Fachbereich
Elektrotechnik und Informatik

FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

Masterarbeit

Ein P2P-basierter Echtzeit-Online-Editor für Code-Reviews und Pair-Programming

vorgelegt von

Melanie Lucht

Erstprüfer: Prof. Dr. rer. nat. Dipl.-Inform. Nane Kratzke
Zweitprüferin: Prof. Dr. Ing. Milena Zachow

Wintersemester 2018/19

Inhaltsverzeichnis

Kapitel 1

Einleitung

In der heutigen Arbeitswelt werden eine Reihe unterstützender Collaboration-Tools eingesetzt, die eine effiziente Zusammenarbeit unabhängig von Zeit und Ort ermöglichen. Die Unterstützung umfasst die Bereitstellung von Dokumenten, Hilfe bei der Organisation und Abwicklung von Projekten und ermöglichen die Kommunikation zwischen Teammitgliedern durch Forensysteme, Textchats, Audio- und Videosysteme. Auch in der Softwareentwicklung wird eine Vielzahl von Online-Tools zur Organisation von Projekten, Kommunikation oder zur Quellcodeverwaltung eingesetzt [Sta]. Häufig sind die Tools Browser-basiert. Sie müssen nicht installiert werden, haben niedrige Einstiegshürden und sind von überall erreichbar. Auf der anderen Seite speichern sie Nutzerdaten häufig auf zentralen Servern, so dass der Nutzer keine Kontrolle über die Nutzung der möglicherweise sensiblen Daten hat. Neue Webtechnologien machen es möglich Daten direkt zwischen Browsern auszutauschen, so dass sich die Möglichkeit eröffnet, die Abhängigkeit von Servern zu verringern. Dadurch ergeben sich interessante Anwendungsfelder für kollaborative Online-Tools, wie die Entwicklung eines Online-Code-Editors, der es ermöglicht gemeinsam Quellcode zu bearbeiten, ohne dass Dritte Zugriff auf die Daten haben.

1.1 Hintergrund

Traditionell bestehen seit den frühen 90er Jahren Webdokumente aus verlinkten statischen HTML-Seiten, die den strukturellen Aufbau einer Internetseite beschreiben. Der Client stellt zunächst eine Anfrage an den Server, der eine vollständige Seite zurückliefert. Es ist nicht möglich auf Ereignisse im Browser interaktiv zu reagieren. Erst mit der Einführung der asynchronen Kommunikation zwischen Client und Server ändert sich das. Der Begriff AJAX[1] beschreibt eine Sammlung von Technologien, die es ermöglichen eine Anfrage an den Webserver zu senden, ohne dass die HTML Seite in Gänze neu geladen werden muss. So können einzelne Teile der Webseite ausgetauscht werden und z.B. auf Nutzereingaben

[1] AJAX - Asynchronous JavaScript and XML

reagiert werden [Wika]. Mit der Entwicklung von HTML5, dessen erster Arbeitsentwurf 2008 veröffentlicht wurde[2], wurden Plugins wie z.B. Flash[3] für die Entwicklung von Anwendungen und das Streamen von Multimediainhalten überflüssig [Wikc; Evo].

Ein weiterer Meilenstein in der Kommunikation zwischen Client und Server wird mit der Einführung des WebSocket Protokolls [FM11] gesetzt, das eine bidirektionale Verbindung ermöglicht. Nach dem Aufbau einer Verbindung durch den Client sind asynchrone Übertragungen in beide Richtungen möglich. Der Server kann die Verbindung aktiv verwenden, um Informationen an den Client zu senden. Bis hierhin setzen alle Technologien eine Client/Server-Architektur voraus.

Mit der Entwicklung von WebRTC (Web Real-Time Communication) [Ber+17] wird dieses Paradigma durchbrochen. Die Technologie ermöglicht den Aufbau einer Echtzeitverbindung zwischen zwei Browsern. Die direkte Peer-to-Peer Verbindung zwischen den Endgeräten erlaubt es auf umfangreiche Servertechnologien zu verzichten und Applikationen zu erstellen, die weniger Entwicklungsaufwand und Bandbreite auf Seiten des Backends benötigen [Lev13]. WebRTC ist ein offener Standard, der von Google entwickelt und seit 2011 vom W3C[4] standardisiert wird. In den folgenden Jahren sind eine Reihe experimenteller Projekte gestartet worden, um Anwendungsfälle zu prüfen und den Umfang der APIs zu testen. Die umfangreichsten Demos und Beispiele sind unter den offiziellen Seiten des Projekts WebRTC Samples [WebS] und unter den Seiten von Khan [Khaa] zu finden.

Häufig stehen vor allem Lösungen für Video- und Audiokonferenzen sowie Screen-Sharing im Fokus. Mit der Technologie lassen sich, unabhängig von großen Anbietern, eigene Telekommunikationslösungen entwickeln. Der Umbruch in der Telekommunikationsbranche zeigt sich besonders bei den Anbietern von Konferenzsystemen, da mit WebRTC Videochats erschwinglich umgesetzt werden können. Weitere Beispiele, die von der Technologie profitieren sind Systeme im Kundensupport, virtuelle medizinische Beratung und Umsetzungen im Bereich der Lehre, da sich Text- und Videochats leichter in Webseiten integrieren lassen [VE14].

WebRTC unterstützt die Echtzeit-Datenübertragung zwischen zwei oder mehreren Clients. Trotzdem wird häufig auf WebSockets zurückgegriffen, obwohl WebRTC zuverlässige bidirektionale Datenkanäle unterstützt und durch die direkte Verbindung der Clients geringere

[2] Die endgültige Fassung wurde 2014 vom W3C (World Wide Web Consortium) veröffentlicht.
[3] https://get.adobe.com/de/flashplayer/
[4] World Wide Web Consortium (https://www.w3.org/standards/)

Latenzzeiten aufweist [RM17; Tok14]. Aus Datenschutzgründen kann WebRTC eine Alternative darstellen, wenn eine Cloud-Lösung, bei der sensible Daten bei einem Drittanbieter gespeichert werden, nicht gewünscht ist. Beispiele für solche Anwendungen sind Textchats, Multiplayer-Spiele oder allgemein Anwendungen in denen Dateien und Daten geteilt werden sollen.

1.2 Motivation

Im November 2017 wurde mit Version 1.0 die erste als stabil und vollständig gekennzeichnete Version der WebRTC APIs veröffentlicht und die kommenden Versionen werden sich hauptsächlich mit der Verbesserung der Interoperabilität beschäftigen und wenig grundlegende Veränderung innerhalb der API bringen [Wor17], so dass der aktuelle Entwicklungsstand eine stabile Grundlage für die Entwicklung eines Prototypen bietet. Dabei wird es in dieser Arbeit nicht primär um die bereits umfangreich behandelten Audio- und Videoanwendungen gehen, sondern um die Umsetzung eines Kollaborations-Tools, das auf zuverlässige Übertragung von Daten angewiesen ist und damit eher ein klassischer Anwendungsfall für eine TCP/IP-basierte Lösung über Websockets wäre.

Als Fallbeispiel wird im Rahmen dieser Arbeit eine Softwarelösung entwickelt, die es ermöglicht spontan Online-Code-Reviews durchzuführen, um sich über Quellcode fachlich auszutauschen. Code-Reviews sind für die beteiligten Entwickler ein wichtiger Bestandteil der Qualitätssicherung [siehe Sma18], da sie früh im Entwicklungszyklus helfen syntaktische Fehler zu erkennen oder zu vermeiden. Außerdem ermöglichen sie eine Kontrolle darüber, ob vereinbarte Programmierkonventionen und Programmierstile eingehalten werden, um für alle Beteiligten verständlichen Quellcode zu entwickeln, der test- und änderbar ist. Da IT-Projekte häufig aus interdisziplinären Teams bestehen, die in unterschiedlichen Umgebungen und Infrastrukturen arbeiten, wird bei der Umsetzung Wert darauf gelegt, dass die Einstiegshürden und Kosten niedrig sind, die Anwendung nicht installiert werden muss und keine Lizenzkosten für die beteiligten Entwickler entstehen. Damit bietet sich die Entwicklung einer Webanwendung auf WebRTC-Basis als Prototyp an.

1.3 Ziel der Arbeit

Das Ziel dieser Arbeit sind Erkenntnisse darüber zu gewinnen, inwieweit es mit der aktuell verfügbaren WebRTC-Technologie möglich ist eine Browser-basierte Echtzeit-Anwendung umzusetzen, wenn man weitestgehend auf eine zentrale Server-Infrastruktur verzichtet.

Um die Frage nach der technischen Eignung der WebRTC-Technologie zu beantworten, wird ein Prototyp in Form eines Echtzeit-Online-Editors entwickelt. Die Durchführung wird von einem Expertenteam begleitet, um die grundlegenden Anforderungen zu definieren und die Nutzbarkeit der Applikation zu evaluieren. Es ist zu klären inwieweit der Einsatz von Servertechnologie minimiert werden kann und welche Gründe es notwendig machen Server einzubinden und auf welche Art diese Faktoren die Nutzung der Applikation beeinflussen.

1.4 Ablauf und Struktur

Zunächst geht es in Kapitel 2 darum einen Einblick in bereits vorhandene Lösungsansätze und Anwendungsfälle im Bereich der Collaboration-Tools zu geben, um den Prototypen einzuordnen. Außerdem soll evaluiert werden, ob es bereits WebRTC basierte Ansätze gibt, die einen Einblick in Architekturen geben und eventuelle Probleme aufzeigen. Kapitel 3 erklärt die technischen Grundlagen der WebRTC-Technologie und zieht Vergleiche mit weiteren Technologien und Alternativen.

In Kapitel 4 werden die Anforderungen an den Prototypen definiert. Ziel ist es auf Grundlage einer Kontextanalyse, die mithilfe eines Expertenteams erstellt wird, einen Anwendungsfall zu definieren, der von der Applikation unterstützt werden soll. Kapitel 5 beschäftigt sich mit der Architektur der Anwendung, wobei zunächst ein Blick auf die Gesamtstruktur geworfen wird und die notwendigen Serverkomponenten beschrieben werden. Anschließend wird die Softwarearchitektur erläutert. Kapitel 6 erläutert Teile der Implementierung des Prototypen. Im Fokus steht der Kommunikationsprozess, sowie die Abläufe innerhalb der Oberfläche.

Kapitel 7 dokumentiert anhand von Softwaretests inwieweit die Applikation die vorgegebenen Anforderungen erfüllt und welche Ergebnisse bei der Frage nach der technischen Eignung der WebRTC Technologie für den Einsatzzweck erzielt wurden. Usability Tests sollen zudem Aufschluss darüber geben, ob die Applikation neben der technischen Eignung auch als gebrauchstauglich wahrgenommen wird. Abschließend fasst das Kapitel 8 die Ergebnisse zusammen und liefert einen Ausblick.

Kapitel 2

Code-Editoren

2.1 Ist-Stand

Unter einem Collaboration-Tool, auch als Collaborative Software oder Groupware bezeichnet, versteht man allgemein Software, die die Zusammenarbeit einer Gruppe unterstützt. Eine grundlegende Definition stammt von Ellis, Gibbs und Rein [EGR91] aus dem Jahr 1991, die unter dem Begriff Groupware ein Computer-basiertes System verstehen, das eine Gruppe von Personen in ihrem Aufgabengebiet oder Ziel unterstützt und eine Schnittstelle für eine geteilte Arbeitsumgebung bietet. Sie kategorisieren die Systeme anhand ihres Grades ihrer räumlichen und zeitlichen Verteilung.

Für Collaboration-Tools ist vor allem die zeitliche Komponente relevant, d.h. ob synchron oder asynchron gearbeitet werden kann. Asynchrone Tools erlauben eine Zusammenarbeit zu unterschiedlichen Zeiten und an verschiedenen Orten. Sie ermöglichen es bspw. über einen bestimmten Zeitraum jederzeit Ressourcen zur Verfügung zu stellen und Abläufe zu protokollieren. Synchrone Tools erlauben Echtzeitkommunikation, d.h. die Teilnehmer agieren zur gleichen Zeit, unabhängig an welchem Ort sie sich befinden [vgl. KWW].

Typische asynchrone Tools sind Programme zur Unterstützung des Projektmanagements, Kalender- und Emailprogrammme oder Forensysteme. In der Softwareentwicklung fallen z.B. Tools zur Quellcodeverwaltung in diese Kategorie, auch Tools, die neben der reinen Versionsverwaltung erweiterte Features anbieten um bspw. Code-Reviews durchzuführen. Typische Anwendungsfälle für synchrones Arbeiten sind Programme zur Audio- und Videoübertragung, Textchats, Screen-Sharing und Tools, die eine Echtzeitbearbeitung von Dokumenten zulassen.

Verfügbare Code-Editoren, als Sonderform der kollaborativen Dokumentenbearbeitung können sowohl asynchrones als auch synchrones Arbeiten unterstützen, je nachdem ob die Dokumente in Echtzeit von verschiedenen Entwicklern bearbeiten werden können oder ob sie das einfache Teilen und Veröffentlichen von Quellcode unterstützen, um Rückmeldung aus einer

Gruppe von Entwicklern zu erhalten oder um selbst eine Hilfestellung für andere zu bieten. Beispiele für solche Editoren sind JSFiddle[1] oder auch LiveWeave[2], die speziell auf das Teilen und Bearbeiten von webbasierten Inhalten spezialisiert sind. Editoren, die eine synchrone Bearbeitung von Quellcode zulassen sind bspw. CodeBunk[3] und CodeShare[4]. Auf der Seite codegeekz.com [cod15] sind weitere Editoren aufgeführt, die auch in Suchmaschinen hoch gelistet werden. Zielgruppen und Fokus der Anwendungen unterscheiden sich erheblich. Während sich CodeShare bspw. auf das Führen von Interviews spezialisiert hat und deshalb neben dem Code-Sharing Audio und Video unterstützt, stellt AWS Cloud9[5] eine komplette Cloud-basierte, integrierte Entwicklungsumgebung zur Verfügung. Eine weitere Variante ist die Integration des Code-Sharings in bereits vorhandene Editorlösungen, wie es das Editor-Plugin Floobits[6] anbietet.

2.2 Technologien

Bei der Einordnung von Collaboration-Tools ist es außerdem interessant auf welcher Architektur die Systeme aufbauen. Es sind verteilte Systeme, die entweder eine zentrale Architektur aufweisen, d.h. alle Clients sind mit einem zentralen Server verbunden, der die komplette Verwaltung übernimmt. Alternativ existiert keine zentrale Verwaltungsinstanz und die Teilnehmer sind direkt über ein Netzwerk miteinander verbunden (Peer-to-Peer Architektur). Zudem gibt es hybride Architekturen, in denen zwar ein Server existiert, dieser aber keine zentrale Rolle bei der Verwaltung der Daten übernimmt [vgl. Wikb].

Welche Technologien bei Online-Editoren zum Einsatz kommen, d.h. ob es sich um Client-Server Strukturen handelt oder P2P Strukturen vorliegen, lässt sich sicher bei Open Source Lösungen bestimmen. Allerdings geben die Browser in ihrer Ressourcenauflistung Aufschluss darüber welche Technologie verwendet wird. Ob und wie WebRTC auf den Seiten eingesetzt wird, lässt sich über die *WebRTC Internals Tools* des Chrome-Browsers[7] ermitteln, der umfangreiche Daten über Verbindungsaufbau und geöffnete Kanäle preisgibt.

[1] https://jsfiddle.net/
[2] https://liveweave.com/
[3] https://codebunk.com/
[4] https://codeshare.io/
[5] https://aws.amazon.com/de/cloud9/?origin=c9io
[6] https://floobits.com/
[7] Aufrufbar über *chrome://webrtc-internals*

Aktuell sind in den in Suchmaschinen gelisteten Online-Code-Editoren Client-Server Strukturen vorherrschend. Bei den oben genannten Editoren sind zudem einige Dienste kostenpflichtig oder einige Features nur nach einem Login zugänglich, was eine umfangreiche Backendanwendung notwendig macht. Tools wie CodeBunk und CodeShare beinhalten Kommunikationslösungen, wie einen Textchat und/oder einen Audio-/Videochat. Während für die Audio-, und Videokommunikation WebRTC eingesetzt wird, wird für für den Textchat und die zu übertragenden Daten der WebRTC-Datenkanal nicht genutzt.

JSFiddle und LiveWeave verwenden die Bibliothek *Together.js*[8] der Mozilla Corporation, um die Echtzeit-Verbindung des Audiochats aufzubauen. In der Dokumentation der Bibliothek wird die Nichtverwendung der WebRTC-Datenkanäle thematisiert. Die mangelnden Unterstützung der Browser, die Notwendigkeit eines Servers um die Verbindung initial herzustellen, sowie der langsamere Verbindungsaufbau sei Grund für die Verwendung des WebSocket Protokolls anstatt des WebRTC Datenkanals [Tog]. Die Bibliothek ist quelloffen, sie ist im Kern seit ca. 4 Jahren nicht bearbeitet worden, so dass viele Probleme zu Beginn der Einführung der Technologie, wie mangelnde Browserunterstützung bei der Einschätzung relevant waren.

Zu einer ähnlichen Einschätzung kommt der Hersteller des Editor-Plugins Floobits [Flo]. Für Entwickler von Software stehen eine Reihe von Desktop-Entwicklungsumgebungen zur Verfügung, die einen großen Funktionsumfang bieten um Entwickler bei ihrer Arbeit zu unterstützen und in der Regel sind dies installierbare Desktop-Programme. Einige lassen sich um die Funktion des kollaborativen Arbeitens erweitern oder Drittanbieter wie Floobits entwickeln entsprechende Plugins. Vorteil ist, dass der Quellcode nicht extra in einen anderen Editor kopiert werden muss und man sich in seiner gewohnten Entwicklungsumgebung befindet. Nachteil ist, dass sich die Entwickler auf einen Editor einigen müssen und Lizenzkosten entstehen können. Floobits verwendet WebRTC für seinen Audio- und Videochat. Die Übertragung der Daten wird über einen zentralen Server geleitet. Begründet wird die Entscheidung mit einem nicht-trivialen Backend, Probleme mit Firewalls, dem komplizierteren Aufbau von P2P Netzwerken und resultierenden Problemen bei der Synchronisierung der Daten. Zudem sind verschiedene Logins zu verwalten, sei es zum Abrechnen der Dienste oder zum Login in externe Systeme, die entsprechende Serverstrukturen erfordern [Flo].

Nichtsdestotrotz gibt es etablierte Code-Editor Umsetzungen, die die Datenübertragung über WebRTC realisiert haben. Ein Beispiel für ein WebRTC-basiertes Tool, dass als

[8] https://togetherjs.com/

offene, dezentrale Lösung implementiert ist, ist das Teletype Plugin für den Editor Atom [Sob17]. Atom als freier Text und Code-Editor baut auf Webtechnologie auf und nutzt das Electron Framework [Ele], um den Editor für alle Plattformen als installierbare Desktop-App verfügbar zu machen. Der initiale Verbindungsaufbau erfolgt über einen Signaling-Server, danach werden keine Daten über zentrale Server geleitet. Das Problem der unterschiedlichen Unterstützung der Browser entfällt bei der Lösung, da Electron den Browser Chromium als Basis verwendet. Das Beispiel zeigt, dass eine alltagstaugliche Umsetzung möglich ist.

Das der WebRTC-DataChannel grundsätzlich eine Alternative zu WebSocket-Lösungen darstellt, beschreiben Pinikas u. a. [Pin+16] in ihrem wissenschaftlichen Artikel über ihre Entwicklung einer Online-Collaboration Plattform. Der Prototyp demonstriert die Möglichkeiten der WebRTC-Technologie im Bereich des Screen-Capturing und simuliert durch den WebRTC-Datachannel eine WebSocket-Verbindung vollständig. Die Implementierung wird in ihrer abschließenden Bewertung als gleichwertige Technologie eingestuft.

Zusammenfassend lässt sich sagen, dass der Einsatz des WebRTC-DataChannels eher selten ist, während sich die Umsetzung von Audio- und Videolösungen stärker etabliert haben. Die Komplexität der Lösungen bspw. für Login, Rechteverwaltung oder komplexe Features erfordern eine Backend-Lösung. Technische Einschränkungen, wie Firewalls werden als hohe Hürden wahrgenommen. Zudem existiert mit WebSockets eine etablierte Technologie, die nicht einfach durch eine experimentelle Technologie ersetzt wird, wenn die Vorteile der WebRTC-Technologie wie Datenschutz weniger Relevanz haben und ohnehin ein Backend-System besteht [Tok14]. Tabelle 2.1 fasst die verschiedenen Typen anhand der genannten Beispiele zusammen, die sich durch ihrer Plattform, Synchronität (zeitliche Komponente), Architektur und Fokus unterscheiden lassen.

	Plattform	Synchronität	Architektur	Fokus
JSFiddle/LiveWeave	Browser	asynchr.	Client-Server	Teilen
CodeBunk/CodeShare	Browser	synchr..	Client-Server	Interviews
AWS Cloud9	Browser	synchr.	Client-Server	Kollab.
Floobits	Plugin	synchr.	Client-Server	Kollab.
TeleType	Plugin	synchr.	P2P/WebRTC	Kollab.

Tabelle 2.1: Einordnung kollaborative Code-Editoren

Kapitel 3

Konnektivität im Browser

Für die Umsetzung des Prototypen sind eine Reihe von Webtechnologien relevant, die in diesem Kapitel erläutert werden. Grundlegend sind zum einen HTML5, JavaScript und CSS3. Eine besondere Relevanz haben Technologien, die Kommunikationswege von Browser zu Servern und von Browsern zu Browsern zur Verfügung stellen.

HTML5 ist die aktuelle Version der Hypertext Markup Language, die seit 2014 nach langer Entwicklungszeit in ihrer finalen Spezifikation vorliegt. Während sie in den Vorgängerversionen eine reine Beschreibungssprache zur Strukturierung von Webseiten war, bringt sie heute eine Reihe von APIs mit, die es möglich machen Desktop ähnliche Webapplikationen zu entwickeln. Wenn von HTML5 und den verbundenen APIs die Rede ist, bezieht sich dies auf die Kombination von HTML als Markup-Language, JavaScript als Programmiersprache und CSS3 als Sprache zur Definition der Styles einer Webseite.

Die neuen Spezifikationen beinhalten eine Reihe von Technologien, wie die semantische Beschreibung von Inhalten, verschiedene Wege um mit einem Server zu kommunizieren, Offline-Speicherung von Daten, das Einbinden von Multimediainhalten, 2D/3D Grafik, Leistungsverbesserungen sowie Zugriff auf Hardware. Zudem wurden die Möglichkeiten des Stylings erweitert, so dass Anwendungen benutzerfreundlich gestaltet werden können [Mozc; Con17]. Im Fokus dieses Kapitels stehen die verfügbaren APIs im Bereich der Konnektivität, speziell der WebRTC APIs im Vergleich zur ihren Alternativen.

3.1 XHR und AJAX

XMLHttpRequest (XHR) ist eine standardisierte JavaScript-API, mit der es möglich ist JavaScript-gesteuert Daten per HTTP-Request zu übertragen, d.h. Daten können dynamisch von einem Webserver angefordert werden. Bevor diese Technologie zu Verfügung stand war es notwendig bei jeder Benutzeraktion eine Anfrage an den Server zu senden und die

Webseite vollständig neu zu laden. Mit XHR lassen sich nach Bedarf Teile der Webseite im Hintergrund austauschen, so dass der Benutzer weniger Verzögerungen wahrnimmt.

Eine einfache Strategie um regelmäßig Aktualisierungen vom Server zu erhalten ist periodisches Prüfen, sogenanntes Polling (Abb 3.1). Der Client sendet in regelmäßigen Abständen Anfragen an den Server und fragt nach Änderungen. Viele moderne Webapplikationen verwenden diese Technologie um im Hintergrund Daten nachzuladen oder Daten zu evaluieren. Die Anfragen verlaufen asynchron, so dass sich Komponenten auf den Seiten nicht blockieren. Der Oberbegriff für diese Technologie ist AJAX, kurz für *Asynchronous JavaScript and XML*. Die regelmäßigen und damit auch unnötigen Anfragen erzeugen einen Overhead, den man reduzieren kann, wenn der Server nicht sofort antwortet, sondern eine festgelegte Zeit wartet, wenn es keine Änderungen in den Daten gegeben hat. Gibt es Änderungen werden sie umgehend versandt und die Verbindung geschlossen, wie auch beim Ablauf der definierten Wartezeit. Der Client öffnet die Verbindung direkt wieder, so dass eine andauernde Verbindung simuliert wird. Dieses Verfahren nennt sich Long-Polling [GIN15, S. 26-28].

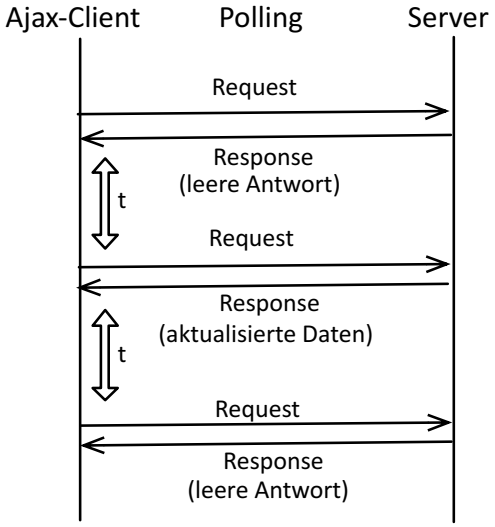

Abb. 3.1: Polling [GIN15, S. 27].

3.2 WebSockets

WebSockets ist ein auf TCP (Transmission Control Protocol) basierendes Kommunikationsprotokoll, das eine bidirektionale Verbindung zwischen einem Client und einem Server aufbaut, der WebSockets unterstützt. Mit der 2011 veröffentlichten WebSocket API [FM11] ist es möglich eine Anfrage an den Server zu senden und benachrichtigt zu werden, wenn Änderungen vorliegen. Das Protokoll ermöglicht Entwicklern auf eine standardisierte (RFC6455) Lösung zurückzugreifen, die eine nachrichtenorientierte, bidirektionale Verbindung aufbaut, über die Text und zusätzlich auch Binärdaten gesendet werden können.

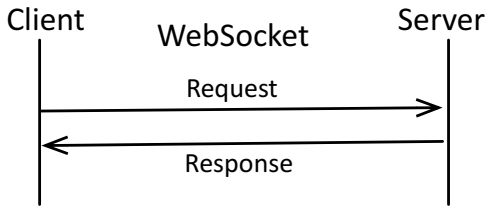

Abb. 3.2: Websockets

TCP bringt eine Reihe nützlicher Eigenschaften mit. Es ist ein zuverlässiges, verbindungsorientiertes, paketvermitteltes Transportprotokoll. Verbindungsorientiert, da die Verbindung zwischen Sender und Empfänger in beiden Richtungen hergestellt und aufrechterhalten wird, paketvermittelt da die Daten durch das Internetprotokoll (IP) in Pakete zerlegt und auf der Empfängerseite wieder zusammengesetzt werden, da sie unterschiedliche Wege durch das Netz nehmen. Auf TCP basiert ein Großteil der Kommunikation im Internet, da das Protokoll von Hause aus garantiert, dass die Daten in der Form ankommen wie sie versendet wurden. Zuverlässig also in dem Sinne, dass alle Daten vollständig und in der richtigen Reihenfolge ankommen [Wik18].

Es existieren eine Reihe von WebSocket-Implementierungen für verschiedene Umgebungen und Sprachen, wie bspw. die Javascript-Bibliothek Socket.io [GIN15, S. 109]. Sie bietet eine Abstraktionsschicht für Clients. Falls diese Websockets nicht unterstützen wird die bidirektionale Verbindung per Long Polling simuliert, so dass in jedem Fall eine Verbindung zustande kommt. Ein Beispiel wie Client und Server miteinander kommunizieren können zeigen folgende Listings.

```
1 const socket = io.connect('http://localhost:3000');
2 socket.on('event1', function(data) {
3  console.log(data);
4  socket.emit('event2', 'Nachricht vom Client');
5 });
```

Listing 3.1: Socket.io auf Clientseite

```
1 const io = require('socket.io').listen(3000);
2 io.sockets.on('connection', function(socket) {
3  socket.emit('event1', 'Nachricht vom Server');
4  socket.on('event2', function(data) {
5   console.log(data);
6  });});
```

Listing 3.2: Socket.io auf Serverseite

Das Beispiel zeigt, wie einfach mit Socket.io eine Verbindung hergestellt werden kann. Öffnet der Client eine Verbindung zum Server, der auf Port 3000 lauscht, wird eine Nachricht an den Client gesendet, der auf das Ereignis reagiert und antwortet.

3.3 WebRTC

WebRTC ist eine Sammlung von Standards, Protokollen und JavaScript APIs, die es ermöglichen eine Peer-to-Peer Verbindung zwischen Browsern aufzubauen, um Audio/Video und Daten zu senden und zu empfangen, d.h. es erweitert den Browser um die Fähigkeit der Echtzeitkommunikation ohne dass zusätzliche Plugins im Browser installiert werden müssen. WebRTC stellt drei primäre Programmierschnittstellen zur Verfügung [LR14, S. 6-8]:

RTCPeerConnection Die RTCPeerConnection API ist für die direkte Kommunikation zwischen den Browsern zuständig. Nachdem eine PeerConnection zustande gekommen ist, können Streams und Daten über diese Verbindung versendet werden.

RTCDataChannel Die RTCDataChannel API erlaubt das bidirektionale Senden von beliebigen Daten über die RTCPeerConnection.

MediaStream Die MediaStream API ist eine Repräsentation eines Audio- oder Videostreams. Die API ermöglicht die Ausgabe und das Senden des Streams.

RTCPeerConnection und RTCDataChannel sind die beiden relevanten APIs für diese Arbeit und werden in den folgenden Kapiteln erläutert. Die MediaStream API wird nicht benötigt da Audio- oder Videostreams nicht Teil dieser Arbeit sind. Die umfangreichen Möglichkeiten der API werden auf den Seiten des Mozilla Developer Networks ausführlich erklärt [Mozd].

3.3.1 RTCPeerConnection

Die RTCPeerConnection API repräsentiert eine WebRTC Verbindung zwischen einem lokalen und entfernten Computer und ist für den Auf- und Abbau der Verbindung verantwortlich. Sie überwacht und koordiniert den kompletten Lebenszyklus einer P2P-Verbindung. Da WebRTC, anders als WebSockets, UDP als Transportprotokoll nutzt, sind eine Reihe weiterer Protokolle notwendig, um bidirektionale Verbindungen möglich zu machen.

UDP (User Datagram Protocol) ist anders als TCP ein verbindungsloses Transport-Protokoll. Es wird keine feste Datenverbindung aufgebaut, die Daten werden direkt von Anwendung zu Anwendung gesendet. Es existieren bis auf eine Prüfsumme keine Sicherungsmechanismen. Es gibt keine Garantie, dass ein gesendetes Paket ankommt, auch nicht, dass Pakete in der richtigen Reihenfolge ankommen oder unverfälscht sind. D.h. eine Anwendung muss sich selbst um Sicherungsmechanismen kümmern oder sie ist unempfindlich gegenüber Datenverlusten.

Die Vorteile liegen in der Übertragungsgeschwindigkeit, da UDP weniger Overhead erzeugt und einfacher aufgebaut ist. Das Protokoll wird bspw. für Multimedia-Anwendungen oder Voice-Over-IP eingesetzt. Hier sorgen verlorengegangene Pakete zwar für eine schlechtere Qualität, die Übertragung verzögert sich aber nicht. Audio- und Videostreaming über WebRTC bedient sich standardmäßig einer nicht-zuverlässigen UDP Verbindung.

Um eine Verbindung zwischen zwei Browsern direkt aufzubauen ist es notwendig die IP-Adresse und Port des Anderen zu kennen. Zwei Prozesse sind hier wichtig, das sogenannte ICE Traversal und der Prozess des Signaling.

ICE

ICE kurz für *Interactive Connectivity Establishment* [IETa] ermöglicht den Verbindungsaufbau zwischen zwei Peers, indem es den Verbindungsweg ermittelt. Im einfachsten Fall sind beide Endgeräte im selben lokalen Netzwerk. Um die Verbindung herzustellen braucht es nur die interne IP-Adresse, die beim Betriebssystem angefragt werden kann. Anders sieht es aus wenn die Verbindung über Netzwerkgrenzen hinweg etabliert werden muss. NAT (Network Address Translation) ermöglicht, dass mehrere Rechner eine im Internet erreichbare Adresse gleichzeitig verwenden können, d.h. intern bekommen sie eine private Adresse zugewiesen und teilen sich nach außen die öffentliche IP-Adresse. Router übernehmen bspw. diese Aufgabe. Auch Firewalls sorgen dafür, dass die direkte Verbindung nicht ohne Weiteres hergestellt werden kann. WebRTC behilft sich mit einem sogenannten ICE Agent, der für die Ermittlung der eigenen IP-Adresse bei einem externen STUN-Server anfragt. Die PeerConnection API kümmert sich um den kompletten ICE Workflow und ermittelt die ICE Candidates (Abb. 3.3).

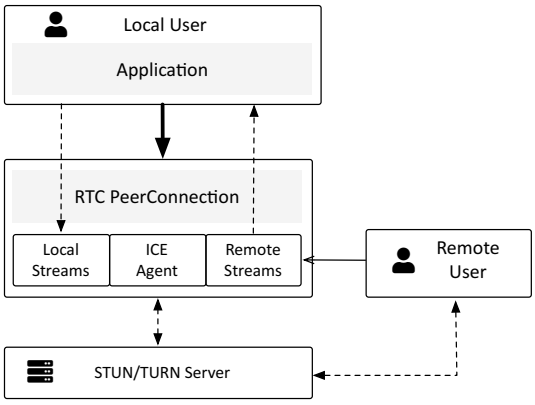

Abb. 3.3: RTCPeerConnection [Gri13]

STUN/TURN

STUN steht für *Session Traversal Utilities for NAT*. Ein STUN-Server gibt einem anfragenden Client seine öffentliche IP-Adresse und den Port zurück, so wie er von außen gesehen wird. In den meisten Fällen ist die Anfrage an einen STUN-Server ausreichend um die Verbindung aufzubauen. Sollte aber z.B. eine Firewall UDP Verbindungen blockieren ist keine direkte

Verbindung möglich. In diesen Fällen kann ein TURN-Server eingesetzt werden. *Traversal Using Relay NAT* ist eine Erweiterung zu STUN. Anders als bei einem STUN-Server, der nach der Ermittlung der IP-Adresse nicht weiter in den Prozess eingreift, werden bei der Nutzung eines TURN-Servers alle Daten über ihn umgeleitet. Er fungiert als Relay-Server, d.h. es handelt sich hier nicht mehr um eine P2P Verbindung [Gri13, S. 42,329].

Listing 3.3 zeigt, wie eine RTCPeerConnection mit der Angabe mehrerer STUN-Server instanziiert wird. Das Beispiel ist verkürzt aus der Dokumentation des Mozilla Developer Networks [Mozf] entnommen. Die Angabe einer Konfiguration ist optional, so dass ein Aufruf ohne Parameter nur in einem lokalen Netzwerk eine Verbindung aufbauen kann. Neben der Übergabe der Server Parameter können noch eine Reihe anderer Angaben, bspw. zur Authentifizierung, gemacht werden.

```
1 const configuration = {
2 iceServers: [{
3  urls: [
4   "stun:stun.example.com",
5   "stun:stun-1.example.com"
6  ]
7 }]};
8 let pc = new RTCPeerConnection(configuration);
```

Listing 3.3: Beispiel: Aufbau einer RTCPeerConnection

Für den STUN/TURN-Server kommen verschiedene Lösungen in Frage. Für Testzwecke gibt es eine Reihe von öffentlichen STUN-Servern. Verlässlich erreichbar sind z.B. die Server von Google[1] oder der öffentliche Server des OpenSource Projekts STUNTMAN[2], eine Implementierung des STUN-Protokolls. TURN-Server sind in der Regel nicht öffentlich und erfordern eine Authentifizierung, da sie als Relay-Server fungieren und der Datentransfer hohe Kosten verursachen kann. Wird ein TURN-Server benötigt ist ein eigener Server deshalb sinnvoll. Eine häufig eingesetzte Implementierung steht mit Coturn[3] zur Verfügung, der STUN- und TURN-Protokoll implementiert.

[1] stun:stun.l.google.com:19302, siehe dazu auch: WebRTC Samples - Trickle ICE [tri]
[2] http://www.stunprotocol.org/
[3] https://github.com/coturn/coturn

Signaling

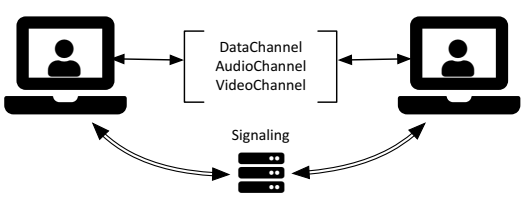

Abb. 3.4: WebRTC Signaling

Durch einen Signaling-Server wird der initiale Verbindungsaufbau koordiniert und geprüft, ob der Gesprächspartner erreichbar ist und eine Verbindung aufgebaut werden kann. Da sich die beiden Peers nicht kennen, braucht es einen sogenannten Signaling Channel, den beide Parteien erreichen können und über den sie initiale Daten austauschen können. Der Prozess der Datenübertragung ist nicht standardisiert und nicht in den WebRTC APIs implementiert. Im Grunde geht es darum definierte META-Daten zu übertragen, um beiden Seiten alle notwendigen Informationen zum Verbindungsaufbau zur Verfügung zu stellen. Für die Übertragung der Daten kann auf etablierte Lösungen oder eigene Implementierungen zurückgegriffen werden. Nach Johnston und Burnett [JB14] können für das Signaling drei Technologien zum Einsatz kommen: HTTP, Websockets oder (mit Einschränkungen) der DataChannel selbst. HTTP ermöglicht Signaling bspw. über den Polling-Mechanismus, Websocket-Lösungen bieten eine bidirektionale Verbindung zum Austausch der Daten an. Ist die Verbindung zwischen zwei Clients bereits zustande gekommen, kann auch der DataChannel selbst zukünftiges Signaling übernehmen, z.B. wenn ein Audiostream nachträglich ergänzt werden soll. Der Signaling-Server kann außerdem noch weitere Funktionen übernehmen, wie Identifikation und Authentifikation des Clients oder die Kontrolle der Session.

Nach Grigorik [Gri13, S. 42,329] gibt es keine eindeutig korrekte Wahl für einen Server, da es von der Art und den Anforderungen der Applikation abhängt welcher sinnvoll ist. Für Voice Over IP und Videoübertragungen existiert bspw. bereits ein Protokoll (SIP), so dass Audio- und Videolösungen vorhandene Strukturen nutzen können. Denkbar sind auch Datenbanken, wie zum Beispiel Cloud Firestore von Google[4], wie sie im Beispiel von Marcus [Mar17] verwendet wird. In den Codelabs [Cod] von Google wird, wie in vielen Tutorials und Implementierungen, ein Signaling-Server auf Basis von Socket.io eingesetzt, aber auch auf eine Fülle von Alternativen verwiesen [Cod; Khab]. Da auch in dieser Arbeit auf die Technologie zurückgegriffen wird, wird in Kapitel 5.1.1 nochmals darauf eingegangen.

[4] https://firebase.google.com/docs/firestore/

SDP

Das Session Description Protocol [IETc] beschreibt das Format der META-Daten einer RTCPeerConnection. Listing 3.4 zeigt die Inhalte der lokalen Session Description einer Offer, wie sie der Chrome-Browser generiert, wenn lediglich ein DataChannel verwendet wird und eine Verbindung zu einem entfernten Peer aufgebaut wird.

```
1  v=0
2  o=- 6743868245450478716 2 IN IP4 127.0.0.1
3  s=-
4  t=0 0
5  a=group:BUNDLE data
6  a=msid-semantic: WMS
7  m=application 53345 DTLS/SCTP 5000
8  c=IN IP4 31.18.241.12
9  a=candidate:827648026 1 udp 2113937151 192.168.1.55 53345 typ host
       generation 0 network-cost 999
10 a=candidate:842163049 1 udp 1677729535 31.18.241.12 53345 typ srflx
       raddr 192.168.1.55 rport 53345 generation 0 network-cost 999
11 a=ice-ufrag:rIov
12 a=ice-pwd:AIa7F8DG9Ob9axJfX7HtQgyz
13 a=ice-options:trickle
14 a=fingerprint:sha-256 F2:3E:34:D8:26:C5:C4:4A:C8:2E:56:2F:D3:CA:33:B2
       :6E:43:DC:92:E6:58:FE:06:C9:6A:E4:17:BF:E6:13:7F
15 a=setup:actpass
16 a=mid:data
17 a=sctpmap:5000 webrtc-datachannel 1024
```

Listing 3.4: Beispiel: Session Description Protocol

Zeile 9 und 10 zeigt die aufgelösten IP-Adressen. Da das Beispiel keine Informationen über mögliche Audio- oder Videoverbindungen enthält, ist es noch überschaubar. Auf eine Änderung an den zu übertragenden Medien, z.B die Ergänzung eines Audio-Channels, würde ein erneuter Austausch der SDP-Daten erfolgen, da die Beschreibung zusätzlich mögliche Audio-Codecs enthalten muss.

Verbindungsaufbau

Die Daten werden in einem Offer/Answer-Mechanismus ausgetauscht, wie er im JSEP (JavaScript Session Establishment Protocol) [IETb] beschrieben wird. Das Generieren eines RTCPeerConnection Objekts setzt folgenden Ablauf in Gang [Gri13, S. 328,329]:

1. Der Initiator PeerA generiert eine *offer* und speichert diese als *local description (sdp)*.
2. Die *offer* wird über den Signaling-Server an Peer B gesendet.
3. Peer B nimmt die *offer* entgegen und speichert sie als *remote description*.
4. Peer B generiert eine *answer* und speichert diese als *local description*.
5. Die *answer* wird über den Signaling-Server an Peer A gesendet.
6. Peer A nimmt die *answer* entgegen und speichert sie als *remote description*.

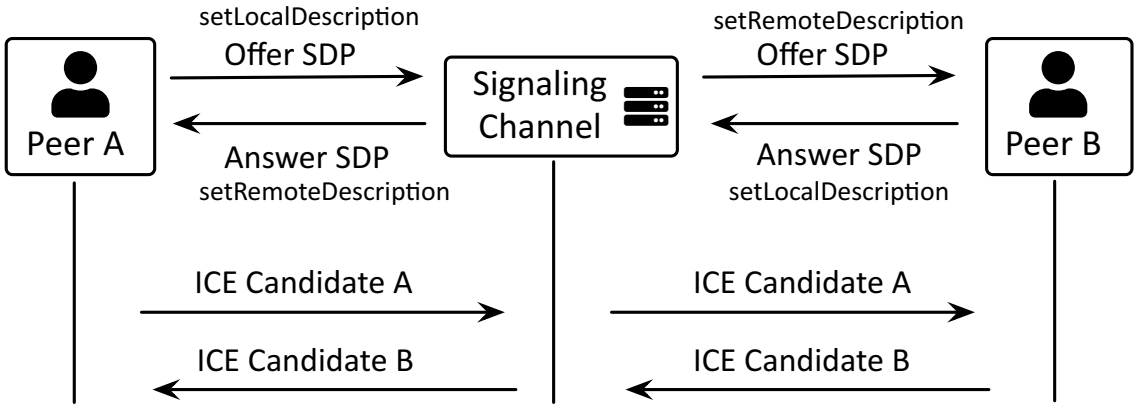

Abb. 3.5: Signaling Prozess

Wurde eine Session Description gesetzt (egal ob lokal oder remote), beginnt der ICE Agent mit dem Prozess mögliche ICE-Candidates zu finden, d.h. IP-Adressen mit denen eine Kommunikation aufgebaut werden kann. Das können lokale Adressen sein, als auch Adressen, die vorab von einem STUN/TURN-Server ermittelt wurden. In Abb. 3.5 ist der Prozess des Signalings der Übersicht halber ohne STUN/TURN-Server beschrieben.

Passend zu Listing 3.4 zeigt Listing 3.5, wie die ICE-Candidates angegeben werden. Durch den Prozess werden die Wege durch das Netz von Rechner zu Rechner offengelegt. Auch bspw. bei einer VPN-Verbindung wird die öffentliche Adresse offengelegt, was u.U. nicht erwünscht ist. Aus diesem Grunde werden z.B. Browser-Plugins angeboten, um WebRTC-Funktionalität abzuschalten.

```
1 candidate:3455323002 1 udp 2113937151 192.168.178.20 61606 typ host
    generation 0 ufrag YHqn network-cost 999
2
3 candidate:842163049 1 udp 1677729535 217.70.192.42 61606 typ srflx
    raddr 192.168.178.20 rport 61606 generation 0 ufrag YHqn network-
    cost 999
```

Listing 3.5: Beispiel: ICE Candidates

3.3.2 RTCDataChannel

SCTP über DTLS

Der RTCDataChannel ermöglicht, wenn die Verbindung der RTCPeerConnection aufgebaut wurde, den bidirektionalen Austausch beliebiger Daten. Im Grunde verhält sich der Channel ähnlich einer WebSocket Verbindung, aber Peer-to-Peer und mit der Möglichkeit die Transportoptionen zu konfigurieren. Aufbauend auf dem letzten Kapitel gibt es bis jetzt lediglich eine UDP Verbindung, die die Eigenschaften einer bidirektionalen Verbindung von sich aus nicht besitzt. Abb. 3.6 zeigt die Protokolle und Dienste, die notwendig sind, um eine verlässliche Verbindung aufbauen zu können, die zusätzlich immer verschlüsselt wird.

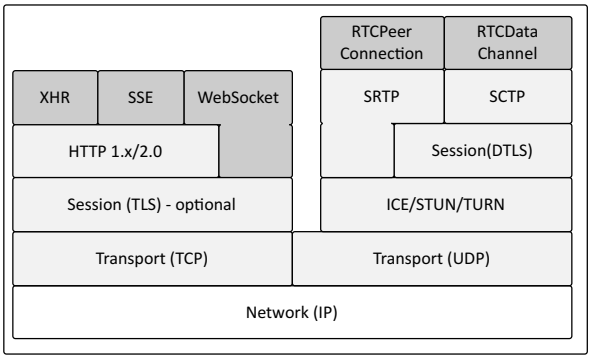

Abb. 3.6: WebRTC protocol stack [Gri13, S. 320]

DTLS, *Datagram Transport Layer Security*, ist zuständig für die Verschlüsselung der übertragenden Daten. Es wird hier nicht näher auf die Verschlüsselungstechnologie eingegangen. DTLS basiert auf TLS[5], das nur in Verbindungen mit zuverlässigen und geordneten Verbindungen, wie TCP funktioniert und deshalb modifiziert wurde. SCTP[6] wurde im Jahr 2000 veröffentlicht und gehört wie TCP zu den zuverlässigen, verbindungsorientierten Netzwerkprotokollen und setzt, wie Abb. 3.6 zeigt, auf DTLS auf und bildet damit das Schlusslicht in der Reihe der verwendeten Protokolle.

[5] Transport Layer Security
[6] Stream Control Transmission Protocol, [IETd]

Das Protokoll bringt einige Eigenschaften mit, die TCP nicht zur Verfügung stellt, wie z.B. die Fähigkeit mehrere Streams zu verwalten [Gri13, S. 341-348], [JB14, S. 221-222]:

- Es werden mehrere voneinander unabhängige Channel unterstützt.
- Jeder Channel kann die Pakete *in-order* oder *out-of-order* übertragen.
- Jeder Channel unterstützt verlässliche oder unverlässliche Übertragung.
- Channel haben Prioritäten, die von der Applikation vergeben werden können.
- Der Transport stellt eine *message-oriented* API bereit. Jede Nachricht kann aufgeteilt und wieder zusammengesetzt werden.
- Der Transport implementiert eine Fluss- und Überlastkontrolle (flow/congestion).
- Der Transport sorgt für eine sichere und vollständige Übertragung der Daten.

Channel-Aufbau

Listing 3.6 zeigt, wie über die RTCPeerConnection ein RTCDataChannel erstellt werden kann, wobei auch mehrere Channel verwaltet werden können.

```
1 let dc_reliable = pc.createDataChannel("Channel1");
```

Listing 3.6: Erstellen eines zuverlässigen RTCDataChannels

Der Aufruf ohne Parameter erstellt automatisch einen verlässlichen Channel. Drei Parameter beeinflussen die Art der Übertragung wobei die letzten beiden nicht gleichzeitig geändert werden können ohne einen Fehler zu generieren [Mozb; Fis].

ordered Gibt an, ob die Nachrichten in der gleichen Reihenfolge ankommen wie sie gesendet wurden. Standard: true.

maxPacketLiveTime Gibt die maximale Anzahl in Millisekunden an, in der versucht wird, eine Nachricht zu übertragen. Standard: null.

maxRetransmits Gibt die maximale Anzahl an erneuten Versuchen an, um eine Nachricht zu senden. Standard: null.

Möchte man also einen Channel so konfigurieren, der sich ähnlich einer UDP-Verbindung verhält, kann der Channel folgendermaßen konfiguriert werden [Fis]:

```
1 let dc_udp = pc.createDataChannel("Channel2", {ordered: false,
    maxPacketLifeTime: null, maxRetransmits: 0 });
```

Listing 3.7: Erstellen eines nicht-zuverlässigen RTCDataChannels

Um eine RTCDataConnection sowie einen RTCDataChannel zu erstellen, um bspw. ein Objekt über den Channel zu senden, benötigt es nur wenige Zeilen Quellcode [Moze]:

```
1 let pc = new RTCPeerConnection();
2 let dc = pc.createDataChannel("Channel");
3 function sendMessage(msg) {
4  let obj = {
5   "message": msg,
6   "timestamp": new Date()
7  }
8  dc.send(JSON.stringify(obj)); // Object -> string
9 }
```

Listing 3.8: Senden eines Objekts über den RTCDataChannel

WebRTC vs WebSocket

WebRTC und WebSocket bieten ähnliche Features an. Beide bauen zuverlässige, bidirektionale Verbindungen auf, wobei WebRTC-Verbindungen sich konfigurieren lassen. Beide Technologien lassen sich direkt miteinander vergleichen.

	WebSocket	DataChannel
Verschlüsselung	**konfigurierbar**	immer
Zuverlässigkeit	zuverlässig	**konfigurierbar**
Delivery (in-order/out-of-order)	geordnet	**konfigurierbar**
Multiplex	nein (Erweiterung)	ja
Transmission	message-oriented	message-oriented
Binärdaten	ja	ja
UTF-8	ja	ja
Kompression	nein (Erweiterung)	nein

Tabelle 3.1: WebSocket vs DataChannel [Gri13, S. 354]

AJAX stellt über HTTP keine bidirektionale Verbindung her, aus diesem Grunde wird hier kein direkter Vergleich gezogen. AJAX kann aber eine dauerhafte Verbindung über Polling simulieren, so dass die Socket.io Bibliothek auf die Technologie zurückgreift, wenn ein Browser kein WebSocket unterstützt.

3.3.3 Interoperabilität

Die aktuelle WebRTC Spezifikation [Ber+17] ist als stabil und vollständig spezifiziert gekennzeichnet. Dies bedeutet aber nicht, dass alle Browser alle Features bereits umgesetzt haben. Nach Caniuse.com [can] unterstützen, bis auf Internet Explorer und Edge, alle Browser die Technologie weitestgehend. Allerdings nicht auf die gleiche Weise. Aus diesem Grund entstand schon sehr früh ein Polyfill-Script namens *adapter.js*, das die Unterschiede in der Umsetzung ausgleichen soll. Zu Beginn 2012/13 glich die Bibliothek nur die unterschiedlichen Präfixe (webkitRTCPeerConnection/mozRTCPeerConnection) aus und stellte Hilfsfunktionen zur Verfügung, um einen Media-Stream an HTML-Elemente zu knüpfen. Über die Jahre ist das Script komplexer geworden. Es manipuliert die WebRTC-APIs und berücksichtigt die verschiedenen Entwicklungsstadien innerhalb der Browser soweit es möglich ist. So können Entwickler gegen die Spezifikation entwickeln und müssen weniger Fallunterscheidungen implementieren. Mit jedem Browser-Update ist es also sinnvoll zu prüfen, ob ein Update der *adapter.js*-Datei notwendig ist und zwar solange bis die Spezifikation vollständig umgesetzt ist [Lev18; wadap18].

Ein wichtiger Unterschied in der Umsetzung der Browser betrifft den RTCDataChannel und das Versenden von Nachrichten. Obwohl die Spezifikation den Umgang mit großen Nachrichten vorgibt, limitieren Mozilla Firefox und Google Chrome, zum aktuellen Stand, die maximale Größe der gesendeten Nachricht unterschiedlich. Die Empfehlung liegt aktuell bei 64kiB, beim Nachrichtenversand zwischen unterschiedlichen Browsern bei 16kiB, wie es die Spezifikation vorgibt. Eine Lösung zum Senden ohne Limitierung, die *SCTP ndata specification*, ist noch im IETF-Draft-Status [Mozg; Blo17].

3.4 Architekturen

Das Kapitel hat verschiedene Möglichkeiten aufgezeigt, wie Browser kommunizieren können, wobei neben der klassischen Client-Server Architektur nun mit WebRTC auch Peer-to-Peer Verbindungen möglich sind.

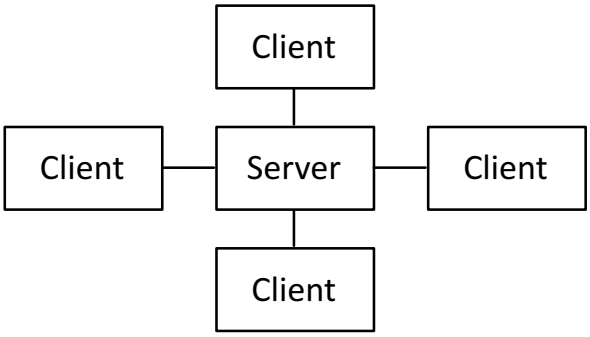

Abb. 3.7: Client-Server

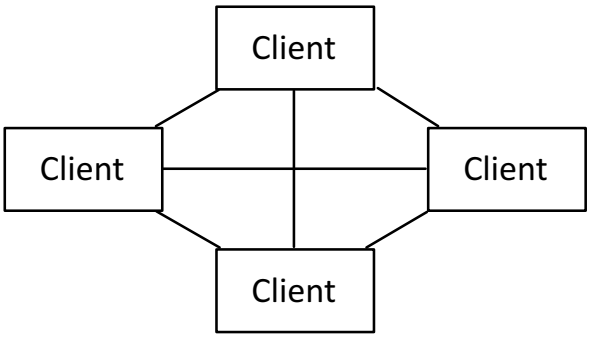

Abb. 3.8: Peer-to-Peer

Client-Server Systeme bestehen aus mindestens zwei kommunizierenden Rechnern oder Prozessen. Clients benötigen Dienste, Server stellen diese Dienste bereit. Klassische HTML-Seiten, AJAX und WebSockets stellen die Grundlage für Webseiten im Internet dar, die vielfach aufgerufen werden und ihre Daten zentral von Servern beziehen, die allgemein erreichbar sind. Die Verantwortlichkeiten sind klar getrennt, die Systeme sind skalierbar und flexibel, aber teurer zu administrieren. Bei dem Datenaustausch über einen RTCDataChannel hingegen sind alle Clients, genannt Peers, gleichberechtigt miteinander verbunden und übernehmen die Rolle des Clients und des Servers, je nachdem ob sie Ressourcen zur Verfügung stellen oder beziehen. Jeder Peer verwaltet seine eigenen Daten. Peer-to-Peer Architekturen haben den Vorteil, dass sie ohne Server als ausfallsicherer gelten, aber das Erkennen der Peers sich schwieriger gestaltet, vor allem wenn sich viele Peers im Netz befinden [Sta18, S. 121-122].

WebRTC-Architektur

Es zeigt sich, dass der Einsatz dreier Server notwendig ist, um eine WebRTC-basierte Architektur aufzubauen. Ein Webserver, der die Applikation ausliefert, ein Signaling-Server, der die initiale Verbindung aufbaut und ein STUN/TURN-Server, der Informationen über die IP-Adressen gibt. Ist die Verbindung aufgebaut, können die Daten Peer-to-Peer ausgetauscht werden. Die Systemarchitektur für WebRTC besteht also aus einer hybriden Architektur, in denen zwar Server existieren, diese aber keine zentrale Rolle übernehmen. Die Verbindung

wird über Server aufgebaut, weil sie für alle Peers erreichbar sind. Danach nutzt man die Vorteile des P2P-Netzwerks, um ohne zentrale Struktur kommunizieren zu können. Bei der Implementierung der WebRTC-Technologie hat dies zur Folge, dass alle Peers den selben Quellcode ausführen und doch ein Kriterium gefunden werden muss wer bspw. mit dem Verbindungsaufbau anfängt, also mit der *Offer* startet (siehe dazu Kapitel 6.1.2).

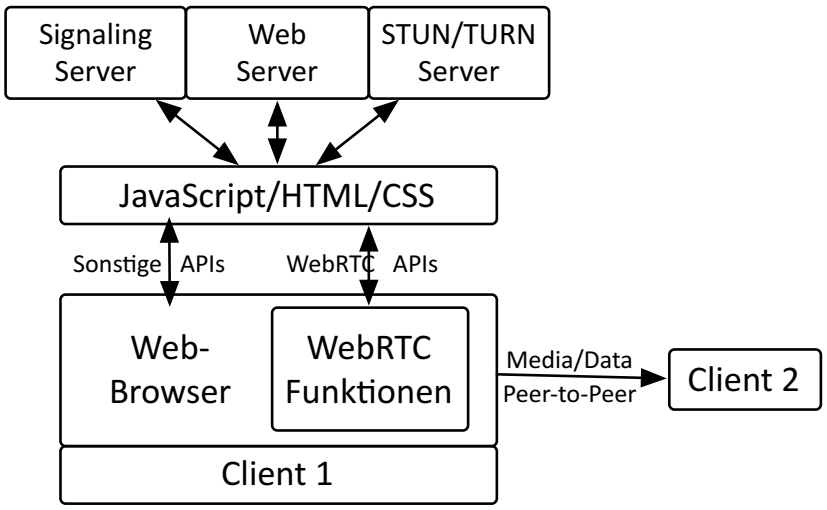

Abb. 3.9: WebRTC im Browser, [nach JB14, S. 3]

Kapitel 4

Anforderungsanalyse

In diesem Kapitel werden die Anforderungen an den Code-Editor festgelegt. Um sie zu definieren, ist es wichtig den Nutzer und den Kontext zu verstehen, in dem die Software verwendet wird. Nutzerforschung basiert auf Beobachtungen, Befragungen und Annahmen, d.h. sie liefert keine exakten Ergebnisse. Es lassen sich aber wichtige Erkenntnisse darüber ableiten, wie Benutzer arbeiten und welche Arbeitsabläufe und Ziele sie haben [vgl. Mos12, S. 56]. Ziel der Anforderungsanalyse ist die Definition von Anwendungsfällen, aus denen ein typisches Szenario abgeleitet werden kann, das beschreibt wie ein Benutzer den Editor konkret benutzt. Dieses Szenario ist Grundlage für den Usability Test in Kapitel 7.2. Zudem wird eine eindeutige und detaillierte Definition der Anforderungen erstellt, die durch funktionale Tests in Kapitel 7.1 überprüft werden. Abschließend wird ein grundlegendes Design vorgestellt, dass Grundlage für die Architektur des Gesamtsystems ist.

4.1 Kontextanalyse

Da spezielle Anwendungsfälle identifiziert werden sollen, die der Code-Editor unterstützt und die Ergebnisse der Analyse weder vergleichbar noch generalisierbar sind, sind qualitative Methoden zum Einsatz gekommen. Eine ausgewählte Expertengruppe aus Software-Entwicklern begleitete das Projekt, um zu Projektbeginn die grundlegenden, minimalen Anforderungen zu definieren und sie im weiteren Verlauf des Projekts zu evaluieren und die Umsetzung zu bewerten.

Das Experteninterview zu Beginn des Projekts fand als geführte, offene Diskussionsrunde statt, in der spezifische Fragen diskutiert und wichtige von unwichtigen Themen voneinander abgegrenzt wurden. In der Literatur wird auch von Fokusgruppen gesprochen, die in den Sozialwissenschaften häufig Anwendung finden. Es sind geführte Diskussionen, in denen Fragen von außen in die Gruppe getragen und offen diskutiert werden [Mos12, S. 68]. Die ausgewählte Gruppe setzte sich aus sechs Personen (Backend-Softwareentwickler und

Projektleitern) zusammen, die sowohl festangestellt als auch freiberuflich an verschiedenen Projekten zusammenarbeiten, d.h. die Teilnehmer kennen sich, sind aber nicht immer vor Ort. Ziel der Diskussion war es Erkenntnisse darüber zu erhalten, wie sich die aktuelle Situation bei der kollaborativen Arbeit aus der Sicht der Entwickler darstellt, welche Arbeitsabläufe als Problem wahrgenommen werden und wie sie mit einem Code-Editor verbessert werden könnten.

4.1.1 Ablauf

Um die Ablenkung gering zu halten wurde die Diskussionsrunde in einem ruhigen Besprechungsraum durchgeführt. Die Inhalte der Diskussion wurden von zwei Protokollanten festgehalten. Folgende Leitfragen wurden vorab definiert, um die Diskussion anzuregen oder Hilfe zu geben, falls der Gesprächsfaden verloren geht. Abschließend wurden die Teilnehmer per Fragebogen einzeln nach ihren Aufgaben im Unternehmen befragt, sowie nach dem ihrer Meinung nach wichtigsten Punkt innerhalb der Diskussion[1].

- Welche Softwaretools unterstützen bei der Zusammenarbeit allgemein? Welche Kommunikationsfunktionen werden genutzt?
- In welcher Form werden Code-Reviews durchgeführt und welche Tools unterstützen bei der Arbeit?
- Wird gemeinsam am Rechner entwickelt und aus welchen Gründen? Kann Software unterstützen und wie kann sie unterstützen?
- Gibt es Datenschutzbedenken bei der Nutzung der Tools, die von Cloudanbietern angeboten werden?

Mögliche Anwendungsfälle

Zwei Anwendungsfälle hatten sich vorab aus Beobachtungen von Arbeitsabläufen zwischen den Entwicklerteams abgeleitet und standen zur Diskussion. Der erste mögliche Anwendungsfall beschreibt die umfangreiche Prüfung des Quellcodes durch Code-Reviews. Bevor Arbeitsergebnisse in einen Entwicklungsstand integriert werden, werden sie manuell durch Mitarbeiter geprüft. Eingesetzt werden Quellcodeverwaltungs-Tools, die neben der Möglichkeit sich die Änderungen im Code anzusehen, auch eine Kommentarfunktionen anbieten, also eine asynchrone Kommunikation zulässt. Die zu klärende Frage war, ob ein Echtzeit-Code-Review den Kommunikationsprozess und die Klärung von Problemen effektiver gestalten könnte.

[1] Leitfaden und Fragebogen zur Diskussionsrunde siehe Anhang A.2

Der zweite mögliche Anwendungsfall beschreibt die Möglichkeit eines AdHoc Code-Reviews, d.h. dass Entwickler Hilfe bei der Lösung eines fachlichen Problems erhalten. Zunächst wird per Textchat über das Problem diskutiert, um einen Ansprechpartner zu finden. Ist der Mitarbeiter vor Ort, ergibt sich die Möglichkeit das Problem zusammen an einem Rechner zu lösen. Diese Art der Zusammenarbeit ähnelt dem Pair-Programming, bei dem zwei Entwickler an einem Rechner arbeiten um gemeinsam Quellcode zu entwickeln. Einer schreibt den Code, während der andere den Code kontrolliert. Allerdings ist es dort vorgesehen längerfristig paarweise zusammen zu arbeiten, während in diesem speziellen Fall nur kurzzeitig Hilfe angefordert wird [vgl. BA04, S.42]. Ist der Mitarbeiter nicht vor Ort bleibt nur die Möglichkeit weiter im Chat darüber zu diskutieren, da kein Tool eingesetzt wird, das ein synchrones, gemeinsames Arbeiten an Quellcode ermöglicht.

4.1.2 Ergebnisse

Um die erhaltenen Informationen auszuwerten und zu strukturieren, wurden die Inhalte der einstündigen Diskussion in einzelne Fakten herunter gebrochen, anschließend gruppiert und mit einer Überschrift versehen. Die folgenden Ergebnisse geben die gruppierten Themen wieder, die am höchsten priorisiert wurden, je nachdem wie intensiv über einzelne Punkte gesprochen wurde und ob im abschließenden Fragebogen ein Thema als besonders wichtig angegeben wurde.

Klassische Code-Reviews

Im Gespräch wurde die Relevanz von Code-Reviews hervorgehoben. Verwendet wird das Tool GitLab[2], das neben der Quellcodeverwaltung eine Reihe von Funktionen zur Verfügung stellt, um den Softwarelebenszyklus zu verwalten. Jeder erstellte Code wird einer formalen und inhaltlichen Prüfung unterzogen bevor er in den bestehenden Code integriert wird. Der zu kontrollierende Code besteht dabei in der Regel nicht aus kleinen Snippets, sondern umfasst größere Codefragmente bis hin zu zusammenhängende Dateien. Außerdem wird die Kommentarfunktion umfangreich zur Zusammenarbeit genutzt, die zwar zeitversetzt beantwortet wird, aber die Entwicklung protokolliert, solange der Merge-Request (die Anfrage auf Integration) offen ist. Es bestand Konsens darüber, dass ein Echtzeit-Editor aus diesen Gründen keine klassische Code-Review Funktionalität übernehmen kann, weil die Prüfungen als abschließender Prozess eingestuft wird, der zu umfangreich ist und dokumentiert werden muss.

[2] https://gitlab.com/

AdHoc Code-Reviews

Ein AdHoc-Review vorab, um Probleme zu lösen, wurde als sehr sinnvoll eingestuft. Kurze, auch spontane Treffen finden in der Regel statt, um Mitarbeiter einzuführen und anzuleiten oder weil akute Probleme besprochen werden müssen. In diesem Anwendungsfall wird die Möglichkeit des Live-Codings begrüßt, auch dann wenn sich beide Teilnehmer im selben Büro befinden. Pair-Programming, klassisch durchgeführt wie in der Vorüberlegung in diesem Kapitel beschrieben, wird so gut wie gar nicht durchgeführt.

Editorfunktionen

Es bestand Einigkeit darüber, dass sich der Editor auf die Kernfunktionalität, das Teilen und kollaborative Bearbeiten von Quellcode, beschränken sollte. Dies wurde besonders im Hinblick auf die Integration von Kommunikationsfunktionen betont. Es haben sich bereits Kommunikationstools etabliert, die parallel genutzt werden. Für die Kommunikation in einem Projekt wird hauptsächlich Slack[3] als Gruppenchat und Google Hangout[4] für Audio- und Videokonferenzen eingesetzt, da die Konferenzen direkt per Link aus dem Kalender oder Slack gestartet werden können.

Des Weiteren wurde ein fokussiertes Bearbeiten des Quellcodes gefordert, so dass der Initiator immer die Kontrolle behält wer den Quellcode bearbeiten kann.

Priorisiert wurde zudem, dass der integrierte Editor sich im Laufe der Entwicklung an die Standards der etablierten Desktop-Softwareentwicklungstools[5] orientiert. Als Beispiel wurde das Syntax-Highlighting genannt.

Initialisierung und Aufrechterhaltung der Session

Die Initialisierung einer Session, der Aufbau der Verbindung und die Möglichkeit des Beitritts zu einer Session sollten sich möglichst einfach und übersichtlich gestalten und die Einstiegshürden niedrig gehalten werden. Für die Einladung zu einer Session waren in der Runde eine generierte URL, die z.B. per Slack versendet werden kann eine adäquate Lösung. Ein weiterer Punkt war die notwendige zeitnahe Rückmeldung des Systems, falls die Verbindung unterbrochen oder von einer Seite abgebrochen wird.

[3] https://slack.com/intl/de-de
[4] https://hangouts.google.com/
[5] Verwendete Softwareentwicklungstools in den Projekten sind IntelliJ, PHPStorm und WebStorm von JetBrains (https://www.jetbrains.com)

Sicherheitsaspekte

Der Punkt Sicherheit bzgl. Cloudanbietern hat, anders als erwartet, keinen hohen Stellenwert eingenommen. Grundsätzlich wird die Problematik sicherheitsrelevanten Code bei Drittanbietern zu hosten gesehen, es gibt aber keine Notwendigkeit sie deshalb nicht zu nutzen. Das liegt zum einen an der Tatsache, dass die Tools von den Arbeitgebern vorgegeben werden, zum anderen überwiegen die Vorteile. Das Hosten der Server im eigenen Unternehmen wurde zwar in Erwägung gezogen, aber als zu aufwändig eingeschätzt.

4.1.3 Zusammenfassung

Die Entwicklerrunde hat gezeigt, dass der Anwendungsfall des AdHoc-Reviews sinnvoll umgesetzt werden kann, in dem kleinere Quellcode-Teile kollaborativ bearbeitet werden können. Der Editor sollte sich bevorzugt auf sein Kernfeature, der Bearbeitung von Quellcode beschränken und sich dementsprechend ähnlich verhalten wie ein Softwareentwicklungstool und Syntax-Highlighting unterstützen. Bereits vorhandene Tools, wie Slack, übernehmen ergänzend die Kommunikation. Wichtig sind Rückmeldungen zum Verbindungsstatus, damit Verbindungsabbrüche registriert werden. Klassische Code-Reviews wurden in der Runde als eine abschließende und umfangreiche Aufgabe im Entwicklungsprozess betrachtet, die mit den vorhanden Tools gut bewältigt werden können.

4.2 Der Prototyp

4.2.1 Features

Folgendes Szenario und die Liste der Minimalfeatures lassen sich aus dem Anwendungsfall des AdHoc-Reviews ableiten. Sie dienen als Vorlage für das Design der Anwendung.

AdHoc-Review Szenario

Softwareentwickler A möchte fachliche Fragen klären, die sich auf einen Teil seines entwickelten Quellcodes bezieht. Er hat die Möglichkeit über einen vorhandenen Kommunikationskanal, wie z.B. Slack, einen Mitarbeiter um Hilfe zu bitten. Er initiiert eine Session im Online-Editor und erhält von der Applikation eine Zufalls-URL, die er dem Mitarbeiter zukommen lässt und ihn so zur Online-Session einlädt. Der Mitarbeiter tritt durch den Aufruf der URL der Session bei. Unabhängig davon ob der Mitarbeiter bereits der Session beigetreten ist oder

nicht, kann der Initiator den Quellcode importieren oder per Copy/Paste in den Editor laden. Der Mitarbeiter, ab jetzt Co-Worker genannt, erhält Einsicht in den Quellcode und kann, nach vorangegangener Freigabe durch den Initiator, Änderungen vornehmen, die markiert werden. Die vorgenommenen Änderungen können exportiert werden.

Minimal-Features

- **F1: Session-Handling**
  Die App bietet die Möglichkeit eine Session zu öffnen und zu schließen. Die App generiert eine URL, mit der ein Mitarbeiter der Session beitreten kann. Wenn die Session geschlossen wird, wird der andere jeweils benachrichtigt.

- **F2: Kollaboration**
  Die Inhalte des Code-Editors werden mit dem Editor des Mitarbeiters synchronisiert, so dass veränderte Codezeilen sichtbar sind. Diese werden farblich markiert. Der Initiator der Session hat die Kontrolle darüber wer editieren darf, d.h. es kann immer nur einer den Quellcode bearbeiten.

- **F3: Chat**
  Der Editor beinhaltet einen Textchat, um eine minimale Kommunikationsmöglichkeit zusätzlich zu externen Tools sicherzustellen.

- **F4: Import/Export**
  Der Quellcode kann von der Festplatte importiert und exportiert werden, d.h. es kann eine Datei eingelesen werden und auf die Festplatte geschrieben werden. Es können Code-Snippets per Copy/Paste eingefügt werden.

- **F5: Code-Editor**
  Die Code-Editor Komponente besitzt Syntax-Highlighting.

Weitere Anforderungen an den Prototypen sind die Umsetzung als Webanwendung und der Datenaustausch über eine Echtzeit-P2P-Verbindung, um den Einsatz an Servertechnologie so gering wie möglich zu halten. Um für die Administration die Installation möglichst einfach zu gestalten, wird der erforderliche Signaling-Server als Docker-Container[6] zur Verfügung gestellt.

[6] Erläuterungen folgen in Kapitel 5.1.5.

4.2.2 Design-Ansatz

In Kapitel 3 wurden die Grundlagen der WebRTC Technologie erläutert, die eine gewisse Architektur vorgibt unabhängig vom Anwendungsfall. Im Folgenden wird auf Grundlage der Anforderungen aus dem vorangegangenen Kapitel ein Design der Anwendung entwickelt, das im Kapitel Architektur weiter ausgeführt wird. Das Gesamtsystem besteht fachlich aus mehreren Teilen, die ihre eigenen Verantwortlichkeiten haben, so dass jede Komponente ihre eigenen Anforderungen erfüllen muss.

Serverseitig ist der Einsatz eines Signaling-Servers nötig, der die Verbindungsdaten der RTCPeerConnection überträgt. Zudem wird ein STUN-Server für die Ermittlung der externen IP-Adresse benötigt, soweit die Applikation nicht in einem lokalen Netz ausgeliefert wird. Auf Clientseite ist eine Webapplikation zu entwickeln, die für die Funktionen des Code-Editors verantwortlich ist und möglichst wenig auf Backendstrukturen zurückgreifen soll. Unabhängig von der Umsetzung der Oberfläche ist der Aufbau der Kommunikation zwischen den Browsern, da es für die Implementierung der Oberfläche nicht relevant ist woher die Daten geladen werden, solange es eine definierte Schnittstelle gibt, die sie liefert.

Der modulare Aufbau, das Prinzip der Trennung der Verantwortlichkeiten (Separation of Concerns [siehe Sta18, S. 67]) hat den Vorteil, dass die Komponenten getrennt voneinander entwickelt, verändert und getestet werden können. Aus diesem Grund besteht das entwickelte Gesamtsystem aus drei Einzelprojekten. Es wurde kein eigener STUN/TURN-Server umgesetzt, da er den Verbindungsaufbau unterstützt und keinen Einfluss auf die Umsetzung der Anforderungen nimmt.

CoSocketSignaling
CoSocketSignaling implementiert den Signaling-Server. Er basiert auf Websocket-Technologie und erlaubt eine bidirektionale Verbindung zur CoWebRTC-Bibliothek herzustellen. Der Server übernimmt nicht nur das Signaling, er verwaltet auch den Session-Status für unterschiedliche Benutzer.

CoWebRTC
CoWebRTC wird als Bibliothek zur Verfügung gestellt. Sie ist für den Aufbau und Abbau der RTCPeerConnection zuständig und stellt Funktionen bereit um Nachrichten über einen zuverlässigen RTCDataChannel versenden.

CoCoding

Die Web-Applikation CoCoding implementiert den Code-Editor zum kollaborativen Bearbeiten des Quellcodes, beinhaltet den Textchat und verfügt über Funktionen um Daten zu importieren und exportieren. Sie wird als sogenannte Single-Page Application umgesetzt, d.h. sie besteht aus einer einzigen Webseite, die ihre Daten selbst verwaltet und ihre Abläufe durch JavaScript steuert.

4.2.3 Anforderungsdefinition

Die folgenden Tabellen spezifizieren die Anforderungen an die einzelnen Komponenten genauer und ordnen sie den definierten Features aus Abschnitt 4.2.1 zu. Zusätzlich zu den Resultaten aus der Kontextanalyse beinhalten die Tabellen auch Ergebnisse aus den Validierungen während des Entwicklungsprozesses, die in Kapitel 7.2 beschrieben werden. Diese Punkte sind durch einen Stern gekennzeichnet. Sie beschreiben die Erwartung an das Verhalten und Qualität der Applikation, erweitert aber die Minimal-Features nicht.

CoSocketSignaling

C1	Ein Client kann eine WebSocket-Session öffnen/schließen	F1
C2	Ein zweiter Client kann der Session beitreten.	F1
C3	Ein dritter Client kann der Session nicht beitreten.	F1
C4	Es können Nachrichten über einen Signaling-Channel gesendet werden.	F1
C5	Wenn ein Client die Session verlässt wird der erste Client benachrichtigt.	F1

Tabelle 4.1: Anforderungsdefinition CoSocketSignaling

CoWebRTC

B1	Öffnen und Schließen der Verbindung über einen Signaling-Server.	F1
B2	Verbinden zweier Clients über einen Signaling-Server.	F1
B3	Verlassen der Verbindung wird gemeldet.	F1
B4	Öffnen eines RTCDataChannels zwischen den Clients.	F2,F3
B5	Senden und Empfangen von Nachrichten über den DataChannel.	F2,F3
B6	Funktionen (sessionStart, sessionStop, onMessage, send) sind implementiert.	F1-F3
B7	Verbindungsaufbau über STUN/TURN Server.	F1

Tabelle 4.2: Anforderungsdefinition CoWebRTC

CoCoding

A1	Eine Session kann über einen Button gestartet werden.	F1
A2	Eine Session kann über einen Button beendet werden.	F1
A3	Eine URL zum Einladen wird über den Aufruf eines Dialogs angezeigt.	F1
A4	Über die Eingabe der generierten URL kann der Session beigetreten werden.	F1
A5	Der Text kann über einen Dialog importiert werden.	F4
A6	Der Text kann über einen Dialog exportiert werden.	F4
A7	Code-Snippets können per Copy/Paste importiert werden.	F4
A8	Über ein Steuerelement kann der Initiator die Schreibrechte erteilen/nehmen.	F2
A9	Die Veränderung des Inhalts im Editor wird synchronisiert.	F2
A10	Die Veränderung des Inhalts im Editor werden farblich markiert.	F2
A11	Der Editor besitzt Syntax-Highlighting für Sprachen (Texttypen).	F5
A12	Der Text im Chat wird übertragen und angezeigt.	F3
A13	Es wird angezeigt, wenn der Gesprächspartner die Session verlässt.	F1
A14*	Die Sprache des Editors kann ausgewählt werden.	F5
A15*	Der Status der Session wird angezeigt.	F1
A16*	Die Oberfläche zeigt an, ob der Benutzer gerade Schreibrechte besitzt.	F2
A17*	Die Session des Initiators bleibt geöffnet, wenn der Co-Worker diese verlässt.	F1
A18*	Die Reaktionszeit einer Antwort des Peers sollte unter 50ms liegen.	F1

4.2.4 Bedienkonzept

Die Applikation besteht nicht nur technisch gesehen aus einer Seite, sie ist vom Designbild her auch als eine Seite konzipiert. Zentrale Komponente ist der Editor (F5), der immer sichtbar ist. Die wichtigsten Funktionen sind über eine Toolbar erreichbar. Die Bearbeitungsfreigabe lässt sich über ein Slider-Element an- und ausschalten (F2). Der Chat (F3) lässt sich über einen Button ein- und ausblenden, da er unter Umständen wenig genutzt wird und deshalb nicht dauerhaft sichtbar sein muss. Die zentralen Funktionen wie das Öffnen und Schließen sowie das Teilen einer Session (F1) sind in der Toolbar direkt erreichbar und sind entweder deaktiviert oder unsichtbar wenn sie nicht verwendet werden können oder sollen. Wurde die Verbindung geöffnet, wird der <Neue Session> Button durch einen <Schließe Session> Button ausgetauscht. Die Dialoge, die das Importieren und Exportieren von Dateien (F4)

ermöglichen, sind ebenfalls direkt in den Toolbar angeordnet. Eine ComboBox enthält eine Auswahl an Sprachen, mit der sich die Sprache des Editors umschalten lässt.

Die visuelle Gestaltung des Editors steht nicht im Fokus dieser Arbeit. Allerdings trägt ein einheitliches und aufgeräumtes Gesamtbild dazu bei, dass der Benutzer sich besser orientieren und die Applikation ohne Anleitung bedienen kann. Aus diesem Grund wurden für die Gestaltung der Oberfläche *Material Design* Komponenten von Google verwendet [Mat], da sie sich an etablierten Gestaltungsvorgaben orientieren. Ergänzend existiert eine Einstiegsseite, die den Editor kurz erklärt. Sie hat sonst keine Funktion, alle Steuerelemente in der Toolbar sind deaktiviert. Ein Button führt auf die Seite des Editors.

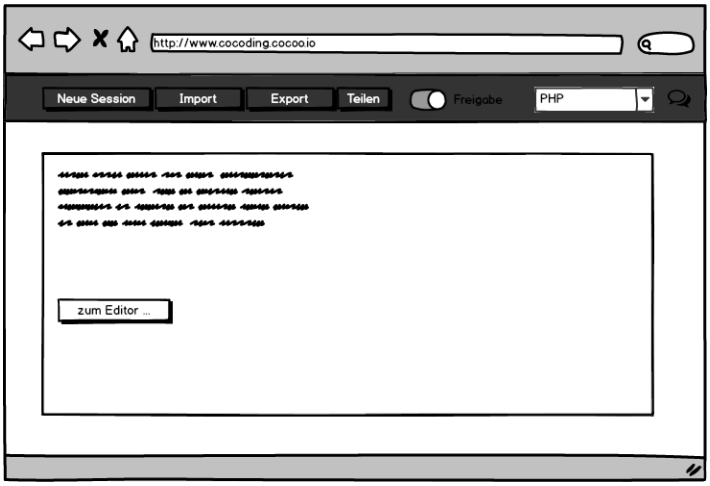

Abb. 4.1: Wireframe der Einstiegsseite

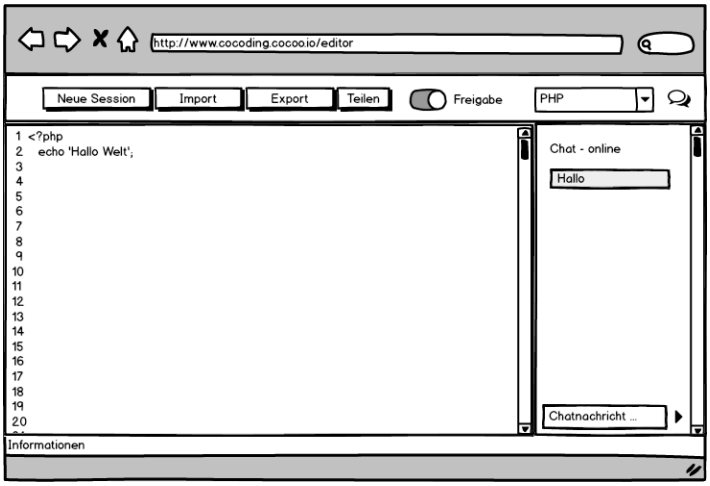

Abb. 4.2: Wireframe des Editors

Kapitel 5

Architektur

Wie in Abschnitt 4.2.2 schon eingeleitet wurde, besteht das Gesamtsystem aus drei Einzelprojekten. Im Kapitel Systemarchitektur werden die einzelnen Komponenten kurz hinsichtlich ihrer Aufgaben und ihrer Rolle im Gesamtsystem betrachtet und die benötigte Infrastruktur erläutert. Das Kapitel Softwarearchitektur gibt einen tieferen Einblick in die Strukturen innerhalb der einzelnen Projekte.

5.1 Systemarchitektur

5.1.1 CoSocketSignaling

JavaScript ist eine höhere Programmiersprache, objektorientiert, aber klassenlos, in der sich objektorientiert, prozedural oder funktional programmieren lässt. Sie benötigt einen Interpreter, der in Browsern als Laufzeitumgebung zur Verfügung steht. JavaScript steht mittlerweile aber auch serverseitig zur Verfügung. Für die Implementierung des Signaling-Servers kommt die Bibliothek Socket.io zu Einsatz, mit der Echtzeitanwendungen unter Node[1] entwickelt werden können. Node basiert auf Googles V8 Javascript Engine und ermöglicht es serverseitige Applikationen zu entwickeln. Es arbeitet nachrichtenbasiert und ist eine speziell für asynchrone Applikationen ausgelegte Engine, d.h. für die Entwicklung eines WebSockets-Servers gut geeignet [GIN15, S. 109-110].

Socket.io stellt eine Reihe zusätzlicher Funktionen zur Verfügung, wie z.B. das Verwalten von Räumen und das Überwachen von Verbindungsabbrüchen. Ein wichtiger Punkt der Anforderungsanalyse war es zu erkennen, wann die Verbindung zum Gesprächspartner abbricht. Ein Signaling-Server auf Socket.io-Basis kann dies von Natur aus erkennen. Es ist nachvollziehbar wann ein Client einen Raum betritt und wann er ihn wieder verlässt. So kann der Server darüber informieren wie viele Mitglieder sich in einem Raum befinden und

[1] https://nodejs.org/

kann die Anzahl der Teilnehmer limitieren. Zudem ist er in der Lage Sessions über IDs zu verwalten. So kann in der Anwendung in der URL eine ID Übergeben werden, die direkt als Raumname übernommen werden kann. Socket.io Lösungen werden in vielen Artikeln und Tutorials verwendet, so dass viel Erfahrung in diesem Bereich vorhanden ist. Für die Entwicklung des Frontends und der Bibliothek ist JavaScript zwingend, für die Serverseite ist die Entwicklung in Node eine sinnvolle Lösung, um die initiale Verbindung aufzubauen. Alle Komponenten sind so in einer einheitlichen Programmiersprache entwickelt und greifen auf die gleichen standardisierten Pakete und Schnittstellen zurück [Spr16, S. 31]. Vorlage für die Implementierung des CoSocketSignaling-Servers ist die Umsetzung eines Servers in den Google Codelabs [Cod], die angepasst und erweitert wurden.

5.1.2 CoWebRTC

Die WebRTC Komponente ist eine JavaScript-Bibliothek, die für die WebRTC-basierte Kommunikation zwischen den Browsern zuständig ist, d.h. unabhängig von der Implementierung der Oberfläche und dem gewählten Framework steuert sie die Aktivitäten zwischen den Browsern. Dies beinhaltet die Initialisierung der RTCPeerConnection und des RTCDataChannels als auch den initialen Verbindungsaufbau über den CoSocketSignaling- und einen STUN/TURN-Server. Nach außen kapselt sie ihre Funktionalität ab und stellt eine einfache Schnittstelle zur Verfügung um eine Session zu erzeugen und beenden zu können und um Nachrichten zu senden und zu empfangen. Zusätzlich benachrichtigt sie die Webapplikation über Änderungen im Verbindungsstatus und den versendeten Nachrichten. Abb. 5.1 zeigt wie CoWebRTC als Schnittstelle zwischen der Oberfläche CoCoding und den Servern fungiert.

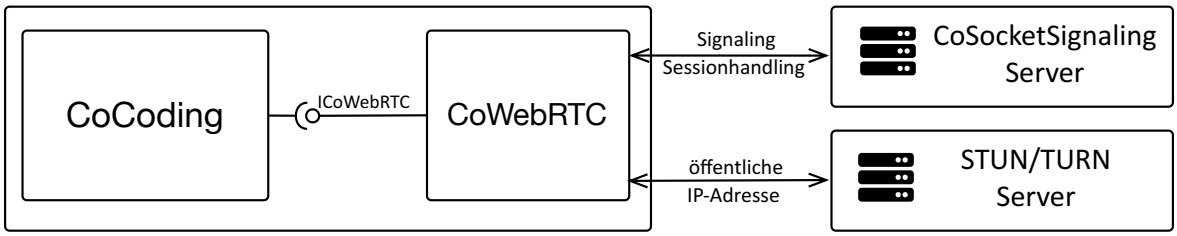

Abb. 5.1: Einordnung der CoWebRTC-Komponente

Bereits vorhandene WebRTC-Lösungen wie EasyRTC[2], SimpleWebRTC[3], spezialisiert auf Audio- und Videoanwendungen, oder auch PeerJS[4] sind sehr umfangreich und geben die Architektur und die Technologie der erforderlichen Server und zum Teil auch des Clients vor. Zudem enthalten sie eine Reihe Optimierungen und Fallunterscheidungen und sind über die Jahre mit der Entwicklung der WebRTC-Technologie gewachsen. Nur eine eigene Umsetzung der WebRTC-Funktionalität gibt die Möglichkeit die Implementierung den Anforderungen entsprechend einfach und ohne Optimierungen umzusetzen, um die Verwendbarkeit für den Anwendungsfall zu prüfen und bewerten zu können.

5.1.3 CoCoding

CoCoding implementiert die Web-Applikation, die wie in Abschnitt 4.2.2 schon erwähnt, als Single-Page Application konzipiert ist. Eine SPA zeichnet sich dadurch aus, dass sie aus einer einzigen Webseite besteht, in der alle Abläufe durch JavaScript gesteuert, der Aufbau durch HTML strukturiert und das Design durch CSS beschrieben wird. Sie verwaltet ihre Daten lokal und implementiert die vollständige Business-Logik, d.h. wenn die Applikation einmal geladen wurde, ist sie vollständig vorhanden und verändert nur die Teile der Oberfläche, die notwendig sind. Ihre Daten bezieht sie über AJAX, WebSockets oder wie in dieser Arbeit über den RTCDataChannel, der von CoWebRTC zur Verfügung gestellt wird, so dass keine weiteren Backend-Server notwendig sind. Der Vorteil einer SPA ist, dass sie sich im Grunde verhält wie eine Desktop-Applikation, universell über einen Webserver erreichbar ist aber nicht installiert werden muss [MP14, S. 20-21].

Eine SPA stellt einige Anforderungen an die Struktur der Applikation. Der Speichern des aktuellen Zustands der Seite, das Routing, Datenhaltung, sowie das Rendern der Oberfläche muss clientseitig implementiert werden und erfordert deshalb für eine strukturierte Umsetzung ein Framework. In dieser Arbeit wird Angular.io[5] eingesetzt. In Kapitel 5.2.5 wird auf die Software-Architektur, die durch den Einsatz von Angular vorgegeben ist, eingegangen.

[2] https://easyrtc.com/
[3] https://www.simplewebrtc.com/
[4] https://peerjs.com/
[5] https://angular.io

5.1.4 Zusammenarbeit

Die Kommunikation zwischen CoCoding und CoWebRTC ist ereignisbasiert umgesetzt, im Speziellen wird auf das Observable Pattern zurückgegriffen, auch Publisher-Subscriber-Pattern genannt. Die Idee dahinter ist, dass eine Komponente Daten publiziert und andere Komponenten sie abonnieren können, d.h. es wird nicht regelmäßig angefragt ob neue Daten vorhanden sind, sondern die Daten werden ausgeliefert [Sta18, S. 124]. Sowohl CowebRTC als auch Angular in CoCoding setzen dieses Konzept auf Grundlage der RXJS-Bibliothek[6] um.

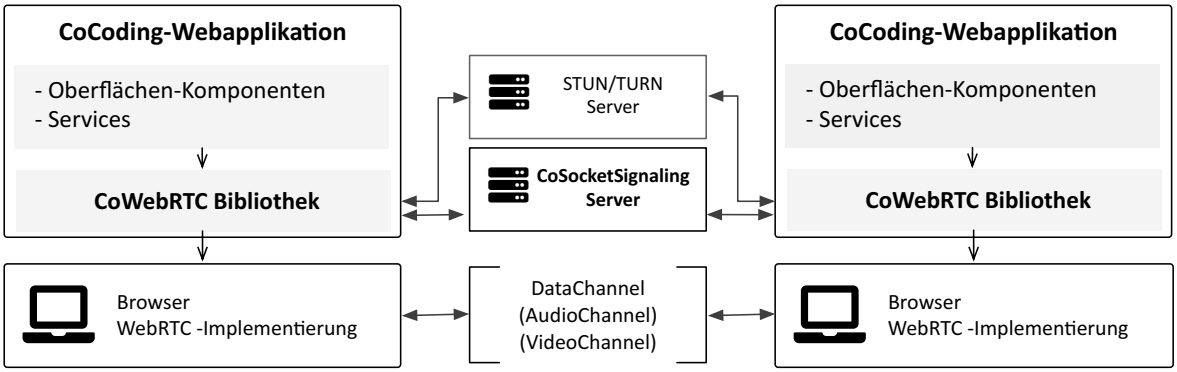

Abb. 5.2: Aufbau des Gesamtsystems

Da CoSocketSignaling die serverseitige Implementierung von Socket.io nutzt, verwendet CoWebRTC die passende clientseitige Implementierung der Bibliothek. In der Kommunikation zwischen Bibliothek und Signaling-Server werden definierte Nachrichten erwartet und versandt, d.h. die Komponenten sind aufeinander abgestimmt. Während die Webapplikation einfach austauschbar ist, wäre zum Austausch der Signaling-Servers eine Erweiterung innerhalb der CoWebRTC Bibliothek notwendig, die andere Kommunikationswege unterstützt. Abbildung 5.2 zeigt den Aufbau des Gesamtsystems.

[6] https://rxjs-dev.firebaseapp.com/

5.1.5 Infrastruktur

Client

Die CoCoding-App, die CoWebRTC beinhaltet, ist komprimiert ca. 4MB groß. Für die Auslieferung der Applikation ist jeder Webserver geeignet, wie bspw. nginx[7] oder Apache Webserver[8]. Da keine serverseitigen Skripte ausgeführt werden, ist es auch möglich die Applikation in Containern zu hosten, die statische Webseiten ausliefern können wie z.B. ein S3-Bucket von AWS[9]. Allerdings sollte der Server so konfiguriert werden, dass er das Routing der Applikation selbst überlässt. Denn es existiert nur eine *index.html* als Startdatei, der Pfad in ein Unterverzeichnis existiert praktisch nicht[10].

Server

Der Kern von Node bildet nicht nur die V8-Engine, einige Bibliotheken sind systemspezifisch in C++ implementiert. Während die CoCoding-App einfach auf einen Webserver kopiert und von dort aus ausgeliefert werden kann, ist dies mit einer Node-Anwendung wie CoSocket-Signaling nur möglich, wenn sie nicht auf systemspezifische Bibliotheken zurückgreift, d.h. die Installation der Module muss in diesem Fall auf dem Zielsystem erfolgen. Um die Auslieferung zu erleichtern, enthält das Projekt ein Dockerfile, mit dem ein Docker-Container [dock18] erstellt werden kann. Es steht zusätzlich ein Docker-Image auf der beiliegenden DVD bereit, aus dem ein Container als Instanz gestartet werden kann. Der STUN/TURN-Server ist keine Eigenentwicklung, es wird eine eigene Instanz des Coturn-Servers genutzt.

Sicherheitsaspekte

Im Hinblick auf eine spätere Veröffentlichung des Projekts, ist es aus Sicherheitsgründen empfehlenswert die Auslieferung der Web-Applikation zu verschlüsseln und sich zusätzlich beim Signaling-Server zu authentifizieren. Die Datenverbindung über den RTCDatachannel ist zwar verschlüsselt, aber der initiale Verbindungsaufbau nicht, so dass dieser von dritter Seite verändert werden könnte. Zudem ist die Konfiguration entsprechender CORS-Regeln (Cross-Origin Resource Sharing, [siehe Moza]) auf dem Webserver notwendig, um den Zugriff auf den Signaling-Server als externe Ressource, falls er auf einem anderen Server installiert wurde als die Applikation, zu ermöglichen.

[7] https://www.nginx.com/
[8] https://httpd.apache.org/
[9] Amazon Web Services: https://aws.amazon.com/de/
[10] Wie bspw. Nginx oder der Apache Webserver konfiguriert werden können, ist in der README.md Datei des CoCoding-Projekts erklärt.

5.2 Softwarearchitektur

5.2.1 Modularisierung

Solange JavaScript in einer Seite nur für kleinere Aufgaben, wie das Prüfen von Eingaben, zuständig ist, ist der globale Namensraum, in dem gearbeitet wird, kein Problem. In größeren Anwendungen ist es notwendig den Quellcode zu strukturieren und zu modularisieren, auch im Hinblick auf den Einsatz von externen Bibliotheken, wie Socket.io.

JavaScript unterstützt erst mit Version ES2015 Module (ECMAScript Modules, ESM) und der Standard ist noch nicht in allen Browsern einheitlich implementiert [ecma; can]. Um Module und Abhängigkeiten zu definieren, haben sich im Laufe der Zeit verschiedene Lösungsansätze entwickelt. Zu nennen ist da z.B. das Module-Entwurfsmuster, das die Kapselung von Daten bspw. über Closures und IIFE[11] vornimmt [Ack16, S. 741]. Um den Umgang zusätzlich mit externen Modulen zu vereinfachen haben sich einige Modulsysteme etabliert, wie AMD[12], das die Definition von Modulen in Browsern ermöglicht und CommonJS[13], das für die Entwicklung auf Serverseite konzipiert wurde. Um Module zu schaffen, die universell einsetzbar sind existiert das Universal Module Definition-Pattern [umd], das in allen Systemen funktioniert, aber größere Dateien hervorbringt [Cal14].

In CoSocketSignaling ist der Einsatz von Modulen durch Node gegeben, das CommonJS verwendet. CoCoding verwendet, vorgegeben durch das Angular-Framework ein eigenes Modulsystem namens NgModules[14]. Um die CoWebRTC-Bibliothek universell verwenden zu können, auch auf Webseiten ohne Modulsystem, ist sie als UMD-Modul konzipiert, die als eine zusammenhängende Datei, zusammen mit den Modulen von denen sie abhängt, eingebunden werden kann.

[11] Immediately-Invoked Function Expression: Ein Funktionsausdruck, der sofort ausgeführt wird und so einen eigenen Scope erzeugt.
[12] Asynchronous Module Definition
[13] http://www.commonjs.org/specs/modules/1.0/
[14] https://angular.io/guide/architecture-modules

5.2.2 Typescript

Da die Umsetzung von ES2015 in den Browsern unterschiedlich schnell umgesetzt wird, hat es sich durchgesetzt entweder in einer ECMAScript Version seiner Wahl zu implementieren und das Ergebnis durch einen Transpiler, wie Babel[15], in eine frühere Version zu konvertieren oder man nutzt eine andere Programmiersprache und transpiliert zu einer JavaScript-Version seiner Wahl. In dieser Arbeit fällt die Wahl auf letzteres, speziell auf TypeScript[16], d.h. der Quellcode ist in TypeScript implementiert und wird in JavaScript transpiliert[17], wobei angegeben werden kann in welche Version übersetzt werden soll. Es enthält Sprachkonstrukte wie z.B. Klassen, Vererbung, Interfaces, Module, anonyme Funktionen und bietet Generics an, also die Konstrukte, die in modernen Programmiersprachen zu finden sind und das Modularisieren vereinfacht. Zudem ergänzt TypeScript JavaScript um eine statische Typisierung, d.h. Entwicklungsumgebungen können Autovervollständigung anbieten und auf Fehler bereits während der Entwicklung reagieren. Aus diesem Grunde es es möglich die Diagramme an die UML-Notation anzulehnen, die die Strukturen im Typescript-Quellcode direkt abbilden, während der übersetzte JavaScript-Code dies nicht mehr ohne Weiteres erkennen lässt.

5.2.3 CoSocketSignaling

Der Signaling-Server ist einfach gehalten und implementiert die Kommunikation in einer zentralen Klasse. Abb. 5.3 zeigt den Aufbau. Das index-Startscript initialisiert alle externen Module und Helferklassen, liest die Konfiguration ein und übergibt sie als Parameter an die Klasse *CoSocket-Signaling*. *Socket.io* und *http* sind als externe Bibliotheken eingebunden, *config* enthält die Konfiguration und *CoLogging* ist eine Helferklasse für das Protokollieren der Meldungen in die Konsole oder in eine Datei. Die möglichen Nachrichten-Tags die gesendet oder empfangen werden können

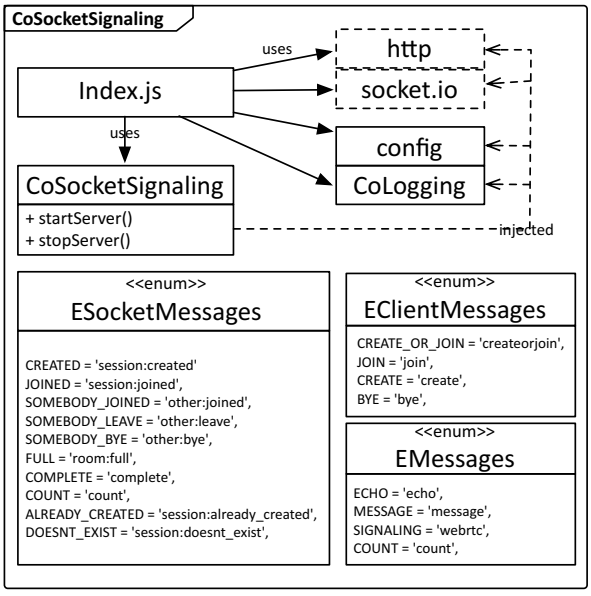

Abb. 5.3: CoSocketSignaling Struktur

[15] https://babeljs.io/
[16] https://www.typescriptlang.org/
[17] Siehe dazu: Compiling vs Transpiling, [Fen12]

sind in drei Enumerations definiert. *EClientMessages* beschreibt die Anfragen, die von einem Client an den Server gesendet werden können. *ESocketMessages* beschreiben die Antworten, die vom Server je nach Status des Channels zurückgegeben werden. *EMessages* sind Nachrichten, die vom Server mit dem gleichen Tag beantwortet werden, wie sie angefragt worden sind. Hier nicht aufgelistet sind Nachrichten, die zwischen der Client- und Serverimplementierung der Socket.io-Bibliothek ausgetauscht werden und sich auf den Verbindungstatus (wie connect/disconnect) beziehen.

Der Server horcht auf Nachrichten, die über Socket.io gesendet werden, verarbeitet sie und sendet eine Antwort an den oder die entsprechenden Adressaten. Listing 5.1 zeigt die entsprechende Implementierung in CoSocketSignaling. Hier nimmt er die Signaling-Anfrage an und leitet sie an den zweiten Client im Raum weiter.

```
1 socket.on(EMessages.SIGNALING, (data: any) => {
2  this.sendSignalingToRoomWithoutSender(socket, EMessages.SIGNALING,
      data.room, data.data);
3 });
```

Listing 5.1: CoSocketSignaling: Signaling-Message

5.2.4 CoWebRTC

Einstiegspunkt der Bibliothek ist ebenfalls eine index-Datei, die alle Objekte instanziiert und initialisiert und sie an die jeweils abhängige Klasse als Parameter übergibt, d.h. keine Klasse instanziiert eine Klasse von der sie abhängt selbst, sondern bekommt sie als Referenz übergeben. Dieses Prinzip der Dependency Injection [Fow18] hat den Vorteil, dass der Code unabhängiger wird, da von außen bestimmt werden kann, welche Art von Objekt übergeben wird. Wichtig ist das Konzept bei der Umsetzung der UnitTests, da anstelle echter Objekte Mocks übergeben werden und so Quellcode unabhängig getestet werden kann.

Abb. 5.4 zeigt die wichtigsten Klassen, die an der WebRTC-Kommunikation beteiligt sind. Die Klasse *CoSocketService* ist für die Socket.io-Kommunikation mit dem CoSocket-Signaling-Server zuständig und leitet alle Events an die Klasse *CoConnection* weiter. *CoPeerConnection* übernimmt alle Funktionen, die mit dem Aufbau der RTCPeerConnection und des RTCDataChannels zu tun haben. *CoConnection* koordiniert die Prozesse und sorgt dafür, dass der Signaling-Server zum Aufbau der Verbindung genutzt werden kann. Zudem leitet die Klasse Nachrichten an den Client weiter, so dass die Oberfläche auf den Verbindungsstatus reagieren kann. Welche Nachrichten erwartet werden können, beinhalten

die Enumerations *ESessionMessages* und *EDataChannelStatus*. Hilfsklassen und externe Module, wie *RXJS*, *Socket.io* oder die benötigte Polyfill-Datei *webrtc-adapter* sind nicht eingetragen. Die Klassen *SocketBuilder* und *PeerBuilder* generieren das Socket.io Objekt und das RTCPeerConnection-Objekt.

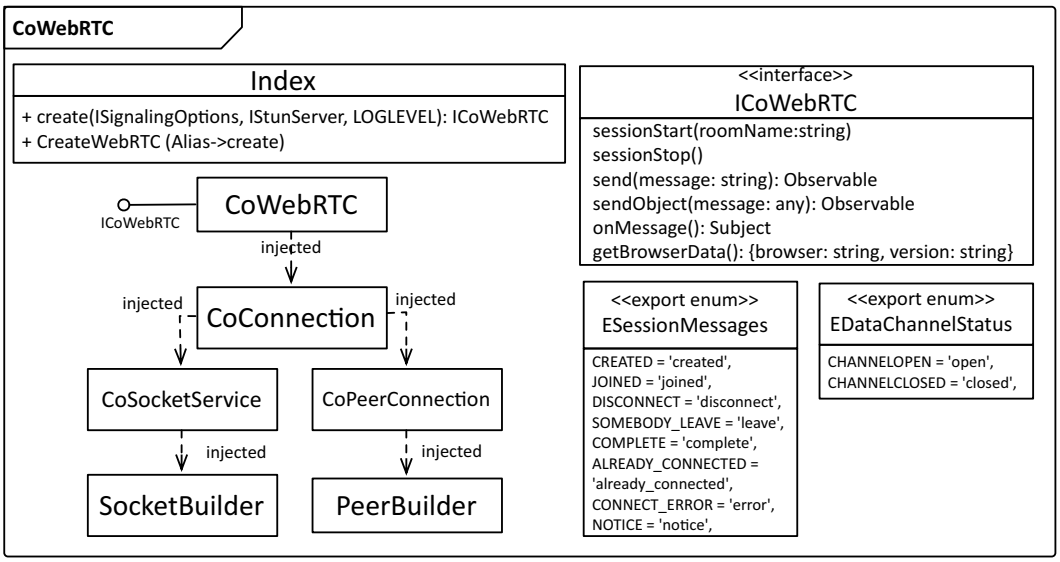

Abb. 5.4: CoWebRTC Struktur

Als Ausblick wäre es denkbar andere Signaling-Varianten in die Struktur zu integrieren. Da lediglich die Klasse *CoSocketService* mit den Nachrichten des CoSocketSignaling-Servers in Kontakt kommt, wäre es möglich an dieser Stelle eine andere Klasse zu verwenden, die von der index-Datei bereitgestellt wird. Funktionen bzgl. des Session-Handlings müssten dann von der *CoConnection* Klasse implementiert werden, d.h sie müsste bspw. selbst ein regelmäßiges Signal über den RTCDataChannel senden, um zu überprüfen, ob die Verbindung noch besteht (keep-alive).

Listing 5.2 zeigt das Erzeugen des CoWebRTC-Objekts durch die Funktion *CreateWebRTC*, sowie das Öffnen einer Session und das Abonnieren der Daten des RTCDataChannels, wie es in einer TypeScript Umgebung wie Angular implementiert werden kann. Zwingend ist die Übergabe der Daten des Signaling-Servers. STUN/TURN-Server und die Angabe wie ausführlich CoWebRTC Aktivitäten loggen soll sind optional. Da TypeScript das Konzept der Module aus ECMAScript 2015 übernommen hat, können Module und Enumerations der Bibliothek importiert werden. Listing 5.3 zeigt den Aufruf, wie er direkt in einer HTML-Seite

verwendet werden kann. Ein UMD-Modul besitzt einen Bibliotheksnamen, das es innerhalb der Browserumgebung eindeutig identifiziert.

```
import { CreateWebRTC, ESessionMessages, EDataChannelStatus} from '
    cowebrtc.umd.js';
let cowebrtc = CreateWebRTC(signaling, stunserver, log);
let roomName = 'Testraum';
cowebrtc.sessionStart(roomName).subscribe({
 next: (data) => {
   switch (data.tag) {
    case EDataChannelStatus.CHANNELOPEN:
      this.messageSubscription = this.webrtc.onMessage().subscribe({
       next: (text) => {
     // text beinhaltet die Daten aus dem RTCDataChannel
       }
     });
     break;
   }},
 error: (error) => {  // ... },
 complete: () => { // ... }
});
```

Listing 5.2: CoWebRTC - Import in Angular

```
<script src='/lib/cowebrtc.umd.js'></script>
<script>
 let cowebrtc = CoWebRTC.create(signaling, stunserver, log);
 let roomName = 'Testraum';
 cowebrtc.sessionStart(roomName).subscribe({
  // ... s.o
 });
</script>
```

Listing 5.3: CoWebRTC - Import in HTML

5.2.5 CoCoding

Angular ist ein komplexes Framework, das sich um den Zugriff auf DOM-Elemente im Browser kümmert, Module und Abhängigkeiten verwaltet, eine Testumgebung beinhaltet und Prozesse für das Ausliefern der Applikation bereitstellt, d.h. das Framework gibt die Architektur der Anwendung maßgeblich vor. Auf Angular selbst soll hier in soweit eingegangen werden, wie es für das Verständnis der Abläufe notwendig ist. Für die Tauglichkeit der WebRTC-Technologie spielt die Auswahl des Frameworks nur eine nachgelagerte Rolle. Allerdings spielt es eine wichtige Rolle, wenn es darum geht eine Anwendung zu konzipieren, die angemessen auf Nutzereingaben reagiert und mit asynchronen Nachrichten aus dem CoWebRTC-Framework umzugehen weiß. Zudem sollten Abläufe testbar und die Applikationsstruktur einen festen Rahmen vorgeben, in der Code, Daten und Design getrennt verwaltet werden[18].

Module

Module bilden in Angular fachlich zusammengehörende Strukturen oder Workflows ab. Jede Angular Applikation besitzt mindestens ein Modul, das Root Module (*AppModule*), das den Bootstrap-Mechanismus zum Starten der App zur Verfügung stellt und andere funktionale Module beinhaltet [ang18]. CoCoding besitzt drei Module: *AppModule* als Root Module, *AppRoutingModule* für das Routing und *MaterialModule*, das alle Module beinhaltet, die zur Darstellung des Material Designs notwendig sind. Neben einer Vielzahl von Angular-Modulen und MaterialDesign-Modulen, die für die Darstellung der Steuerelemente innerhalb des Browsers zuständig sind, verwendet CoCoding externe Module:

- *MonacoEditorModule* von Kumar [Kum18] integriert den Monaco-Code-Editor[19] von Microsoft in die Modulstruktur von Angular. Monaco ist eine umfangreiche Editor-Lösung für den Browser, die z.B. von Natur aus Syntax-Highlighting mitbringt und stetig um neue Funktionen erweitert wird[20].
- *ClipboardModule* von Lin [Lin18] integriert die JavaScript-Bibliothek clipboard.js[21] in Angular, mit der Text in die Zwischenablage kopiert werden kann. Der generierte URL-Link kann so in die Zwischenablage kopiert und versendet werden.

[18] Vue.js (https://vuejs.org) oder React (https://reactjs.org/) wären eine Alternative, auch wenn sie nicht so umfangreich sind wie Angular (siehe [comp18]).
[19] https://microsoft.github.io/monaco-editor/
[20] Auch die Editoren CodeMirror (https://codemirror.net/) oder Ace (https://ace.c9.io/) könnten alternativ eingesetzt werden.
[21] https://clipboardjs.com/

Komponenten

Angular ist ein Komponenten-basiertes MVC-Framework[22], d.h. eine Applikation besteht aus einem hierarchischen Baum von Komponenten, die Bestandteile der Oberfläche repräsentieren. Komponenten bestehen aus einem Template, das die Oberflächenstruktur definiert und die Daten präsentiert sowie einer Komponenten-Klasse, die die Oberfläche steuert. Style-Sheets lassen sich für jede Komponente separat definieren. Komponenten definieren *Views*, die aus einer Reihe Oberflächenelementen bestehen. Die Verbindung zwischen Applikationslogik und Darstellung wird über sogenannte Property-Bindings und Event-Bindings hergestellt, d.h. eine Komponente kann Werte in eine untergeordnete Komponente über sogenannte Input-Bindings hineinreichen (als Parameter), die untergeordnete Komponente kann ihre Eltern-Komponenten über Output-Bindings (Events) informieren [ang18]. Außerdem können übergeordnete Komponenten direkt auf Funktionen und Eigenschaften ihrer Elemente zugreifen, indem sie über den sogenannten ViewChild-Decorator auf sie zugreifen. Jede Applikation enthält mindestens eine Komponente, unter die alle anderen einsortiert sind.

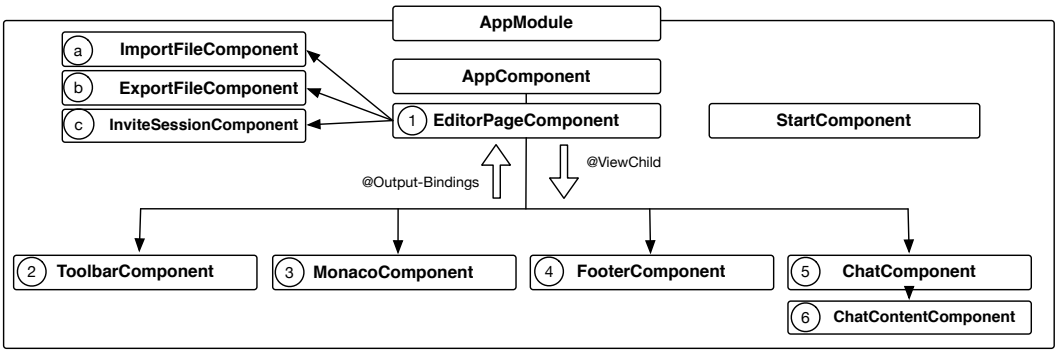

Abb. 5.5: CoCoding: Hauptkomponenten der App

Abb. 5.5 zeigt die Hauptkomponenten der Applikation und ihre Abhängigkeiten. Abb. 5.6 zeigt die verbundenen Elemente auf der Oberfläche. *AppComponent* ist die Root-Komponente und der Startpunkt der App. *EditorPageComponent* ist die zentrale Klasse der App, die alle Aktivitäten koordiniert. Sie nimmt Ereignisse aus den untergeordneten Komponenten über Event-Bindings entgegen, um Aktionen auszulösen. Sie ruft, wenn sich Änderungen ergeben, Funktionen in den untergeordneten Komponenten auf, um ihre Inhalte anzupassen.

[22]MVC: model-view-controller, oder in Angular auch MVVM: model-view-viewmodel

Bspw. wird der Klick auf den Button *Import* in der *ToolbarComponent* registriert, ein Event ausgelöst, das die *EditorPageComponent* entgegen nimmt und daraufhin das Dialogfenster zum Importieren von Text öffnet.

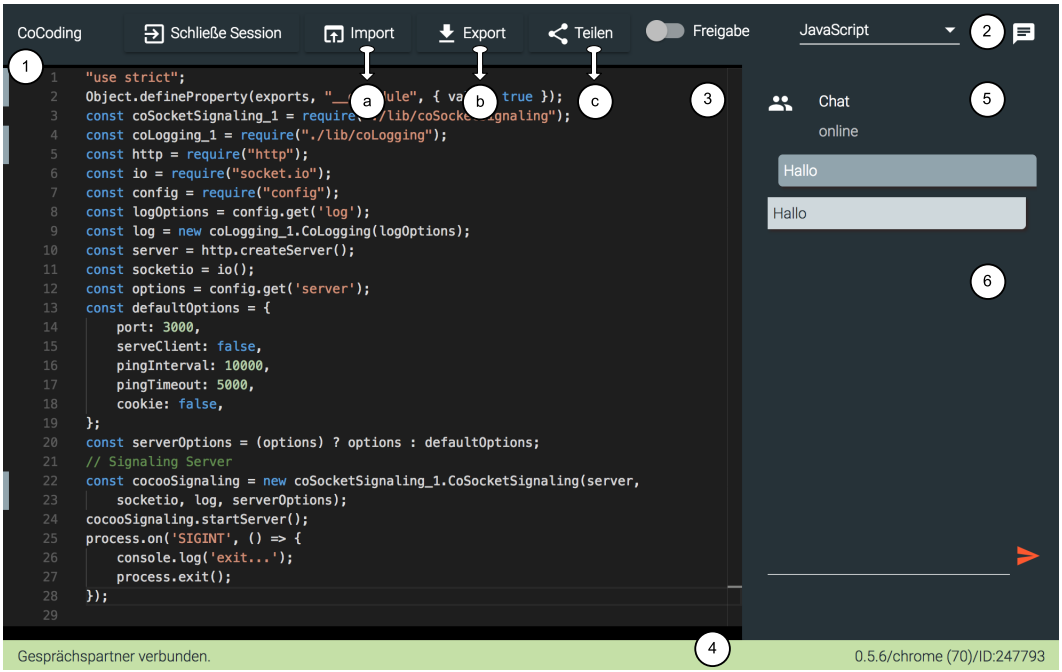

Abb. 5.6: CoCoding: GUI der Applikation

(a) Dialog: Import (b) Dialog: Export (c) Dialog: Teilen

Abb. 5.7: CoCoding: Dialoge

Die Dialogfenster (Abb. 5.7) werden nur von *EditorPageComponent* geöffnet und werden als Layer über dem Editor angezeigt. Die einzelnen Komponenten spiegeln mit ihren Verantwortlichkeiten innerhalb der Architektur die Anforderungen an die Applikation namentlich direkt wider.

Services

Daten und Logik, die nicht speziell mit einer *View* verbunden sind, werden von Service-Klassen zur Verfügung gestellt. Services sind ein Kernbestandteil der Architektur in Angular. Sie kapseln Applikationslogik und speichern Daten, die von einer oder mehreren Komponenten benötigt werden. Services werden von der App pro Modul einmalig instanziiert und per Dependency Injection durch den Konstruktor an die Komponente übergeben.

Der wichtigste Service ist der *WebRTCService*, da er die Kommunikation mit der CoWebRTC Bibliothek koordiniert. Die *EditorPageComponent* ist ein Subscriber der Nachrichten, die aus dem Service geliefert werden und gibt Daten entsprechend an andere Komponenten weiter oder nimmt Daten von anderen Komponenten, wie der *MonacoComponent* oder der *ChatComponent*, an und leitet sie an den *WebRTCService* weiter. Abb. 5.8 zeigt den Einsatz der Services in Bezug auf den Datenfluss und der Datenverwaltung innerhalb der Applikation.

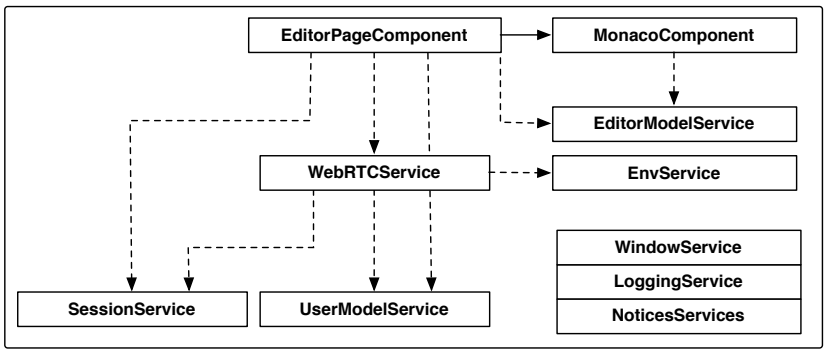

Abb. 5.8: CoCoding: Services

Der *SessionService* verwaltet den aktuellen Online-Status der App, der *UserModelService* verwaltet den Status des Benutzers, d.h. ob er der Besitzer der Session ist und ob er gerade Schreibrechte besitzt. Beide Services bieten die Möglichkeit Daten zu setzen und die Veränderungen über Observables zu abonnieren. So können Änderungen über die Grenzen des Komponentenbaums übermittelt werden. Der *EditorModelService* verwaltet die Daten des Monaco-Editors, um Änderungen im Datenfluss zu koordinieren. *EnvService* lädt die Umgebungsvariablen der Applikation, z.B. die URLs der Server. Der Übersichtlichkeit halber sind *LoggingService* und *NoticesService* in der Abbildung nicht mit den Komponenten verbunden, da sie von fast allen Komponenten zum Protokollieren in die Konsole und der Rückgabe von Statusmeldungen verwendet werden. *WindowService* abstrahiert den Zugriff auf das globale Windows Objekt im Browser.

Kapitel 6

Implementierung

Das Kapitel fokussiert sich auf drei Bereiche. Zum einen gibt es einen tieferen Einblick darüber wie die Kommunikation zwischen den Komponenten umgesetzt ist, zum anderen wird der Ablauf der Oberfläche näher erläutert. Zum Abschluss des Kapitels wird die Projektstruktur erklärt.

6.1 Kommunikation

6.1.1 Signaling-Prozess

Wie generell ein Signaling-Prozess abläuft wurde in Kapitel 3.3 erläutert. Da der CoSocket-Signaling-Server neben dem Signaling auch die Verwaltung der Session übernimmt, ist der Ablauf umfangreicher. Das Sequenzdiagramm in Abb.6.1, angelehnt an Loreto und Romano [LR14], beschreibt das Zusammenspiel zwischen CoWebRTC und dem Server.

Der Aufbau der Session beginnt mit einer *connect*-Anfrage an den Signaling-Server. Antwortet der Server mit einem *connect*, wird eine *createorjoin*-Anfrage an den Server gesendet. Der Server prüft wie viele Clients bereits im Raum sind. Existiert der Raum nicht, wird er geöffnet. Ist er bereits vorhanden, darf der Client eintreten. Sind bereits zwei Clients im Raum lehnt er die Anfrage ab. Sobald zwei Clients anwesend sind, wird eine *completed*-Nachricht an beide Teilnehmer versendet und der Signaling-Prozess kann beginnen. Der Client, der den Raum geöffnet hat, beginnt mit der Initiierung des Prozesses durch die Generierung der Offer. Wenn eine RTCSessionDescription gesetzt wurde, beginnt die Suche nach passenden P2P Kandidaten (ICE-Candidates). Dieser Prozess läuft parallel zum Austausch der SessionDescription ab (Abb. 6.2).

Ein Verlust der Verbindung oder ein manuelles Schließen der Session löst eine *disconnect*-Nachricht aus, die an die Oberfläche weitergereicht wird. Die Applikation hat die Möglichkeit entsprechend darauf zu reagieren.

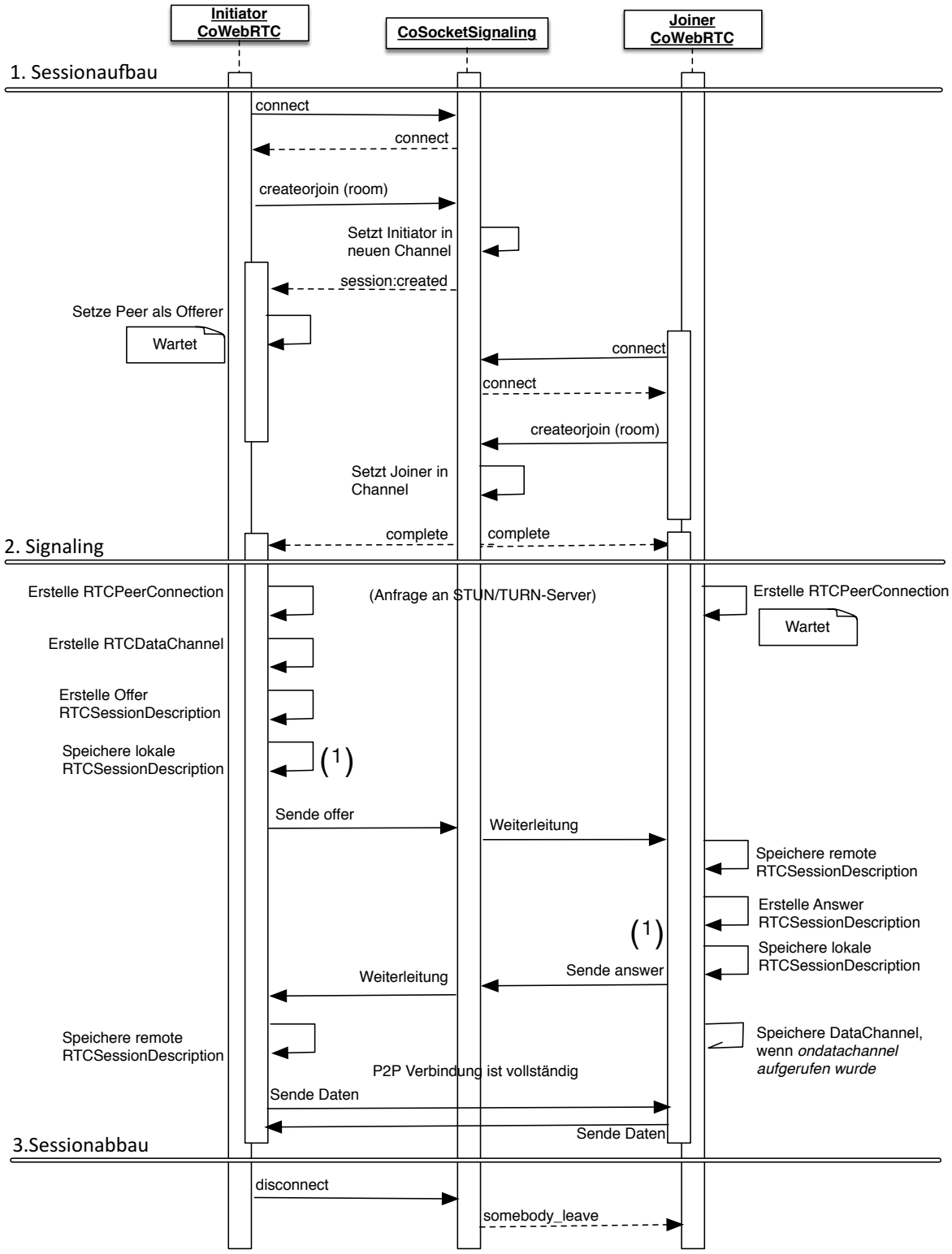

Abb. 6.1: WebRTC-Verbindung und Signalingprozess

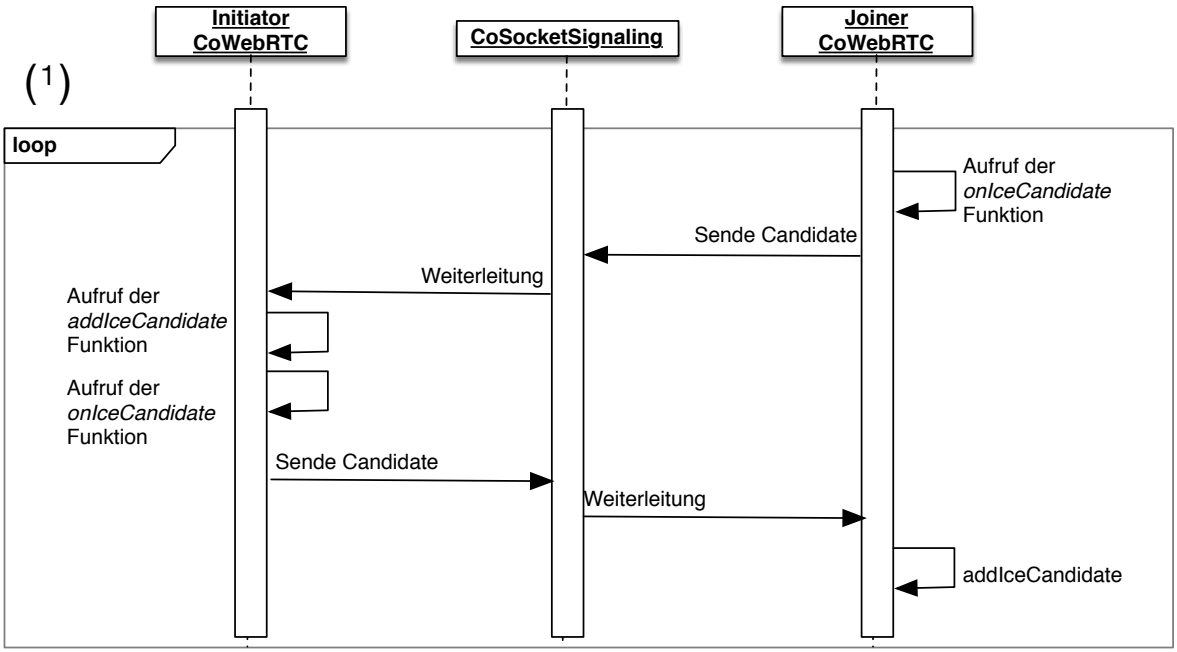

Abb. 6.2: ICE Candidates

6.1.2 WebRTC-Verbindung

Öffnen der Verbindung

Die Implementierung der WebRTC-Verbindung orientiert sich an den Umsetzungen der Google Codelabs [Cod] und Loreto und Romano [LR14]. Listings 6.1-6.5 zeigen, wie in der Klasse CoPeerConnection das Signaling und der Aufbau der WebRTC Verbindung zusammenhängen[1]. Die Abstraktionsebene der WebRTC-APIs ist hoch, so dass sich mit wenigen Zeilen Code eine Verbindung aufbauen lässt. Die Implementierung beinhaltet keine Optimierungen und prüft nicht auf Sonderfälle.

Der Initiator einer Verbindung ist immer der Client, der den Raum auf dem Signaling-Server erzeugt hat, so dass immer klar ist wer die Offer startet. In dieser Implementierung generiert der Offerer den RTCDataChannel. Der zweite Client wartet darauf, dass der Channel geöffnet wird und verbindet sich.

[1] Der Übersichtlichkeit halber ohne Log-Einträge und Fehlerbehandlung.

```
1  private peerC: RTCPeerConnection;
2  private mainDataChannel: IMetaDataChannel;
3
4  private createPeerConnection(isOfferer: boolean) {
5   this.peerC = this.peerBuilder.create();
6   this.peerC.onicecandidate = this.callBackIceCandidate;
7   if (isOfferer) {
8    this.makeAnOffer(this.peerC);
9   } else {
10   this.waitForOffer(this.peerC);
11  }
12 }
```

Listing 6.1: CoWebRTC: createPeerConnection (Auszug)

```
1  private makeAnOffer(peerConnection: RTCPeerConnection) {
2   this.mainDataChannel.channel = this.createReliableRTCDataChannel(
       peerConnection, this.mainDataChannel.name);
3   this.handleDataChannelEvents(this.mainDataChannel.channel);
4   this.createOffer(peerConnection);
5  }
```

Listing 6.2: CoWebRTC: makeAnOffer (Auszug)

```
1  private waitForOffer(peerConnection: RTCPeerConnection) {
2   peerConnection.ondatachannel = (event) => {
3    this.mainDataChannel.channel = event.channel;
4    this.handleDataChannelEvents(this.mainDataChannel.channel);
5   };
6  }
```

Listing 6.3: CoWebRTC: waitForOffer (Auszug)

```
1  private createOffer(peerConnection: RTCPeerConnection) {
2   peerConnection.createOffer()
3    .then((description) => {
4     peerConnection.setLocalDescription(description)
5      .then(() => {
6       this.sendMessageForSignaling(peerConnection.localDescription);
7      })
8    })
9  }
```

Listing 6.4: CoWebRTC: createOffer (Auszug)

```
1 private createReliableRTCDataChannel(peerConnection: RTCPeerConnection
    , name: string): RTCDataChannel {
2  return peerConnection.createDataChannel(name, null);
3 }
```

Listing 6.5: CoWebRTC: createReliableRTCDataChannel (Auszug)

Die Funktion *createOffer* in Listing 6.4 generiert die SessionDescription und startet den Austausch der Daten über die Funktion *sendMessageforSignaling* über die Socket.io-Verbindung an den Signaling-Server. Der zweite Client erhält die Signaling-Nachricht (siehe auch Listing 5.1) und generiert eine *Answer* entsprechend wie in Abbildung 6.1 beschrieben. Der PeerBuilder instanziiert wie in folgendem Listing 6.6 die RTCPeerConnection mit den entsprechenden Angaben zum STUN/TURN-Server.

```
1 public create(): RTCPeerConnection {
2   [...] // Abfragen ob Stun-Server gesetzt ist
3   return new RTCPeerConnection(this.pcConfig.servers);
4 }
```

Listing 6.6: CoWebRTC: PeerBuilder (Auszug)

Senden und Empfangen über den DataChannel

Listings 6.7 und 6.8 zeigen wie eine Nachricht über den RTCDataChannel gesendet und empfangen werden kann. Eine empfangene Nachricht wird über einen Stream nach außen weitergeleitet, so dass die Web-Applikation ihn über die *onMessage*-Funktion der Bibliothek abonnieren kann.

```
1 private sendMessage(msg: string) {
2  this.mainDataChannel.channel.send(msg);
3 }
```

Listing 6.7: CoWebRTC: sendMessage

```
1 private handleDataChannelEvents = (channel: RTCDataChannel) => {
2  [...] // Andere Events
3  channel.onmessage = (event: MessageEvent) => {
4   const messageText: string = event.data;
5   this.messageNotification$.next(messageText);
6 };}
```

Listing 6.8: CoWebRTC: handleDataChannelEvents (Auszug)

6.1.3 Nachrichtenfluss

Abb. 6.3 zeigt, wie am Beispiel einer Änderung im Code-Editor Daten von Browser zu Browser übertragen werden. Das Ändern des Textes löst in der *MonacoComponent* ein Event aus, das durch das Property-Binding an die *EditorPageComponent* übertragen wird. Die Nachricht wird in ein spezielles Nachrichtenformat konvertiert (Listing 6.9), das kennzeichnet welche Art von Aktion mit der Nachricht verbunden ist. Über die CoWebRTC-Bibliothek wird die Nachricht über den RTCDataChannel an Peer2 gesendet, der auf das Event *onMessage* reagieren kann. Die Nachricht wird anhand der Kennung an den richtigen Adressaten, in diesem Fall die *MonacoComponent* weitergeleitet, um die Inhalte anzupassen.

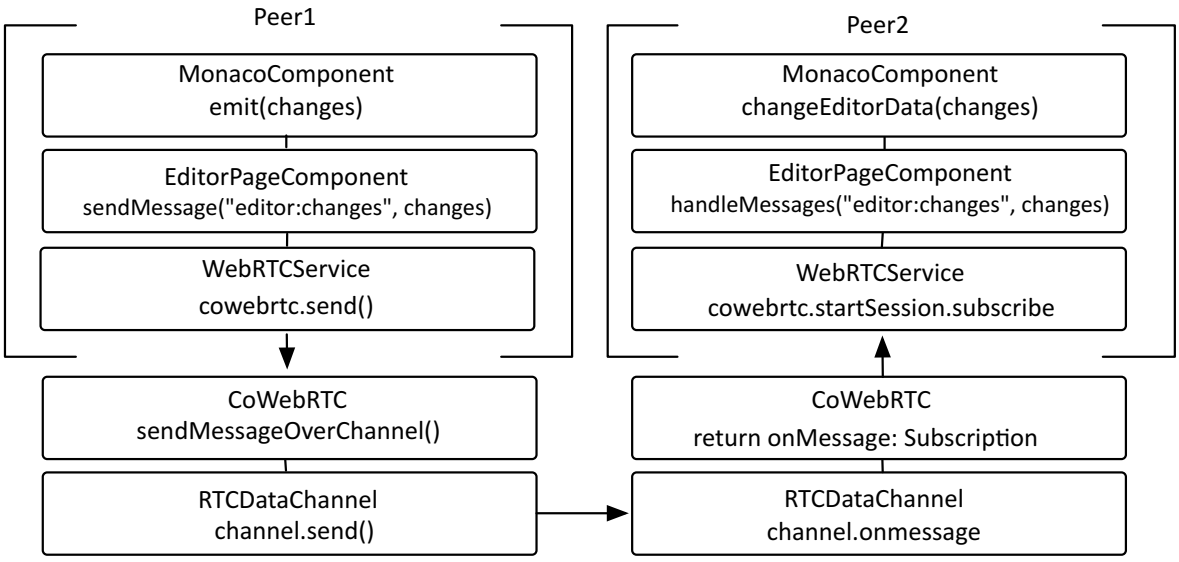

Abb. 6.3: Nachrichtenfluss durch die Komponenten

```
export interface IWebRTC {
 action: EWebRTCAction; // Action-Tag
 data: IWebRTCData;  // Datentyp der Nachrichten
}
export enum EWebRTCAction {
 CHATMESSAGE = 'chat:message',  // Chatnachricht
 EDITORMESSAGE = 'editor:all',  // Vollstaendiger Inhalt des Editors
 EDITORCHANGES = 'editor:changes', // Aenderungen im Editor
 EDITORWRITABLE = 'action:writable', // Aenderung des Schreibstatus
 PING = 'ping', // PING Nachricht
 PONG = 'pong' //PONG Nachricht
}
```

Listing 6.9: Struktur der Übertragungdaten

6.2 CoCoding

6.2.1 Die Weboberfläche

Im Folgenden werden die Abläufe zum Teilen und Bearbeiten von Dateien im Code-Editor anhand von Screenshots zusammenfassend beschrieben[2].

Der Benutzer gelangt über die Infoseite zur zentralen Seite der Applikation. Dort kann er über den <Neue Session> Button eine Session öffnen und sich über den <Teilen> Button eine URL generieren lassen, mit der sich ein Co-Worker einladen lässt. Abb. 6.4 zeigt den Zustand der Oberfläche nach dem Öffnen der Session und dem Aufruf des Teilen-Dialogs. Der Verbindungsstatus im Footer gibt an, dass die Applikation auf eine Verbindung wartet.

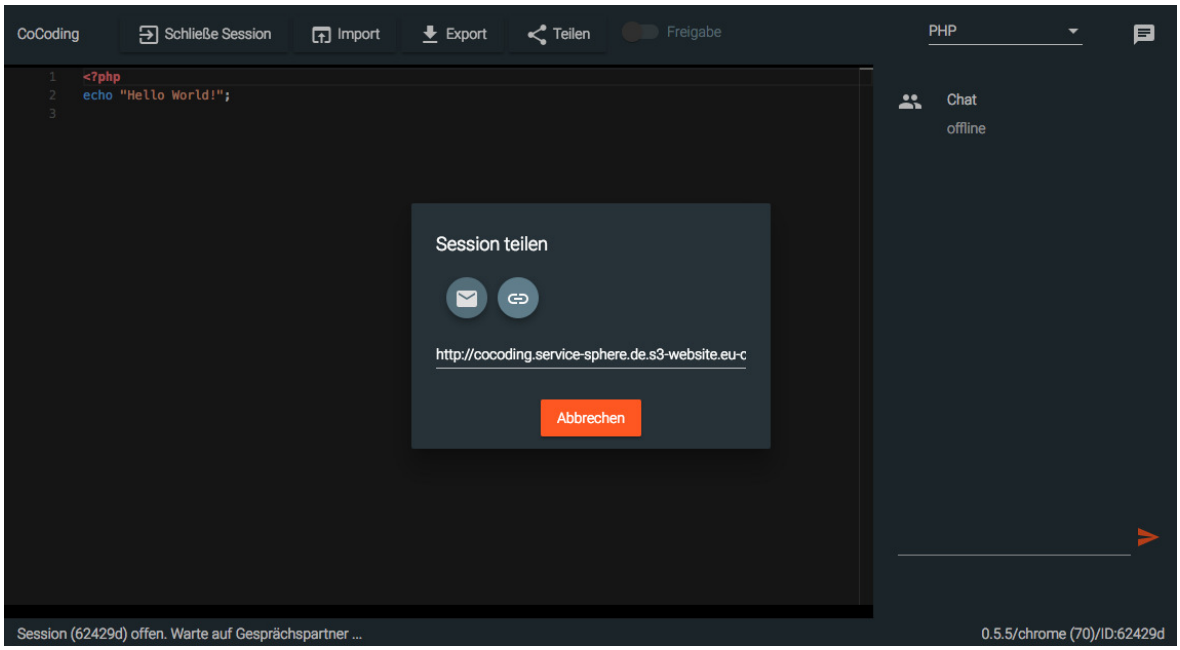

Abb. 6.4: CoCoding: Start und Teilen der Session

Nachdem der Co-Worker der Session beigetreten ist, wird eine Verbindung über den RTCDataChannel aufgebaut und Änderungen im Editor, Chatnachrichten sowie die Schreibfreigabe des Editors können übermittelt werden.

[2]Auf der beiliegenden DVD ist der vollständige Ablauf des Live-Tests mit Screenshots dokumentiert.

Die grüne Farbe des Footers gibt an, dass der Benutzer online ist und Schreibrechte besitzt. In der Toolbar sind nur noch die Buttons zu sehen, die notwendig sind, d.h. der Co-Worker sieht keinen <Teilen> Button und keinen <Freigabe> Slider (siehe Abb. 6.5 und 6.6).

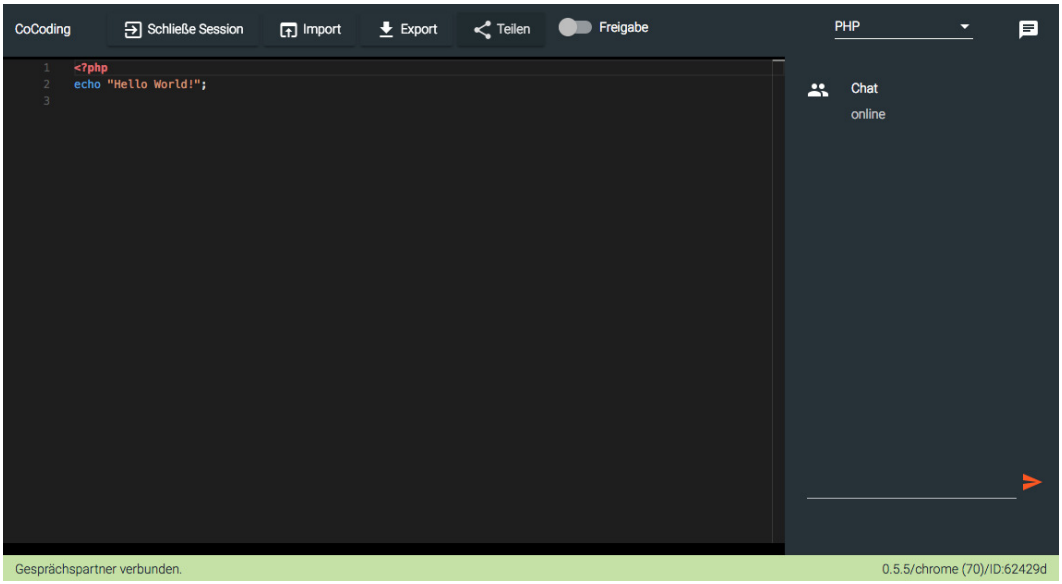

Abb. 6.5: CoCoding: Offene Verbindung des Initiators

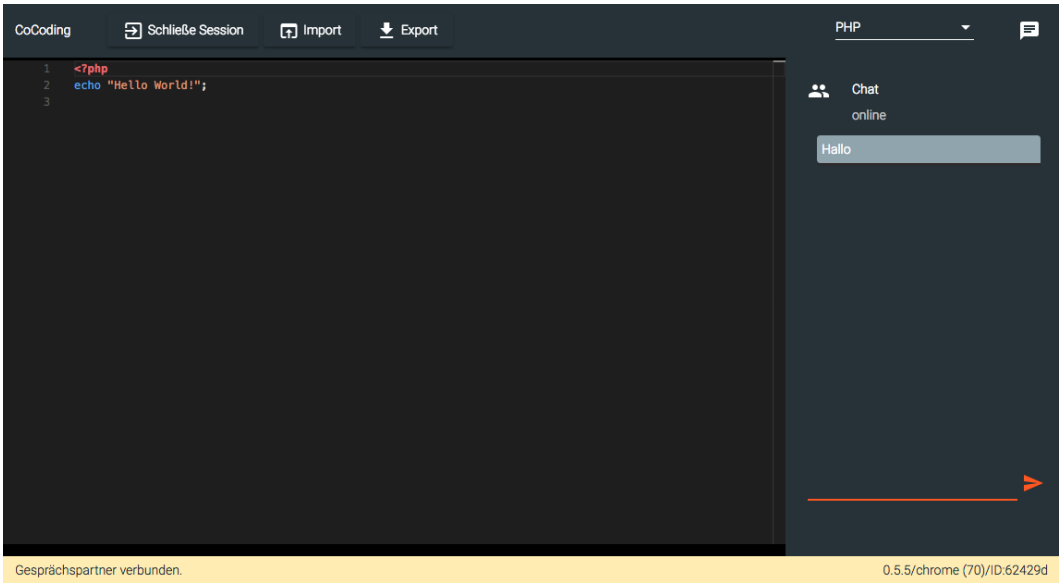

Abb. 6.6: CoCoding: Offene Verbindung des Co-Workers

Abb. 6.7 und 6.8 zeigen die Oberflächen, nachdem eine Datei importiert und der Initiator dem Co-Worker Schreibrechte erteilt hat. Die Farbe der Toolbar wechselt beim Initiator zu gelb und beim Co-Worker zu grün, um anzuzeigen, dass sich der Status verändert hat. Der Co-Worker kann in seinem Editor Änderungen am Text vornehmen, die mit dem Editor des Initiators synchronisiert werden.

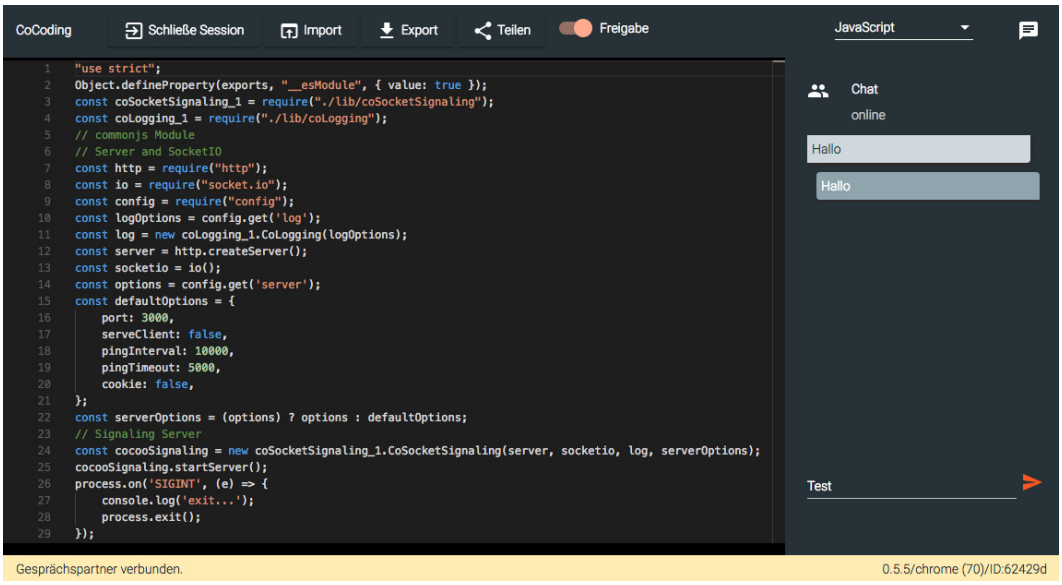

Abb. 6.7: CoCoding: Editors des Initiator nach der Freigabe

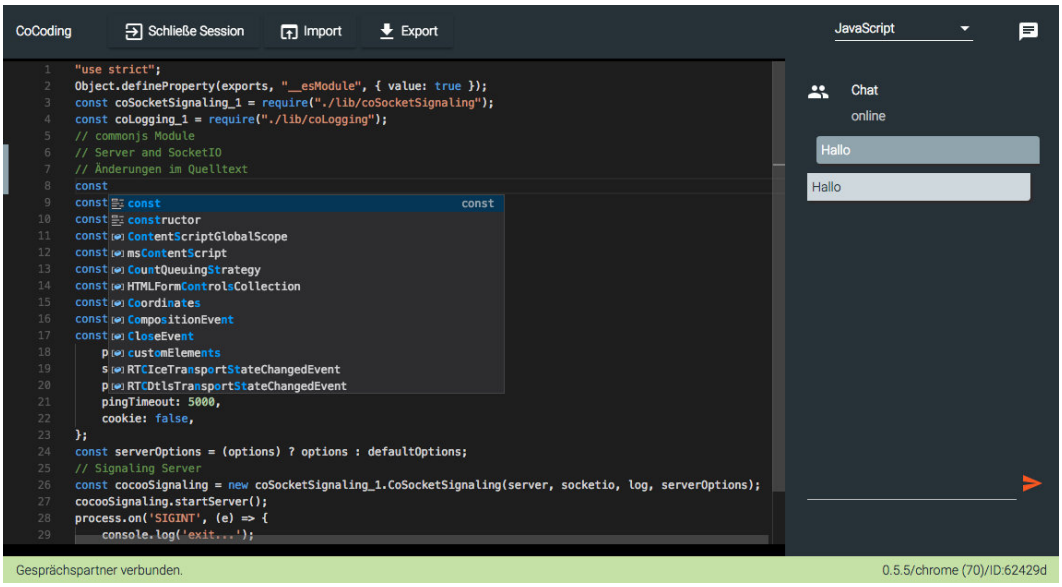

Abb. 6.8: CoCoding: Editors des Co-Workers nach der Freigabe

Abbildungen 6.9 und 6.10 zeigen den Export des Inhalts des Editors in eine Datei. Da in einem Browser nicht direkt auf die Festplatte gespeichert werden kann (im folgenden Kapitel 6.2.2 wird darauf näher eingegangen) wird das Speichern durch ein Dateidownload simuliert und das Speichern-Fenster des Betriebssystems angezeigt.

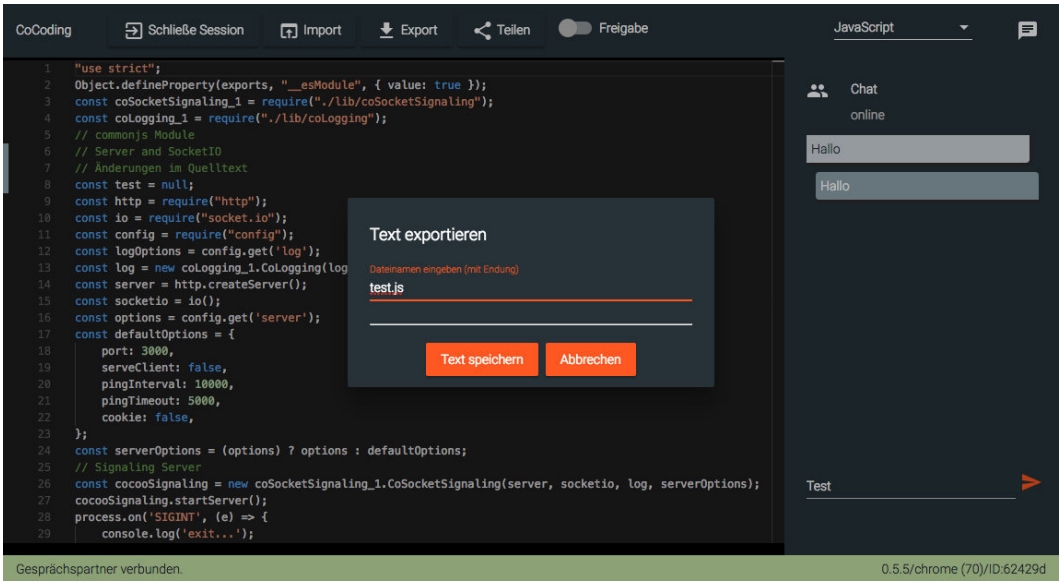

Abb. 6.9: CoCoding: Dialog Export einer Datei

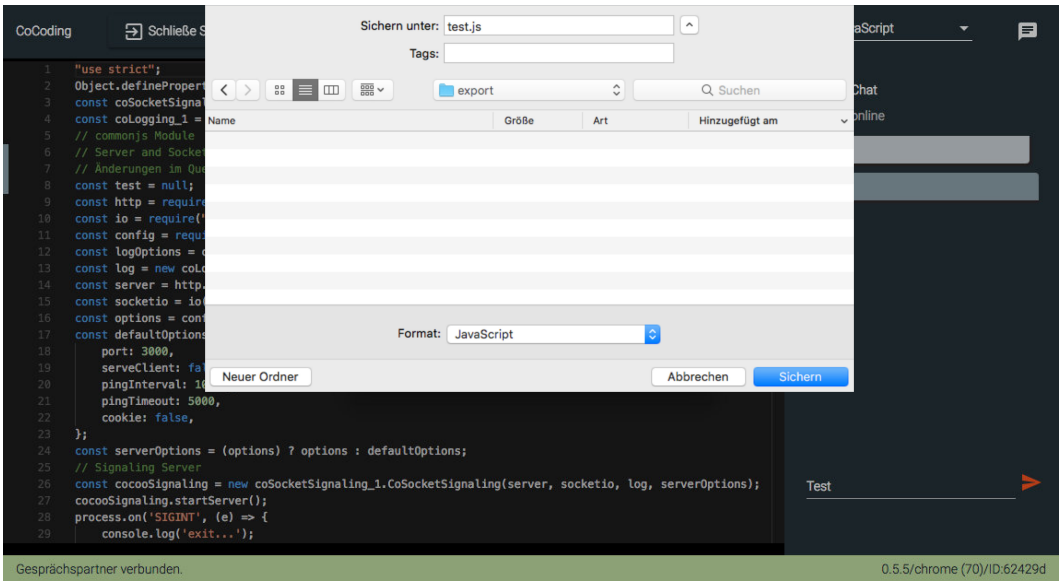

Abb. 6.10: CoCoding: OS-Export der Datei

Beendet der Co-Worker die Session wird eine Benachrichtigung an den Initiator gesendet. Die Session bleibt für den Initiator geöffnet, um erneut eine Einladung versenden zu können.

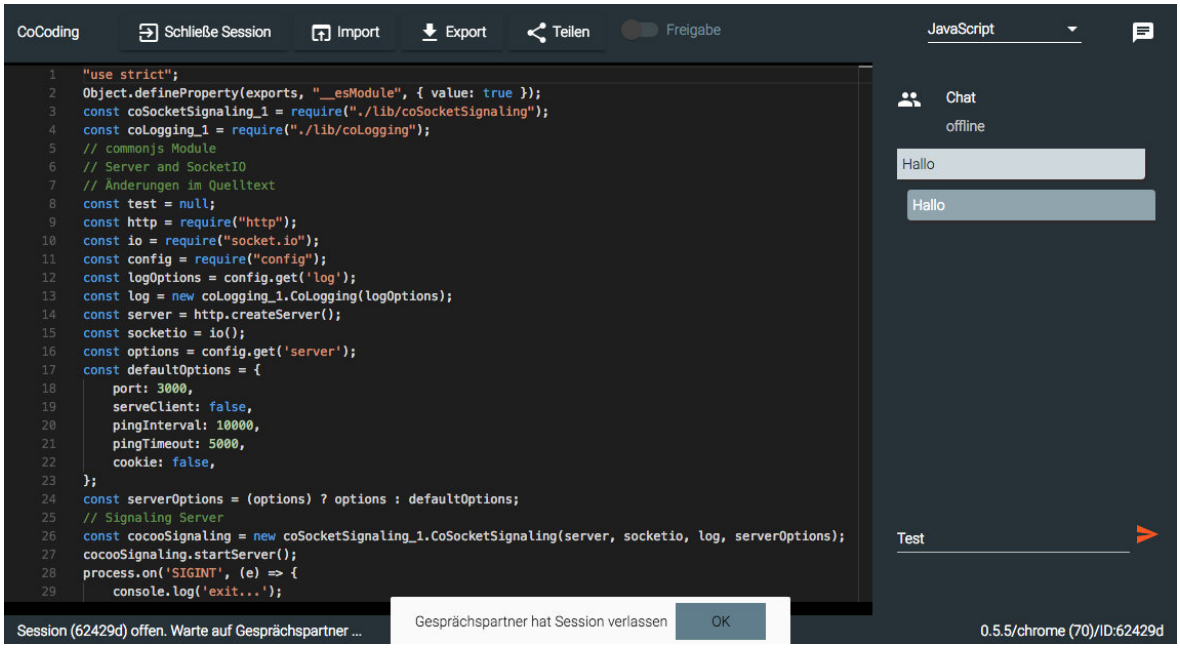

Abb. 6.11: CoCoding: Co-Worker verlässt die Session

6.2.2 Die FileAPI

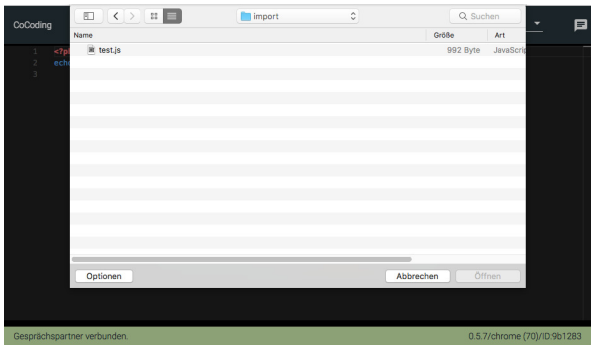

Abb. 6.12: Textimport

Das Importieren und Exportieren von Dateien aus dem lokalen Dateisystem des Nutzers ist durch die Verwendung der FileAPI [Con18] möglich. Allerdings ist der Zugriff auf lokale Dateien aus Sicherheitsgründen beschränkt. Nur wenn der Benutzer die Datei explizit über das Öffnen-Auswahlfenster des Betriebssystems auswählt, ist es möglich sie zu importieren. Die Daten können in Textform als auch in Binärform eingelesen werden. Der Export ist nicht ohne Weiteres möglich, ein Schreiben von Dateien ist nicht erlaubt. Allerdings können lokal generierte Inhalte heruntergeladen werden. Diesen Umstand nutzt die Bibliothek *FileSaver.js* von Grey [Gre11], die in CoCoding beim Export einer Datei zum Einsatz kommt.

6.3 Projektstruktur

Da die Grundstruktur der Projekte ähnlich ist, wird hier am Beispiel des CoWebRTC Projekts die Dateistruktur erläutert. Alle Projekte beziehen ihre externen Module aus dem npm-Repository[3]. npm ist ein Paketmanager für JavaScript-basierte Projekte und Bestandteil von Node, d.h. er ist automatisch auf jedem System vorhanden auf dem Node installiert wurde, lässt sich aber auch ohne Node nutzen. Welche Module in welcher Version benötigt werden wird in der Datei *package.json* hinterlegt. Alle Module müssen im ersten Schritt installiert werden und liegen dann im Verzeichnis /node_modules vor. Die Sourcen des Projekts liegen im Verzeichnis /src, die Tests in /test.

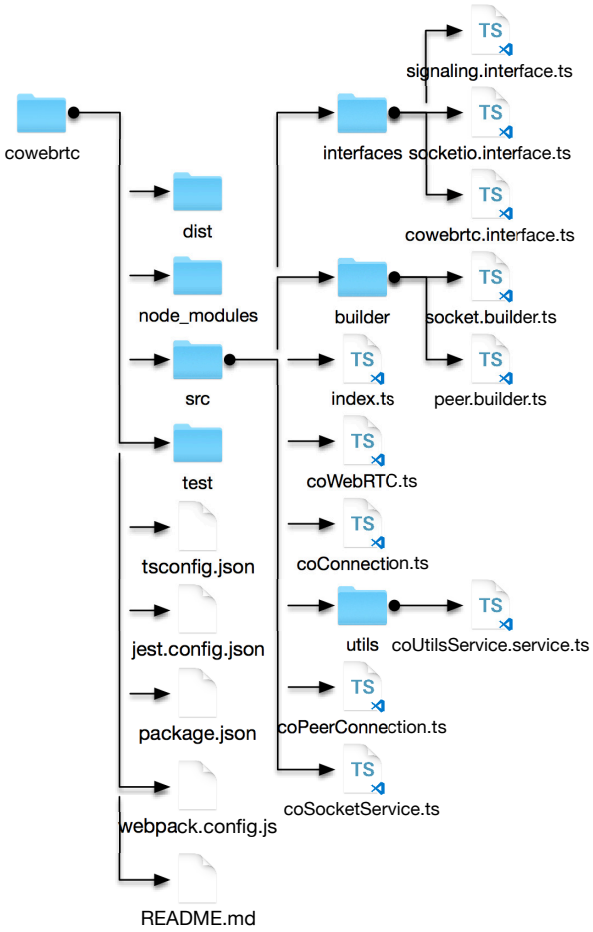

Abb. 6.13: Projekt CoWebRTC

Alle Projekte werden durch den Typescript-Transpiler in JavaScript Dateien konvertiert, CoWebRTC zusätzlich noch mit seinen Abhängigkeiten als UMD-Modul zusammengepackt und komprimiert. Dieser Vorgang wird durch den Bundler Webpack[4] übernommen. Angular Projekte besitzen von Haus aus Webpack als Bundler und können deshalb effizient zusammengepackt und optimiert werden. CoWebRTC ist von der Konzeption her geeignet als eigenes npm-Modul im offiziellen npm-Repository zur Verfügung gestellt zu werden. Aktuell liegt die Bibliothek in CoCoding als sogenanntes Asset vor, als externe Ressource, die mit ausgeliefert wird. Alle Produktionsversionen der Projekte liegen final im /dist-Verzeichnis vor. Im Root-Ordner sind Konfigurationsdateien für TypesScript, Softwaretests und Webpack zu finden. Wie die Projekte im einzelnen erstellt werden, wird in der Datei README.md in jedem Projekt ausführlich erklärt.

[3] https://www.npmjs.com/
[4] https://webpack.js.org/

Kapitel 7

Evaluation

Das Kapitel behandelt die Prüfung des Prototypen auf seine Funktionalität und seine Gebrauchstauglichkeit. Es soll sichergestellt werden, dass die definierten Anforderungen aus Kapitel 4 erfüllt werden, d.h. es wird getestet ob die technische Eignung gegeben ist. Der zweite Teil evaluiert mithilfe von Usability Tests, ob die Applikation im Hinblick auf das erarbeitete Szenario als nutzbar empfunden wird und ob der Einsatz der WebRTC Technologie einen Einfluss auf diese Einschätzung nimmt.

7.1 Test der Funktionalität

7.1.1 Strategie

Softwaretests prüfen die Korrektheit auf verschiedenen Ebenen. Grundsätzlich orientiert sich dieser Arbeit an den Vorschlägen für JavaScript-basierte Projekte von Zaidman [Zai18], der drei Typen unterscheidet:

- In Unit-Tests werden kleinere Programmteile in Isolation von anderen Programmteilen getestet. Die Einheiten können einzelne Methoden, Klassen oder auch Komponenten umfassen und werden in der Regel zeitnah zur Programmierung erstellt [siehe auch Fra14, Kap.9.1.1].
- Integrationstests überprüfen das Zusammenspiel zwischen Komponenten, d.h. einzelne Komponenten werden nach und nach zusammengeführt, bis zum Schluss alle Komponenten zur Verfügung stehen und komplette Anwendungsfälle durchlaufen werden können [siehe auch Fra14, Kap.9.3].
- UI Tests, auch bezeichnet als E2E (End-to-End) Tests oder wie bei Elliot [Ell16] und Franz [Fra14] auch als Funktionale Tests bezeichnet, überprüfen ob eine Applikation die Anforderungen aus Sicht der User erfüllt, d.h. ob sich die Anwendung im Browser so verhält wie erwartet.

Da die einzelnen Projekte ihre eigenen Verantwortlichkeiten haben und auf verschiedenen Ebenen fungieren, besitzen die Projekte ihre eigenen Testumgebungen und werden in der Auswertung getrennt voneinander betrachtet. Hauptsächlich kommen in den Tests Integrationstests zum Einsatz, da die Kommunikation zwischen den Komponenten den wichtigsten Teil einnehmen. UnitTests prüfen bspw. die Existenz von Schnittstellen, Komponenten und Konfigurationen.

Eine Besonderheit im Umgang mit WebRTC macht im Projekt CoWebRTC einen Test-Client notwendig. WebRTC ist eine Technologie, die als API im Browser zur Verfügung steht (siehe Abb. 5.2), d.h. alle Anforderungen, die eine Verbindung über eine RTCPeerConnection benötigen, müssen in einem Browser getestet werden, da die Funktionalität sonst gar nicht zur Verfügung steht. D.h. die Tests benötigen eine Hilfsapplikation im Browser, um Funktionen, die die Bibliothek zur Verfügung stellt, ausführen zu können. Über ein Button kann die Funktion ausgelöst und die erwarteten Ausgaben in der Konsole geprüft werden. Diese Tests werden als Integrationstests gewertet, da sie den Verbindungsaufbau über die WebRTC-Technologie im Zusammenspiel mit dem Signaling-Server testen und nicht die Oberfläche selbst. Anders wird dies im CoCoding Projekt gewertet. Eine Aktion im Browser führt zu Reaktionen auf der Oberfläche, die in den Tests berücksichtigt werden müssen, deshalb sind diese Tests klassische E2E Tests, die zum einen das erwartete Verhalten der Oberfläche testen, aber auch einen Integrationstest über alle Komponenten durchführen.

7.1.2 Testumgebungen

Frameworks

Für JavaScript existiert eine Fülle von Testframeworks. Anders als in Java oder PHP, in der es Standard Frameworks, wie JUnit und PHPUnit gibt, bringen viele JavaScript-Frameworks ihre eigenen Testtools mit. Da JavaScript sowohl client- als auch serverseitig eingesetzt wird, sind die Frameworks unterschiedlich ausgelegt und übernehmen unterschiedliche Funktionen innerhalb der Testumgebungen.

Die wichtigsten Aufgaben, die von einem oder einer Kombination von Frameworks übernommen werden muss sind zum einen die Möglichkeit Tests zu strukturieren und zu organisieren. Zum anderen müssen Funktionen bereitgestellt werden, um Bedingungen zu prüfen und es müssen Verfahren bereitstehen, um Komponenten simulieren zu können. Zudem sollte ein Framework in der der Lage sein die Tests angemessen zu protokollieren, auszugeben und Reports zu erzeugen. Da kein Framework alle Anforderungen erfüllt, werden in der Regel

mehrere Test-Frameworks kombiniert, um für die Anforderung und Umgebung (client- oder serverseitig) die passende Lösung[1] zu finden [siehe Goj18].

In dieser Arbeit fiel die Wahl für Unit- und Integrationstest auf das Framework Jest. Das Framework ist verglichen mit den etablierten Frameworks noch recht neu und basiert auf Jasmine. Der Vorteil ist, dass alle Tests im Projekt, unerheblich ob auf Client- oder Serverseite, einheitlich von einem Framework bearbeitet werden können. Es ist sowohl für Node als auch für Angular Projekte verwendbar. Zudem kann Jest mit Typescript-Projekten umgehen, da es den Code vorab transpilieren kann und die Tests selbst können in Typescript programmiert werden. Um Zugriff auf eine Browser-Umgebung zu erhalten, wurde, sowohl im CoWebRTC-Testclient als auch in der CoCoding-App, mit einer Kombination aus Jest und der Node-Bibliothek Puppeteer[2] gearbeitet. Über das Framework kann ein Browser über die ChromeDevTools gesteuert werden, ohne ihn explizit starten zu müssen, d.h. die Tests laufen im sogenannten Headless-Modus des Chrome- oder Chromiumbrowsers.

Alle Tests generieren ein HTML Protokoll, die für die einzelnen Projekte im Anhang gelistet sind. Alle Tests haben zudem gemein, dass sie einheitlich organisiert sind. Die nummerierten Tests sind immer Teil einer TestSuite, die sich an der Syntax der BDD (Behaviour Driven Tests) orientiert, die das Verhalten der Software beschreibt.

```
1 describe('Session', () => {
2  it('T2: Verbinden zweier Clients ueber Signaling',() => {
3  ...
4  })
5 })
```

Listing 7.1: Beispiel für eine Testbeschreibung

Die Integrationstest, sowie die E2E Tests prüfen mit einem Chrome- oder Chromium Browser (>Version 60). Für die funktionalen Tests, als auch für die Usability Test in Kapitel 7.2, stehen zwei konfigurierte Systeme zur Verfügung, eine lokale Testumgebung, sowie eine Live-Testumgebung.

[1] Beispiele sind Frameworks wie Mocha (https://mochajs.org/), Jasmine (https://jasmine.github.io/) oder Jest (https://jestjs.io/) und für UI-Tests Tools wie z.B. Protractor (https://www.protractortest.org/)
[2] https://github.com/GoogleChrome/puppeteer

Lokale Testumgebung

CoSocketSignaling-Server und CoCoding-App stehen zusammen auf einen RaspberryPi3 mit installiertem Node in Version 10.9 und nginx als Webserver zur Verfügung. Wenn kein Zugang zu einem Netzwerk (LAN/WLAN) besteht wird zusätzlich ein WLAN-Router eingesetzt, der ein lokales Netz zur Verfügung stellt. Die Lösung ist unabhängig von Infrastrukturen einsetzbar und aufgrund der kleinen Maße des RasperryPi3 transportabel. Diese Umgebung wurde für den Usability-Test in Kapitel 7.2 eingesetzt, da kein Internetzugriff zur Verfügung stand und nur lokal getestet werden konnte.

Live Testumgebung

Eine Live-Umgebung wurde auf den Servern der Amazon Web Services eingerichtet. CoSocketSignaling läuft auf einer EC2 Instanz mit installiertem Node (v10.9), die CoCoding-App in einem S3-Bucket. Als STUN/TURN-Server dient ein Coturn-Server, der ebenfalls auf einer EC2-Instanz installiert wurde. In dieser Umgebung wurde ein Live-Test durchgeführt, der im folgenden Kapitel beschrieben wird.

7.1.3 Oberflächentest

Nach Franz [Kap10.2.1 Fra14] kann ein Oberflächentest sowohl ein funktionaler Test als auch unter die Kategorie Usability-Test fallen. In dieser Arbeit wird er als funktionaler Test eingestuft.

Auch wenn mit Unit-, Integrations- und E2E Tests die meisten Anforderungen geprüft werden konnten, blieben einige Sonderfälle ungetestet, die aufgrund von Einschränkungen gesondert betrachtet werden mussten. Aus Sicherheitsgründen hat ein Test-Framework bspw. keinen Zugriff auf das Auswahlfenster einer Datei im Betriebssystem. So wurde z.B. der Import und der Export der Dateien durch einen E2E Test in soweit geprüft, dass die korrekten Fenster und Eingaben zur Verfügung stehen, aber durch den Live-Test ergänzend geprüft, ob die Dateien tatsächlich geladen und gespeichert wurden. Ein weiteres Beispiel ist die Anforderung des Kopierens von Text durch Copy/Paste in den Editor.

Möchte man sicherstellen, dass die Verbindung über einen STUN/TURN-Server korrekt funktioniert, ist es notwendig Browser-Instanzen auf unterschiedlichen Rechnern in getrennten Netzen zu betrachten. Denn aus einem lokalen Netzwerk heraus wird beim STUN-Server zwar nach der externen IP-Adresse angefragt, aber bei der Auswahl des Verbindungsweges

nicht genutzt. Dies lässt sich durch das Auslesen der SDP-Daten in der Konsole erkennen. Im Live-Test wird der vollständige Verbindungsweg angezeigt.

Aus diesen Gründen wurden Tests durch einen manuellen Oberflächentest in der Live-Umgebung ergänzt, der durch zwei Personen an unterschiedlichen Standorten durchgeführt wurde. Der Live-Test folgt einem Testplan, der im Laufe der Entwicklung an die Anforderungen angepasst wurde und in der finalen Version alle Anforderungen an den Editor beinhaltet. Trotzdem werden in den folgenden Auflistungen nur die ergänzenden Tests aufgeführt. Der Test wurde mit zwei Chrome-Browsern (Version 69) durchgeführt. [3]

7.1.4 Durchführung

CoSocketSignaling

Beim Signaling-Server wird die Kommunikations-Schnittstelle getestet. Die Tests[4] konzentrieren sich auf die Abfolge der gesendeten und empfangenden Nachrichten und simulieren die Anfragen beider Clients und werten die Reaktionen aus, d.h. sie überprüfen eine Abfolge an Nachrichten und erwarten, dass definierte Antworten zurückgesendet werden. Folgende Tabelle ordnet die Tests den Anforderungen aus Kapitel 4.2.3 zu. Die Beschreibungen der Tests beziehen sich der Einfachheit halber auf das erwünschte Verhalten, nicht auf die Einzelereignisse. Die Unit-Tests überprüfen die Initialisierung des Servers. Sie sind in einer Suite zusammengefasst und sind keiner Anforderung direkt zugeordnet.

	C1	C2	C3	C4	C5	Test
Integrations-Tests						
T1: Ein Client kann Session öffnen/schließen	x					ok
T2: Zweiter Client kann beitreten		x				ok
T3: Dritter Client kann nicht beitreten			x			ok
T4: Client1 kann Client2 Signaling-Message senden				x		ok
T5: Client2 verlässt Session => Rückmeldung					x	ok
Unit-Tests						
Testsuite D1: Konfiguration						ok

Tabelle 7.1: Funktionale Tests: CoSocketSignaling

[3] Im Anhang unter A.4.5 und mit Screenshots auf der beiliegenden DVD enthalten.
[4] Testprotokoll unter A.4.1

CoWebRTC

In den Tests der WebRTC-Bibliothek werden die einzelnen Schritte des Verbindungsaufbaus und des Session-Handlings getestet. Da die WebRTC-Funktionalität eine Implementierung innerhalb des Browsers darstellt, kann sie nur in einer Browser-Umgebung getestet werden. Die Umwege über die umfangreichen GUI-Mechanismen der CoCoding Oberfläche können Fehler schwer auffindbar machen, deshalb wurden in CoWebRTC die Tests über einen Test-Client durchgeführt. Listing 7.2 zeigt, wie der Test auf einem geöffneten DataChannel, ohne Vor- und Nachbedingungen, implementiert ist. Durch das Framework Puppeteer werden zwei Browser-Instanzen des Chromium Browsers erzeugt, die Funktion der Bibliothek über einen Button aufgerufen und dann kontrolliert, ob die Ausgaben in der Konsole den Erwartungen entsprechen.

```
1  it('T4: Oeffnen eines RTCDataChannels zwischen zwei Clients', async ()
       => {
2    page.on('console', async (msg) => {
3     messageArray1.push(msg.text());
4    });
5    page2.on('console', async (msg) => {
6     messageArray2.push(msg.text());
7    });
8
9    page.bringToFront();
10   await page.click('button[name=btnStart]');
11   await page.waitFor(testConfig.defaultAwait);
12
13   page2.bringToFront();
14   await page2.click('button[name=btnStart]');
15   await page2.waitFor(testConfig.defaultAwait);
16   expect(messageArray1).toContainEqual('ch:open');
17   expect(messageArray2).toContainEqual('ch:open');
18   await page.waitFor(testConfig.defaultAwait);
19   });
```

Listing 7.2: CoWebRTC: Test auf geöffneten DataChannel (Auszug)

Die Unit-Tests sind auch hier in einer Suite zusammengefasst und testen die Existenz und das korrekte Verhalten der Schnittstelle nach außen. Das Testprotokoll und ein Screenshot

des Test-Clients sind im Anhang unter A.4.3 und A.4.4 zu finden. Tabelle 7.2 zeigt die Zuordnung der Tests zu den Anforderungen (Kapitel 4.2.3) .

	B1	B2	B3	B4	B5	B6	B7	Test
Integrations-Tests								
T1: Öffnen/Schließen der Session über Signaling	x							ok
T2: Verbinden zweier Clients über Signaling		x						ok
T3: Verlassen der Verbindung wird gemeldet			x					ok
T4: Öffnen eines RTCDataChannels zwischen Clients				x				ok
T5: Senden/Empfangen über RTCDataChannel					x			ok
T6: Extern Connect (STUN/TURN)							x	ok
Unit-Tests								
Testsuite D1: Schnittstellen						x		ok
Live-Tests								
L5: SDP Konsole: ICE Candidates über STUN/TURN							x	ok

Tabelle 7.2: Funktionale Tests: CoWebRTC

CoCoding

In CoCoding werden E2E-Tests über die Oberfläche vorgenommen, um die Anforderungen zu prüfen. Wie im Test-Client simuliert der Test die Bedienung der Oberfläche mit zwei Browser-Fenstern, um die Verbindung, den Datenaustausch und zusätzlich die Reaktion der GUI zu testen. Da die Anzahl der Tests größer ist als in den zwei vorangegangen Tests, werden Zuordnung (Tabelle 7.3) und Testbezeichnung (Tabelle 7.4) getrennt aufgelistet. Die App beinhaltet zudem eine Reihe von Unit-Tests, die hier nicht gelistet sind. Sie überprüfen lediglich die Existenz der Komponenten. Das Protokoll ist im Anhang unter A.4.2 zu finden.

Eine besondere Erklärung benötigt der Test L12, der den Ping-Wert über den Live-Test abprüft und nicht automatisiert über einen der E2E Tests. Interessant ist hier die Antwortzeit von App zu App nach dem Nachrichtendurchlauf durch die Komponenten wie in Kapitel 6.1.3 beschrieben und zwar in entfernten Netzen, nicht lokal. Denn dies simuliert die Verzögerung zwischen dem Tippen eines Buchstabens und der Synchronisierung im entfernen Browser. Bei einem Click auf das Chat-Icon in Oberfläche kann deshalb eine PING-Anfrage ausgelöst werden, die vom verbundenen Browser mit einem PONG beantwortet wird.

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18
E2E																		
T1	x																	
T2		x																
T3				x														
T4			x															
T5					x													
T6						x												
T7								x										
T8									x									
T9												x						
T10										x								
T11													x					
T12														x				
T13															x			
T14																x		
T15																	x	
Live-Test																		
L9							x											
L11											x			x				
L12																		x
L16						x												
L17					x													

Tabelle 7.3: Funktionale Tests: CoCoding

Nr	Test-Beschreibung	Test
E2E		
T1	Client öffnet Session mit Button <Neue Session>	ok
T2	Client schließt Session mit Button <Schliesse Session>	ok
T3	Session kann per URL beigetreten werden (DataChannel offen)	ok
T4	Button <Teilen> öffnet Dialog. Die URL wird angezeigt.	ok
T5	Button <Import> öffnet Dialog, Datei kann importiert werden	ok
T6	Button <Export> öffnet Dialog, Dateiname kann für Export angegeben werden	ok
T7	Slider <Freigabe> sendet Editorfreigabe an den zweiten Client	ok
T8	Geänderter Quelltext von Client1 wird an Client2 übertragen	ok
T9	Nach dem Absenden von Text im Chat wird er übertragen und angezeigt	ok
T10	Veränderungen im Editor werden seitlich mit einem Decorator markiert	ok
T11	Wenn ein Client die Session verlässt wird eine SnackBar mit Meldung angezeigt	ok
T12	In der Toolbar ist eine ComboBox zum Umstellen der Sprache des Editors	ok
T13	Im Footer wird der Session-Status angezeigt	ok
T14	Farbwechsel im Footer beim Ändern der Schreibrechte	ok
T15	Session des Initiators bleibt offen wenn Co-Worker Session schliesst	ok
Live-Test		
L9	Text kann per Copy/Paste eingefügt werden	ok
L11	Umschalten der Sprache im Editor durch eine ComboBox	ok
L12	Durchschnittlicher Ping-Wert liegt unter 50ms	ok
L16	Im Datei-Export Fenster wird das Datei-Speichern Fenster des OS geöffnet	ok
L17	Im Datei-Import Fenster wird das Datei-Laden Fenster des OS geöffnet	ok

Tabelle 7.4: Testbeschreibung CoCoding

7.2 Usability Test

In den vorangegangenen Kapiteln wurden die Phasen der Analyse, Konzepterstellung und Entwicklung des Prototypen erläutert, sowie funktionale Tests durchgeführt, die zeigen, dass die technischen Anforderungen erfüllt werden. Im Mittelpunkt dieses Kapitels steht der Prozess der Evaluierung durch den Nutzer.

Nach Moser [Mos12, S. 220] ist es wichtig während der Entwicklung Nutzertests durchzuführen, um früh Unstimmigkeiten zu finden, die durch Vereinfachung oder durch Missverständnisse entstanden sind. Um die Nutzbarkeit der CoCoding-App zu prüfen, sind formative Usability Tests[5] zum Einsatz gekommen, um qualitative Aussagen zur Verbesserung der App zu erhalten und um Probleme beim Einsatz der WebRTC-Technologie aufzudecken.

Zu Beginn der Umsetzungsphase (Prototyp ohne vollständige Funktionalität) hat es drei kurze Treffen mit einzelnen Entwicklern gegeben, die darum gebeten wurden zu prüfen, ob der Seitenaufbau und der Ablauf verständlich ist. Moser [Mos12] bezeichnet diese kurzen 5-10 Minuten Treffen als Hallway-Tests. Sie sind informell und subjektiv, verhelfen aber früh zu Feedbacks. Die Ergebnisse der Gespräche sind in die Anforderungsdefinitionen in Kapitel 4.2.3 eingeflossen, wurden implementiert und durch Softwaretests geprüft.

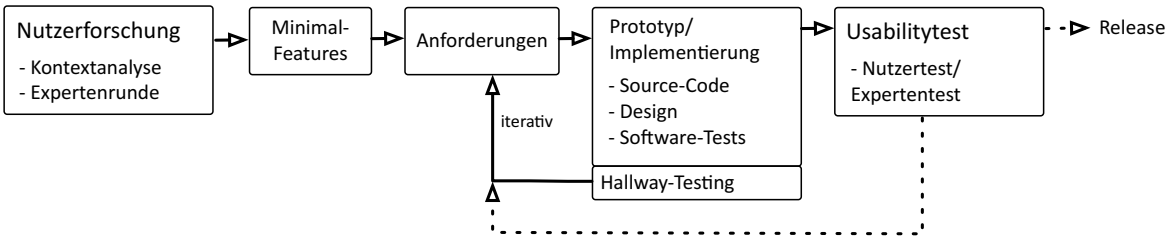

Abb. 7.1: Iterativer Design-Prozess

Zum Ende der Implementierungsphase wurde der erste funktionsfähige Prototyp durch eine Kombination aus Nutzer- und Expertentest evaluiert, der im Mittelpunkt des folgenden Kapitel steht. Der Prozess der iterativen Entwicklung zeigt Abb. 7.1. Er ist an Moser [Mos12, S. 14] angelehnt, der den Prozess als User-Experience-Design-Prozess bezeichnet, der von Konzept bis zum Release in Iterationen durchgeführt wird. Die Applikation hat im Rahmen dieser Arbeit eine Zyklus-Phase durchlaufen um das erste Release fertigzustellen.

[5] Der Begriff Usability ist in der DIN-ISO 9241-11 definiert und beschreibt das „Ausmaß in dem ein Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen."

7.2.1 Planung und Ziele

Der Usability-Test soll prüfen, ob die WebRTC-Technologie Einfluss auf die Nutzbarkeit der Applikation nimmt und folgende Fragen klären: Fällt es auf, dass es sich hier um eine WebRTC-basierte Kommunikation handelt? Entstehen Fehler, die durch die Technologie zustande kommen? Gibt es Verzögerungen oder fehlt Rückmeldung, die eindeutig der Technologie zuzuordnen ist? Werden Funktionen vermisst, die typischerweise von einem Backend-Server geleistet werden? Ziel des Test war es aus Beobachtungen, Äußerungen und Mitschriften der Tester Rückschlüsse auf eventuelle Probleme zu ziehen.

Der Prototyp wurde vorab durch einen Oberflächentest in der lokalen Testumgebung (siehe Abschnitt 7.1.2) geprüft, um sicherzustellen, dass die Nutzer nicht auf schwerwiegende Fehler treffen, die das System unbenutzbar machen. Bis auf die Anforderung (Nr. A10), dass Änderungen im Editor in geeigneter Form gekennzeichnet werden sollen, waren alle vereinbarten Minimalanforderungen implementiert.

Das AdHoc-Review Szenario aus Kapitel 4.2.1 diente als Grundlage für die Durchführung des Tests, d.h. es sollte geprüft werden, ob eine Gruppe von Testern Aufgaben nach Vorlage des Szenarios ohne Hilfen lösen können. Die Testgruppe bestand aus Softwareentwicklern, die in Projekten zusammenarbeiten und sich persönlich kennen. Die Zusammenstellung der Gruppe wich von der Gruppe zur Anforderungsanalyse leicht ab, d.h. nicht alle kannten die Anforderungen auf die man sich vorab geeignet hatte. Die Gruppe bestand nicht aus Usability-Experten, aber sie sind mit Usability-Verfahren vertraut, so dass es im Test nicht nur um die Bewältigung des Szenarios ging, sondern auch darum gebeten wurde auf einige Heuristiken zu achten, die in der Methode der heuristischen Evaluation Anwendung finden [Mos12, S. 232].

Der Grund für die Vermischung der Methodik ist, dass Entwickler es gewohnt sind funktionale Tests durchzuführen und das Szenario unter Umständen als funktionaler Test abgearbeitet worden wäre. Der Oberflächentest hat vorab diese Vorgänge geprüft und keine Fehler aufgezeigt. Deshalb war es wichtig, dass sich die Tester von dieser Vorgehensweise gedanklich lösen und sich stärker auf den Usability-Test einlassen. Nach Sarodnick und Brau [SB06, S. 244] hilft die Kombination der beiden Methoden die Relevanz der Probleme und die Verbesserungsvorschläge besser einzuschätzen.

7.2.2 Durchführung

An dem Test haben insgesamt sechs Entwickler teilgenommen, die ihre eigenen Arbeits-Laptops mit installierten Chrome-Browser verwendeten. Der Test fand im Büro des Teams, in ihrer normalen Arbeitsumgebung statt, um eventuelle Störungen, Ablenkungen miteinzubeziehen. Als technische Umgebung stand die lokale Installation (siehe 7.1.2) auf dem Rasperry Pi3 zur Verfügung. Für den Test wurde ein extra WLAN erzeugt ohne Verbindung ins Internet, da fremde Systeme nicht ohne Weiteres in das System vor Ort integriert werden können und so eine autarke Installation getestet werden konnte.

Der Zeitrahmen für die Bearbeitung der Aufgaben lag bei einer halben Stunde, bei der nur mit dem Testpartner gearbeitet und eine gemeinsame Lösung gefunden werden sollte. Die Methode ähnelt der Methode des „lauten Denkens“, bei der Tester ihr Handeln und ihre Erwartungen laut beschreiben. Da die Applikation eine Kollaborationslösung ist, bot es sich an paarweise die Aufgaben abzuarbeiten und direkt miteinander zu diskutieren. Nach Sarodnick und Brau [SB06, S. 171-172] regt diese Form (Constructive Interaction) eine natürliche Diskussion über die abzuarbeitenden Schritte an, da es meist ungewohnt für Tester ist, laut über ihre Schritte zu sprechen. Videoaufnahmen konnten nicht gemacht werden, d.h. der Testablauf wurde beobachtet und protokolliert. Im Anschluss gab es die Möglichkeit in einer freien Diskussion über die gefundenen Probleme zu sprechen.
Folgende, offen formulierte Arbeitsschritte wurden vorgegeben und sollten selbstständig im Team abgearbeitet werden[6]:

- Öffnen einer Session und Einladen eines Mitarbeiters (falls möglich über Slack)
- Importieren und/oder Kopieren von Text in den Editor
- Bearbeitungsfreigabe an den Mitarbeiter geben und nehmen
- Abwechselnde Bearbeitung des Texts
- Exportieren des Texts
- Nutzung des Chats

Zudem sollte auf die Einhaltung einiger Heuristiken geachtet werden. Es existieren eine Reihe von Vorschlägen für solche Heuristiken, sie hängen von der Art des Projekts ab. Basierend auf der ISO-Norm 9241-110 und Erfahrungen aus Projekten und Recherchen haben Sarodnick und Brau [SB06] einen Satz Heuristiken entwickelt, die hier zugrunde gelegt wurden.

[6] Der vollständige Bogen ist im Anhang (A.3) und auf der DVD zu finden.

Die Tester wurden gebeten, die Fragen schriftlich zu beantworten, ihre Erwartung an das Verhalten zu beschreiben und eine Einschätzung abzugeben für wie schwerwiegend sie das Problem halten.

- **Sind alle Funktionen vorhanden, um die Aufgaben zu bewältigen?**
  Die Frage zielt auf die Aufgabenangemessenheit ab, d.h. ob die Applikation grundsätzlich alle Funktionen beinhaltet, die notwendig sind, um das Szenario zu bewältigen und eine Kollaborationssitzung aufzubauen und abzuhalten.
- **Werden reale Arbeitsbedingungen abgedeckt?**
  Die Frage nach der Prozessangemessenheit soll prüfen, ob die Tester das Tool auch in ihrer täglichen Arbeitswelt nutzen würden.
- **Gibt das System ausreichend Rückmeldung?**
  Je nach Verbindungs-Status und Rolle des Benutzers verhält sich die Applikation unterschiedlich. Die Frage zielt darauf ab, ob die Selbstbeschreibungsfähigkeit der Applikation ausreichend ist, d.h. ob immer klar ist in welchem Status sich die Applikation gerade befindet. Beispiele: Ist die Verbindung zustande gekommen? Wer arbeitet gerade im Editor?
- **Ist das System konsistent bzgl. Dialoge, Buttons, Beschreibungen, Sprache und Formulierungen?**
  Die Gestaltung nimmt Einfluss darauf, wie selbstbeschreibend eine Anwendung ist und ob verstanden wird, wie die Applikation funktioniert und wie man zum Ziel kommt.
- **Sind Daten und/oder Texte verloren gegangen?**
  Diese Frage bezieht sich auf die System- und Datensicherheit und soll Aufschluss darüber geben, ob während der Bearbeitung unter Umständen durch die WebRTC-Technologie Fehler in der Datenübertragung aufgetreten sind.

7.2.3 Auswertung

In die Auswertung fließen mehrere Datenquellen ein. Zum einen wurden die Tester während ihrer Bearbeitung beobachtet und die Fortschritte während der Bearbeitung registriert. Zum anderen wurden die schriftlichen Aufzeichnungen der Tester kategorisiert und ausgewertet. Ob eine Verbindung zwischen den Browsern zustande gekommen ist, konnte indirekt über die Protokoll-Datei des CoSocketSignaling-Servers festgestellt werden. Ob eine P2P-Verbindung zustande gekommen ist, lässt sich über die Konsolen-Einträge des Browsers prüfen, die während des Tests geöffnet waren. Sarodnick und Brau [SB06, S. 244] empfehlen bei der

Auswertung Auswertungsraster zu erstellen, um die Probleme systematisch und übersichtlich zusammenzufassen und über die Häufigkeit die Relevanz zu bestimmen, da sie die wissenschaftlich korrekte Auswertung der qualitativ erhobenen Daten als schwierig einstufen. Aus diesem Grund wurden bei der Auswertung nach der Sammlung und Zählung aller Aussagen, die Inhalte nach ihrem Ort des Auftretens gruppiert und zusammengefasst.

Beobachtungen

Die Beobachtungen sind subjektiv aber sie geben ein allgemeines Bild vom Ablauf des Usability-Tests. Die vorgegebenen Arbeitsschritte wurden von den Testern innerhalb von 15 Minuten ohne Probleme abgearbeitet. Das Öffnen und Schließen der Session funktionierte im Chrome-Browser ohne Probleme. Das Kopieren der URLs, um einer Session beizutreten funktionierte ebenso reibungslos, wie auch der Wiedereintritt in eine Session nachdem sie einmal verlassen wurde. Bei der Vergabe der Schreibberechtigung zeigte sich, dass diese vom Co-Worker über den Schreibtisch hinweg verbal angefordert wurde und wieder Bescheid gegeben wurde, wenn die Änderungen als abgeschlossen betrachtet wurden. Bei der Bearbeitung des Quellcodes traten keine Probleme auf, das Importieren und Exportieren von Text dauerte subjektiv betrachtet recht lang. In dieser Phase wurde vor allem über die Fähigkeiten des Editors diskutiert und Vergleiche mit Desktop Varianten gezogen, die sich komfortabler und einheitlicher beim Syntax-Highlighting und der Code-Completion verhalten. Der Chat funktionierte bei allen Teilnehmern ohne Probleme.

Fragebögen

Die Auswertung der Diskussionen und Fragebögen zur Bewertung der Heuristiken spiegeln die Beobachtungen wider und ergeben ein recht einheitliches Bild über die relevanten Probleme, da einige Punkte entweder von allen Testern oder nur vereinzelt angesprochen wurden.

Die Einträge lassen sich in fünf Kategorien zusammenfassen. Abb. 7.2 zeigt die Verteilung der Aussagen auf die Themen Editor, Dateihandling, Systemstatus/Rückmeldung, Datenverbindung bezogen auf WebRTC und dem Chat. Die Angaben sind in Prozent sowie der Anzahl an Nennungen angegeben, wobei mehrfache Nennungen mehrfach gezählt wurden, um die Aussagen zu gewichten. Die meisten Einträge betreffen die Funktionalität und das Verhalten des Code-Editors und des Datei-Handlings[7].

[7] Protokolle und Auswertungen sind auf der beiliegenden DVD zu finden.

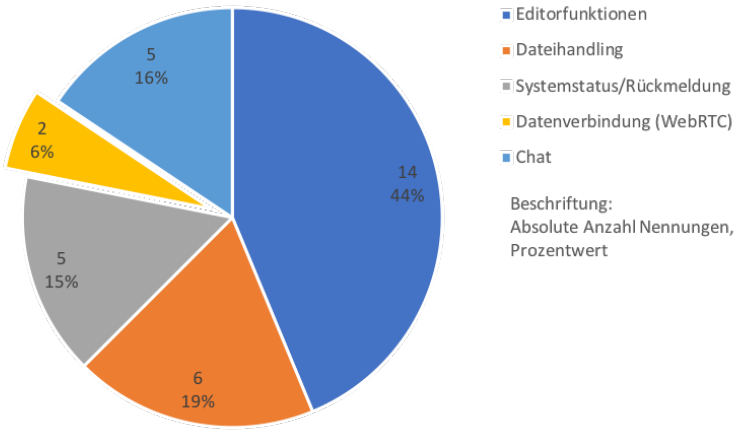

Abb. 7.2: Auswertung der Aussagen, gruppiert

Sechs der 14 Nennungen im Bereich Editor beziehen sich auf die Aussage, dass der Editor neben einer Kennung welche Texte verändert worden sind auch kenntlich machen sollte, wo sich der Schreibende gerade mit dem Cursor im Editor befindet um ihm folgen zu können. Diese Einschätzung wurde von allen Testern, in unterschiedlicher Ausprägung, angegeben. Die Einschätzung aus der Diskussion, dass der Editor hinter Desktop-Umsetzungen noch zurückbleibt, spiegelt sich auch in den Fragebögen wider.

Die meisten Einträge lassen sich als Wunsch nach einer Verbesserung eines Features oder als neues Feature einstufen. Ein häufig genannter Wunsch war, dass an mehreren Dateien parallel gearbeitet werden kann. Gelöst wurde dies im Test, indem mehrere Browser-Reiter geöffnet wurden und dort einzelne Sessions mit verschiedenen Dateien erstellt wurden. Von Einzelnen wurde der Umgang mit Dateien als umständlich eingestuft und ein schnellerer Export der Dateien gewünscht. Einige Punkte beziehen sich auf eine Verbesserung der Kommunikation über die Oberfläche, wie der Wunsch die Schreibfreigabe über ein Button in der Oberfläche anzufordern. Der Chat sollte mitteilen, dass eine Nachricht eingegangen ist und der Chatverlauf sollte nach Wiederaufnahme einer Verbindung nicht gelöscht werden. Die Frage nach der Konsistenz der Dialoge, Buttons, Sprache und Formulierungen wurden von allen bejaht und keine Unstimmigkeiten gefunden.

Datenverbindung / WebRTC

Der Punkt Datenverbindung insbesondere die WebRTC-Verbindung gilt es gesondert zu betrachten. Der Verbindungsaufbau zwischen zwei Chrome-Browsern funktionierte ohne Probleme, während der Versuch eine Verbindung mit dem Firefox-Browser herzustellen zunächst scheiterte, mit der Meldung keine ICE-Candidates finden zu können. Der Aufbau zwischen den Browsern Chrome und Firefox liefert in seltenen Fällen diesen Fehler und nur unter der Bedingung, dass der Firefox die Offer des Chrome-Browsers annimmt. Ein erneuter Verbindungsversuch ist dann meist wieder erfolgreich. Der Fehler als solches wurde nicht als kritisch eingestuft, die fehlende Rückmeldung der Oberfläche, dass eine Verbindung nicht hergestellt werden konnte, hingegen schon. So wurde verhindert, dass der Tester den Aufruf nochmal startete.

Die Tester überprüften im Editor Grenzfälle, z.B. ob die Synchronisation der Daten gestört oder verhindert werden kann. Längere Texte (im Test im Chrome größer als 64kb) wurden nicht mehr übertragen. Die Implementierung des RTCDataChannels berücksichtigt weder Komprimierung noch die verschiedenen Obergrenzen an Daten, die in den unterschiedlichen Browsern versendet werden können. D.h. durch die entsprechende Implementierung wäre auch die Übertragung längerer Textpassagen möglich, stellen aber in diesem Kontext Grenzfälle dar.

7.3 Zusammenfassung

Der Usability-Test hat gezeigt, dass die Aufgaben des AdHoc-Reviews problemlos bearbeitet werden konnten. Probleme wurden darin gesehen, dass Ereignisse nicht entsprechend kenntlich gemacht werden oder Meldungen fehlen. Der wichtigste Punkt in diesem Zusammenhang ist die fehlende Anzeige des Editors, wo aktuell editiert wird. Weitere Anmerkungen und Probleme bezogen sich hauptsächlich auf den Umgang mit einzelnen oder mehreren Dateien, da die FileAPI Einschränkungen mit dem Umgang mit Dateien hat. Abbildung 7.2 zeigt, dass nur wenige Angaben einen direkten Bezug auf die WebRTC-Technologie hatten. Die beiden angezeigten Ausnahmen zeigen, dass noch Probleme mit der Datenübertragung durch die unterschiedliche Implementierungen der Browser bestehen und die Implementierung optimiert werden muss. Alle angegebenen Probleme lassen sich aber grundsätzlich beheben und stellen kein Hinderungsgrund dar, die eingesetzten Technologien zu verwenden und die Applikation weiter zu verbessern.

Kapitel 8

Fazit und Ausblick

Der Entwicklungsprozess bestehend aus Anforderungsanalyse, Konzeption, Entwicklung des Prototypen, sowie die Überprüfung auf technische Eignung und Nutzbarkeit durch Software- und Usability-Tests hat abschließend betrachtet zu interessanten und umfangreichen Ergebnissen geführt.

Die Recherche hat gezeigt, dass sich die WebRTC-Technologie im Bereich Audio- und Videoanwendungen etabliert hat, während die Nutzung des WebRTC-DataChannels seltener ist und eher auf WebSockets in einer Client-Server Architektur gesetzt wird, vor allem wenn ohnehin bereits Backend-Systeme bestehen. Online-Editoren, in denen Quellcode gemeinsam bearbeitet werden kann, fokussieren sich häufig auf die Interview-Situation mit Bewerbern, d.h. es geht nicht zentral um die kollaborative Bearbeitung von Dateien, wie es in der hier entwickelten Lösung der Fall ist. Die unterschiedlichen Implementierungen der WebRTC-APIs und im Speziellen des RTCDataChannels in den Browsern scheinen zudem weiterhin eine Hürde darzustellen.

Ziel der Arbeit war es zu prüfen, ob die aktuell verfügbare WebRTC-Technologie mittlerweile geeignet ist einen realen Anwendungsfall umzusetzen, d.h. zum einen ging es um die technische Eignung der Technologie und zum anderen um die Frage, ob die Umsetzung ohne zentrale Server-Infrastruktur als nutzbar empfunden wird. Aus diesem Grund wurde ein Prototyp entwickelt, dessen Entwicklung von einer Expertengruppe begleitet wurde und der anhand von Softwaretests und Usability-Tests seine Eignung unter Beweis stellen sollte.

Durch eine Diskussionsrunde mit Experten und Entwicklern wurde zu Beginn ein einfaches Szenario entwickelt, dass als sinnvoll eingestuft wurde und Entwickler bei ihrer Arbeit unterstützt. Dies besteht darin spontane Online-Sessions zu erstellen, die einem Entwickler ermöglichen Unterstützung bei der Umsetzung von Quellcode zu erhalten, in dem abwechselnd kollaborativ am Text gearbeitet werden kann, um die Ergebnisse zu übernehmen und zu speichern. Das Szenario ist Grundlage für eine Liste von Anforderungen an die Applikation, die während der Umsetzungsphase durch regelmäßige Treffen überprüft und erweitert wurde,

so dass es möglich war während der Entwicklung umfangreiche Software-Tests durchzuführen um die technische Eignung zu überprüfen. Die Gebrauchstauglichkeit des funktionsfähigen Prototypen konnte abschließend durch einen Usability-Test bestätigt werden.

Es hat sich gezeigt, dass ein WebRTC-basiertes System nicht ohne Server auskommt. Zum einen müssen sich die Clients zunächst finden und benötigen deshalb für den initialen Aufbau der Verbindung einen Signaling-Server, der diese Aufgabe übernimmt. Zudem machen es die Übertragungsprotokolle notwendig einen externen STUN/TURN-Server in den Verbindungsaufbau mit einzubeziehen, da die externen IP-Adressen der Peers ermittelt werden müssen. Ist die Verbindung erst einmal hergestellt, werden die Daten direkt von Peer zu Peer übertragen.

Aufgrund der besonderen Architektur wurde das Projekt für die Umsetzung in drei Einzelprojekte aufgeteilt. Das Frontend besteht aus der Weboberfläche namens CoCoding und einer WebRTC-Bibliothek namens CoWebRTC, die sich um die WebRTC-basierte Kommunikation kümmert. Für den initialen Verbindungsaufbau wurde der Signaling-Server CoSocketSignaling auf WebSocket-Basis entwickelt. Die Entscheidung, das Frontend in die CoCoding-App und die WebRTC-Bibliothek aufzuteilen hat sich als praktische Lösung erwiesen, da das Testen der Bibliothek losgelöst von der Oberfläche effektiv und ohne Störfaktoren umzusetzen war. So konnte immer unterschieden werden, wo ein Fehler seine Ursache hatte. Die Weiterentwicklung, bspw. das Komprimieren der Daten, kann zukünftig im CoWebRTC-Projekt vorgenommen werden.

Die einfachste Umsetzung des RTCDataChannels hat sich für den Anwendungsfall als geeignet herausgestellt. Die funktionalen Tests bestätigen, dass die Technologie in der Lage ist eine verlässliche Verbindung aufzubauen. Auch die Usability-Tests haben gezeigt, dass die Technologie für den Anwendungsfall geeignet ist, aber die Implementierung noch optimiert werden muss. Für die Übertragung großer Dateien ist es notwendig die Daten zu komprimieren und aufzusplitten. Diese Funktionalität bringt WebRTC nicht mit, sie muss zusätzlich implementiert werden. Es hat sich gezeigt, dass die Interoperabilität bzgl. der Browser noch nicht optimal ist. Besonders die unterschiedliche Implementierung der Limits für die Datenübertragung erschweren die Verwendung von verschiedenen Browsern. Soweit man sich aktuell auf die gleichzeitige Verwendung eines Browsers desselben Herstellers einigt, sind keine Probleme zu erwarten. Trotzdem ist es notwendig, aufgrund der voranschreitenden Umsetzung der WebRTC-Spezifikation innerhalb der Browser, stetig Anpassungen an den eigenen entwickelten Systemen vorzunehmen. Laut W3C ist die Interoperabilität einer der wichtigste Punkte, die aktuell auf der Agenda stehen.

Der Usability-Test hat noch weitere interessante Ansätze gezeigt, wie die Applikation und allgemein die Entwicklung Browser-basierte Applikationen weiter vorangetrieben werden könnte. Zum einen wurden die Funktionen des Code-Editors mit denen von Desktop-Lösungen verglichen, die Vorteile mit sich bringen, sei es in der Verwaltung vollständiger Projekte, der Fehlererkennung oder Code-Completion. Browser-Lösungen können dies durch die Beschränkungen im Umgang mit Dateien nicht oder noch nicht leisten. Der Umgang mit Dateien und Projekten ist noch nicht optimal, aber auch hier sind APIs in der Entwicklung, die den Umgang zukünftig vereinfachen könnten, wie die FileAPI oder auch die FileSystemAPI, die noch im Draft-Status sind und weiterentwickelt werden.

CoCoding als Browser-basierter Online-Editor muss also Kompromisse eingehen. Die Vorteile sind die allgemeine Erreichbarkeit, keine notwendige Installation und im Zuge dessen besteht die einfache Möglichkeit regelmäßig Updates auszuliefern. Zudem werden Daten nur von Browser zu Browser übertragen und bleiben unter der Kontrolle des Nutzers. Eine Alternative ist der Ansatz eine Desktop-Lösung mit einer Kollaborationsfunktion auszustatten und so die Vorteile beider Welten zu kombinieren, soweit sich das Entwicklerteam auf einen Editor einigt.

Zudem ist die Professionalisierung der JavaScript-Entwicklung ein Punkt, der nicht außer Acht gelassen werden sollte. Es ist mit den aktuellen Frameworks und Umgebungen möglich komplexe Applikationen zu entwickeln, die strukturiert und umfassend testbar sind und bereits jetzt durch den Einsatz von Sprachen wie TypeScript oder durch zukünftige ECMAScript-Versionen weniger fehleranfällig werden.

Abschließend kann man sagen, dass die WebRTC-Technologie bereit für den produktiven Einsatz ist, wenn die Bereitschaft besteht, sich dauerhaft mit den Veränderungen und Anpassungen der Implementierungen in den Browsern auseinanderzusetzen. Der Einsatz von Adaptern und Polyfill-Dateien wird noch eine Weile notwendig sein, d.h. eigene Implementierungen müssen regelmäßig überprüft und angepasst werden.

Anhang A

Anhang

A.1 Inhalt der DVD

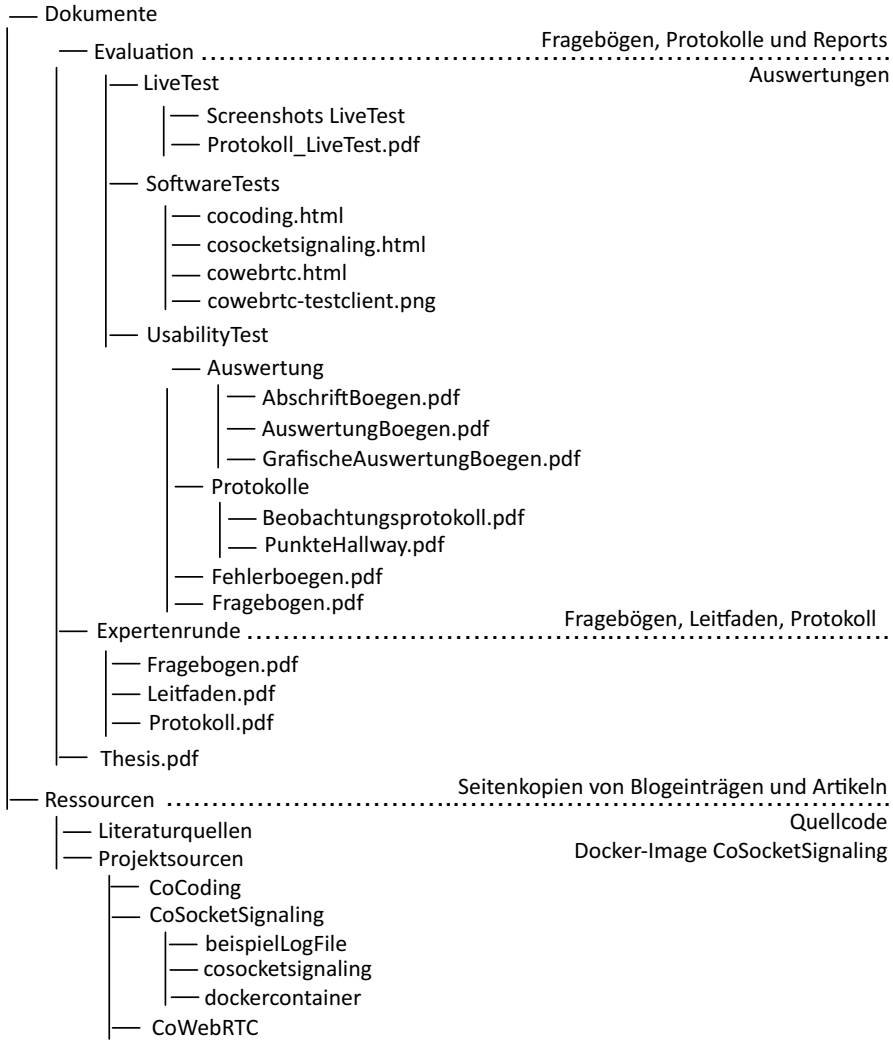

Abb. A.1: Inhalt der DVD

A.2 Expertenrunde

A.2.1 Diskussionsleitfaden

Leitfaden
Expertentreffen am 05.06.2018

Zusammenarbeit und Tools allgemein
- In welchen Situationen arbeitet ihr mit anderen Personen zusammen?
- Nutzt ihr Softwaretools und welche?

Code Reviews
- In welcher Form werden CodeReviews durchgeführt?
- Unterschiede im Büro/HomeOffice?
- Nutzt ihr Softwaretools und welche?
- Vorteile/Nachteile? Gründe warum oder warum nicht?

AdHoc CodeReviews/Pair-Programming
- Werden AdHoc CodeReviews durchgeführt?
- Wie müsste die Softwareunterstützung aussehen, damit sie hilfreich unterstützen kann?

Sicherheit/Cloud-Anbieter/Installation
- Ist es wichtig, dass Kommunikations/Collaborationsdaten nicht bei Cloudanbietern gespeichert werden?
- Ist es wichtig, dass Software nicht installiert werden muss?

A.2.2 Fragebogen

Fragebogen
Expertentreffen am 05.06.2018

Bitte beantworte abschließend kurz ein paar Fragen.

1. Was sind deine Aufgaben im Arbeitsalltag?

2. Gibt es einen Punkt, der Dir bei der Diskussion besonders wichtig war? Wenn ja, welcher?

3. Gibt es ergänzende Punkte, die bei der Diskussion nicht angeprochen wurden?
   Wenn ja, welche?

4. Ist es in Ordnung Dich für zukünftige Termine oder Tests (auch online) einzuladen?

_____ _____

Name Emailadresse

Vielen Dank für die Teilnahme.

A.3 Fragebogen Usability Test

Usability Evaluation

Expertentreffen am 18.09.2018

Ablauf

- WLAN: `cococo`, pwd: `cocoding`
- URL: `192.168.0.100`

WICHTIG: Bitte führe die Aufgaben unter Berücksichtigung der Punkte auf der zweiten Seite aus und protokolliere Unstimmigkeiten und Dinge, die Dir auffallen.

Aufbau der Verbindung

- Einer von euch ist der Besitzer der Session und startet sie.
  Nachdem die Session gestartet ist, teile den Link mit Deinem Testpartner.
  URL: `192.168.0.100/editor;id=<sessionname>`

Textbearbeitung

- Besitzer der Session:
  Bitte kopiere oder importiere einen Text von Deiner Festplatte, z.B. text, php, javascript
  (der Import hat aktuell noch eine Größenbeschränkung von 16kb)
- Gib den Text zur Bearbeitung frei
- Bearbeitet den Text abwechselnd
- Exportiere den Text (bitte anderen Dateinamen angeben als das Original!)

Chat

- Versendet Chatnachrichten

Name: _____ Browserversion: _____

- Sind alle Funktionen vorhanden, um die Aufgaben zu bewältigen?

- Werden reale Arbeitsbedingungen abgedeckt?

- Gibt das System ausreichend Rückmeldung?

- Ist das System konsistent bzgl. Dialoge, Buttons, Beschreibungen, Sprache und Formulierungen?

- Sind Daten und/oder Texte verloren gegangen?

- Fallen sonstige Punkte auf?

A.4 Softwaretest-Protokolle

A.4.1 CoSocketSignaling

CoSocketSignaling TestReport

Start: 12.11.2018 (09:04:25)

8 tests -- 8 passed / 0 failed / 0 pending

D1 - Konfiguration	**should: Server mit Standard-Konfiguration initialisieren**	passed in 0.005s
D1 - Konfiguration	**should: Server initialisieren und die Standard-Konfiguration überschreiben**	passed in 0s
D1 - Konfiguration	**should: Log Debug-Infos, wenn Aufruf Constructor**	passed in 0.002s
Session	**T1: Ein Client kann Session öffnen/schließen**	passed in 0.52s
Session	**T2: Ein zweiter Client kann beitreten**	passed in 0.505s
Session	**T3: Ein dritter Client kann nicht beitreten.**	passed in 0.507s
Session	**T4: Client1 kann Client2 eine Signaling Message senden**	passed in 0.505s
Session	**T5: Wenn Client Session verlässt => Rückmeldung an Client1**	passed in 0.508s

Abb. A.2: Testprotokoll CoSocketSignaling

A.4.2 CoCoding

CoCoding TestReport

Start: 12.11.2018 (15:05:41)

15 tests -- 15 passed / 0 failed / 0 pending

> Session	**T1: Client öffnet Session mit Btn-Neue Session**	passed in 2.486s
> Session	**T2: Client schliesst Session mit Btn-Schliessen Session**	passed in 2.434s
> Session	**T3: Session kann per URL beigetreten werden (DataChannel offen)**	passed in 4.245s
> Session	**T11: Wenn ein Client die Session verlässt wird eine SnackBar mit Meldung angezeigt.**	passed in 4.934s
> Session	**T13: Im Footer wird der Session-Status angezeigt.**	passed in 4.526s
> Session	**T15: Session des Initiators bleibt offen wenn Co-Worker Session schliesst.**	passed in 4.557s
> Kollaboration	**T7: Slider <Freigabe> sendet Editorfreigabe an den zweiten Client.**	passed in 5.434s
> Kollaboration	**T8: Geänderter Quelltext von Client1 wird an Client2 übertragen**	passed in 5.561s
> Kollaboration	**T9: Nach dem Absenden von Text im Chat wird er übertragen und angezeigt**	passed in 4.879s
> Kollaboration	**T14: Farbwechsel im Footer beim Ändern der Schreibrechte.**	passed in 4.51s
> Toolbar	**T4: Button <Teilen> öffnet Dialog. Die URL wird angezeigt.**	passed in 3.367s
> Toolbar	**T5: Button <Import> öffnet Dialog, Datei kann importiert werden.**	passed in 2.941s
> Toolbar	**T6: Button <Export> öffnet Dialog, Dateiname kann für Export angegeben werden.**	passed in 2.175s
> Toolbar	**T12: In der Toolbar ist ein ComboBox zum Umstellen der Sprache.**	passed in 2.16s
> Editor	**T10:Veränderungen im Editor werden seitlich mit einem Decorator markiert.**	passed in 2.817s

Abb. A.3: Testprotokoll CoCoding

A.4.3 CoWebRTC

CoWebRTC TestReport

Start: 12.11.2018 (10:22:59)

13 tests -- 13 passed / 0 failed / 0 pending

> Session-STUN	**T6: STUN Server Connect schreibt ext. IP in SDP String**	passed in 1.655s
> RTCDataChannel	**T4: Öffnen eines RTCDataChannels zwischen zwei Clients**	passed in 1.915s
> RTCDataChannel	**T5: Senden und Empfangen von Daten über den Datachannel**	passed in 1.975s
Session	**T1: Öffnen und Schließen der Session über Signaling (created/disconnected)**	passed in 0.416s
Session	**T2: Verbinden zweier Clients über Signaling (created/joined)**	passed in 0.405s
Session	**T3: Nach geöffneter Verbindung verlässt Client2 die Session, => Client1 Benachrichtigung (leave)**	passed in 0.405s
D1 - Schnittstellen	**schreibt bei der Initialisierung eine Info in das Logfile**	passed in 0.002s
D1 - Schnittstellen	**createConnection startet die Session (Aufruf createConnection)**	passed in 0.001s
D1 - Schnittstellen	**closeConnection beendet die Session (Aufruf closeConnection)**	passed in 0s
D1 - Schnittstellen	**sendMessage sendet eine Nachricht über Daten Kanal**	passed in 0.001s
D1 - Schnittstellen	**sendObject sendet eine Nachricht verpackt in einem Objectüber den Daten Kanal**	passed in 0.001s
D1 - Schnittstellen	**onMessage liefert die Nachrichten als Observer**	passed in 0s
D1 - Schnittstellen	**getBrowserData liefert die aktuellen Browser Informationen**	passed in 0.001s

Abb. A.4: Testprotokoll CoWebRTC

A.4.4 CoWebRTC: Test-Client

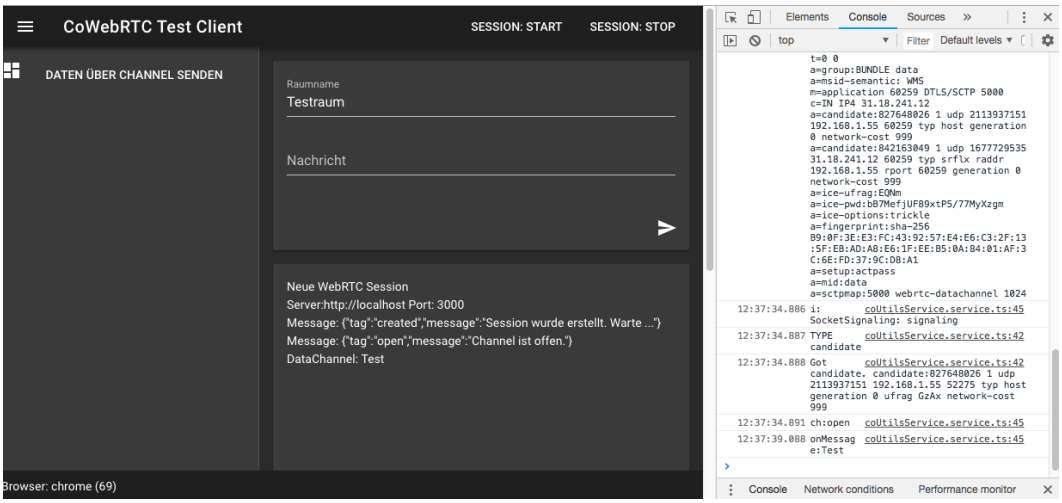

Abb. A.5: CoWebRTC: Test-Client

A.4.5 Protokoll Live-Test

Initiatorsicht Co-Worker-Aktionen

Nr	—> Beschreibung der Aktion, Erwartete Reaktion		ok?
L1	—> Aufruf der CoCoding WebApp und Bestätigung der Einleitungsseite. GUI vollständig geladen. Keine Fehler in der Konsole. Editor zeigt Quellcode mit Synntaxhighlighting		ja
L2	—> Öffnen der Session über den Button <Neue Session> Statusleiste: Session(id) offen. Warte auf Gesprächspartner …		ja
L3	—> Aufruf des Buttons <Teilen> und Kopieren der URL (über Link-Button) Dialog offen.Link wird angezeigt und kann in den Zwischspeicher gespeichert werden. —> Versenden der URL an Co-Worker, Warten auf Verbindung		ja
	—> URL in den Browser eingeben Statusleiste ist gelb: Gesprächpartner verbunden	L4	ja
L5	Statusleiste ist grün: Gesprächpartner verbunden, SDP Info in der Konsole zeigt externe IP-Adressen (siehe candidates)		ja
L6	—> Versenden einer Chatnachricht (Test) Chattext wird in die Liste übertragen Die Anzeige im Chat steht auf “online“		ja
	Chat-Text wird empfangen und in die Liste eingestellt —> Versenden einer Chatnachricht (Test)	-> L6	ja
L7	—> Warte auf Antwort Chat-Text wird empfangen und in die Liste eingestellt (andere Farbe, versetzt)		ja
L8	—> Freigabe des Editors über den Freigabeschieber Statusleiste wird gelb, Freigaberegler rot, Editor ist schreibgeschützt —> Warten auf Änderungen im Text		ja
	Statusleiste wird grün, kein Freigaberegler, Editor ist freigegeben —> Copy/Paste von “Hello Welt!“ in die zweite Zeile	L9	ja
L10	Text ändert sich von echo“Hello World!“ auf “Hello Welt!“ Am linken Rand wird ein Decorator angezeigt.		ja
L11	Der Editor hat Farbtemplate für php. —> Über die Combobox eine andere Sprache als php einstellen Die Keywords verlieren ihre Farbe.		ja
L12	—> Ping-Test starten per Klick auf das Chat-User Icon (20x) Durchschnittlicher Wert liegt nicht höher als 50ms, gut unter 30ms		ja 19,95
L13	—> Wegnahme der Freigabe über den Freigabeschieber Statusleiste wird grün, Freigaberegler grau, Editor ist freigegeben —> Warte, dass Gesprächspartner die Verbindung verlässt		ja
	—> Button <Schließe Session> Statusleiste: Session ist geschlossen	-> L15	ja
L14	Anzeige im Footer, dass der Gesprächspartner die Session verlassen hat. Statusleiste: Session(id) offen. Warte auf Gesprächspartner …		ja
L15	—> Button <Schließe Session> Statusleiste: Session ist geschlossen		ja
L16	—> Button<Export> aufrufen, Namen eingeben, Text Speichern Dialog OS -> Text wird auf HD gespeichert		ja
L17	—> Seite neu laden, Code Text ist wieder: Hello World! Button<Import> aufrufen., Dialog OS, Datei kann ausgewählt werden und wird in den Editor geladen, Code Text ist Hello Welt!		ja

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

[Ack16] Philip Ackermann. *JavaScript: Das umfassende Handbuch*. Rheinwerk Computing, 2016.

[ang18] *Angular.io - Architektur Overview*. [Online], Zugriff: 24.10.2018. 2018. URL: https://angular.io/guide/architecture.

[BA04] Kent Beck und Cynthia Andres. *Extreme Programming Explained: Embrace Change (2Nd Edition)*. Addison-Wesley Professional, 2004.

[Ber+17] A. Bergkvist u. a. *WebRTC 1.0: Real-time Communication Between Browsers. W3C Candidate Recommendation*. [Online], Zugriff: 14.08.2018. 2017. URL: https://www.w3.org/TR/webrtc/.

[Blo17] Mozilla.org Blog. *Large Data Channel Messages*. [Online], Zugriff: 24.10.2018. 2017. URL: https://blog.mozilla.org/webrtc/large-data-channel-messages/.

[Cal14] David Calhoun. *What is AMD, CommonJS, and UMD*. [Online], Zugriff: 15.10.2018. 2014. URL: https://www.davidbcalhoun.com/2014/what-is-amd-commonjs-and-umd/.

[can] *Can I Use*. [Online], Zugriff: 15.10.2018. URL: https://caniuse.com/.

[Cod] Google Codelabs. *Real Time Communication with WebRTC*. [Online], Zugriff: 20.09.2018. URL: https://codelabs.developers.google.com/codelabs/webrtc-web/.

[cod15] codegeekz.com. *12 Best Code Editors for Real Time Collaboration*. [Online], Zugriff: 12.09.2018. Apr. 2015. URL: https://codegeekz.com/12-best-code-editors-for-real-time-collaboration/.

[comp18] *React vs Angular vs Vue.js: A Complete Comparison Guide*. [Online], Zugriff: 31.10.2018. 2018. URL: https://medium.com/front-end-hacking/react-vs-angular-vs-vue-js-a-complete-comparison-guide-d16faa185d61.

[Con17] World Wide Web Consortium. *HTML5.2, W3C Recommendation*. Zugriff: 15.09.2018. 2017. URL: https://www.w3.org/TR/2017/REC-html52-20171214/.

[Con18] World Wide Web Consortium. *File API, W3C Working Draft, 6 Nov 2018*. Zugriff: 09.11.2018. 2018. URL: https://www.w3.org/TR/FileAPI/.

[dock18] *What is a Container?* [Online], Zugriff: 01.11.2018. 2018. URL: https://www.docker.com/resources/what-container.

[ecma] *ECMAScript compatibility table*. [Online], Zugriff: 15.10.2018. URL: http://kangax.github.io/compat-table.

[EGR91] Clarence A. Ellis, Simon J. Gibbs und Gail Rein. "Groupware: Some Issues and Experiences". In: *Commun. ACM* 34.1 (Jan. 1991), S. 39–58. URL: http://doi.acm.org/10.1145/99977.99987.

[Ele] Electron. *Build cross platform desktop apps with JavaScript, HTML, and CSS*. [Online], Zugriff: 10.06.2018. URL: https://electronjs.org/.

[Ell16] Eric Elliot. *Javascript Testing: Unit vs Functional vs Integration*. [Online], Zugriff: 31.08.2018. Apr. 2016. URL: https://www.sitepoint.com/javascript-testing-unit-functional-integration/.

[Evo] *The Evolution of the Web*. [Online], Zugriff: 14.08.2018. URL: http://www.evolutionoftheweb.com/.

[Fen12] Steve Fenton. *Compiling vs Transpiling*. [Online], Zugriff: 01.09.2018. 2012. URL: https://www.stevefenton.co.uk/2012/11/compiling-vs-transpiling/.

[Fis] Jim Fisher. *How does reliability work in RTCDataChannel?* [Online], Zugriff: 12.09.2018. URL: https://jameshfisher.com/2017/01/17/webrtc-datachannel-reliability.html.

[Flo] Floobits.com. *Floobits FAQ. Why is Floobits centralized?* [Online], Zugriff: 10.06.2018. URL: https://floobits.com/help/faq.

[FM11] I. Fette und A. Melnikov. *The WebSocket Protocol. RFC 6455*. 2011. URL: http://www.rfc-editor.org/rfc/rfc6455.txt.

[Fow18] Martin Fowler. *Inversion of Control Containers and the Dependency Injection pattern*. [Online], Zugriff: 24.10.2018. 2018. URL: https://martinfowler.com/articles/injection.html.

[Fra14] K. Franz. *Handbuch zum Testen von Web- und Mobile-Apps: Testverfahren, Werkzeuge, Praxistipps*. 2. Aufl. Xpert.press. [E-Book Version]. Springer Berlin Heidelberg, 2014.

[GIN15] P.L. Gorski, L.L. Iacono und H.V. Nguyen. *WebSockets: Moderne HTML5-Echtzeitanwendungen entwickeln*. Hanser, 2015.

[Goj18] GojkoAdzic. *Choosing the right JavaScript testing tool*. [Online], Zugriff: 31.08.2018. Feb. 2018. URL: https://gojko.net/2018/02/25/javascript-testing-tools.html.

[Gre11] Eli Grey. *Saving generated files on the client-side*. Zugriff: 09.11.2018. Juli 2011. URL: https://eligrey.com/blog/saving-generated-files-on-the-client-side/.

[Gri13] Ilya Grigorik. *High Performance Browser Networking*. Sebastopol, CA: OReilly Media, 2013.

[IETa] IETF. *Interactive Connectivity Establishment (ICE)*. [Online], Zugriff: 15.09.2018. URL: https://tools.ietf.org/html/rfc5245.

[IETb] IETF. *JavaScript Session Establishment Protocol (JSEP)*. [Online], Zugriff: 15.09.2018. URL: https://tools.ietf.org/html/draft-ietf-rtcweb-jsep-24.

[IETc] IETF. *SDP for the WebRTC*. [Online], Zugriff: 15.09.2018. URL: https://tools.ietf.org/id/draft-nandakumar-rtcweb-sdp-01.html.

[IETd] IETF. *Stream Control Transmission Protocol*. [Online], Zugriff: 15.09.2018. URL: https://tools.ietf.org/html/rfc4960.

[JB14] Alan B. Johnston und Daniel C. Burnett. *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*. 3. Aufl. USA: Digital Codex LLC, 2014.

[Khaa] Muaz Khan. *WebRTC Experiments & Demos*. [Online], Zugriff: 28.05.2018. URL: https://www.webrtc-experiment.com.

[Khab] Muaz Khan. *WebRTC Signaling*. [Online], Zugriff: 20.09.2018. URL: https://github.com/muaz-khan/WebRTC-Experiment/blob/master/Signaling.md.

[Kum18] Atul Kumar. *ngx-monaco-editor*. [Online], Zugriff: 01.09.2018. 2018. URL: https://github.com/atularen/ngx-monaco-editor.

[KWW] Byron Kask, Sarah Wood und Brett Williams. "Synchronous and Asynchronous Communication:Tools for Collaboration". In: *Etec510 Wiki* (). [Online], Zugriff: 14.08.2018. URL: http://etec.ctlt.ubc.ca/510wiki/Synchronous_and_Asynchronous_Communication:Tools_for_Collaboration.

[Lev13] Tsahi Levent-Levi. "4 Facts You Need to Know about P2P in WebRTC". In: *WebRTC, BlogGeek.me* (Sep. 2013). [Online], Zugriff: 01.06.2018. URL: https://bloggeek.me/4-p2p-webrtc-facts/.

[Lev18] Tsahi Levent-Levi. "What is WebRTC adapter.js and Why do we Need it?" In: *WebRTC, BlogGeek.me* (Jan. 2018). [Online], Zugriff: 01.11.2018. URL: https://bloggeek.me/webrtc-adapter-js/.

[Lin18] Sam Lin. *ngx-monaco-editor*. [Online], Zugriff: 01.09.2018. 2018. URL: https://github.com/maxisam/ngx-clipboard.

[LR14] Salvatore Loreto und Simon Romano. *Real-Time Communication with WebRTC. Peer-to-Peer in the Browser*. OReilly Media, 2014.

[Mar17] David Marcus. *Insanely Simple WebRTC Video Chat Using Firebase*. [Online], Zugriff: 20.09.2018. 2017. URL: https://websitebeaver.com/insanely-simple-webrtc-video-chat-using-firebase-with-codepen-demo.

[Mat] Angular Material. *Angular Material*. [Online], Zugriff: 15.09.2018. URL: https://material.angular.io/.

[Mos12] Christian Moser. *User Experience Design*. X.media.press. Berlin: Springer, 2012.

[Moza] Mozilla.org. *Cross-Origin Resource Sharing, CORS*. [Online], Zugriff: 01.09.2018. URL: https://developer.mozilla.org/de/docs/Web/HTTP/CORS.

[Mozb] Mozilla.org. *MDN web docs, Create RTCDataChannel*. Zugriff: 15.09.2018. URL: https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/createDataChannel.

[Mozc] Mozilla.org. *MDN web docs, HTML5*. Zugriff: 15.09.2018. URL: https://developer.mozilla.org/de/docs/Web/HTML/HTML5.

[Mozd] Mozilla.org. *MDN web docs, Media Capture and Streams API (Media Stream)*. Zugriff: 15.09.2018. URL: https://developer.mozilla.org/en-US/docs/Web/API/Media_Streams_API.

[Moze] Mozilla.org. *MDN web docs, RTCDataChannel.send()*. Zugriff: 15.09.2018. URL: https://developer.mozilla.org/en-US/docs/Web/API/RTCDataChannel/send.

[Mozf] Mozilla.org. *MDN web docs, RTCIceServer*. Zugriff: 15.09.2018. URL: https://developer.mozilla.org/en-US/docs/Web/API/RTCIceServer.

[Mozg] Mozilla.org. *MDN web docs, Using WebRTC data channels*. Zugriff: 01.10.2018. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Using_data_channels#Understanding_message_size_limits.

[MP14] Michael Mikowski und Josh Powell. *Single Page Web Applications: JavaScript End-to-end*. 1. Aufl. Shelter Island, NY, USA: Manning Publications Co., 2014.

[Pin+16] Nikos Pinikas u. a. *Extension of the WebRTC Data Channel Towards Remote Collaboration and Control*. Sep. 2016. URL: https://www.researchgate.net/publication/311312434_Extension_of_the_WebRTC_Data_Channel_Towards_Remote_Collaboration_and_Control.

[RM17] R.Langmann und M.Stiller. *Cloud-basiert steuern - aber wie?* [Online], Zugriff: 01.06.2018. Juni 2017. URL: https://www.computer-automation.de/steuerungsebene/steuern-regeln/artikel/142203/4/.

[SB06] Florian Sarodnick und Henning Brau. *Methoden der Usability Evaluation: Wissenschaftliche Grundlagen und praktische Anwendung*. 3. Aufl. Hogrefe Verlag, Bern, 2006.

[Sma18] SmartBear. *The State of Code Quality 2017. Trends & Insight into Dev Collaboration*. Zugriff: 04.05.2018. 2018. URL: https://smartbear.com/de/resources/ebooks/the-state-of-code-review-2017.

[Sob17] Nathan Sobo. *Code together in real time with Teletype for Atom*. [Online], Zugriff: 10.06.2018. Nov. 2017. URL: http://blog.atom.io/2017/11/15/code-together-in-real-time-with-teletype-for-atom.html.

[Spr16] Sebastian Springer. *Node.js: Das Praxisbuch*. 2. Aufl. Rheinwerk Verlag, 2016.

[Sta] Stackshare.io. *Discover & compare tech stacks*. [Online], Zugriff: 17.08.2018. URL: https://stackshare.io/collaboration.

[Sta18] Gernot Starke. *Effektive Software-Architekturen: Ein praktischer Leitfaden*. 8. Aufl. München: Hanser, 2018.

[Tog] Together.js/Mozilla. *Together.js Documentation. Technology Overview*. [Online], Zugriff: 16.08.2018. URL: https://togetherjs.com/docs/#technology-overview.

[Tok14] TokBox Blog. *WebRTC Data Channels vs WebSockets*. [Online], Zugriff: 01.06.2018. Sep. 2014. URL: https://tokbox.com/blog/webrtc-data-channels-vs-websockets/.

[tri] *WebRTC samples, Trickle ICE*. [Online], Zugriff: 01.09.2018. URL: https://webrtc.github.io/samples/src/content/peerconnection/trickle-ice/.

[umd] *UMD (Universal Module Definition) patterns for JavaScript modules that work everywhere.* [Online], Zugriff: 15.10.2018. URL: https://github.com/umdjs/umd.

[VE14] Christopher John Vitek und Pilip E. Edholm. *WebRTC for Enterprises: History and Use Cases*. USA: CreateSpace Independent Publishing Platform, 2014.

[wadap18] *WebRTC Adapter*. [Online], Zugriff: 01.11.2018. 2018. URL: https://github.com/webrtchacks/adapter.

[WebS] *WebRTC Samples*. [Online], Zugriff: 28.05.2018. URL: https://github.com/webrtc/.

[Wika] Wikipedia. *Ajax (programming)*. [Online], Zugriff: 01.06.2018. URL: https://en.wikipedia.org/wiki/Ajax_(programming).

[Wikb] Wikipedia. *Groupware*. [Online], Zugriff: 11.06.2018. URL: https://de.wikipedia.org/wiki/Groupware.

[Wikc] Wikipedia. *HTML5*. [Online], Zugriff: 01.06.2018. URL: https://de.wikipedia.org/wiki/HTML5.

[Wik18] Wikipedia. *Transmission Control Protocol*. [Online], Zugriff: 01.09.2018. 2018. URL: https://de.wikipedia.org/wiki/Transmission_Control_Protocol.

[Wor17] World Wide Web Consortium. *WebRTC becomes design-complete strengthening the Web Platform as a solid actor in the telecommunications arena*. [Online], Zugriff: 14.05.2018. 2017. URL: https://www.w3.org/2017/11/media-advisory-webrtc10-cr.html.en.

[Zai18] Vitali Zaidman. *An Overview of JavaScript Testing in 2018 – Welldone Software – Medium*. [Online], Zugriff: 31.08.2018. Feb. 2018. URL: https://medium.com/welldone-software/an-overview-of-javascript-testing-in-2018-f68950900bc3.