: Eine ordentliche Darstellung von Quelltexten ohne Textformatierungen ist nicht möglich.

3.3 Tabellarische Gegenüberstellung

Nach dem Aufstellen der Anforderungen und der Betrachtung beider CRS lassen sich diese Ergebnisse wie folgt tabellarisch gegenüberstellen:

	StuReSy	Pingo	Gewünscht
komplett webbasiert	x	✓	✓
kommt ohne Server aus	x	x	✓
Code-Formatierungsoptionen	✓	x	✓
Code-Ausführung	x	x	✓

Tabelle 3.1: Tabellarischer Vergleich verschiedener CRS-Systeme.

4 Entwurf und Implementierung eines modernen CRS für die Programmierlehre

Im Rahmen dieser Arbeit wird ein CRS entworfen und als Prototyp implementiert, welches die besagten Kern-Anforderungen erfüllt. Die entstehende Software heißt Weclare (ein Akronym für „**W**eb **C**lassroom **R**esponse (System)“). Der gesamte Quelltext zu dem Projekt befindet sich in einem öffentlichen GitHub-Repository¹ und eine öffentlich zugängliche Version der Software kann unter <https://weclare.de> aufgerufen werden. Insgesamt hat die entstandene Implementierung einen Umfang von rund 6.000 Zeilen Code (vergleichbar mit dem Vorbild StuReSy). Die Software wird als freie Software unter einer GPL-3.0-Lizenz veröffentlicht (vgl. [11]).

Eine weitere, beiläufige Anforderung an das neue System: Zur Evaluierung soll Weclare auch mit bestehenden Fragesätzen aus StuReSy getestet werden. Aus diesem Grund wird außerdem ein Werkzeug entwickelt, welches Fragesätze vom XML-basierten StuReSy-Format in das JSON-basierte Weclare-Format konvertiert. Dieser Konverter wird ebenfalls in JavaScript geschrieben und ist als Kommandozeilen-Werkzeug entworfen. Es benötigt daher die Node.js-Laufzeitumgebung, um JavaScript außerhalb des Browsers ausführen zu können. Auch dieses Hilfsprogramm kann in einem öffentlichen GitHub-Repository² gefunden werden.

Die Umsetzung der wichtigsten Kern-Aspekte von Weclare wird auf den kommenden Seiten exemplarisch beschrieben.

¹<https://github.com/pReya/weclare>

²<https://github.com/pReya/weclare-sturesy-converter>

4.1 Implementierung als Single-Page-Application mit dem React-Framework

Um eine webbasierte Anwendung zu erstellen, die ohne einen zentralen Anwendungs-Server auskommt, muss die Software komplett clientseitig in einem Browser ausgeführt werden. Da JavaScript die einzige Sprache (neben dem neuen WebAssembly-Bytecode – siehe Kapitel 5.2) ist, die von allen Browsern ausgeführt werden kann, wird folglich auch die gesamte Anwendung in JavaScript entworfen. Ein klassisches Backend, also eine Datenschicht (üblicherweise handelt es sich bei den meisten Web-Applikationen mindestens um eine Zwei-Schichten-Architektur) existiert nicht, bzw. ist in die Präsentationsschicht integriert. Das bedingt die Kategorisierung der Anwendung als „Fat Client“.

Durch die vollständig clientseitige Ausrichtung bietet es sich an, die Software als sogenannte Single-Page-Application (SPA) zu implementieren. Herkömmliche Webseiten, sogenannte Multi-Page-Applications (MPA) laden während ihrer Lebenszeit mehrfach neue Seiten vom Server, zum Beispiel jedes Mal, wenn innerhalb der Anwendung eine neue Ansicht dargestellt werden soll. Eine SPA dagegen lädt nur eine einzige Webseite sowie das zugehörige JavaScript-Programm und verändert diese eine Seite im weiteren Verlauf dynamisch. Neue Daten werden bei Bedarf asynchron (ohne Blockieren der Seite) nachgeladen, es wird jedoch keine komplette Seite geladen. (vgl. [15, S. 1])

Damit wird auch eine maximale Autarkie gegenüber dem Webserver erreicht, da dieser nur mit wenigen Requests (im einfachsten Fall ein Request pro Nutzer) umgehen muss. Die Anforderungen an den Webserver, auf dem die Anwendung bereitgestellt wird, sind sehr gering – er muss lediglich statische Dateien (HTML, JavaScript, Bilder, Schriften, etc.) bereitstellen. Damit kann die Anwendung zum Beispiel auf dem kostenlosen Hosting-Angebot von GitHub³ betrieben werden, so wie es auch bei der öffentlich zugänglichen Weclare-Instanz unter <https://weclare.de/> der Fall ist.

Als Framework für die Implementierung einer solchen SPA wird das React-Framework⁴ ausgewählt. Das Open-Source-Projekt existiert seit 2013 und wird von Facebook finanziert und gefördert. Es gehört zu den populärsten Frameworks zum Erstellen von Benutzeroberflächen und Web-Applikationen (vgl. [29]) und ist dem Autor bereits vertraut.

³Offizielle Webseite: <https://pages.github.com/>

⁴Offizielle Webseite: <https://reactjs.org/>

Um Implementierungsdetails nachzuvollziehen, erfolgt an dieser Stelle eine Einführung in einige Grundkonzepte von React.

4.1.1 Komponenten, Eigenschaften und Zustand

Die elementaren Bausteine einer React-Anwendung sind Komponenten. Die React-Dokumentation beschreibt die Aufgabe von Komponenten wie folgt [19, Header-Sektion]:

„Build encapsulated components that manage their own state, then compose them to make complex UIs.“

Ein Komponente ist eine autarke und wiederverwendbare Einheit und kapselt meistens sowohl Struktur, Aussehen als auch Logik. Komponenten werden häufig als Klasse implementiert (können aber in einfachen Fällen auch als Funktion implementiert werden – siehe Quelltext 4.6) und erben von der Klasse `React.Component`. Valide Komponenten müssen über eine `render()`-Funktion verfügen, die HTML oder andere React-Komponenten zurückliefert. Ein Beispiel einer simplen, statischen Komponente sieht so aus:

```
1 import React from "react";
2
3 class Greeting extends React.Component {
4   render() {
5     return React.createElement("div", null, "Hello there!");
6   }
7 }
```

Quelltext 4.1: Einfache React-Komponente ohne JSX-Syntax.

Da die Syntax des Aufrufs `React.createElement()` nicht so kompakt ist, wie die Notation eines XML-Tags (`<div> . . </div>`), wird im React-Umfeld üblicherweise eine Syntax-Erweiterung namens JSX (JavaScript Syntax Extension) verwendet, um React-Komponenten einfacher beschreiben zu können. JSX wird mit einem Compiler während des Build-Prozesses in herkömmliches JavaScript umgewandelt. Äquivalent zum letzten Beispiel ist daher die folgende Variante unter Einbeziehung von JSX-Syntax:


```
1 import React from 'react';
2
3 class Greeting extends React.Component {
4   render() {
5     return <div>Hello there!</div>;
6   }
7 }
```

Quelltext 4.2: Einfache React-Komponente mit JSX-Syntax.

Um Komponenten dynamisch zu machen, können über sogenannte Eigenschaften (Properties, kurz: props) Daten an Komponenten übergeben werden. Auf diese kann mit dem props-Objekt zugegriffen werden:

```
1 import React from "react";
2
3 class Greeting extends React.Component {
4   render() {
5     return <div>Hello, {this.props.name}!</div>;
6   }
7 }
```

Quelltext 4.3: Komponenten erhalten Daten über ihre Eigenschaften (props).

Eigenschaften werden, wie andere HTML-Attribute auch, hinter den Namen einer Komponente innerhalb des zugehörigen Tags in die Instanziierung einer Komponente integriert:

```
1 import React from "react";
2
3 class GreetAllFriends extends React.Component {
4   render() {
5     return (
6       <div>
7         <Greeting name="Michael" />
8         <Greeting name="Karla" />
9       </div>
10    );
11  }
12 }
```

Quelltext 4.4: Eigenschaften werden wie normale HTML-Attribute verwendet.

Eigenschaften werden also verwendet, um Daten in Komponenten hineinzureichen. React kümmert sich dann automatisch um das Aktualisieren der Ansicht, sobald sich eine Eigenschaft ändert (dieses „reaktive“ Prinzip ist auch der Namensgeber für das Framework). Dabei verwendet React sehr schnelle und intelligente Algorithmen, um immer nur diejenigen Elemente einer Seite zu aktualisieren, die sich auch tatsächlich geändert haben – der gesamte Prozess nennt sich Reconciliation (vgl. [18]). Dadurch können Single-Page-Applications tendenziell schneller agieren als Multi-Page-Applications, vorausgesetzt es werden keine neuen Daten abgefragt.

Eigenschaften können nicht modifiziert werden – sie sind wie auch die meisten anderen Objekte innerhalb von React „immutable“. Wenn Daten innerhalb einer Komponente modifiziert werden sollen, gehören sie in den internen, modifizierbaren Zustand – also in das state-Objekt dieser Komponente:

```
1 import React from "react";
2
3 class BusyOrNot extends React.Component {
4   state = {
5     busy: false
6   };
7
8   toggleBusy() {
9     this.setState(prevState => ({
10      busy: !prevState.busy
11    }));
12  }
13
14  render() {
15    return (
16      <div>
17        <div>This user is {busy ? "busy" : "not busy"}!</div>
18        <button type="button" onClick={this.toggleBusy}>
19          Change Busy State
20        </button>
21      </div>
22    );
23  }
24 }
```

Quelltext 4.5: Jede Komponente kann über einen modifizierbaren Zustand verfügen.

Bedingt durch die statische Natur der Eigenschaft, forciert React einen unidirektionalen Datenfluss. Daten können nur von „oben nach unten“ (in Bezug auf die Baumstruktur im Document-Object-Model einer Seite) durch eine Anwendung fließen. Möchten zwei Komponenten an unterschiedlichen Stellen auf die gleichen Daten zugreifen, dann sollten diese oberhalb der beiden Komponenten in einer gemeinsamen Eltern-Komponente gehalten werden (vgl. [17]).

```
1 import React from "react";
2
3 function Son(props) {
4   return <p>I am the son of {props.parent}</p>
5 }
6
7 function Daughter(props) {
8   return <p>I am the daughter of {props.parent}</p>
9 }
10
11 class Parent extends React.Component {
12   state = {
13     parentName: "Peter Parent"
14   };
15
16   render() {
17     return (
18       <div>
19         <Son parent={this.state.parentName} />
20         <Daughter parent={this.state.parentName} />
21       </div>
22     );
23   }
24 }
```

Quelltext 4.6: „Lifting state up“: Mehrere Komponenten greifen auf die gleichen Daten zu.

Da dieses Muster bei großen Anwendungen aber schnell zu sehr aufwendigem „Durchstecken“ von Eigenschaften (sogenanntes „Prop Drilling“) durch mehrere Komponentenebenen führt, gibt es eine populäre Erweiterung für React zur Verwaltung eines einzigen, globalen Zustandes in der gesamten Anwendung: das Flux-Muster.

4.1.2 Globale Zustandsverwaltung mit Redux

Das Flux-Entwurfsmuster ist ebenfalls eine Entwicklung von Facebook. Prinzipiell handelt es sich um ein abstraktes Entwurfsmuster, das in vielen Sprachen angewendet werden kann. Etabliert hat es sich jedoch gerade in Kombination mit React-Anwendungen. Die

bekannteste Implementierung, die auch in dieser Arbeit verwendet wird, hört auf den Namen Redux⁵.

Im Redux-Konzept geht es darum, eine zentrale Zustandsverwaltung für eine Anwendung einzurichten, eine sogenannte „Single Source of Truth“. Dieser zentrale Ort wird als Store bezeichnet. Ein Store beinhaltet typischerweise solche Daten, die für die gesamte Anwendung relevant sind. Parallel dazu kann es aber weiterhin Komponenten geben, die einen eigenen, lokalen Zustand verwalten, wenn dieser nicht für die gesamte Anwendung relevant ist. (vgl. [22])

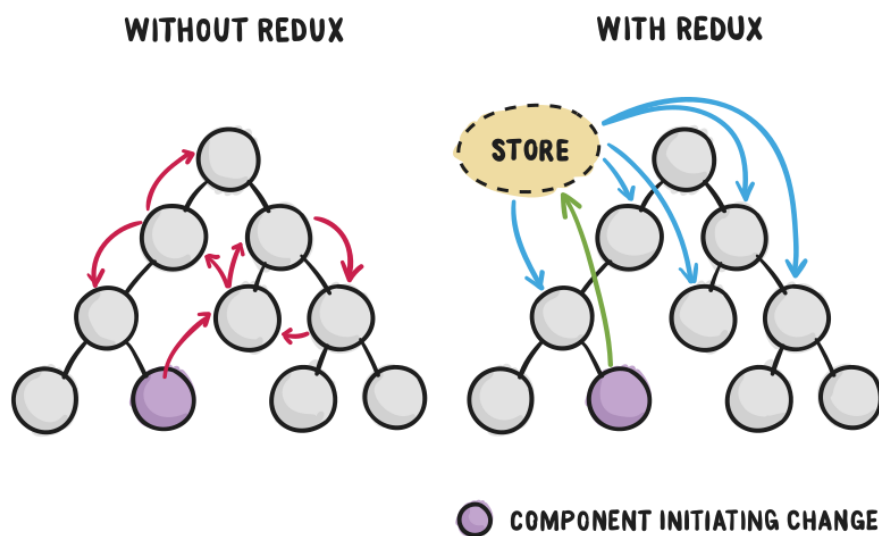


Abbildung 4.1: Der Redux-Store verwaltet den globalen Zustand einer React-Anwendung und ist die „Single Source of Truth“.⁶

Einzelne React-Komponenten können mit einem Store durch einen Publish/Subscribe-Mechanismus verbunden werden. In Redux wird diese Verbindung durch die connect-Funktion zur Verfügung gestellt. Die gewünschten Daten aus dem Store können in einer mapStateToProps-Funktion angegeben werden und stehen der Komponente dann als Eigenschaften zur Verfügung. (vgl. [24, Abschnitt „Implementing Container Components“])

⁵Offizielle Webseite: <https://redux.js.org/>

⁶Bildquelle: <https://css-tricks.com/learning-react-redux/> (aufgerufen am 24.04.2019).

Sobald sich die Daten im Store ändern, wird die verbundene Komponente sofort benachrichtigt und bei einer Änderung der Eigenschaften auch neu gerendert (reaktives Prinzip). Um eine möglichst lose Kopplung zwischen den Komponenten zu realisieren, wird deswegen empfohlen, die Verbindung zu einem Store in einer (nicht sichtbaren) Container-Komponente zu realisieren (vgl. [24, Abschnitt „Presentational and Container Components“]). Im folgenden Beispiel wird die eigentliche, sichtbare Header-Komponente mit einem unsichtbaren Container versehen, der die notwendigen Daten aus dem Store innerhalb der Komponente unter der `status`-Eigenschaft verfügbar macht.

```
1 import { connect } from "react-redux";
2 import Header from "../Header";
3
4 const mapStateToProps = state => ({
5   status: state.connection.status
6 });
7
8 export default connect(mapStateToProps)(Header);
```

Quelltext 4.7: Über den `connect`-Aufruf beim Exportieren der Komponente wird sie mit dem Store verbunden. (aus: `src/client/components/ClientHeaderContainer.js`)

Änderungen in einem Store müssen mithilfe von Actions realisiert werden. Bei einer Action handelt es sich lediglich um ein Objekt, welches die Art der Änderung in einem Store beschreibt (vgl. [20, Abschnitt „Actions“]). Um das wiederholte Schreiben langer Objekt-Literale zu erleichtern, werden die Actions üblicherweise von einer ActionCreator-Funktion erzeugt (vgl. [20, Abschnitt „Action Creators“]):

```
1 export function addQuestion(newQuestion) {
2   return {
3     type: "ADD_QUESTION",
4     payload: {
5       newQuestion
6     }
7   };
8 }
```

Quelltext 4.8: Ein Action-Objekt ist die Beschreibung einer Änderungsoperation und wird in einem ActionCreator erzeugt.

Die eigentliche Implementierung einer Änderungsoperation erfolgt in der zugehörigen reducer-Funktion. Die Operationen in einem reducer sind stets pure Funktionen, das heißt, sie liefern immer das gleiche Ergebnis bei gleichen Eingabe-Parametern und sie haben keine Seiteneffekte. Die Parameter des reducers sind der aktuelle Zustand des Stores und die eingehende Action, der Rückgabewert ist der neue Zustand des Stores. Die Signatur eines reducers lautet also $(previousState, action) \Rightarrow newState$ (vgl. [23]). Es folgt ein Beispiel für einen einfachen Reducer zum Hinzufügen einer Frage zu einem Fragenkatalog:

```
1 const questionEditor = (state = [], action) => {
2   switch (action.type) {
3     case "ADD_QUESTION": {
4       return [... state, createNewQuestion()];
5     }
6   }
7 };
```

Quelltext 4.9: In einem Reducer werden die Änderungsoperationen eines Stores als pure Funktion implementiert.

Der gesamte Datenfluss innerhalb einer typischen Redux-Anwendung ist im folgenden Schaubild zusammengefasst:

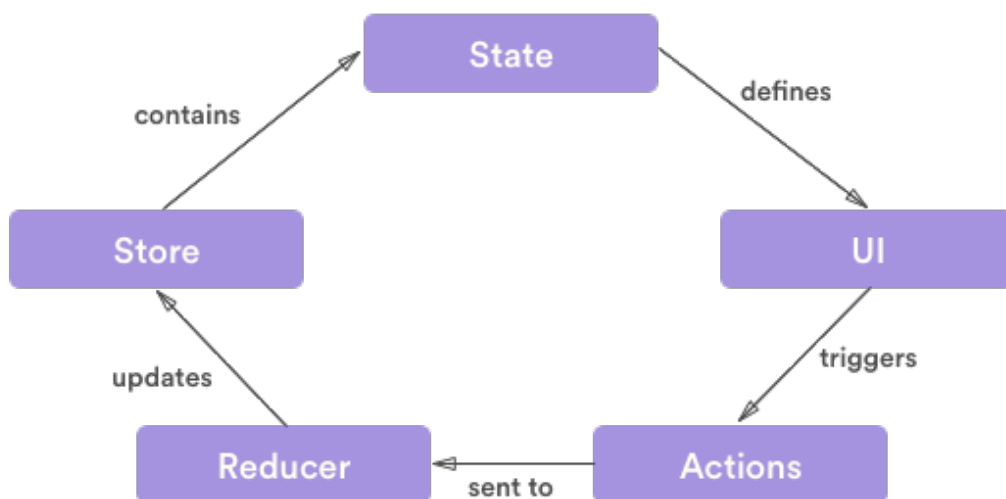


Abbildung 4.2: Der Datenfluss einer typischen Redux-Anwendung⁷

4.1.3 Konkrete Umsetzung am Beispiel des Fragen-Editors

Um das neue CRS zu implementieren, muss zunächst eine sinnvolle Aufteilung in React-Komponenten erfolgen. Exemplarisch wird an dieser Stelle einer der Hauptbestandteile der Anwendung besprochen: der Fragen-Editor.

Das zweiteilige Design, bestehend aus einer Fragenliste in der Seitenspalte und einem Fragen-Inhaltsbereich, das sowohl bei StuReSy als auch bei Pingo zum Einsatz kommt, wird beibehalten.

Eine mögliche Komponenten-Hierarchie für den Fragen-Editor wird in Abbildung 4.3 veranschaulicht. Die äußerste Komponente, der `QuestionEditorContainer`, ist nicht sichtbar. Es handelt sich dabei lediglich um eine Container-Komponente, welche die notwendigen Daten aus dem Store holt und dann an ihre Kinder-Komponenten als Eigenschaften weitergibt.

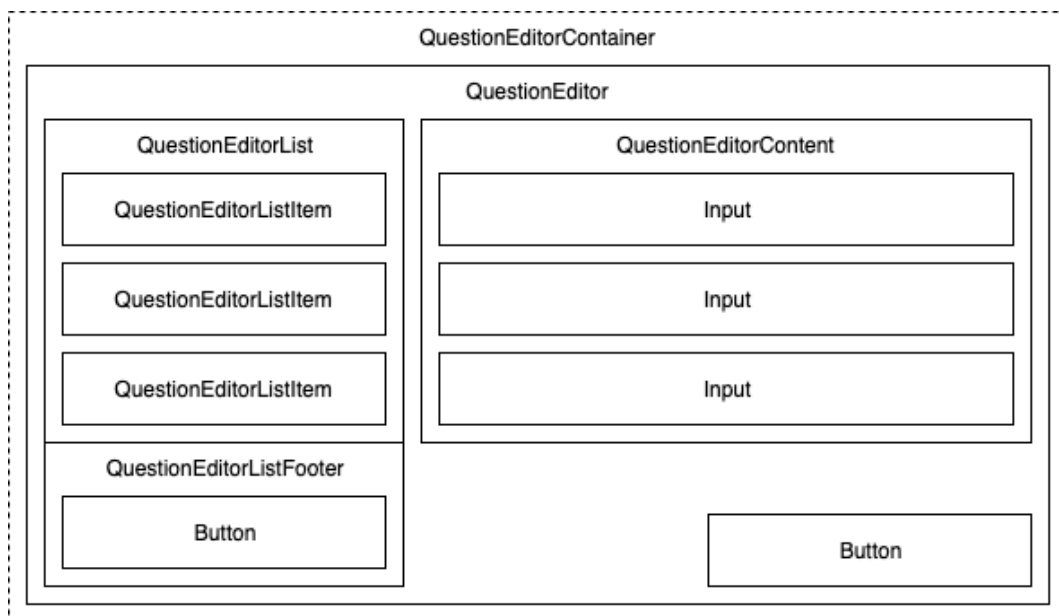


Abbildung 4.3: Mögliche Komponenten-Hierarchie für den Fragen-Editor.

⁷Bildquelle: <https://hackernoon.com/thinking-in-redux-when-all-youve-known-is-mvc-c78a74d35133> (aufgerufen am 06.05.2019).

4 Entwurf und Implementierung eines modernen CRS für die Programmierlehre

Die tatsächliche Implementierung entspricht nahezu vollständig diesem Bild (einzig die Komponente `QuestionEditorListItem` wurde aus Gründen der Drag-and-Drop-Funktionalität in der Seitenleiste nicht so implementiert).

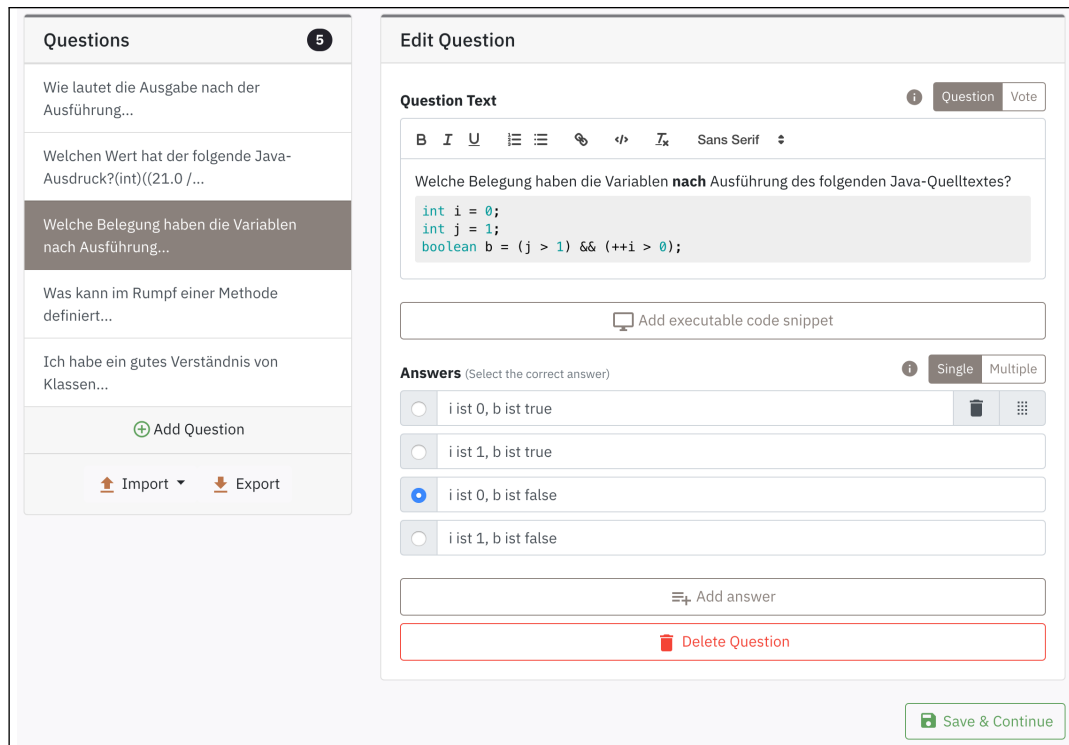


Abbildung 4.4: Fertige Implementierung des Fragen-Editors im React-Framework.

Alle Daten, mit denen der Fragen-Editor arbeitet, werden in einem zugehörigen Redux-Store gehalten. Die möglichen Änderungsoperationen finden sich dementsprechend in einem zugehörigen Reducer wieder, dessen Schnittstelle wie folgt aussieht:

```
1 export const questionEditor = (state = [], action) => {
2   switch (action.type) {
3     case ADD_QUESTION: {...}
4     case EDIT_QUESTION_TEXT: {...}
5     case EDIT_QUESTION_CODE: {...}
6     case EDIT_QUESTION_MODE: {...}
7     case EDIT_QUESTION_TYPE: {...}
8     case DELETE_QUESTION: {...}
9     case DELETE_ANSWER: {...}
10    case ADD_ANSWER: {...}
11    case EDIT_ANSWER_TEXT: {...}
12    case SET_CORRECT_SINGLE_ANSWER: {...}
13    case SET_CORRECT_MULTI_ANSWER: {...}
14    case LOAD_QUESTIONS: {...}
15    case SORT_QUESTION: {...}
16    case SORT_ANSWER: {...}
17    default: {
18      return state;
19    }
20  }
21 };
```

Quelltext 4.10: Auszug aus dem Reducer für den Fragen-Editor (aus: `src/server/reducers/questions.js`)

Ein Beispiel für die Datenstruktur, die den Zustand des Fragen-Editors darstellt und im Store abgelegt wird:

```
1 {
2   questionEditor: [
3     {
4       id: 'c6U9v1',
5       type: 'question',
6       mode: 'single',
7       text: '<p>First question: Will this program really work?</p>',
8       questionIdx: 0,
9       code: 'public class ApplicationTest [...]',
10      answers: [
11        {
12          id: 'EkzauV',
13          text: 'Yes',
14          isCorrect: true
15        },
16        {
17          id: 'CKEzce',
18          text: 'No',
19          isCorrect: false
20        }
21      ]
22    }
23  ]
24 }
```

Quelltext 4.11: Zustand des Fragen-Editors innerhalb des Redux-Store mit einer einzigen Frage.

4.2 Peer-to-Peer-WebRTC-Verbindungen mit PeerJS

Um Unabhängigkeit von einem dedizierten, zentralen Anwendungsserver zu erlangen, der einerseits Wartungsaufwand bedeutet und außerdem einen „Single Point of Failure“ darstellt, werden Verbindungen direkt zwischen den Teilnehmern aufgebaut. Jeder Weclare-Nutzer kann zum Start einer Sitzung, also zur Laufzeit, entscheiden, welche Rolle er in der aktuellen Sitzung einnehmen will (Server oder Client). Der Rechner des Dozenten agiert typischerweise als Server, die Rechner der Studenten sind Clients. Somit ergibt sich eine klassische, zentralisierte Architektur des verteilten Systems in Form einer Stern-Topologie. Unabhängig von der Wahl wird jedoch stets der gleiche Code vom Server geladen. Das Programm ist in dieser Hinsicht omnipotent.

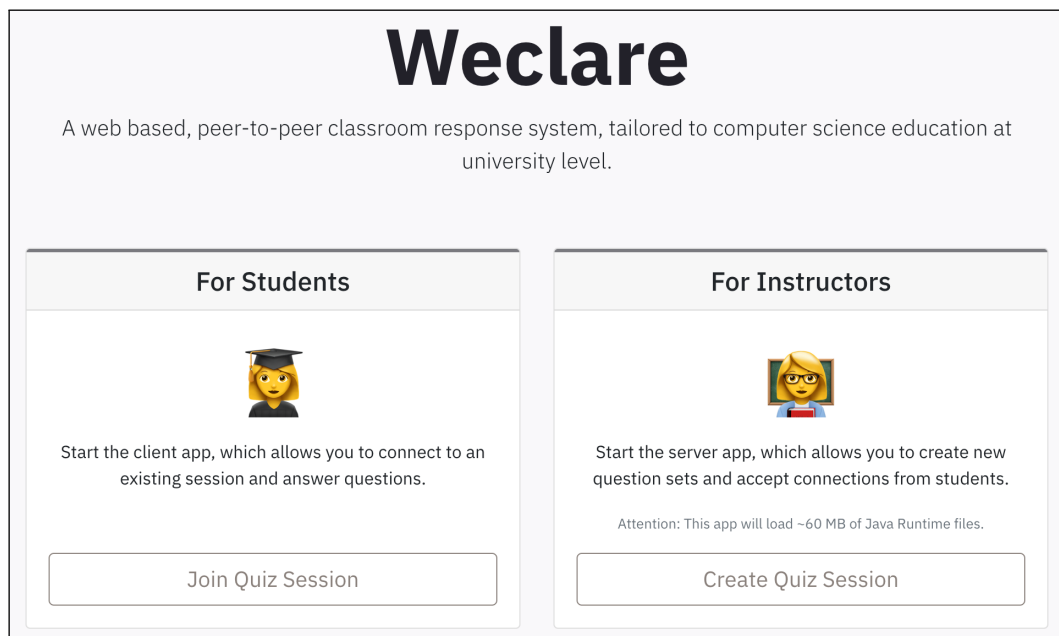


Abbildung 4.5: Am Start einer Sitzung kann der Nutzer entscheiden, ob er als Client oder als Server teilnehmen möchte. Unabhängig von der Wahl wird das gleiche Programm geladen.

Mit diesem Verhalten erfüllt die Software die Definition einer Peer-to-Peer Architektur, zum Beispiel gemäß Andrew Tanenbaum [30, S. 62]:

„Aus einem übergeordneten Blickwinkel heraus sind die Prozesse, die ein Peer-to-Peer-System bilden, alle gleich. Die Funktionen, die ausgeführt werden

müssen, werden also von jedem Prozess des verteilten Systems dargestellt. Folglich ist die meiste Interaktion zwischen Prozessen symmetrisch: Jeder Prozess agiert gleichzeitig als Client und als Server [...].“

Jedoch trifft dies nur auf den Zeitraum vor dem Start einer Sitzung zu. Sobald eine Sitzung begonnen hat, handelt es sich um eine klassische Client-Server-Struktur. Die Bezeichnung als Peer-to-Peer-Architektur entspricht daher nicht der gängigen Vorstellung eines vollvermaschten Peer-to-Peer-Netzwerks, bei dem jeder Teilnehmer mit jedem anderen Teilnehmer verbunden ist.

Die Tatsache, dass ein Teilnehmer entscheiden kann, ob er als Server oder Client teilnehmen will, bedeutet auch, dass Aspekte wie Network Address Translation (NAT) und Firewalls beim Austausch der IP-Adressen (Signaling) berücksichtigt werden müssen.

Der Zugriff auf die Schnittstellen des Betriebssystems (wie etwa das Netzwerk) durch Browser-Skripte ist aus Sicherheitsgründen stark eingeschränkt. Viele bestehende webbasierte CRS (z.B. Pingo, vgl. [8]) verwenden das WebSocket-Protokoll zur Kommunikation zwischen Client und Server. Der WebSocket-Standard beschreibt jedoch nur Möglichkeiten zum Verbinden mit einem bestehenden Server aus dem Browser heraus, nicht zum Kreieren einer Verbindung (vgl. [5]). Eine Peer-to-Peer-Architektur mit WebSockets ist daher nicht realisierbar.

Die einzige Möglichkeit, eine omnidirektionale Verbindung zwischen zwei Browsern zu realisieren, ist der relativ neue und offene WebRTC-Standard (Web Real-Time Communication)⁸. WebRTC wird hauptsächlich für Multimedia-Echtzeit-Anwendungen eingesetzt und seit 2017 von allen großen Browsern (Google Chrome, Mozilla Firefox, Opera, Apple Safari und Microsoft Edge) unterstützt. Viele Video- und Audiotelefonie-Lösungen (z.B. Skype⁹ oder Discord¹⁰) basieren inzwischen auf dem WebRTC-Protokoll. Neben Audio- und Videoinhalten können aber auch beliebige andere Daten über sogenannte RTCDataChannel übertragen werden. WebRTC beinhaltet keine Anweisungen für den Austausch der IP-Adressen zwischen beteiligten Parteien. Dieser Teil, das sogenannte Signaling, ist nicht Teil des Standards und muss selbst implementiert werden (vgl. [7]).

⁸Offizielle Webseite: <https://webrtc.org/>

⁹Offizielle Webseite: <https://web.skype.com/>

¹⁰Offizielle Webseite: <https://discordapp.com/>

Der Aufwand dafür ist groß, weil viele verschiedene Netzwerk-Situationen berücksichtigt werden müssen. Somit wird beim Weclare-Prototypen eine Open-Source-Bibliothek namens PeerJS¹¹ verwendet, die WebRTC in eine simple API kapselt und ein Signaling-Verfahren beisteuert.

Die (in Kapitel 2.2) formulierte Anforderung, keinen dedizierten Server zu benötigen, kann aufgrund des notwendigen Signalings also nicht vollständig erfüllt werden und muss präzisiert werden: Ein Server wird lediglich zum Austausch der IP-Adressen der Teilnehmer benötigt. Nach dem Austausch ist kein Server mehr notwendig. Anwendungsdaten werden nie über einen zentralen Server, sondern immer nur zwischen den einzelnen Teilnehmern verschickt. Der notwendige Signaling-Server muss nicht vom Weclare-Anwender betrieben werden – ein öffentlicher Server kann verwendet werden. In der Standard-Einstellung verwendet PeerJS einen kostenlosen und öffentlichen Signaling-Server, der von den PeerJS-Autoren betrieben wird.

Die PeerJS-Library ermöglicht den Aufbau einer Datenverbindung zwischen zwei Browsern mit sehr simplen Aufrufen: Zunächst müssen Client und Server ein Peer-Objekt erzeugen. Als Parameter kann an dieser Stelle eine (auf dem Signaling-Server noch nicht verwendete) alphanumerische ID übergeben werden, unter welcher der zugehörige Peer beim Signaling-Server bekannt gemacht wird. Optional kann hier außerdem ein eigener Signaling-Server angegeben werden.

Anschließend können an dem neuen Peer-Objekt diverse Callback-Methoden registriert werden, die den weiteren Gebrauch regeln. So kann zum Beispiel der Server seine neu erstellte ID kundtun und auf eingehende Verbindungen und Daten reagieren:

¹¹Offizielle Webseite: <https://peerjs.com/>

```
1 import Peer from "peerjs";
2
3 const peer = new Peer("server-id");
4
5 peer.on("open", id => {
6   console.log(`Successfully created peer: ${id}`);
7 });
8
9 peer.on("connection", conn => {
10  console.log(`New client connected: ${conn.peer}`);
11  conn.on("data", data => {
12    switch (data.type) {
13      case "answer":
14        // Do something
15        break;
16      default:
17        // Noop
18    }
19  });
20 });
```

Quelltext 4.12: Verbindungsaufbau mit der PeerJS-Bibliothek auf der Server-Seite. (aus: `src/server/actions/server.js`)

Auf der Gegenseite, beim Client kann die Verbindung einfach über die `connect()`-Methode aufgebaut werden, die als Parameter die ID des gewünschten Peers erhält. Anschließend kann über das zurückgelieferte `Connection`-Objekt eine Nachricht verschickt werden. (vgl. [26])

```
1 import Peer from "peerjs";
2
3 const peer = new Peer();
4 const connection = peer.connect("server-id");
5 connection.send("Hello world!");
```

Quelltext 4.13: Verbindungsaufbau mit der PeerJS-Bibliothek auf der Client-Seite. (aus: `src/client/actions/client.js`)

Um dem Flux-Muster (siehe Kapitel 4.1.2) treu zu bleiben, wird die gesamte Netzwerk-Kommunikation innerhalb von `ActionCreator`-Funktionen implementiert. Da Netzwerk-Aufrufe keine puren Funktionen sind und asynchron erfolgen müssen, können sie nicht in

einen Reducer integriert werden. Um solche asynchronen Actions zu realisieren, wird Redux um eine sehr simple Middleware namens `redux-thunk`¹² erweitert. Diese Middleware erlaubt es, asynchrone Aufrufe in ActionCreator-Methoden unterzubringen (vgl. [21]). Da viele dieser Netzwerkaufrufe keine Änderungen im Store bewirken, wird der ActionCreator seinem Namen nicht mehr treu, denn am Ende wird, entgegen seinem Zweck, keine Action mehr zurückgegeben.

```
1 export function sendAnswers(answerIdxArray) {
2   return (dispatch, getState) => {
3     const {
4       client: { connection = null, currentQuestion = null }
5     } = getState();
6
7     if (
8       connection &&
9       currentQuestion &&
10      typeof answerIdxArray !== "undefined"
11    ) {
12      const msg = {
13        type: "answer",
14        payload: {
15          questionIdx: currentQuestion.questionIdx,
16          answerIdxArray,
17          userId: connection.provider.id
18        }
19      };
20      connection.send(msg);
21    }
22  };
23 }
```

Quelltext 4.14: ActionCreator zum Versenden von Antworten vom Client zum Server. (aus: `src/client/actions/client.js`)

¹²Offizielle Webseite: <https://github.com/reduxjs/redux-thunk>

4.3 Text- und Code-Formatierung mit Quill und CodeMirror

Quelltexte können in zwei verschiedenen Kontexten in Fragestellungen auftauchen: Zum Einen gibt es unvollständige, kurze Quelltext-Fragmente, die in einen Fließtext eingebunden werden sollen, wie in diesem Beispiel:

```
„Wie viele String-Parameter hat die folgende Java-Methode?  
public String m(int i, int s, boolean b) { ... }“
```

Dabei handelt es sich um ein syntaktisch unvollständiges und nicht ausführbares Java-Fragment. Um die Lesbarkeit dieses Fragments zu erhöhen, sollte es dennoch optisch deutlich vom Fließtext unterscheidbar sein. Es bietet sich an, den Code in einer Monospace-Schriftart zu formatieren und (wenn möglich) eine syntaktische Einfärbung (Syntax Highlighting) anzuwenden.

Da die Entwicklung einer eigenen Editor-Komponente komplex ist, und es in diesem Bereich bereits eine große Auswahl an Bibliotheken gibt, wird auf eine bestehende Implementierung zurückgegriffen. Dabei müssen folgende Anforderungen von einer solchen Bibliothek erfüllt werden:

- **Integration in React:** Muss sich leicht in das deklarative React-Framework einbinden lassen.
- **WYSIWYG (What You See Is What You Get):** Der Editor muss stets eine Vorschau aller Formatierungen darstellen, und es soll nicht zwischen einem Markup- und einem Vorschau-Modus gewechselt werden müssen. Formatierungen sollen über Buttons eingefügt werden können.
- **Text-Formatierungen:** Der Editor muss über simple Text-Formatierungen verfügen (zum Beispiel Fettschrift und Kursivierung).
- **Monospace-Schriftart:** Eine Möglichkeit zum Verwenden einer Monospace-Schriftart muss vorhanden sein.
- **Syntax-Highlighting:** Syntaktische, farbliche Hervorhebungen von Code-Fragmenten müssen unterstützt werden (üblicherweise durch ein Plugin/Erweiterung mit einer zusätzlichen Bibliothek wie Highlight.js¹³).

¹³Offizielle Webseite: <https://highlightjs.org/>

Nach dem Erwägen und Ausprobieren diverser Bibliotheken (z.B. Draft.js¹⁴, Slate¹⁵, Prose-mirror¹⁶) fiel die Wahl auf den Quill-Editor¹⁷, der alle genannten Anforderungen erfüllt. Quill ist nicht primär für den Einsatz mit React vorgesehen. Daher wird der Editor in Form der Wrapper-Bibliothek react-quill-Bibliothek¹⁸ verwendet, die Quill in React-Komponenten bündelt.



Abbildung 4.6: Bearbeitung einer Fragestellung im StuReSy-Fragen-Editor.

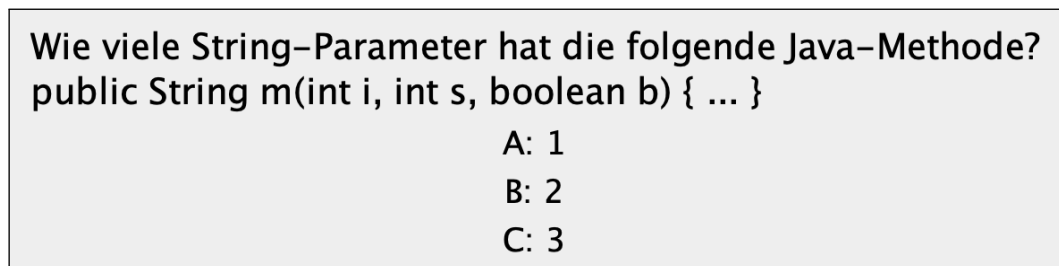


Abbildung 4.7: Darstellung eines Code-Fragments in StuReSy (ohne manuell vorgenommene Formatierungen).

¹⁴Offizielle Webseite: <https://draftjs.org/>

¹⁵Offizielle Webseite: <https://www.slatejs.org/>

¹⁶Offizielle Webseite: <https://prosemirror.net/>

¹⁷Offizielle Webseite: <https://quilljs.com/>

¹⁸Offizielle Webseite: <https://github.com/zenoamaro/react-quill>

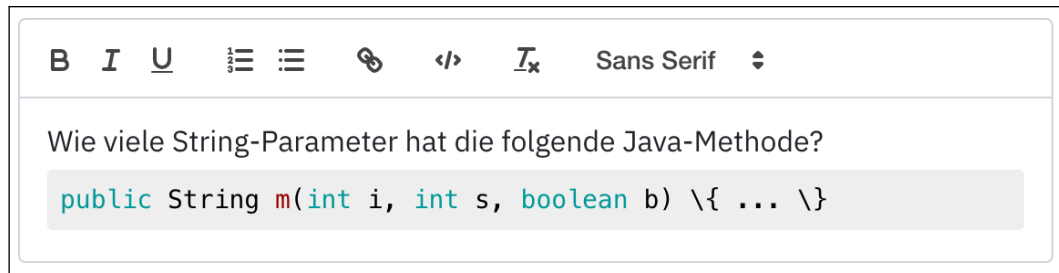


Abbildung 4.8: Bearbeitung einer Fragestellung im Quill-Editor von Weclare.

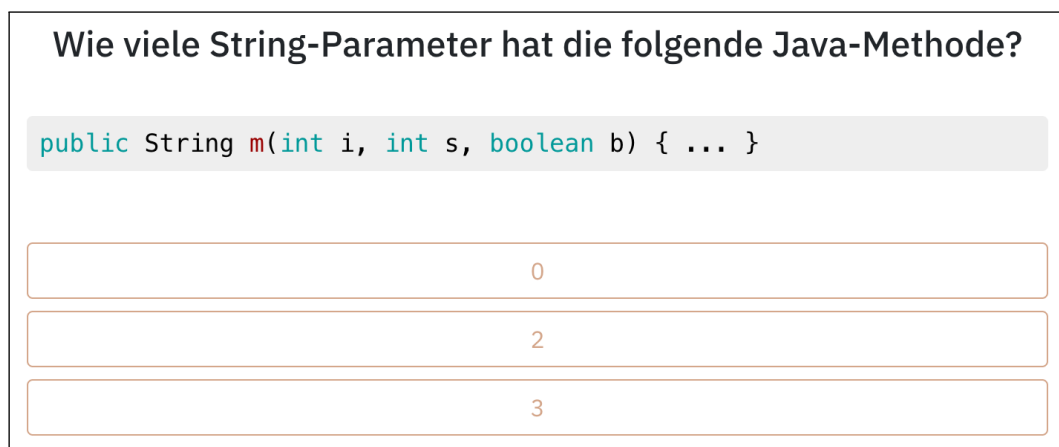
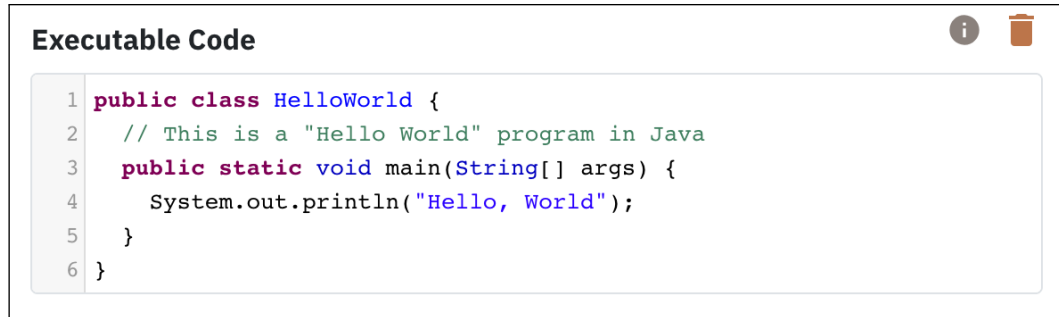


Abbildung 4.9: Darstellung eines Code-Fragments in Weclare (Quelltext als Code-Block ausgezeichnet).

Der zweite Kontext von Quelltexten in Fragestellungen ist das Einfügen von vollständigem, ausführbarem Code, in Form ganzer Java-Klassen. Derartige Code kann in Weclare direkt im Browser ausgeführt werden. Solch ein interaktives Code-Beispiel kann allerdings nur einmal pro Frage vorhanden sein und ist auch nicht Teil des Fließtextes. Eine entsprechende Editor-Komponente zum Einfügen des Codes sollte also auch funktional darauf zugeschnitten sein: Text-Formatierungen sind nicht mehr notwendig, stattdessen rücken Funktionen wie die korrekte Einrückung von Code-Zeilen und Zeilen-Nummerierungen in den Vordergrund.

Der Quill-Editor kann nicht in einem „Code only“-Modus betrieben werden, so dass eine zweite, unabhängige Editor-Komponente für interaktive Code-Abschnitte integriert wird.

Mit CodeMirror¹⁹ existiert eine beliebte Bibliothek für diesen Zweck, die allerdings nicht explizit für den Einsatz im React-Framework bestimmt ist, so dass auch sie in Form einer Wrapper-Bibliothek namens react-codemirror²⁰ zum Einsatz kommt.

The image shows a code editor window titled "Executable Code". It contains a Java program with the following code:

```
1 public class HelloWorld {
2     // This is a "Hello World" program in Java
3     public static void main(String[] args) {
4         System.out.println("Hello, World");
5     }
6 }
```

Abbildung 4.10: Darstellung eines ausführbaren Java-Quelltexts in Weclare mittels der CodeMirror-Bibliothek.

¹⁹Offizielle Webseite: <https://codemirror.net/>

²⁰Offizielle Webseite: <https://github.com/scniro/react-codemirror2>

4.4 Code-Ausführung im Browser via DoppioJVM

Die grundlegende Programmierausbildung im Department Informatik an der HAW Hamburg erfolgt in der Programmiersprache Java, weswegen auch die Funktionen von Weclare auf Java zugeschnitten sind. Normalerweise wird Java in einem zweistufigen Prozess ausgeführt. Zunächst wird eine plattformabhängige JVM geladen. Innerhalb dieser JVM wird der Java-Compiler (javac) aufgerufen. Dieser compiliert Java-Quelltext zu Java-Bytecode. Der Java-Bytecode kann dann von der JVM ausgeführt werden. Dieser Ablauf ist in der Abbildung 4.11 unter a) dargestellt.

Um Java in einem Browser auszuführen ergeben sich konzeptionell mindestens drei verschiedene Möglichkeiten, die in Abbildung 4.11 mit den Buchstaben b) - d) gekennzeichnet sind.

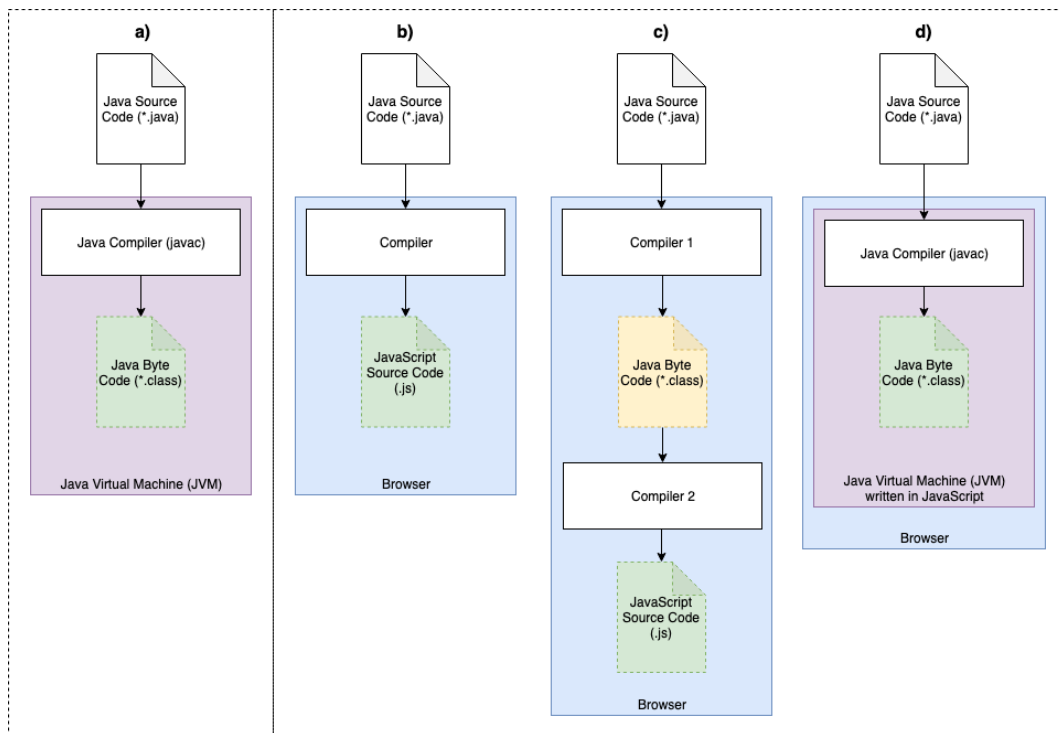


Abbildung 4.11: Ausführungsmöglichkeiten eines Java-Quelltexts im Browser.

- **b):** Java-Quelltext wird mit einem Compiler direkt in JavaScript umgewandelt: Diese Variante ist konzeptionell sehr einfach. Zwar gibt es einige Programme, welche Java-Quelltext in JavaScript übersetzen können (z.B. JSweet²¹ oder GWT²²), jedoch sind alle diese Programme entweder in Java selbst oder in dritten Programmiersprachen verfasst. Ein solcher Compiler, der selbst in JavaScript geschrieben wurde und damit auch im Browser ausgeführt werden kann, ist nicht bekannt. Es wäre möglich eines der besagten Programme auf einem separaten Compile-Server einzusetzen. Dies würde aber eine weitere Abhängigkeit von einem Server bedeuten und damit den formulierten Kern-Anforderungen widersprechen.
- **c):** Java-Quelltext wird mithilfe eines ersten Compilers in Java-Bytecode übersetzt. Anschließend wird der Java-Bytecode mit einem zweiten Compiler in JavaScript übersetzt. Auch hier ergibt sich das gleiche Problem wie in b). Zwar gibt es Programme, die den jeweiligen Teilschritt übernehmen könnten, jedoch ist keins davon in JavaScript verfasst, so dass es keine browserkompatible Lösung dafür gibt.
- **d):** Wie auf einem lokalen Rechner wird eine JVM zur Ausführung verwendet, die allerdings selbst in JavaScript implementiert wurde. Mithilfe dieser JVM können existierende Java-Programme wie zum Beispiel der javac-Compiler ausgeführt werden, um Java-Quelltext in Java-Bytecode zu übersetzen und das Ergebnis kann anschließend in der gleichen JVM ausgeführt werden.

In Anbetracht der Tatsache, dass die Ausführung der Code-Beispiele im Browser stets die gleichen Ergebnisse liefern muss, wie die Ausführung in einer lokalen JVM (zum Beispiel auch in Bezug auf Fehler beim Kompilieren), so ist die Variante d) zu bevorzugen und wird im Rahmen dieser Arbeit in das entstandene CRS integriert.

Die benötigte JVM-Implementierung in JavaScript existiert bereits in Form eines Universitätsprojekts der PLASMA-Forschungsgruppe der University of Massachusetts in Amherst, USA. Unter dem Namen DoppioJVM wurde dort im Jahr 2014 im Rahmen einer wissenschaftlichen Arbeit eine solche JVM erschaffen [32, S. 8]:

„DoppioJVM is a robust prototype Java Virtual Machine (JVM) interpreter that operates entirely in JavaScript. DoppioJVM implements all 201 bytecode instructions specified in the second edition of the Java Virtual Machine Specifi-

²¹Offizielle Webseite: <http://www.jsweet.org/>

²²Offizielle Webseite: <http://www.gwtproject.org/>

ation, supports multithreaded programs, runs multiple languages that run on top of the JVM, and implements many of the complex mechanisms and native functionality that JVM programs expect.“

Voraussetzung für die Nutzung von DoppioJVM ist ein weiteres Projekt der gleichen Forschungsgruppe namens BrowserFS²³. Die JVM benötigt für ihre Funktion Zugriff auf ein Dateisystem, welches der Browser als Laufzeitumgebung normalerweise nicht anbietet. Wer JavaScript außerhalb des Browser auf dem Server bzw. in der Kommandozeile einsetzt, kann aber dank der Node.js-Laufzeitumgebung auf Dateisysteme zugreifen. Genau diese populäre Node.js-API emuliert BrowserFS und bringt sie in den Browser. Dabei werden verschiedene Datenquellen unterstützt: Die Dateien können zum Beispiel im LocalStorage des Browser oder aber komplett im Arbeitsspeicher abgelegt werden (vgl. [25]).

Für den Betrieb der DoppioJVM werden zwei Verzeichnisse mit verschiedenen Datenquellen konfiguriert: Die Bestandteile der Java Runtime Environment (JRE) werden auf dem Webserver abgelegt und anschließend beim Zugriff über asynchrone HTTP-Requests nachgeladen (bekannt als „Asynchronous JavaScript and XML“ (AJAX)). Die Quelltext- und Bytecode-Dateien, die kompiliert und ausgeführt werden, liegen komplett im Arbeitsspeicher.

Das asynchrone Nachladen der JRE ist der größte Schwachpunkt der Code-Ausführung via DoppioJVM. Die Dateigröße der JRE liegt bei rund 62 Megabyte, die der Server-Rechner herunterladen muss, um Java-Code ausführen zu können. Bei langsamer Internetverbindung kann das einige Minuten dauern.

Nachdem das BrowserFS-Dateisystem konfiguriert ist, kann DoppioJVM mit einer relativ einfachen Schnittstelle verwendet werden:

²³Offizielle Webseite: <https://github.com/jvilk/BrowserFS>

```
1 new Doppio.VM.JVM(  
2   {  
3     doppioHomePath: "/sys",  
4     classpath: [".", "/sys/", "/tmp/"]  
5   },  
6   (err, jvmObject) => {  
7     jvmObject.runClass("Loader", [classname], exitCode => {  
8       if (exitCode !== 0) {  
9         console.log("JVM exited with an error");  
10      } else {  
11        console.log("JVM exited successfully");  
12      }  
13    });  
14  }  
15 );
```

Quelltext 4.15: Instanziierung der DoppioJVM (aus: src/server/actions/doppio.js).

Die DoppioJVM muss bei jeder Verwendung neu instanziiert werden. Somit ist es erforderlich, dass die Kompilierung und Ausführung des Java-Codes aus der Fragestellung in einem Aufruf erfolgt. Dazu bietet Java die passenden Klassen und Methoden um Quelltexte programmatisch zu kompilieren und auszuführen:


```
1 import javax.tools.*;
2 import java.lang.reflect.*;
3 import java.io.*;
4 import java.net.*;
5
6 public class Loader {
7     public static void main(String[] args) {
8         if (args.length == 0) {
9             System.out.println("No class was found.");
10            System.exit(1);
11        }
12        String className = args[0];
13        String sourceFile = "/tmp/" + className + ".java";
14        String classFile = "/tmp/" + className + ".class";
15
16        System.out.println("Compiling found class '" + className + "'.");
17
18        JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();
19        int result = compiler.run(null, null, null, sourceFile, "-d", "/tmp/"
20            );
21
22        if (result == 0) {
23            try {
24                System.out.println("Compilation successful. Executing...");
25
26                URLClassLoader classLoader = new URLClassLoader(
27                    new URL[] {new File(classFile).toURI().toURL()}, ClassLoader.
28                        getSystemClassLoader());
29                Class<?> c = classLoader.loadClass(className);
30                Method m = c.getDeclaredMethod("main", String[].class);
31                m.invoke(null, new Object[] {});
32                classLoader.close();
33
34                System.out.println("Execution successfull");
35            } catch (Exception e) {
36                e.printStackTrace();
37            }
38        } else {
39            System.out.println("Could not compile");
40        }
41    }
42 }
```

Quelltext 4.16: Java-Loader, der programmatisch Quelltext kompiliert und ausführt. (aus: public/doppio/Loader.java)

4 Entwurf und Implementierung eines modernen CRS für die Programmierlehre

Ebenso wie die Netzwerk-Kommunikation ist auch die Instanziierung der DoppioJVM komplett in Form von ActionCreator-Funktion in Weclare integriert, um dem Flux-Muster zu entsprechen. In einem Store wird dabei auch der Inhalt des kleinen Terminal-Fensters gespeichert, welches die Ein-und-Ausgabe-Streams der DoppioJVM abfängt.

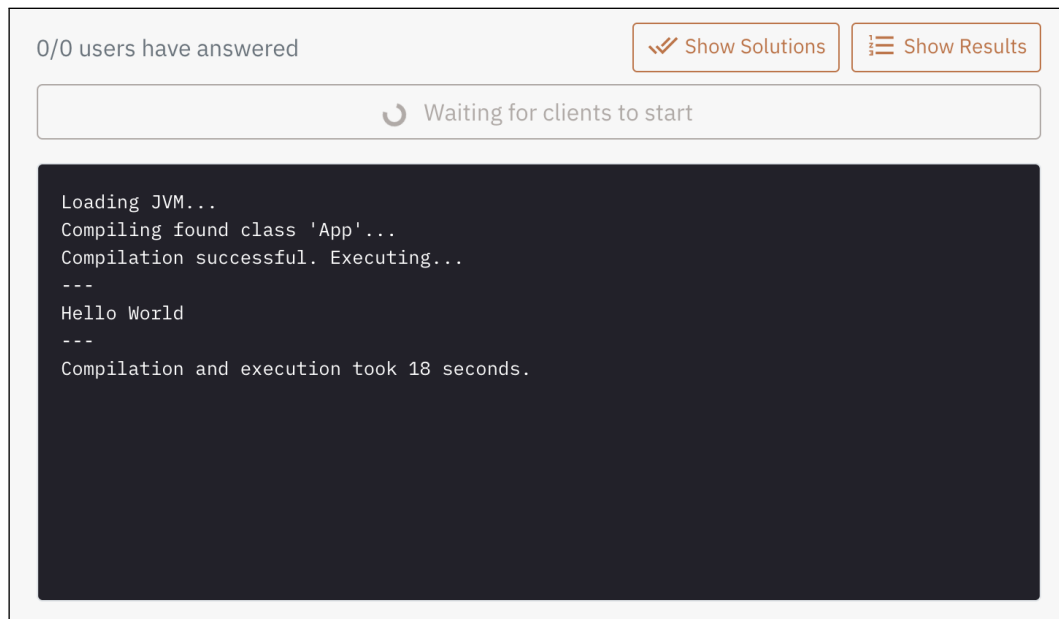


Abbildung 4.12: Ausführung einer Java-Klasse in Weclare mittels DoppioJVM.

5 Fazit und Ausblick

5.1 Diskussion der Ergebnisse

Für jede Anforderung wird einzeln geprüft, ob die Umsetzung gelungen ist und welche Erfahrungen bei der Implementierung gemacht wurden.

Web-Applikation

Die Umsetzung eines CRS als Web-Applikation bereitete keine Probleme. Das React-Framework stellte sich als gute Wahl für dieses Projekt heraus. Die bereitgestellten Konzepte und Bibliotheken waren einfach zu verstehen und ermöglichten eine sehr schnelle Entwicklung. Langfristige Wartbarkeit ist durch die Größe der React-Gemeinde ebenfalls gegeben.

Eine wichtige Erfahrung während der Umsetzung war der Umgang mit dem großen JavaScript-Ökosystem. Die Menge an verfügbaren und kostenlosen, OpenSource-JavaScript-Bibliotheken ist enorm groß, allerdings zeigte sich, dass es oft schwer ist, hochwertige Bibliotheken zu finden. Sowohl für die Code-Formatierung und die Code-Ausführung als auch für viele andere Funktionen wurden zahlreiche Bibliotheken ausprobiert und später wieder verworfen. Die Qualität und vor allem die Aktualität der Projekte stellte sich häufig erst zu einem späten Zeitpunkt als problematisch heraus. Die Anzahl und Aktivität der Entwickler sowie die Häufigkeit von Aktualisierungen waren oft bessere Kriterien für die Auswahl einer Bibliothek als ihr anfänglicher Funktionsumfang.

Peer-To-Peer-Verbindungen

Die Implementierung von Peer-to-Peer-Verbindungen auf Basis von WebRTC erwies sich als schwierig. Der WebRTC-Standard ist leider noch nicht fertiggestellt und Abweichungen zwischen den Implementierungen der verschiedenen Browser-Hersteller sind noch vorhanden¹. WebRTC produktiv mit breiter Browser-Kompatibilität einzusetzen, ist aufwendig.

Die Verwendung von WebRTC-Wrappern wie PeerJS, die die Schnittstelle vereinfachen und das Signaling regeln, erscheint gerade in kleinen Projekten unausweichlich zu sein, da eine eigene Implementierung sehr aufwändig ist. Bedauerlicherweise gibt es bisher nur wenige und nicht sehr aktuelle Open-Source-Bibliotheken für diese wichtige Aufgabe. Es bleibt zu hoffen, dass die Stabilität der Schnittstelle und damit auch die Qualität der Wrapper-Bibliotheken nach dem Ende der WebRTC-Standardisierung ein höheres Niveau erreicht, damit die Technologie zukünftig einfacher verwendet werden kann.

Das fertige System funktioniert zwar im Regelfall (sog. „Lucky Path“) einwandfrei, allerdings ist die Zuverlässigkeit von PeerJS problematisch. Verbindungsverlust der Teilnehmer während einer Sitzung oder die Verwendung zu alter oder zu neuer Browser-Versionen können zu Fehlerzuständen führen. Ob WebRTC eine gute Wahl für den Praxisbetrieb in Lehrveranstaltungen ist, muss daher mit entsprechenden Tests in der tatsächlichen Netzwerk- und Teilnehmerumgebung getestet werden.

Code-Formatierung

Es gibt eine große Anzahl von sehr aktiven JavaScript-Editor-Bibliotheken, allerdings bieten nur wenige davon auch eine überzeugende und aktuelle React-Integration an.

Beide verwendeten Editor-Bibliotheken, Quill und CodeMirror, wirken ausgereift und leistungsfähig. Im Praxiseinsatz wird dieser Eindruck aber teilweise durch qualitativ verbesserungswürdige und teilweise veraltete React-Wrapper-Bibliotheken zunichte gemacht.

Dennoch funktioniert die Code-Formatierung in Weclare ohne Auffälligkeiten. Sowohl kleine Code-Fragmente als auch ganze Java-Klassen können mit wenigen Klicks in eine Fragestellung eingefügt und übersichtlich formatiert werden.

¹Siehe: <https://github.com/webrtcchacks/adapter>

Code-Ausführung im Browser

Weclare zeigt, dass die Ausführung von Java-Quellcode im Browser möglich ist. Wie praktikabel diese Funktion im Alltag von Dozenten ist, bleibt fraglich. Vor allem das notwendige Laden der JRE sorgt dafür, dass die Ausführungsdauer von Code-Beispielen oft unangenehm hoch ist. Bei geladener Runtime können Code-Beispiele meistens innerhalb von 15 bis 30 Sekunden kompiliert und ausgeführt werden. Je nach Geschwindigkeit der Internetverbindung kann sich dieser Wert beim ersten Laden der Runtime aber auf mehrere Minuten verlängern. Die Funktion der Code-Ausführung ist aber optional und Weclare kann auch ohne sie eingesetzt werden.

Während der Implementierung von Weclare lag der Fokus auf der Umsetzung der erwähnten Kern-Anforderungen, so dass einige Funktionen, die in StuReSy vorhanden sind, nicht in Weclare umgesetzt werden konnten:

- Freitext-Fragen werden nicht unterstützt.
- Die Visualisierung der erhaltenen Antworten in Form von Diagrammen und der Export als Grafik oder CSV wird nicht unterstützt.
- Es können nur vollständige Fragesätze importiert werden. Das Importieren einzelner Fragen aus einem Fragesatz wird nicht unterstützt.

Außerdem gibt es Schwachstellen in der Implementierung von Weclare, die adressiert werden müssen, sollte ein Praxiseinsatz in Erwägung gezogen werden:

- Unit- und Integrationstest für die Anwendung sind notwendig.
- Die Instantiierung der DoppioJVM sollte in einem eigenen Thread erfolgen und deswegen in einen WebWorker ausgelagert werden.
- Die Robustheit und das Error-Handling der WebRTC-Verbindung muss erhöht und mit verschiedenen Browsern und Netzwerk-Konstellationen getestet werden.
- Grundsätzlich unterstützt das verwendete Bootstrap-CSS-Framework responsives Design für verschiedene Bildschirmgrößen, allerdings wurde während der Entwicklung fast nur mit Desktop-Geräten getestet. Eine Optimierung für mobile Geräte ist notwendig.

5.2 Ausblick

Der Weclare-Prototyp zeigt zwar, dass es möglich ist, dank einer JVM-Implementierung in JavaScript einen Java-Quelltext in der Browser-Umgebung zu kompilieren und anschließend auszuführen, ohne dafür einen Compiler-Server zu verwenden. Gleichzeitig zeigen sich bei dieser Vorgehensweise auch deutliche Probleme:

- **Größe der Java Runtime:** Die Java-Laufzeitumgebung ist etwa 62 Megabyte groß und muss erst vom Client heruntergeladen werden. Trotz schneller Internetverbindungen ist das eine beträchtliche Datenmenge, die auch die mobile Nutzung der Code-Ausführung einschränkt.
- **Performance:** Die DoppioJVM ist um ein Vielfaches langsamer als eine native JVM. Das hat mehrere Gründe, zum Beispiel die Qualität der Implementierung oder der geringere Grad der Optimierung (Ahead-Of-Time-kompilierte vs. interpretierte Sprache).

In der Recherche-Phase dieser Arbeit schien es so, als könne WebAssembly eine Lösung für die genannten Probleme darstellen.

WebAssembly ist ein Standard der bereits von allen großen Browser-Herstellern unterstützt wird (Microsoft, Mozilla, Apple, Google). Es handelt sich bei WebAssembly um ein sogenanntes Compile-Target. Also eher nicht um eine Programmiersprache, die manuell von Entwicklern geschrieben wird (obwohl das möglich ist), sondern um ein Ausgabeformat eines Compilers. Viele andere, beliebte Sprachen können nach WebAssembly kompiliert werden. Gut unterstützt wird das momentan zum Beispiel von den Sprachen C, C++ und Rust. WebAssembly erfüllt damit die gleiche Aufgabe wie Java-Bytecode: Es ist ein plattformunabhängiges und kompaktes Binärformat, das in einer virtuellen Maschine ausgeführt werden kann.

Damit ermöglicht WebAssembly zukünftig ganz neue Möglichkeiten: Software-Entwickler können Web-Anwendungen in beliebigen Sprachen verfassen und sind nicht mehr an JavaScript gebunden. Außerdem können bestehende Anwendungen relativ einfach in die Browser-Umgebung portiert werden und die Leistungsfähigkeit von Web-Anwendungen gesteigert werden (WASM kann als Ahead-of-Time-kompilierte Sprache tendenziell schneller ausgeführt werden als eine interpretierte und JIT-kompilierte Sprache wie JavaScript).

Es gibt bereits Compiler, die Java-Bytecode nach WebAssembly konvertieren können (z.B. TeaVM²). Allerdings ist der TeaVM-Compiler selbst in Java geschrieben und dafür gedacht, vorab und einmalig ausgeführt zu werden, um eine Java-Anwendung im Browser ausführen zu können. Der Compiler selbst kann also noch nicht im Browser ausgeführt werden.

Theoretisch ließe sich dieser Compiler selbst nach WebAssembly kompilieren und könnte dann im Browser funktionieren. Praktisch ist das allerdings aufgrund vieler kleiner Einschränkungen des (noch relativ jungen) WebAssembly-Standards noch nicht so einfach möglich und der Einsatz von WebAssembly für diese Arbeit konnte nicht realisiert werden.

Mit steigender Popularität und steigendem Funktionsumfang von WebAssembly ist es sehr wahrscheinlich, dass es bald einen Java-zu-WebAssembly-Compiler geben wird, der selbst in WebAssembly implementiert wurde oder problemlos nach WebAssembly übersetzt werden kann. Damit ließe sich dann im Browser Java-Quellcode in WebAssembly konvertieren und ausführen ohne das lästige Laden der Java Laufzeitumgebung und mit tendenziell besserer Performance als DoppioJVM (vgl. [12]).

Mit WebAssembly wäre es außerdem leichter möglich, das Feature der Code-Ausführung im Browser um weitere Sprachen zu erweitern und damit den potenziellen Nutzerkreis für Weclare zu erweitern.

²Offizielle Webseite: <http://teavm.org/>

Literatur

- [1] *Blogpost: PINGOs Zukunft ist gesichert – Verstetigung von PINGO*, <https://blogs.uni-paderborn.de/pingo/pingos-zukunft-ist-gesichert-uebergabe-von-pingo/>, zuletzt besucht am 07.05.2019.
- [2] *Blogpost: Umzug von PINGO zu coactum*, <https://blogs.uni-paderborn.de/pingo/umzug-von-pingo-zu-coactum/>, zuletzt besucht am 07.05.2019.
- [3] C. H. Crouch und E. Mazur, “Peer Instruction: Ten years of experience and results”, *American journal of physics*, Jg. 69, Nr. 9, S. 970–977, Apr. 2001. Adresse: http://wayback.archive-it.org/all/20121206172800/http://www4.uwm.edu/ltc/srs/faculty/docs/Mazur_Harvard_SRS2.pdf.
- [4] *E-Assessment-Wiki des ELAN e.V.* https://ep.elan-ev.de/wiki/Audience_Response, zuletzt besucht am 07.05.2019.
- [5] I. Fette und A. Melnikov, “The WebSocket Protocol”, RFC Editor, RFC 6455, Dez. 2011. Adresse: <https://www.rfc-editor.org/rfc/rfc6455.txt>.
- [6] C. Fies und J. Marshall, “Classroom Response Systems: A Review of the Literature”, *Journal of Science Education and Technology*, Jg. 15, Nr. 1, S. 101–109, März 2006. Adresse: <https://link.springer.com/article/10.1007/s10956-006-0360-1>.
- [7] *Getting Started with WebRTC*, <https://www.html5rocks.com/en/tutorials/webrtc/basics/#toc-signaling>, zuletzt besucht am 07.05.2019.
- [8] *GitHub-Repository: Pingo*, <https://github.com/PingoUPB/PINGOWebApp/>, zuletzt besucht am 07.05.2019.
- [9] *GitHub-Repository: StuReSy Client*, <https://github.com/sturesy/client>, zuletzt besucht am 07.05.2019.
- [10] *GitHub-Repository: StuReSy Server*, <https://github.com/sturesy/server>, zuletzt besucht am 07.05.2019.

- [11] *GPL-3.0-Lizenz für Weclare*, <https://github.com/pReya/weclare/blob/master/LICENSE>, zuletzt besucht am 24.04.2019.
- [12] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai und J. Bastien, "Bringing the Web Up to Speed with WebAssembly", *SIGPLAN Not.*, Jg. 52, Nr. 6, S. 185–200, Juni 2017. Adresse: <http://doi.acm.org/10.1145/3140587.3062363>.
- [13] R. Kaleta und T. Joosten, *Clickers in the Classroom: Analyses from the University of Wisconsin System Project*, <https://www.educause.edu/ir/library/pdf/EDU06283.pdf>, Okt. 2006.
- [14] D. Kundisch, J. Magenheim, M. Beutner, P. Herrmann, W. Reinhardt und A. Zokye, "Classroom response systems", *Informatik-Spektrum*, Jg. 36, Nr. 4, S. 389–393, 2013. Adresse: <https://link.springer.com/article/10.1007/s00287-013-0713-0>.
- [15] A. Mesbah und A. van Deursen, "Migrating multi-page web applications to single-page AJAX interfaces", *CoRR*, Jg. abs/cs/0610094, 2006. Adresse: <http://arxiv.org/abs/cs/0610094>.
- [16] C. Nupur, "World Wide Web and Its Journey from Web 1.0 to Web 4.0", *International Journal of Computer Science and Information Technologies*, Jg. 5, Nr. 6, S. 8096–8100, 2014. Adresse: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.666.6445&rep=rep1&type=pdf>.
- [17] *Offizielle React-Dokumentation: Lifting State up*, <https://reactjs.org/docs/lifting-state-up.html>, zuletzt besucht am 29.04.2019.
- [18] *Offizielle React-Dokumentation: Reconciliation*, <https://reactjs.org/docs/reconciliation.html>, zuletzt besucht am 29.04.2019.
- [19] *Offizielle React-Webseite*, <https://reactjs.org/>, zuletzt besucht am 09.04.2019.
- [20] *Offizielle Redux-Dokumentation: Actions*, <https://redux.js.org/basics/actions>, zuletzt besucht am 09.04.2019.
- [21] *Offizielle Redux-Dokumentation: Async Actions*, <https://redux.js.org/advanced/async-actions>, zuletzt besucht am 09.04.2019.
- [22] *Offizielle Redux-Dokumentation: Motivation*, <https://redux.js.org/introduction/motivation>, zuletzt besucht am 09.04.2019.

- [23] *Offizielle Redux-Dokumentation: Reducers*, <https://redux.js.org/basics/reducers>, zuletzt besucht am 09.04.2019.
- [24] *Offizielle Redux-Dokumentation: Usage with React*, <https://redux.js.org/basics/usage-with-react>, zuletzt besucht am 09.04.2019.
- [25] *Offizielle Schnittstellen-Dokumentation von BrowserFS*, <https://jvilk.com/browserfs/1.4.1/index.html>, zuletzt besucht am 29.04.2019.
- [26] *Offizielle Schnittstellen-Dokumentation von PeerJS*, <https://peerjs.com/docs.html>, zuletzt besucht am 29.04.2019.
- [27] W. D. Posdorfer, "Prototyp einer Freien Software für interaktive Vorlesungen mit Echtzeitabstimmungen", Bachelorarbeit, Universität Hamburg, Dez. 2012.
- [28] K. Siau, H. Sheng und F. F.-H. Nah, "Use of a classroom response system to enhance classroom interactivity", *IEEE Transactions on Education*, Jg. 4, S. 398–403, Aug. 2006. Adresse: <https://ieeexplore.ieee.org/abstract/document/1668284>.
- [29] *Stack Overflow Entwickler-Umfrage 2019: Web-Frameworks*, <https://insights.stackoverflow.com/survey/2019#technology--most-loved-dreaded-and-wanted-web-frameworks>, zuletzt besucht am 29.04.2019.
- [30] A. S. Tanenbaum und M. van Steen, *Verteilte Systeme: Prinzipien und Paradgimen*. Pearson Studium, 2008, ISBN: 978-3-8273-7293-2.
- [31] *Team-Seite von Pingo*, <http://trypingo.com/en/team/>, zuletzt besucht am 07.05.2019.
- [32] J. Vilk und E. D. Berger, *Doppio: Breaking the Browser Language Barrier*, <https://plasma-umass.org/doppio-demo/paper.pdf>, 2014.

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

Hamburg, 9. Mai 2019

Moritz Stückler