



**Hochschule für Technik
und Wirtschaft Berlin**

University of Applied Sciences

Groupware Editor zum gemeinsamen Arbeiten mit verschiedenen Werkzeugen auf einem Web-Whiteboard

Abschlussarbeit

zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)

an der

Hochschule für Technik und Wirtschaft Berlin
Fachbereich Wirtschaftswissenschaften II
Studiengang Angewandte Informatik

1. Prüfer: Prof. Dr. Burkhard Messer
2. Prüfer: Dipl.-Ing.(FH) Ingo Wiederoder

Eingereicht von Suat Secmen

Berlin, 21. September 2015

Einleitung	4
Inhalt und Aufbau der Arbeit	4
Motivation	4
Arbeitsmittel	5
Grundlagen	5
JavaScript	5
NodeJS	6
Canvas	6
Events	6
Git	7
SCSS (Sassy Cascading Stylesheet)	7
Analyse	7
Anforderungen	7
Aufgabe	8
Entwurf	9
Protokoll	9
Modulare Funktionen	10
Webinterface	11
Werkzeug und Werkzeugeinstellungen	12
Verlauf	12
Datum	13
Formen	13
Textwerkzeug	16
Stift- und Pinselwerkzeug	17
Radierwerkzeug	17
Tabellenwerkzeug	17
Auflösung	18
Temporäre Leinwände	19
Implementierung	19
Prototypen	19
Events	24
SCSS	24
Server	24
Dokumentation	26

Test	26
Einleitung	26
Leinwandtest	27
Werkzeugetest	27
Verlaufstest	28
Chat-Test	28
Servertest	29
Ergebnis	30
Literatur	34
Verzeichnisse	36
Glossar	36
Abkürzungen	37
Anhang	37
Eigenständigkeitserklärung	38

Einleitung

Inhalt und Aufbau der Arbeit

In meiner Abschlussarbeit möchte ich ein Whiteboard für den Webbrowser entwickeln. Es soll möglich sein, auf einer Zeichenfläche mit verschiedenen Werkzeugen zu zeichnen, Texte zu schreiben und verschiedene Formen und Objekte zu erstellen. Die erstellten Zeichnungen und Objekte sollen in Echtzeit auf anderen Geräten dargestellt und ergänzt werden, wodurch das gemeinsame Zeichnen und Planen ermöglicht wird.

Die Anwendung wird client- und serverseitig in der Sprache JavaScript entwickelt. Der Client und der Server kommunizieren über einen Websocket miteinander, wodurch die Anfragen und Antworten erst in Echtzeit realisiert werden können.

Es wird objektorientiert und modular programmiert, wodurch das Erweitern ohne Bearbeitung anderer Dateien möglich ist. Da ich meine Arbeit dokumentieren und zum Generieren der Dokumentation JSDoc benutzen möchte, bin ich gezwungen mich an Programmierstile zu halten, die von JSDoc unterstützt werden.

Einige Begriffe in dieser Abschlussarbeit enthalten hochgestellte Zahlen in eckigen Klammern. Die Idee stammt von großen Verlagen wie Springer und Thieme. Diese Zahlen dienen als Verlinkung zu den Quellenangaben von Wörtern, Wortgruppen, Sätzen und Absätzen und sind am Ende der Arbeit unter „Literatur“ aufzufinden.

In manchen Erklärungen im Glossar sind Ausdrücke wie „(hier)“ enthalten. Es dient als Hinweis, dass diese Begriffe verschiedene Bedeutungen haben können, jedoch in der Abschlussarbeit die Rede von einer ganz bestimmten Bedeutung ist. So kann ein „Server“ beispielsweise eine Hardware (ein Rechner) sein, jedoch wird der Begriff in dieser Abschlussarbeit ausschließlich im Sinne einer Software verwendet.

Alle Bilder und Grafiken wurden, falls nicht anders angegeben, selbsterstellt.

Ich programmiere und entwickle gerne auf Englisch. Aus diesem Grund werden meine Prototypen, Funktionen und Methoden, Variablen und Attribute, Kommentare und die Dokumentation in der Anlage auf Englisch sein. Um die Konsistenz der Abschlussarbeit beizubehalten werden die Namen hier auf Deutsch erwähnt.

Motivation

Kollaborative Echtzeitsoftware fand ich schon immer hochinteressant. Mit Seiten wie Google Docs^[1] und Cloud9^[2] arbeite ich sehr gerne (auch zu meiner HTW-Zeit um gemeinsam mit anderen Kommilitonen Mitschriften zu einem Fach zu sammeln oder für Prüfungen zu lernen), wo das gleichzeitige Arbeiten auf demselben Dokument ohne Überschreibungen und Konflikten mit anderen Benutzern möglich war. Erfahrung mit Websockets und Groupware hatte ich nicht und deshalb konnte ich mir nicht vorstellen, wie so etwas in Echtzeit funktionieren konnte. An

der Hochschule für Technik und Wirtschaft kam ich durch ein kleines, freiwilliges Spiel¹ und später durch ein DJ-Programm² zum ersten Mal mit NodeJS in Kontakt und habe gelernt, kleine Websocketserver zu programmieren.

Meine erste Idee für die Abschlussarbeit war ein kleines Spiel zu entwickeln. Nach einem kurzen Gespräch mit Herrn Messer wurde mir empfohlen, eine Groupware zu erstellen. Die Aufgabe ein Echtzeit-Whiteboard zu entwickeln fand ich sehr ansprechend und herausfordernd.

Arbeitsmittel

An der Abschlussarbeit wird nahezu ausschließlich von Zuhause aus gearbeitet. Ich besitze einen iMac mit einem Zweitbildschirm. Meinen Editor habe ich meistens auf dem iMac-Bildschirm geöffnet und auf dem Zweitbildschirm einen Browser (Google Chrome oder Safari) und einen Web Inspector. Außerdem arbeite ich, falls ich mal nicht am Schreibtisch bin, mit meinem MacBook Pro mit Retina-Display. Damit das Arbeiten mit zwei verschiedenen Geräten erst möglich ist, ohne dass ich mir selbst den aktuellsten Stand zusenden muss, arbeite ich mit Git und einem privaten Repository auf BitBucket (bitbucket.org). Der Editor meiner Wahl auf meinen beiden Macs ist „Coda 2“ von Panic^[3] für OS X. Dieser Editor kombiniert den Editor unter anderem mit einem FTP-Client, einem Terminal, einer Datenbankverwaltung (SQL) und einer Versionsverwaltung. Trotz der integrierten Versionsverwaltung in Coda habe ich diese Funktion selten benutzt, da sie weniger Funktionalität bietet und fehlerhaft ist. Aus diesem Grund nutze ich für die Versionierung mit Git die App „Tower“^[4] für OS X (ein GUI für Git).

Grundlagen

JavaScript

JavaScript ist eine Skriptsprache, die ursprünglich für Webbrowser entwickelt wurde um statische HTML Seiten dynamisch und interaktiv zu machen und die HTML-Struktur (das DOM) zu bearbeiten, nachdem der Server das HTML Dokument bereits an den Client gesendet hat (und der Server keinen weiteren Einfluss mehr auf das Dokument hat). Heute wird JS auch u. a. für Server verwendet (z. B. mit NodeJS).

JS ist eine prototypenbasierte, also eine objektorientierte, aber klassenlose Programmiersprache. Mit Prototypen ist es möglich den Konstrukturfunktionen Methoden (Funktionen) und Attribute (Variablen) zuzuweisen. Die Konstrukturfunktionen werden mit dem „new“ Operator instanziiert, wodurch dem Objekt alle Prototypmethoden übergeben werden.^{[5][6]}

¹ Freiwilliges Spiel „PixelCannon“ (www.pixelcannon.de) u. a. entwickelt von mir für das Fach „Webentwicklung“ (Semester 4).

² Belegaufgabe „HTWebJay“ (www.htwebjay.ml) u. a. entwickelt von mir für das Fach „Audio- und Videotechnik“ (Semester 5).

Einer der beliebtesten und bekanntesten Bibliotheken für JS ist jQuery.^[7] jQuery ist eine schnelle und kleine Bibliothek, wodurch das Manipulieren vom HTML Dokument, das Ausführen von AJAX Anfragen, das Hinzufügen und Entfernen von Events und vieles mehr mit viel weniger Code und viel einfacher umgesetzt werden kann.^[8]

NodeJS

NodeJS ist eine Plattform, mit der JS-Entwickler Webserver erstellen können. Durch die Installation und Implementierung von Modulen ist es JS möglich, mit dem Dateisystem zu arbeiten, also u. a. Dateien auszulesen und anzulegen, eine Verbindung zu Datenbanken herzustellen und diese zu verarbeiten und bearbeiten, Websockets zu erstellen und vieles mehr. Das Inkludieren von anderen JS-Dateien ist in JS normalerweise nicht möglich. Da jedoch Node serverseitig arbeitet und einen direkten Zugriff zur Datenstruktur hat (sie also im Gegensatz zum clientseitigen JS nicht herunterladen muss) liefert Node eine „require“ Funktion, wodurch das Einbinden und Ausführen von Modulen ermöglicht wird.^[9]

Das Node-Modul „socket.io“ ermöglicht es ganz einfach mit wenigen Zeilen einem Port zu lauschen und einen Websocket zu erstellen. Das Modul bietet u. a. sogenannte „Channels“^[10] (Kanäle/Räume), wodurch man nach Zuweisung von Clients in die erwähnten Channels Informationen Channel-separiert broadcasten und somit jedem anderen Client im selben Channel zusenden kann. Auf diese Weise kann man z. B. einen Chat mit verschiedenen Räumen erstellen, wo die Benutzer separiert miteinander kommunizieren können, ohne andere Räume zu behindern.

Canvas

Das HTML Canvas-Element ist eine „Leinwand“, worauf man (üblicherweise mit JS) zeichnen kann. Canvas hat etliche Methoden zum Zeichnen von Pfaden (Linien), Rechtecken, Kreisen (nicht zu verwechseln mit Ellipsen), Texten und Bildern. Außerdem liefert es Methoden zum Klären (Leeren) und Auslesen der Leinwand. Durch das Abhören der Maus (siehe „Events“) ist es dadurch möglich, beispielsweise eine Kurve zu zeichnen, indem bei Bewegung der Maus eine neue Linie zum aktuellen Standpunkt des Mauszeigers zeichnet.

Events

Events bzw. Ereignisse werden von Systemen oder Benutzern ausgelöst. Das Drücken einer Taste oder das Bewegen der Maus würde beispielsweise in JS das KeyDown-^[11] oder MouseMove-Event^[12] auslösen. Durch das Abhören von solchen Events ist es möglich Verzögerungsfrei die entsprechenden Aktionen auszuführen, wie z. B. das Verschieben von Objekten, indem man auf ein Objekt mit der Maus drückt und mit gedrückter Maustaste die Maus bewegt.

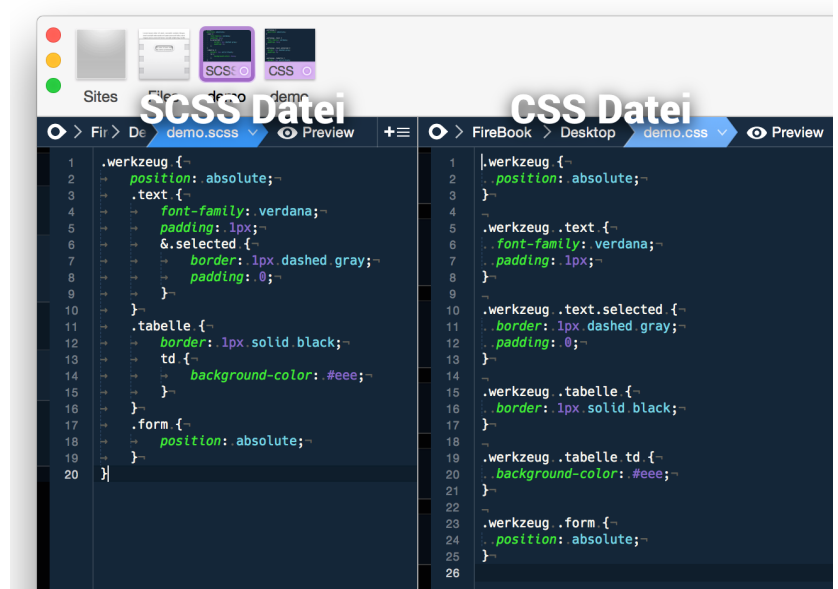
Git

Git ist eine kostenlose und offene Versionsverwaltungssoftware. Sie ermöglicht es Dateien an eine Repository zu senden, wobei nur die Veränderungen an der Datei gespeichert werden. Dadurch ist es möglich, den ganzen Verlauf von Projekten zu protokollieren, ohne dass die Repository zu groß wird. Außerdem ermöglicht es die Zusammenarbeit mit Anderen, ohne dass jemand den gerade selbstgeschriebenen Quelltext überschreibt. Falls es doch zu Konflikten kommen sollte, da mehrere Benutzer an derselben Datei an derselben Stelle etwas geändert haben, so wird es dem Benutzer angezeigt, der den Code eines anderen Benutzers überschrieben „hätte“ (also dem letzteren Benutzer). Der Benutzer hat nun die Möglichkeit Hand anzulegen und die Konflikte zu lösen.

Git erlaubt es außerdem noch durch sog. „Branches“ vom Projekt abzuschweifen und beispielsweise neue Funktionalitäten zu testen und im Notfall alles zu verwerfen und zurückzusetzen.

SCSS (Sassy Cascading Stylesheet)

SCSS ist eine Style-Sprache mit einer sehr ähnlichen Syntax wie CSS. Da es CSS an Funktionen wie Verschachtelungen und Variablen fehlt, was der Übersicht schaden kann, wird für diese Abschlussarbeit SCSS benutzt, welches mehr Funktionalitäten bietet und mit einem kürzeren Code längere CSS Dateien generiert. Unter „Arbeitsumfeld“ wurde erwähnt, dass mit Coda 2 für OS X gearbeitet wird. Dieser Editor ermöglicht das Installieren von Plugins. Durch das SASS-Plugin^[13] für Coda 2 werden beim Abspeichern von .scss Dateien die .css Dateien sofort automatisch generiert und ebenfalls abgespeichert.



Analyse

Anforderungen

Der Client benötigt eine große Leinwand, auf der man u. a. mit Werkzeugen wie Stift und Pinsel, einem Textwerkzeug, einem Tabellenwerkzeug und Formwerkzeugen zeichnen und Objekte erzeugen kann. Außerdem soll es möglich sein, einige Objekte wieder zu verschieben und zu entfernen. Es soll möglich sein, dass multiple Clients zur selben Zeit ohne Konflikte zeichnen können. Wenn jedoch beispielsweise zwei Clients dasselbe Objekt zur selben Zeit verschieben möchten, so soll es nur dem ersten ermöglicht werden.

Eine Versionierung ermöglicht es, die letzten Aktionen rückgängig zu machen und wiederherzustellen. Jeder Benutzer sollte jedoch nur seine eigenen Zeichnungen rückgängig machen können.

Durch einen kleinen Chat ist es möglich, mit anderen Benutzern zu kommunizieren.

Da es viel interessanter ist, jemandem beim Zeichnen zusehen zu können, als die Zeichnung erst zu sehen, nachdem der andere Client fertig gezeichnet und die Maustaste losgelassen hat, muss die Zeichnung während man gerade am Zeichnen ist temporär gebroadcastet werden. Dies muss ebenfalls für Texte (während man tippt) und für Formen (während man z. B. ein Rechteck zieht) realisiert werden.

Aufgabe

Der Server sollte so wenig wie möglich tun, damit er sehr zuverlässig und schnell arbeiten kann und so wenig wie möglich verarbeiten oder berechnen, sondern stattdessen die Informationen von einem Client zu den anderen Clients weiterreichen soll. Jeder Client sollte nun die erhaltenen Informationen selbstständig validieren und verarbeiten.

Da jedoch sehr wahrscheinlich jeder Client eine andere Uhrzeit hat und dies zu Problemen bei der Ordnung von Zeichnungen führen wird („welche Zeichnung war zuerst da? Die Eigene oder die des Anderen?“) muss der Server jedem Client seine Uhrzeit mitteilen. Um außerdem noch sicherzustellen, dass sich kein Benutzer als ein anderer ausgibt, werden die Benutzer-IDs vom Server übergeben. Eine obligatorische Aufgabe für den Server wäre eine Broadcast-Funktion. Dies ermöglicht es beim Erhalt einer Nachricht von einem Client diese Nachricht an alle anderen Clients zu senden, außer an den Client, von dem die Nachricht kam.

Sobald die Verbindung eines Clients zum Server unterbrochen wurde und alle anderen Clients den Server verlassen, sollte es dem Client nach nachträglicher Verbindung trotzdem möglich sein, den letzten Stand der Zeichnung zu sehen. Deshalb sollte der Server eine Grafik des letzten Stands besitzen.

Der Server wird jeden Benutzer bei einer unterbrochenen Verbindung darauf hinweisen. Auf diese Weise ist es beispielsweise möglich, einen Benutzer sofort nachdem er das Browserfenster schließt, im Chat als abwesend anzuzeigen.

Schließlich sollte es noch „Räume“ geben. Je nach Raum-ID kommuniziert man mit anderen aus demselben Raum. Dadurch werden Konflikte mit anderen Teams verhindert. Der Client ermittelt die Raum-ID durch die URL (die ihm beispielsweise vom Team zugeschickt wurde) und teilt es dem Server beim Verbinden mit. Beim Verbinden ohne Raum-ID wird dem Benutzer eine zufällige Raum-ID zugewiesen und der Benutzer kann diesen nun weiterleiten. Das Node Modul „socket.io“ stellt bereits einiges dieser Funktionalitäten zur Verfügung (so werden die Benutzer-IDs für alle Benutzer bereits generiert und die Implementierung von Räumen (Channels) und das Broadcasten ist ebenfalls ohne viel Aufwand anwendbar).

Der Client sollte alle Zeichnungen, die er erhält, sofort nach Erhalt nachzeichnen. Es werden nicht viele Funktionen für das Verarbeiten von Antworten vom Server benötigt. Beim Client

liegt die Herausforderung in der Umsetzung der Werkzeuge und der Effizienz (besonders bei vielen verbundenen Benutzern, die gleichzeitig zeichnen).

Eines der wichtigsten und professionellsten Bedingungen wäre meiner Meinung nach die modulare Entwicklung. Es soll einfach möglich sein, beispielsweise ein weiteres Werkzeug hinzuzufügen, ohne andere Dateien bearbeiten zu müssen. Da manchmal die ganze Leinwand neu gezeichnet werden muss, muss der Verlauf in richtiger Reihenfolge abgearbeitet werden und, da alle Werkzeuge modular sind, das richtige Werkzeug ausgewählt und ausgeführt werden, ohne dass es vom Verarbeiter fest einprogrammiert (hardcoded) wurde. Dies kann durch eine einmalige Registrierung jedes Werkzeugs vor der Benutzung realisiert werden.

Entwurf

Protokoll

Zur Kommunikation zwischen Server und Client wird das Node und JS Modul socket.io verwendet. Alle gesendeten Informationen bestehen aus einem Event-Namen und den Daten. Wenn die gesendeten Daten aus einem String bestehen, werden sie als String an den Server gesendet und auch als String von den Clients empfangen. Bestehen die gesendeten Daten jedoch aus einem Array oder einem JS Objekt, so werden sie in einen JSON-String umgewandelt und anschließend versendet.^[14]

Da der Server so wenig Arbeit wie nur möglich haben soll, gibt es nur wenige Event-Namen: „benutzerId“: Die Benutzer-ID sendet der Server an den Client gleich nach der erfolgreichen Verbindung.

„url“: Die URL (Adresse) sendet der Server dem Client, wenn für den Client ein neuer Raum generiert wurde.

„verbunden“ und „getrennt“: Das sendet der Server allen Clients im selben Raum, sobald ein Benutzer die Verbindung mit dem Server hergestellt oder unterbrochen hat.

„datum“: Das sendet der Client zuerst dem Server mit der eigenen lokalen Uhrzeit. Danach sendet der Server dem Client seine eigene Uhrzeit. Auf diese Weise ist es möglich, die Differenz zu berechnen und der eigenen Uhrzeit immer draufzuschlagen oder abzuziehen.

„*“: Das ist der Broadcast-Event-Name. Den sendet der Client dem Server und der Server leitet die Nachricht direkt an alle anderen Clients im Raum weiter. Die Clients enthalten die Broadcasts ebenfalls mit dem „*“ Event-Namen.

„fehler“: Für jegliche entstandene Fehler kann der Server den Client mit dem „fehler“-Protokoll darauf hinweisen. Dies wird beispielsweise benötigt, wenn der Benutzer auf ein Raum zugreifen möchte, was gar nicht (mehr) existiert.

Die Daten der Events müssen einem festen Protokollschema entsprechen. Bei „benutzerId“ und bei „url“ kommt die Antwort, da nur ein einziger Wert erwartet wird, als String an. Die Fehlermeldung „fehler“ kann als Zahl gesendet werden. Wenn der Client jede Fehlermeldung mit dem Fehlercode kennt, so ist in Zukunft möglich, dass jeder Client verschiedene

Fehlermeldungen anzeigen kann (z. B. in verschiedenen Sprachen). Außerdem spart es Datenverkehr, eine Zahl zu senden, statt einer langen Fehlermeldung.

Beim „getrennt“ wird ebenfalls ein String gesendet, dieser enthält die Benutzer-ID der getrennten Person.

Das Datum („datum“) wird zuerst vom Client an den Server gesendet. Dieser enthält die aktuelle clientseitige Uhrzeit als Zeitstempel (JS liefert hierbei die vergangenen Millisekunden seit dem 01. Januar 1970 00:00:00 UTC zurück^[15]). Der Server wird nun die eigene Zeit als „serverzeit“ und die empfangene Zeit als „clientzeit“ an den Client zurücksenden. Dadurch kann der Client die genaue Serverzeit berechnen, auch wenn die Anfrage etwas länger gedauert hat.

Beim Broadcast („*“) wird immer ein Objekt erwartet. Dieser enthält immer einen „typ“ Wert. Verschiedene Objekte (wie z. B. Werkzeugobjekte oder das Cursorobjekt) können sich dann beim Serverobjekt registrieren wodurch eine bestimmte Methode von diesen Objekten ausgeführt wird, sobald deren „typ“ empfangen wird. Auf diese Weise wird das Projekt modular und erweiterbar.

Modulare Funktionen

JS ist eine klassenlose prototypenbasierte Sprache, jedoch ist es möglich, mit dem „new“ Operator Objekte von Funktionen zu erstellen. Durch Prototypen kann man nun den Objekten noch „Methoden“ (Funktionen) geben. Auf diese Weise kommt man zu ähnlichen Ergebnissen wie Sprachen mit Klassen. Wie in Sprachen mit Klassen sollen unter anderem alle Werkzeuge jeweils in eigene JS Dateien verlagert werden. Die Werkzeuge sollen sich automatisch überall registrieren, solange man sich an den Bauplan eines Werkzeugs (wie bei einem „Klasseninterface“) hält. Zum Registrieren wäre es nötig, einem Hauptobjekt ein Objekt des Werkzeugs zu übergeben und das Werkzeug in die HTML Datei einzubinden, damit es automatisch ausgeführt wird. Durch die Einhaltung an den Bauplan eines Werkzeugs ist es nun möglich, in Routinen das Werkzeug automatisch zu benutzen, wenn beispielsweise die Leinwand neu gezeichnet werden muss. Außerdem kann das Werkzeug automatisch ausgeführt werden, wenn man ein Broadcast empfängt, wenn also ein anderer Benutzer mit dem Pinselwerkzeug etwas malt, dass dann automatisch die Pinselfunktion ausgeführt wird. Neben den Werkzeugen bekommen auch u. a. der Verlauf, die Leinwandverarbeitung, der Server und die Werkzeugverarbeitung (z. B. für die Werkzeugleiste) eine eigene Datei mit einer klassenähnlichen Strukturierung (Konstruktorfunktion und Prototypmethoden). Da manche Prototypen Stile (CSS) benötigen sollte es möglich sein, die Stile Dateigetrennt hochzuladen und somit das Bearbeiten von anderen Stildateien zu verhindern.

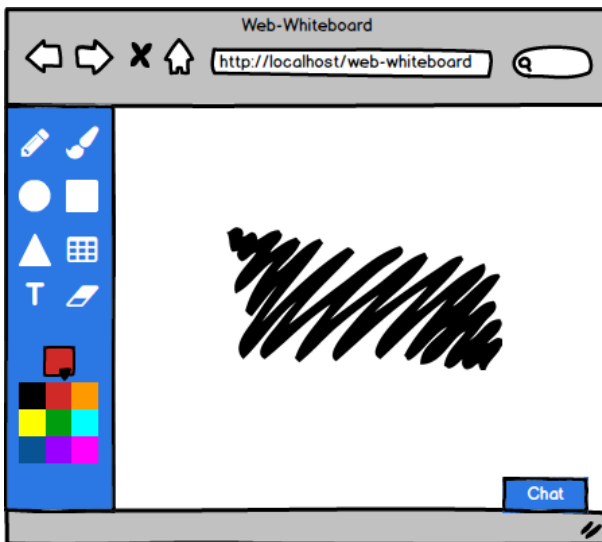
Webinterface

Der Client besteht Größtenteils aus einer Zeichenleinwand. Auf der linken Seite ist eine kleine Seitenleiste zu sehen. Diese beinhaltet die Werkzeuge, dynamische Werkzeugeinstellungen je nach Auswahl eines Werkzeugs (z. B. die Pinselgröße), die Funktionen „Rückgängig“ und „Wiederherstellen“ und eine Farbpalette.

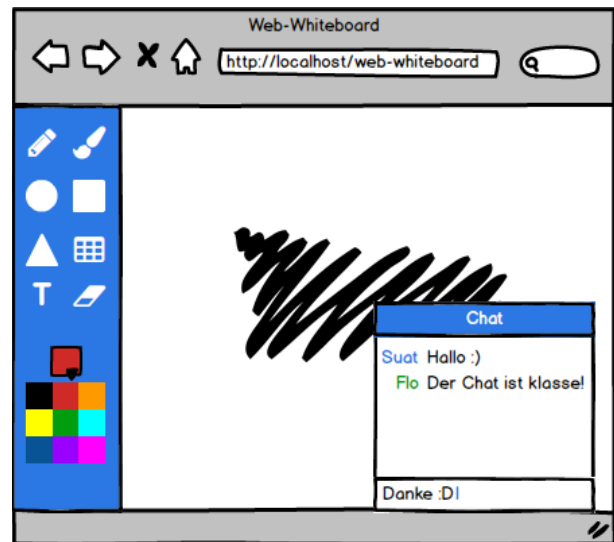
Die Leinwand sollte scrollbar sein, damit Benutzer auf kleineren Bildschirmen nicht benachteiligt werden und nur einen Bruchteil des Bildes sehen können. Die maximale Leinwandgröße sollte begrenzt werden, um viel zu große Bilder zu verhindern und dadurch die Leistung zu gefährden.

An einer unteren Ecke ist eine kleine Leiste zu sehen, die bei einem Klick sich zu einem kleinen Chat öffnet. Dieser Chat besteht aus einem Eingabefeld und dem Chatverlauf. Der Chat soll wieder minimiert werden können, um das Zeichnen nicht zu behindern.

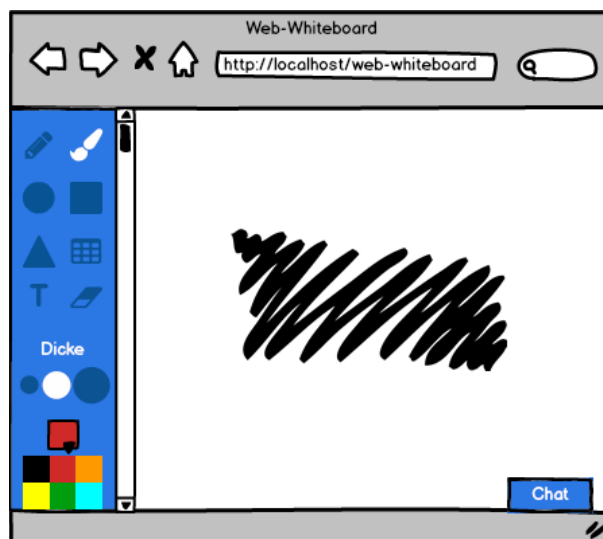
Hier einige Mockups, erstellt mit Balsamiq³:



geschlossener Chat



offener Chat



Auswahl eines Werkzeugs, Darstellung der Werkzeugeinstellungen.

³ Balsamiq (<http://balsamiq.com>) ermöglicht das Erzeugen von Mockups.

Werkzeug und Werkzeugeinstellungen

Werkzeuge sollen automatisch in die Seitenleiste hinzugefügt werden, ohne die HTML-Datei zu oder eine fremde Funktion zu bearbeiten. Außerdem soll es möglich sein, Werkzeugen Einstellungen zu geben. So kann z. B. einem Pinsel- oder Radierwerkzeug die Größe und einem Rechteck- und Dreieckwerkzeug der Abrundungswert gegeben werden.

Da alle Werkzeuge eine eigene Funktion mit Prototypen besitzen, können durch ein lokales Objekt in der Werkzeugkonstruktorfunktion Informationen zum Werkzeug hinterlegt werden. Auf diese Weise kann man einer Werkzeugfunktion ebenfalls multiple Werkzeuge zuweisen, da beispielsweise der Stift, der Pinsel und der Radierer auf gleiche Weise funktionieren, kann alles von einer einzigen Werkzeugfunktion verarbeitet werden. Das Rechteck, die Ellipse und das Dreieck funktionieren ebenfalls sehr ähnlich, deshalb gilt hier das Gleiche. Das „multidimensionale“ JS Objekt besteht nun in der obersten Ebene aus dem Werkzeugnamen (z. B. „stift“ und „pinsel“). Dieses beinhaltet nun ein weiteres Objekt mit dem „symbol“ für die Seitenleiste (z. B. ein Bild), optional gefolgt von den „einstellungen“, was ebenfalls ein Objekt ist. Dieses beinhaltet den Namen der Einstellung (z. B. „pinselgröße“) mit einer „beschriftung“ und einem „typ“. Es gibt einige vordefinierte Typen, wie „schaltflächen“, „kleineSchaltflächen“, „regler“, „kontrollkästchen“ und andere benötigte Elemente. Je nach „typ“ muss das „einstellungen“ Objekt weitere Werte beinhalten, wie z. B. beim „regler“ den „minimalwert“ und „maximalwert“.

Die Werkzeugeigenschaften werden alle automatisch beim initialisieren des Werkzeugs, bzw. die Einstellungen beim Klick auf das Werkzeug verarbeitet. Auf diese Weise ist es schnell und sauber möglich, Werkzeugen Einstellungen geben, ohne HTML in JS zu benutzen oder mit redundantem Code zu arbeiten.

Verlauf

Um die Funktionen „Rückgängig“ und „Wiederherstellen“ umsetzen zu können, wird ein „Verlauf“ benötigt. Dieser beinhaltet in der richtigen Reihenfolge alle Schritte, die ein Benutzer unternommen hat (z. B. Zeichnung eines Rechtecks, Erstellen eines Textfeldes, Verschieben des Textfeldes). Da jeder Benutzer Schritte rückgängig machen kann, ist es auch nötig, dass jeder Benutzer die Schritte von jedem Benutzer im Verlauf protokolliert hat. Auf diese Weise kann ein Benutzer einfach nur die Aktion „rückgängig“ broadcasten und durch die „benutzerId“ ist es nun jedem möglich, den letzten Schritt dieses Benutzers „vorübergehend“ zu löschen. Dabei kann dem Verlaufsobjekt einfach ein „rückgängig“ Wert übergeben werden, welches beim „Wiederherstellen“ wieder entfernt wird.

Es wäre ineffizient die ganzen Verläufe von jedem Benutzer zu protokollieren. Es würde sehr viel Speicher verbrauchen und beim Zeichnen ruckelig werden, weshalb die maximale Anzahl an rückgängig machbaren Schritten begrenzt werden muss.

Datum

Da jeder Client eine leicht abweichende Uhrzeit im System eingestellt hat muss der Client die Uhrzeit des Servers holen und mit dieser Zeit weiterrechnen. Bei verschiedenen Zeiten würde es zu Fehlern beim Nachzeichnen von Verläufen kommen, so würden Zeichnungen eines Clients, dessen Zeit eine Minute vorgeht, eine Minute lang über jeder anderen Zeichnung liegen (man könnte die Zeichnung also eine Minute lang nicht überdecken). Aus diesem Grund sollte die Zeit des Servers so genau wie möglich ermittelt werden.

Der Plan ist, dass der Client seine eigene Zeit ermittelt, diesen an den Server sendet, der Server keinerlei Berechnungen durchführt, sondern die empfangene Zeit mit der eigenen Zeit nun zurück an den Client sendet.

Der Client ermittelt nun wieder seine Zeit. Die Differenz der gesendeten Zeit und der aktuellen Zeit ist der „Ping“ und die Hälfte des Pings ist die geschätzte Latenz.

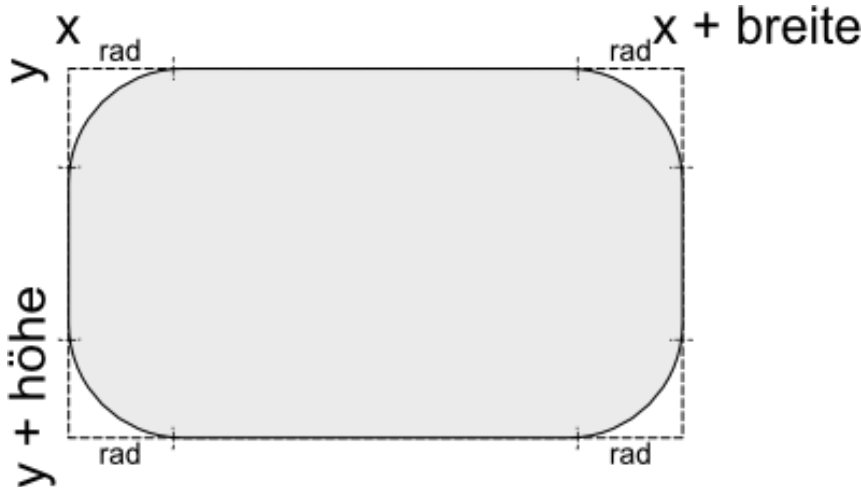
Wenn man nun die empfangene Serverzeit mit der Latenz zusammenaddiert erhält man die geschätzte Serverzeit.

Formen

Canvas stellen bereits einige Funktionen zum Zeichnen von Formen bereit. So kann man beispielsweise mit der Funktion „`fillRect`“ und den Parametern `x`, `y`, Breite und Höhe ein Rechteck auf der Leinwand zeichnen. Leider dient es jedoch nur zum Zeichnen von Rechtecken ohne abgerundete Ecken. Das Zeichnen von abgerundeten Rechtecken ist mit dieser Funktion nicht möglich. Außerdem fehlt es dem Canvas auch an einer Ellipsenfunktion. Es ist nur möglich, einen Kreis mit einem Radiuswert zu zeichnen. Eine Funktion zum Zeichnen von Dreiecken gibt es gar nicht. Aus diesem Grund müssen die Formen mit Pfaden selbst gezeichnet werden.

Abgerundete Rechtecke:

Gegeben ist ein Abrundungswert (rad) von 0px bis beispielsweise 200px (je nach Benutzereinstellung). Das Rechteck wird von den Mausclickkoordinaten bis zu den Mausloslasskoordinaten gezeichnet. Durch die beiden Koordinaten ist es nun möglich, die Breite und die Höhe des Rechtecks zu ermitteln ($Breite = |x1 - x2|$ und $Höhe = |y1 - y2|$). x und y können nun jeweils den geringeren Wert bekommen ($x = \min(x1, x2)$ und $y = \min(y1, y2)$).



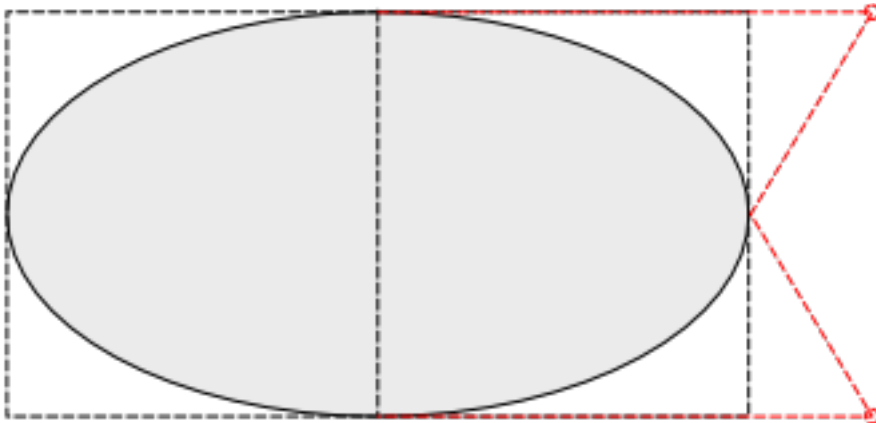
Um die Abrundungen zu zeichnen gibt es in Canvas die „quadraticCurveTo“ Funktion. Als Parameter werden zuerst die Kontrollpunktkoordinaten und dann die Koordinaten angegeben, bis wohin die Kurve gezeichnet werden soll.

Die erste Linie wird nun von $[x + rad, y]$ bis $[x + Breite - rad, y]$ gezeichnet (obere mittlere Linie). Von dort aus wird eine Kurve gezeichnet. Als Kontrollpunktkoordinaten werden $[x + Breite, y]$ angegeben und als Zielkoordinate $[x + Breite, y + rad]$. Die nächste Linie geht bis $[x + Breite, y + Höhe - rad]$ und die nächste Kurve hat die Kontrollpunktkoordinaten $[x + Breite, y + Höhe]$ und geht bis $[x + Breite - rad, y]$. Es folgt die untere mittlere Linie, die bis $[x + rad, y]$ geht und dann die Kurve mit den Kontrollpunktkoordinaten $[x, y]$ und den Zielkoordinaten $[x, y + Höhe - rad]$. Schließlich noch die Linie bis $[x, y + rad]$ und die letzte Kurve mit dem Kontrollpunkt $[x, y]$ bis $[x + rad, y]$.

Wenn der Abrundungswert (rad) 0 ist, so können die Kurven übersprungen werden oder gleich die zuvor erwähnte „fillRect“ Funktion benutzt werden.

Ellipsen:

Um Ellipsen zu zeichnen kann mit der „bezierCurveTo“ Funktion gearbeitet werden. Dieser benötigt im Gegensatz zur „quadraticCurveTo“ Funktion zwei Kontrollpunktkoordinaten.



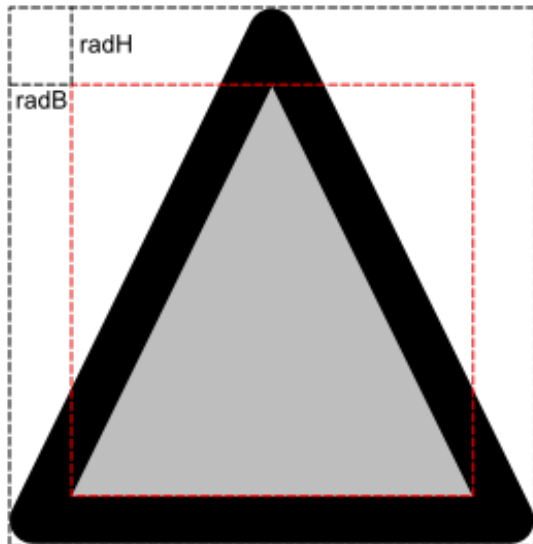
Die erste Kurve wird von $[x + \text{Breite} / 2, y]$ (die obere Mitte) bis $[x + \text{Breite} / 2, y + \text{Höhe}]$ (die untere Mitte) gezeichnet. Dabei werden die Kontrollpunkte auf die Koordinaten $[x + \text{Breite} / 2 + \text{Breite} * 2 / 3, y]$ und $[x + \text{Breite} / 2 + \text{Breite} * 2 / 3, y + \text{Höhe}]$ gesetzt. Auf diese Weise entsteht eine halbe Ellipse. Die andere Seite der Ellipse wird mit derselben Formel gezeichnet, jedoch wird diesmal die $\text{Breite} * 2 / 3$ vom Mittelpunkt ($x + \text{Breite} / 2$) abgezogen statt hinzuaddiert.

Dreieck:

Um ein Dreieck auf einer Leinwand zu zeichnen wird eine Linie von $[x + \text{Breite} / 2, y]$ bis $[x + \text{Breite}, y + \text{Höhe}]$ gezogen (die Linie geht runter, dadurch entsteht ein Dreieck, wo die Spitze oben ist). Die zweite Linie wird bis $[x, y + \text{Höhe}]$ gezogen und schließlich das Dreieck mit der Linie bis $[x + \text{Breite} / 2, y]$ geschlossen. Durch das Beobachten der Maus kann man ermitteln, ob die Maus nach oben oder nach unten gezogen wurde. Auf diese Weise ist es möglich, je nach Bewegungsrichtung die Dreiecksspitze oben oder unten zu zeichnen. Um das Dreieck umzukehren müsste man die y-Koordinaten vertauschen, d. h. alle Koordinaten mit y ersetzen durch $y + \text{Höhe}$ und alle Koordinaten mit $y + \text{Höhe}$ ersetzen durch y .

Ein abgerundetes Dreieck zu zeichnen ist mathematisch etwas komplexer. Aus diesem Grund habe ich mir überlegt, dem Dreieck eine Umrandung zu geben. Man kann der Leinwand ausrichten, dass man die Linien „abgerundet“ verbinden möchte (`lineJoin = "round"`). Wenn man nun die Füllung des Dreiecks um den Abrundungswert (`rad`) verkleinert und dann dem Dreieck eine Umrandung in derselben Farbe wie die Füllung und einer Dicke von `rad` übergibt, so entsteht ein abgerundetes Dreieck mit viel weniger Arbeit.

Durch einen kurzen Test konnte ich herausfinden, dass das Ergebnis meiner Meinung nach am besten wirkt, wenn man zwei verschiedene Abrundungswerte hat, einen für die Höhe und einen für die Breite. Der Benutzer wählt einen Wert zwischen 0% und 10% aus und $radB$ (für die Breite) wird dann ermittelt durch $Breite \text{ des Dreiecks } * \text{ Prozentwert}$ und $radH$ (für die Höhe) durch $Höhe \text{ des Dreiecks } * \text{ Prozentwert}$. Die Dicke der Umrandung wird durch $Mittelwert \text{ von Breite und Höhe } * \text{ Prozentwert}$ ermittelt.

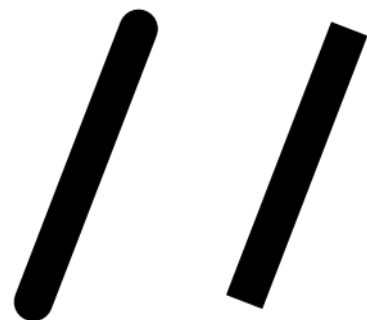


Linie:

Linien auf Leinwände zu zeichnen ist die Grundfunktionalität und man muss daran nicht viel ändern. Im Gegensatz zu den anderen Formen muss für Linien nicht einmal die Breite und die Höhe ermittelt werden. Durch die Klickkoordinaten und Loslasskoordinaten ist es sehr einfach möglich, eine Linie zu zeichnen. Um dem Benutzer mehr Freiheit zu bieten kann man ihm zwei Optionen für die Linien bieten. Die wichtigere Linienbreite um dünne und dicke Linien zu zeichnen und den eher subtileren Linienverbindungsstil. Wie bereits beim Dreieck erwähnt ist es möglich, den Stil von Verbindungen zu ändern. So kann man Linien eckige, spitze und runde Verbindungskanten geben. Da bei einer einzigen Linie ohne jegliche Winkel eine eckige und eine spitze Kante denselben Effekt haben benötigt der Benutzer nur zwei Optionen.

abgerundet

eckig



Textwerkzeug

Da das Textobjekt nicht nur einmal erzeugt und nach der Erzeugung nicht weiter bearbeitbar sein soll, sondern der Text weiterhin geändert werden können soll, darf das Textobjekt nicht in das Canvas-Element gezeichnet werden, sondern muss als HTML-Element zur Verfügung stehen. Auf diese Weise kann man, wenn das richtige Werkzeug ausgewählt ist, mit einem Klick auf das Textfeld das Element „auswählen“, was durch eine gestrichelte animierte

Umrandung dargestellt wird. Nun ist das Bearbeiten des Textes wieder möglich. Da durch die Umsetzung durch das HTML-Element das Textfeld (in der Z-Koordinate) „über“ dem Canvas-Element sein würde, würden Zeichnungen auf dem Text nicht möglich sein, die Zeichnung würde hinter dem Text erscheinen. Dies könnte man in Zukunft umgehen, indem man Texte „rasterisieren“ kann, wodurch das Textelement in die Leinwand gezeichnet wird und das Element aus dem HTML-DOM gelöscht wird.

Stift- und Pinselwerkzeug

Beim Pinselwerkzeug wird beim Klick auf die Leinwand die Koordinate gemerkt und angefangen die Mausbewegung zu beobachten. Sobald die Maus bewegt wird, wird von der Klickkoordinate bis zur Bewegungskordinate eine Linie gezeichnet. Bei der nächsten Mausbewegung wird die Linie von der letzten Koordinate bis zur neuen Mausposition gezogen. Da bei einer Bewegung viele Zwischenschritte registriert werden, werden viele kurze Linien gezeichnet. Dadurch entsteht der Eindruck einer runden Kurve. Der einzige Unterschied zwischen dem Stift- und dem Pinselwerkzeug ist die Dicke der Linie.

Radierwerkzeug

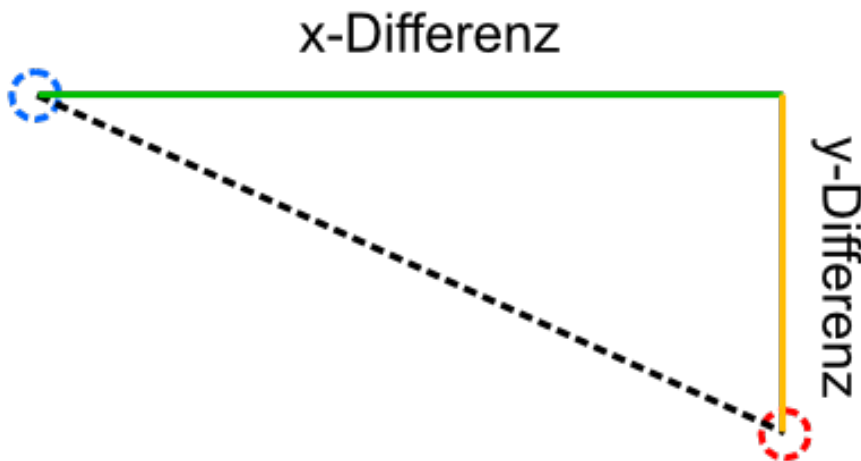
Die Leinwand bietet keine Funktion, um zirkuläre Flächen zu löschen. Die einzige Möglichkeit Flächen zu löschen benötigt die Koordinaten und die Dimensionen, um rechteckförmig die angegebene Stelle zu löschen. Eine Möglichkeit wäre es, mit der „clip“-Funktion eine kreisförmige Stelle der Leinwand zu definieren und dann mit der rechteckförmigen Löschfunktion die Stelle zu löschen. Da die clip-Funktion den Rest der Leinwand schützt, würde nur die kreisförmige Stelle gelöscht werden.^[16]

Ich halte diese Möglichkeit bei einem Radierwerkzeug, was beim Ziehen der Maus mit der clip-Funktion einen kreisförmigen Bereich auswählt und löscht, für ineffizient. Aus diesem Grund habe ich mir überlegt, einfach die Hintergrundfarbe der Leinwand zu nehmen und in der Farbe über die Zeichnung drüber zu malen. Für diese Funktion kann man einfach dieselbe Funktion wie für das Pinselwerkzeug nehmen, ignoriert jedoch die Auswahl der Farbe in der Werkzeugleiste und malt immer in weiß.

Tabellenwerkzeug

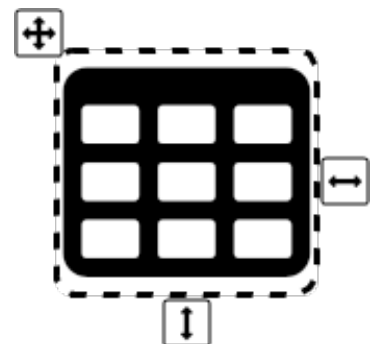
Die wohl einfachste Möglichkeit eine Tabelle zu erzeugen, ist das `<table>` HTML-Element zu verwenden. Nachdem man einen Rechteck gezogen hat (wie auch bei den Formen), wird eine Tabelle erzeugt, die anfangs nur eine Zelle (d.h. eine Zeile mit einer Spalte) hat. Nun könnte man beim Auswählen der Tabelle Funktionsschaltflächen anzeigen, womit man weitere Zeilen und weitere Spalten erzeugen kann. Jedoch wäre eine alternative und interessante Methode zum Erzeugen von neuen Zeilen und Spalten das Ziehen von Linien. Je nachdem, ob man die Linie horizontal oder vertikal zieht, „spaltet“ man die Tabelle auf und erzeugt eine neue Zeile oder Spalte. Zum Ermitteln, ob die Linie nun horizontal oder vertikal gezogen wurde, benötigt man die Klick- und Loslasskoordinaten. Aus den beiden Koordinaten kann nun die Richtung der

Linie ermittelt werden. Es wäre nun möglich, den Winkel zu berechnen, jedoch ist die einfachere Lösung gleichzeitig die effizientere. Man berechnet nur die Differenz der beiden x- und y-Koordinaten. Ist die x-Differenz größer als die y-Differenz, so wurde eine horizontale Linie gezeichnet. Ansonsten ist es eine vertikale Linie.



Der blaue Kreis (oben links) symbolisiert die Klickposition, der rote Kreis (unten rechts) die Loslassposition. Die grüne Linie (oben) hat eine Länge von 110px, die orange Linie (rechts) eine Länge von 50px. Da die x-Differenz (die grüne Linie) größer ist als die y-Differenz (orange Linie), würde eine horizontale Linie gezeichnet werden, wodurch die Tabelle in zwei Zeilen aufgeteilt werden würde.

Um nachträglich (nachdem die Tabelle erzeugt wurde) die Tabelle zu verschieben oder zu skalieren (die Größe zu ändern), kann die Tabelle „ausgewählt“ werden. Bei einem Klick auf die Tabelle (ohne die Maus zu bewegen und somit eine Linie zu ziehen und die Tabelle aufzuteilen) wird die Tabelle als „ausgewählt“ markiert und bekommt eine animierte, gestrichelte Umrandung, wie beim Textelement. Neben der Umrandung wird um das Element drei Halterungspunkte erstellt, wobei eines sich rechts mittig, eines sich unten mittig und eines sich oben links befindet. Der rechte Halterungspunkt dient zum Verändern der Breite, die untere zum Veränderung der Höhe und die obere linke zum Verschieben des Tabellenelements.



Auflösung

Da beispielsweise Smartphones und einige Apple Produkte, wie der iMac 5k und das MacBook Pro Retina eine höhere dpi haben, sollte die Leinwand doppelt so groß erstellt als dargestellt werden. Dies erreicht man, indem man dem Canvas-Element via HTML eine doppelte Breite und Höhe gibt (z.B. 2000px), via CSS jedoch die Breite und die Höhe wieder auf die gewollte Größe zurücksetzt (z.B. auf 1000px). Beim Zeichnen von Linien und Formen auf der Leinwand

muss man nun alle Koordinaten und Dimensionen verdoppeln, da die Leinwand eigentlich doppelt so groß ist, wie sie dargestellt wird und beispielsweise die Mauskoordinaten nicht mehr den tatsächlichen Koordinaten der Leinwand entsprechen.

Temporäre Leinwände

Beim Zeichnen von Linien merkt sich die Leinwand die letzte Position wohin gezeichnet wurde und von diesen Koordinaten aus wird die Linie fortgesetzt. Diese letzte Position setzt man mit der „moveTo“ Funktion. Auf diese Weise weiß die Leinwand von wo die Linie gezeichnet werden soll.

Wenn jedoch nun zwei oder mehr Benutzer gleichzeitig auf ein und derselben Leinwand zeichnen würden, so würde der eine Benutzer beim Zeichnen einer Linie die Linie vom anderen Benutzer fortsetzen. Beim gleichzeitigen Zeichnen würde eine Zickzack-Zeichnung entstehen, wobei Linien zwischen der Linie von einem Benutzer zur Linie vom anderen Benutzer gezogen werden würden. Um dies zu verhindern müssen temporäre Leinwände erzeugt werden. Diese dienen nur dafür die Zeichnung von anderen Benutzern darzustellen, bis der andere Benutzer die Maus losgelassen und somit die Zeichnung beendet hat. Nachdem die Linie fertiggezeichnet wurde kann sie auf die Hauptleinwand ohne Unterbrechung mit nur einem Durchlauf gezeichnet werden, wodurch es keine Kollisionen mehr mit der Zeichnung des Benutzers geben kann.

Die temporären Leinwände werden ebenfalls für das Zeichnen von Formen benötigt. Während man Formen zeichnet, soll die Form beim Bewegen der Maus jedes Mal neu gezeichnet werden, um während des Ziehens mit der Maus die Form bereits sehen zu können und nicht erst beim Loslassen der Maus. Hierfür muss man die Leinwand bei jeder Bewegung der Maus löschen und die Form neu zeichnen. Wenn man dies auf der Hauptleinwand tun würde, müsste man nach dem Löschen des Bildes der Leinwand die gesamte Zeichnung wiederherstellen.

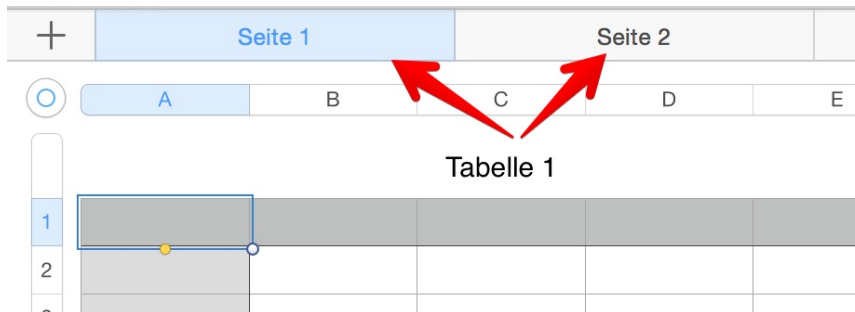
Implementierung

Prototypen

Hauptprototyp:

Es wurde ein Hauptprototyp erzeugt, welches Referenzen zum Serverprototypen, Werkzeugverarbeitungsprototypen, Leinwandverarbeitungsprototypen und Verlaufsprototypen beinhaltet. Außerdem ist es möglich, Werkzeuge beim Hauptprototypen zu registrieren. Auf diese Weise ist jedes Werkzeug bekannt und kann in Zukunft in Routinen berücksichtigt, verarbeitet, abgefragt und aufgerufen werden.

Da das Objekt des Hauptprototypen Objekte aller anderen Prototypen beinhaltet, ist es möglich, multiple Objekte des Hauptprototypen zu erzeugen und multiple Zeichenflächen zu erstellen. Dadurch könnte man mit wenig Aufwand in Zukunft so eine Funktion einprogrammieren, was Microsoft Excel und Numbers (die Tabellenverarbeitungssoftware von Apple) mit den multiplen Tabellen und Seiten bietet:



Serverprototyp:

Der Serverprototyp stellt beim Initialisieren eine Verbindung zum Server her. Außerdem stellt es Funktionen, Objekte und Variablen zur Verfügung, womit man die eigene Benutzer-ID abfragen, die Serverzeit ermitteln und Daten broadcasten kann. Darüber hinaus ist es anderen Prototypen möglich, eine Funktion in das „broadcast“-Objekt oder in die „verbunden“- und „getrennt“-Arrays zu registrieren, wodurch eine beliebige Funktion aufgerufen werden kann, sobald man ein Broadcast empfängt oder ein Benutzer die Verbindung zum Server herstellt oder diese trennt. Schließlich muss man noch ein Objekt erweitern, was wie eine Art „enum“ arbeitet. Somit ist es möglich, mit einer Variable zu arbeiten, die bei jedem Client denselben Wert hat und diese statt einer längeren Zeichenkette zu broadcasten. So muss jeder Client bei dem Broadcasttypen „3“ beispielsweise wissen, dass das Tabellenwerkzeugobjekt gemeint ist. Da das Aufzählungsobjekt bei jedem Eintrag um eins erweitert wird, hat jeder Schlüssel einen einzigartigen Integer wodurch das Objekt eindeutig identifiziert werden kann.

Leinwandverarbeitungsprototyp:

Dieser Prototyp enthält Referenzen zu den dauerhaften und temporären Canvas-Elementen. Er leitet beim Empfangen von temporären Zeichnungen, also während man gerade am Zeichnen ist, bei Formen und Pinsel z. B. ohne dass man die Maus losgelassen hat, und beim Fertigstellen von Zeichnungen, also bei Formen und Pinsel z. B. sobald die Maus losgelassen wurde, die Aktionen an das entsprechende Werkzeug weiter. Das Werkzeug, was streng nach Regel aufgebaut wurde (wie bei einem Interface) wird nun aufgefordert die empfangenen Daten zu rekonstruieren. Bei temporären Zeichnungen werden temporäre Leinwände (Canvas-Elemente) erzeugt. Diese Funktionen werden ebenfalls vom Leinwandverarbeitungsprototypen bereitgestellt.

Des Weiteren ist der Prototyp dafür verantwortlich die entsprechenden Werkzeuge aufzurufen, wenn die gesamte Zeichnung neu gezeichnet werden muss. Der

Leinwandverarbeitungsprototyp stellt ebenso die Rückgängig- und Wiederherstellenfunktionen bereit und injiziert die Schaltflächen in die Werkzeugleiste.

Werkzeugverarbeitungsprototyp:

Der Prototyp erzeugt und stellt eine Werkzeugleiste dar und enthält Referenzen zu den HTML Elementen Seitenleiste, Farben, Werkzeugen und Werkzeugeinstellungen. Er ist dafür verantwortlich die Werkzeuge und Farben in die Seitenleiste zu integrieren und bei Werkzeug- oder Farbauswahl die Veränderung darzustellen (z. B. die Umrandung der richtigen Farbe) und die entsprechenden Objekte aufzurufen. Es ist anderen Objekten, wie z. B. den Werkzeugobjekten möglich, das Werkzeugverarbeitungsobjekt nach der aktuell gewählten Farbe, nach dem gewählten Werkzeug und nach der Werkzeugseitenleistenbreite zu fragen. Das Objekt verarbeitet beim Auswählen von Werkzeugen die Werkzeugeinstellungen und bindet sie unter den Werkzeugen in der Werkzeugleiste ein.

Wenn das Werkzeugobjekt eine optionale „farbeGewechselt“ Funktion besitzt so wird diese mit der gewählten Farbe als Parameter automatisch aufgerufen, sobald die Farbe gewechselt wurde. Dies ist nützlich, wenn man z. B. ein zuvor erstelltes Textelement ausgewählt hat und die Farbe wechselt.

Verlaufsprototyp:

Der Verlaufsprototyp hat ein Objekt worin alle Schritte mit dem exakten Zeitstempel und der Benutzer-ID hinterlegt wird. In dem Objekt werden alle Parameter hinterlegt die zum Rekonstruieren des Schritts benötigt werden (beim Verschieben eines Textelements z. B. die neuen Koordinaten, beim Zeichnen einer Kurve den ganzen Verlauf der Mausbewegung einschließlich der Farbe und Werkzeugeinstellungen). Wenn nun nach einem Internetausfall von einem anderen Benutzer ein Schritt stark verspätet ankommt, so wird es dank des Zeitstempels an die richtige Position des Objekts hinzugefügt. Durch das Neuzeichnen wird nun die Reihenfolge beachtet und die Zeichnung erscheint über den alten und unter den neuen Zeichnungen.

Der Verlaufsprototyp enthält außerdem noch das wichtige Verlaufstypobjekt. Registrierte Werkzeuge werden dort, wie bei einem „enum“, mit einem eindeutigen Integer hinterlegt und durch die Frage nach dem Integer kann nun das Werkzeugobjekt ermittelt werden. Somit kann beim Empfangen von Broadcasts mit dem Verlaufstyp-ID durch das Verlaufstypobjekt das Werkzeugobjekt ermittelt und die erforderlichen Funktionen ausgeführt werden.

Diverse Prototypen:

Ich habe ein Verzeichnis erzeugt, worin diverse Prototypen mit kleineren selbstständigen Funktionalitäten hineingetan werden. Diese Prototypen werden weder vom Hauptprototypen noch von anderen Prototypen initialisiert oder verwendet.

Eines der sonstigen Prototypen ist der Mauszeigerprototyp. Er arbeitet komplett eigenständig und benötigt nur das Serverobjekt zum Broadcasten und Abfangen von Events. Die Aufgabe

des Mauszeigerprototypen ist das Anzeigen, Verschieben und Ausblenden des Mauszeigers der anderen Benutzer. So wird ein Bild eines Mauszeigers mit dem Namen des Benutzers bei allen anderen Benutzern angezeigt, wenn der Benutzer seine Maus bewegt und das Bild wird ausgeblendet, sobald der Mauszeiger das Fenster verlässt oder die Verbindung unterbrochen wurde („getrennt“-Event des Serverprototypen).

Ein weiterer sonstiger Prototyp ist der Auswahlprototyp. Durch das Erstellen eines Auswahlobjekts und das Übergeben von entweder einem HTML Element oder gewünschten Koordinaten und Dimensionen ist es möglich, einen animierten Rahmen zu erzeugen. Dieser symbolisiert die aktuelle Auswahl eines Elements. Die einzigen Methoden die dieser Prototyp zur Verfügung stellt ist das Aktualisieren der Position und der Dimension und das Entfernen des Objekts und Elements.

Hallo Welt!

Der zurzeit letzte diverse Prototyp ist der Chat. Auch der Chat arbeitet unabhängig von anderen Prototypen und benötigt nur den Server zum Senden und Empfangen von Broadcasts. Es ist möglich, den Chat zu minimieren oder wieder zu vergrößern und da der Chat bei der Leinwand einige Maus-Events registriert, ist es möglich, den Chat beim Zeichnen kurzfristig auszublenden und beim Loslassen der Maus wieder einzublenden. Auf diese Weise wird kein Verlassen der Leinwand registriert, wenn man mit der Maus beim Zeichnen auf den Chat raufrollt und wird somit nicht beim Zeichnen behindert, außerdem vergrößert das Ausblenden des Chats während man zeichnet die Sicht auf die Leinwand.

Werkzeugprototypen:

Um ein neues Werkzeug zu erzeugen muss man nur eine neue JS Datei (und optional, falls benötigt, eine SCSS Datei) einbinden, die den Bauregeln eines Werkzeugprototypen folgt. Es muss keine andere Datei bearbeitet bzw. erweitert werden. Jeder Werkzeugprototyp kann multiple auf ähnliche Weise funktionierende Werkzeuge erzeugen. So kann ein Formprototyp ein Rechteck, eine Ellipse und einen Dreieck erzeugen, wobei ein Zeichenprototyp einen Stift, einen Pinsel und einen Radierer zur Verfügung stellen kann.

Der neue Werkzeugprototyp benötigt in der Konstruktorfunktion ein Objekt für die Werkzeugeinstellungen. Diese Einstellungen beinhalten den Namen aller Werkzeuge als Schlüssel mit einem weiteren Objekt als Wert. Dieses Objekt wiederum enthält das Symbol und optional Einstellungen für das Werkzeug, welche in der Seitenleiste dargestellt werden, sobald das Werkzeug ausgewählt wurde. Neben den Werkzeugeinstellungen wird ebenfalls ein Objekt benötigt, was die eingestellten Werte der Einstellungen beinhaltet. Dieses Objekt definiert gleichzeitig auch die Standardwerte.

Neben diesen benötigten Variablen müssen Werkzeuge auch einige benötigte Methoden bereitstellen. Diese Methoden werden automatisch von höheren Prototypen aufgerufen, sobald sie benötigt werden. Die „initialisiereEreignisse“-Methode wird aufgerufen, sobald ein

Werkzeug in der Seitenleiste gewählt wurde. Als Parameter empfängt der Werkzeugprototyp die Werkzeug-ID, dadurch kann der Prototyp das ausgewählte Werkzeug identifizieren. Die Methode kann nun verschiedene benötigte Events zuweisen. So kann das Formwerkzeug einige Mausevents zuweisen, wodurch das Zeichnen einer Form realisiert werden kann. Alle Events müssen den Namen „werkzeug“ tragen. Auf diese Weise können die vorherigen Werkzeugevents automatisch gelöscht werden, sobald ein neues Werkzeug ausgewählt wurde. Die ebenfalls obligatorische „broadcast“-Methode wird aufgerufen, sobald ein Broadcast empfangen wurde, die diese Werkzeug-ID als Typen angegeben hat. Diese Methode dient dazu empfangene Daten auf gleiche Weise nachzubilden. Da die empfangenen Daten den gleichen Aufbau haben wie die gesendeten Daten versteht nur jeder Prototyp seine eigene Schnittstelle. Auf diese Weise können beliebige und beliebig viele Daten gesendet werden, ohne sich an ein Muster zu halten oder einen anderen Prototyp bearbeiten zu müssen.

Schließlich benötigt jeder Werkzeugprototyp die „nachzeichnen“-Methode. Diese Methode wird automatisch aufgerufen, sobald das Bild aus irgendeinem Grund nachgezeichnet werden muss. Gründe dafür können sein, dass Formen oder Zeichnungen entfernt werden müssen (beispielsweise ausgelöst durch die „Rückgängig“-Funktion). Die Leinwand muss nun komplett neu gezeichnet werden, da die Leinwand nicht mit Elementen arbeitet, wo es einfach möglich ist, beispielsweise das letzte erzeugte Element zu löschen. Stattdessen kann der Leinwand nur neue Zeichnungen, Formen und Bilder oben drauf gezeichnet werden (ähnlich dem Prinzip eines echten Leinwands).

Neben den obligatorischen Methoden, die jeder Werkzeugprototyp benötigt, gibt es noch optionale Methoden. Diese Methoden werden nur aufgerufen, wenn der Prototyp diese bereitstellt. Die „deinitialisiereEreignisse“ ist die Gegenmethode von „initialisiereEreignisse“. Sie macht einiges rückgängig, was die höheren Prototypen automatisiert nicht rückgängig machen können, da das Werkzeug etwas speziellere Funktionen ausgeführt hat. So benötigt das Textwerkzeug diese Methode um bei einem Wechsel des Werkzeugs die Auswahlrechtecke von Textelementen auszublenden und das Klicken auf die Textelemente komplett zu verhindern (ein Klick auf ein Element verursacht dadurch, dass das darunter liegende Element den Klick erhält). Auf diese Weise können keine Fehler mit den Events entstehen, wenn man mit dem Stiftwerkzeug über ein Textelement zeichnet. Die ebenfalls optionale Methode „farbeGewechselt“ wird automatisch vom Werkzeugverwaltungsprototypen ausgeführt, sobald eine andere Farbe gewählt wurde. Das wird bei Werkzeugen benötigt, die Elemente auswählen können. Nach der Auswahl eines Elements und dem Wechseln der Farbe erhält das ausgewählte Element dadurch die neue Farbe. Die letzte optionale Methode ist die „neuzeichnenVorbereiten“-Methode. Diese Methode wird kurz vor der „neuzeichnen“-Methode aufgerufen, falls sie existiert. Auf diese Weise können Elemente gelöscht werden, die nicht automatisch durch das Leeren der Leinwand entfernt werden können.

Events

Dank jQuery ist es noch einfacher mit Events zu arbeiten. Das Besondere hierbei ist, dass man den Events Namen (Identifikationen) geben kann, wodurch man ganz bestimmte Events oder eine Sammlung an Events deaktivieren kann.

So wird beispielsweise beim Auswählen des Pinselwerkzeugs dem Leinwandelement ein Event zum Drücken, Bewegen, Loslassen, Verlassen und Betreten der Maus hinzugefügt und den Events der Name „werkzeug“ gegeben. Gleichzeitig kann der Mauszeigerprototyp dem Leinwandelement Events zum Bewegen, Verlassen und Betreten der Maus hinzufügen mit dem Namen „mauszeiger“ und der Chatprototyp Events zum Drücken und Loslassen der Maus mit den Namen „chat“. Beim Wechseln des Werkzeugs wird nun vom Werkzeugverwaltungsobjekt jedes „werkzeug“ Event wieder entfernt, die „mauszeiger“- und „chat“-Events bleiben erhalten.

SCSS

Alle Objektdaten die etwas via CSS stylen müssen bekommen eine eigene .scss Datei. Durch die Möglichkeit in SCSS alles zu verschachteln, kann es zu keinen Konflikten kommen. Die SASS-Dateien wurden, wie auch die JS-Dateien, modular entwickelt. So bekommt die Werkzeugleiste eine eigene Datei, wie auch die diversen Prototypen, wie der Chat und der Mauszeiger, und Werkzeuge, wie das Text- und Tabellenwerkzeug. Wer nun beispielsweise ein neues Werkzeug erstellen möchte, der braucht nur eine neue JS- und eine neue SCSS Datei anzulegen. Es ist nicht nötig (und nicht empfohlen) andere Dateien zu bearbeiten. Auf diese Weise kann es auch zu keinen Konflikten in der Git kommen, da jeder an seiner eigenen Datei arbeitet.

Server

Der Server dient neben der Kommunikation zwischen den Clients, zum Ermitteln einer gemeinsamen der Zeit und zum Erhalten von Informationen, wenn ein Client eine Verbindung aufbaut oder die Verbindung trennt auch zum Verwalten der Räume. Sobald ein Client das Whiteboard aufruft ohne Angabe einer Raum-ID, so wird eine neue Raum-ID generiert. Um das Erraten einer Raum-ID zu erschweren wird ein Raum durch einen zufälligen String ersetzt. Um die Strings so kurz wie möglich zu halten und Dopplungen zu vermeiden wird die Raum-ID Zahl in eine andere Basis umgewandelt und eine einmalig zufällig generierte Zeichenkette zum Repräsentieren der neuen Zahl verwendet.

Am Anfang der Sitzung (beim Starten des Servers) wird einmalig eine Zeichenkette generiert, wo jedes erlaubte Zeichen nur einmal vorkommen kann, z. B. „dt4PioZye...“. Der erste Raum (Raum-ID: 0) würde bei dieser Zeichenkette die ID „d“ bekommen. Der zweite Raum die ID „t“, der dritte die „4“ usw. Ist die Zeichenkette 61 Zeichen lang, so ist es möglich, bis zu 61 verschiedene Räume zu erstellen mit nur einem Zeichen. Wird nun der 62. Raum erstellt, so lautet die ID „td“ (da „d“ für 0 steht. Bei Dezimalzahlen kommt nach der 9 die 10 und nicht 00). Auf diese Weise ist es möglich, die Zeichenkette voll auszuschöpfen und die ID kurz und

merkbar halten. Falls das Whiteboard nun so erfolgreich werden würde und es würden beispielsweise 13.835.789 Räume erstellt werden, so könnte man die generierte ID „25pu“ telefonisch schneller und einfacher sagen oder mit dem Smartphone schneller tippen, als die komplette Zahl.

Die Formel, für das Ermitteln, ab welcher Zahl die Länge der ID sich ändert, lautet:

$$61^x$$

Möchte man ermitteln, wie viele Zahlen man mit nur 6 Ziffern decken kann, so rechnet man:

$$61^6 = 51.520.374.361$$

Subtrahiert man das Ergebnis mit 1, so erhält man die letzte fünfstellige ID (die aus 5x dem letzten Zeichen der Zeichenkette besteht).

Die Formel zum Generieren der Zeichenkette geht folgendermaßen:

Zuerst berechnet man den Modulo von der Zahl und der Anzahl der Zeichen:

$$\text{Modulo} = \text{Zahl} \% \text{Anzahl}$$

Auf diese Weise bekommen wir eine Zahl zwischen 0 und Anzahl - 1. Dadurch können wir unser erstes Zeichen ermitteln. Nun wird die Zahl mit dem Modulo subtrahiert (wodurch es zu einem Vielfachen von der Anzahl wird) und durch die Anzahl geteilt:

$$\text{Zahl} = (\text{Zahl} - \text{Modulo}) / \text{Anzahl}$$

Ist nun die Zahl größer als 0, so wird dieser Schritt wiederholt und das nächste ermittelte Zeichen wird an den Anfang unserer Zeichenansammlung getan. Dies wird solange wiederholt, bis die Zahl 0 ist.

Dokumentation

Alle selbstgeschriebenen JS Dateien vom Server und vom Client wurden ausführlich dokumentiert. Bei der Dokumentation ist zuerst eine Beschreibung der Funktion/Methode aufzufinden, gefolgt von den Beschreibungen und Datentypen der Parameter (falls vorhanden). Schließlich folgt (falls vorhanden und nötig) die Rückgabebeschreibung ebenfalls mit dem Datentypen. Einige Objekte enthalten einen Beispielcode wie der Aufruf zu erfolgen haben sollte. Andere Methoden und Parameter enthalten einen Verweis zu anderen Methoden, Dateien oder Quellen.

Beim Erstellen der Dokumentation wurden die Regeln von JSDoc eingehalten. Schließlich wurde mit dem JSDoc Modul die Dokumentation in einem eigenen Verzeichnis generiert. Beim Aufruf der „index.html“-Datei wird dem Benutzer in der Seitenleiste eine schnelle Verlinkung zu den jeweiligen Dateien angezeigt. Die Beispielcodes wurden automatisch farblich hervorgehoben.

BroadcastType

Contains all types of sendable and receivable broadcasts. The index is the number of broadcasts and has to be incremented after each extension. The broadcast types are constants (shouldn't be edited ever) and have to be written in upper case.

Source: [Server.ptt.js, line 10](#)

Example

```
BroadcastType.NAME = BroadcastType.index++;  
var type = BroadcastType.NAME;
```

(readonly) HistoryType :number

Enum for history types.

Type:

- number

Properties:

Name	Type	Description
properties	Object	Contains the integer value of this enum as key with an object with details as value. The object contains the toolClassName and the toolName. The "index" key returns the last added index value and increases automatically by registering more tools.

Source: [History.ptt.js, line 9](#)

Example

```
HistoryType.TESTT00L; // 3  
HistoryType.properties["TESTT00L"].toolName; // "TestT00L"
```

Debug
Draw
History
Main
Selection
Server
Shape
Table
Text
Tools

Global

BOARDMAXHEIG
BOARDMAXWIDT
BroadcastTyp
HistoryType
debug
lcfirst
hash
main
MULTIPLIER
ToolSettingT
ucfirst
UNDOSTEPS

Test

Einleitung

Für die Tests wurde JsUnit benutzt, ein Framework zum clientseitigen Testen von JS. Es wurde inspiriert von JUnit, dem Testframework von Java, und liefert daher die gleichen assert-Funktionen.^{[17][18]}

JsUnit bietet neben einfachen synchronen Tests auch asynchrone Tests. Diese können nützlich sein beim Testen von Serververbindungen.

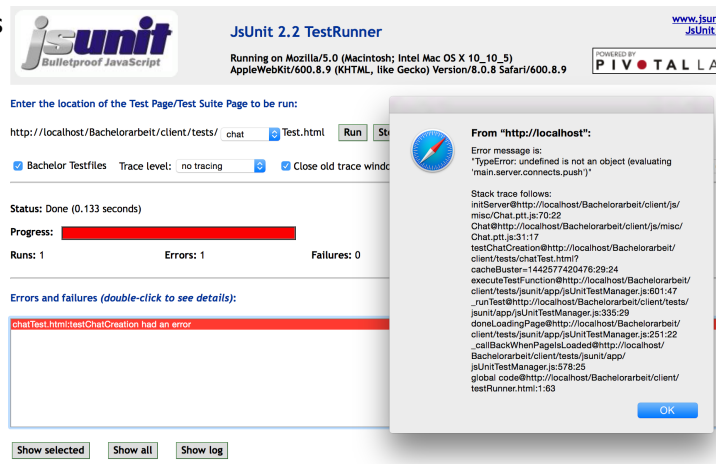
Leinwandtest

Im Leinwandtest wird das Erstellen von temporären Leinwänden getestet. Hierfür wird eine temporäre Leinwand mit einer ID 12 erstellt und zuerst einmal geprüft, ob das Erstellen erfolgreich war und die Leinwand als temporär erkannt wird. Es wird der Leinwand ein zufällig generierter Floatwert (Kommazahl) deklariert und initialisiert, dieser Wert wird im Laufe des Tests zum Überprüfen benötigt. Nun wird im Leinwandprototypen erneut eine temporäre Leinwand mit der ID 12 erzeugt. Da diese Leinwand bereits existiert sollte die Funktion die alte temporäre Leinwand zurückgeben. Dies wird geprüft, indem nach dem Wert des generierten Floatwertes gefragt wird. Es wird noch eine temporäre Leinwand erzeugt, diesmal mit der ID 13. In diesem Fall wird jedoch geprüft, ob der zufällig generierte Floatwert nicht deklariert wurde, dadurch wird ausgeschlossen, dass die Variable global deklariert wird. Nun werden die Anzahl der Leinwände gezählt. Da der Konstruktor des Leinwandprototypen automatisch eine globale (nicht temporäre) Leinwand erzeugt wird eine um eins höhere Zahl erwartet, also drei Leinwände. Schließlich wird die zuerst generierte Leinwand gelöscht und geprüft, ob sie erfolgreich gelöscht werden konnte. Da nach dem Erzeugen der Leinwand noch eine weitere Referenz zur selben Leinwand erzeugt wurde wird nun ebenfalls geprüft, ob diese Leinwand erfolgreich als gelöscht markiert wurde.

Werkzeugetest

Dieser Test testet nicht den Werkzeugverwaltungsprototypen, sondern vorher in einem Array angegebene Werkzeugprototypen. Hierfür werden nacheinander in einer Schleife Objekte der Werkzeugprototypen erzeugt und geprüft, ob sie dem erwarteten Muster eines Werkzeugs nachgebaut wurden. Nach dem Erzeugen eines Objekts wird geprüft, ob das Objekt korrekter Weise in den Werkzeugeinstellungen den Namen des Werkzeugs und zu jedem Werkzeug ein Symbol für die Seitenleiste bereitstellt. Wenn das Werkzeug nun ebenfalls Einstellungen bietet, so werden die Einstellungen nacheinander geprüft, ob sie einen korrekten Typen, wie z. B. „Schaltflächen“ oder „Regler“ definiert. Während der Überprüfung der Typen werden die jeweiligen Einstellungen gezählt und schließlich überprüft, ob die Anzahl der definierten Standardwerte exakt der Anzahl der Einstellungen entsprechen.

Nach der Überprüfung der Werkzeugeinstellungen werden einige obligatorische Methoden geprüft. Da jedes Werkzeug das Initialisieren von Events, das Broadcasten und das Neuzeichnen beherrschen muss wird nun der Reihe nach geprüft, ob die Methoden „initialisiereEvents“, „broadcast“ und „neuZeichnen“ existieren. Erst wenn ein Werkzeug fehlerfrei getestet werden konnte springt die Schleife zum nächsten Werkzeug. Da JS keine



Interfaces bietet und man deshalb beim Erzeugen neuer Werkzeuge Methoden oder Variablen falsch benennen oder vergessen kann, keine weitere Überprüfung stattfinden wird und der Fehler nicht sofort sondern erst nach einer bestimmten Aktion erscheinen kann bietet dieser JsUnit-Test eine sinnvolle und hilfreiche Überprüfung. Hierfür muss nach dem Erzeugen eines Werkzeugs nur der Name des Werkzeugs in das Schleifenarray hinzugefügt und der Test ausgeführt werden.

Verlaufstest

Der Verlaufsprototyp muss zuverlässig funktionieren und das manuelle Testen von multiplen Benutzerverläufen geht nur mit zeitaufwändigeren Tests wo der Server laufen, man mehrere Fenster öffnen und nacheinander zeichnen und die Konsole beobachten muss. Der Verlaufstest hingegen testet automatisiert, ob das Hinzufügen von Verlaufstypen erfolgreich ist und sie den richtigen Wert bekommen. Nach dem Test wird nur ein lokaler (eigener) Wert in den Verlauf hinzugefügt. Dies simuliert eine eigene Zeichnung die in den Verlauf eingetragen wurde. Danach wird ein fremder Wert mit einem Zeitpunkt vor dem Zeitpunkt des lokalen Eintrages hinzugefügt. Nun wird erneut überprüft, ob der letzte Wert der eigene Wert ist und der neue erste Wert (da der Zeitpunkt vor dem lokalen Zeitpunkt liegt) die des simulierten anderen Benutzers ist.

Nach der Bearbeitung des Verlaufsprototypen ist es nun möglich, diese leicht komplexen Tests automatisiert zu überprüfen und Fehler ohne viel Aufwand zu erkennen.

Chat-Test

Dieser Test dient zum Überprüfen ob der eigenständige Chat auch fehlerfrei funktioniert, ohne wie beim Verlaufstest den Server zu starten, ein weiteres Fenster zu öffnen und sich selbst Nachrichten zu schreiben. Da der Chat Funktionen des Servers ausführen wollen würde wurde für diesen Test der Serverprototyp als kleines Objekt mit Funktionen für diesen Test neu erzeugt. Die Funktion zum Ermitteln der Zeit liefert nur die lokale Zeit und die Funktion zum Broadcasten einer Nachricht speichert die zu sendenden Werte in eine Variable die schließlich geprüft werden kann. Da der Chat die „broadcastEmpfangen“-Funktion an das Serverobjekt übergibt, damit der Server beim Empfangen von Chatnachrichten die Werte an die Funktion weiterleiten kann, wird noch ein leeres Objekt hierfür erzeugt.

Der Test startet damit ein Objekt des Chats zu erzeugen. Daraufhin wird geprüft, ob der Konstruktor des Chats den Broadcasttypen erfolgreich mit der richtigen ID erzeugt hat. Nun wird das Absenden der Nachricht simuliert. Da noch keine Nachricht eingegeben werden konnte und somit kein Broadcast stattfinden sollte wird beim Auslesen des letzten Broadcasts weiterhin der Initialwert erwartet. Nun wird eine Testnachricht eingetragen und die Nachricht erneut verschickt. Der letzte Broadcast sollte nun die Nachricht und den Zeitstempel enthalten. Es wird außerdem noch geprüft, ob die Anzahl der Nachricht im Verlauf des Chats „1“ entspricht. Da es nicht weiter wichtig ist wie die Schlüssel vom Broadcast heißen (z. B. wie der Schlüssel der Nachricht und des Zeitstempels heißt), da der Chat selbst das Senden und

Empfangen verarbeitet, werden diese Daten nicht weiter geprüft, es wird lediglich eine Benutzer-ID „12“ angehängen und dieselben Daten an die „broadcastEmpfangen“-Funktion des Chats weitergeleitet. Auf diese Weise wurde nun das Empfangen einer Nachricht von einem Fremden simuliert, ohne die Schlüssel des Objektes zu kennen. Schließlich wird geprüft, ob die erste Nachricht im Verlauf die Eigene und die zweite Nachricht im Verlauf von Benutzer-ID „12“ ist und die Anzahl der Nachrichten „2“ entspricht.

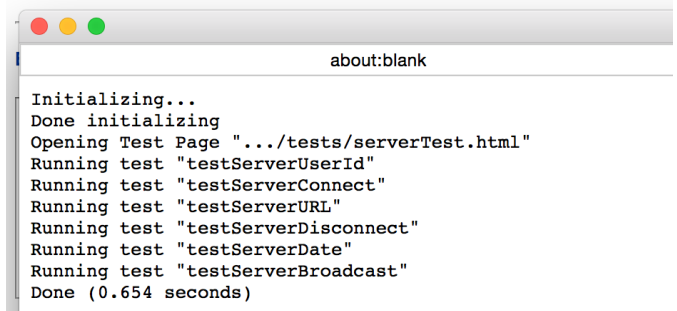
Status: Done (0.654 seconds)

Progress:

Runs: 6

Errors: 0

Failures: 0

A screenshot of a web browser window with a title bar containing three colored circles (red, yellow, green). The address bar shows "about:blank". The main content area displays a list of test logs. The logs start with "Initializing..." and "Done initializing". Then, it says "Opening Test Page \"../tests/serverTest.html\"". This is followed by a series of "Running test" entries for "testServerUserId", "testServerConnect", "testServerURL", "testServerDisconnect", "testServerDate", and "testServerBroadcast". The final line of the log is "Done (0.654 seconds)".

```
Initializing...
Done initializing
Opening Test Page "../tests/serverTest.html"
Running test "testServerUserId"
Running test "testServerConnect"
Running test "testServerURL"
Running test "testServerDisconnect"
Running test "testServerDate"
Running test "testServerBroadcast"
Done (0.654 seconds)
```

Servertest

Der Servertest ist ein etwas komplexerer Test. In dem Test wird eher weniger der Serverprototyp getestet, sondern die Anfragen und Antworten des Node Servers. Um eine Verbindung zu Servern in JsUnit aufbauen zu können muss asynchron gearbeitet werden (da sonst JsUnit sofort nach dem Laden der Seite die Tests startet, bevor die Verbindung zum Server hergestellt werden kann. Dies wurde recht einfach mit der Variable „setUpPageStatus“ umgesetzt. Solange diese Variable den Wert „running“ wartet JsUnit. Sobald die Variable den Wert „complete“ enthält startet JsUnit die Tests.

Der Servertest verbindet sich mit dem Server, sendet und empfängt einige Daten (diese werden zwischengespeichert) und wenn die Kommunikation mit dem Server komplett beendet wurde, dann werden die JsUnit Tests ausgeführt.

Beim Test wird zuerst ein Client mit dem Server verbunden und es wird eine Raum-ID (die URL) und die Benutzer-ID erwartet. Sobald die Raum-ID empfangen wurde wird ein weiterer Client zum selben Raum verbunden, dieser erwartet ebenfalls seine Benutzer-ID. Sobald der zweite Client seine Benutzer-ID bekommen und zwischengespeichert hat sendet der Benutzer eine Broadcastnachricht mit einem zufällig generierten String. Sobald nun der erste Client eine Broadcastnachricht empfängt wird die Nachricht und der Sender zwischengespeichert und der zweite Client trennt die Verbindung zum Server.

Da beim Verbinden und Trennen der Verbindung zum Server jeder Benutzer im Raum kontaktiert wird, wird ebenfalls gelauscht, ob ein Client sich verbunden oder sich getrennt hat. Nachdem der zweite Client erfolgreich die Verbindung hergestellt hat sollte der erste Client mit der Benutzer-ID des zweiten Clients kontaktiert werden. Nun sendet der erste Client sein Datum (die lokale Uhrzeit) an den Server und erwartet nun die gleiche Uhrzeit und die Uhrzeit des Servers als Antwort.

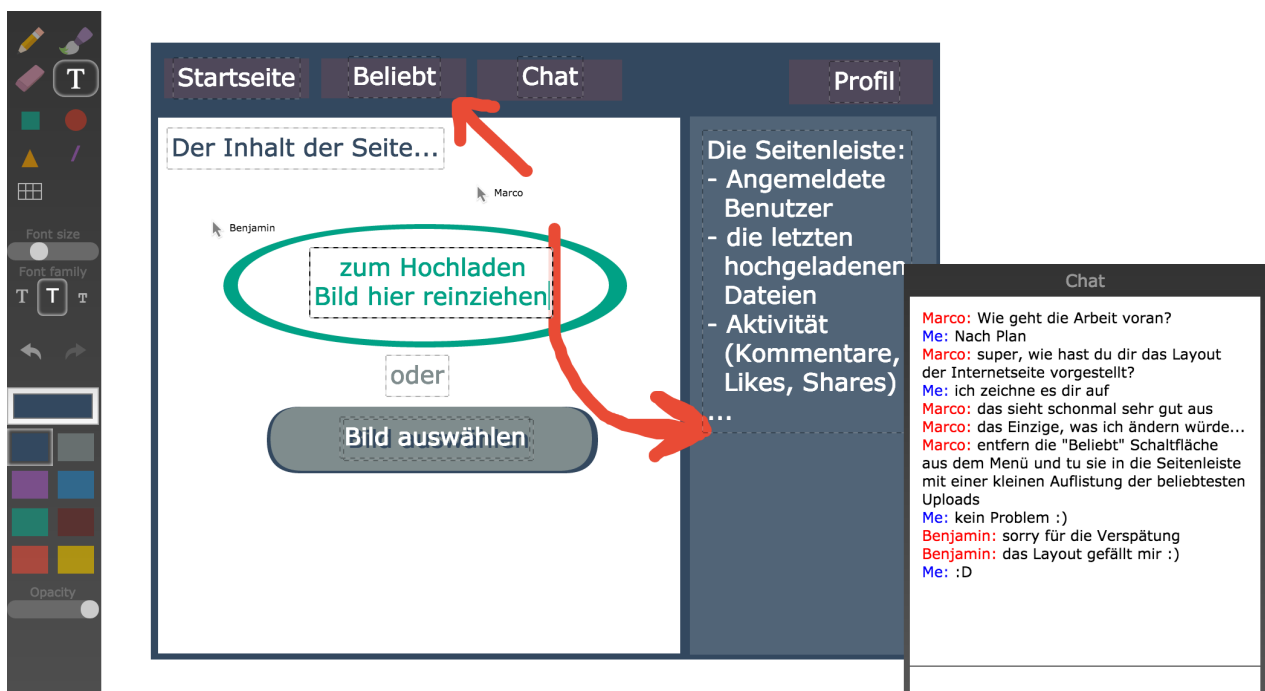
Sobald alle Anfragen gesendet und Antworten erhalten wurden wird der JsUnit Test ausgeführt. Die zwischengespeicherten Daten werden nun in beliebiger Reihenfolge überprüft. Es wird geprüft, ob die URL ausschließlich aus erlaubten Zeichen besteht, die Benutzer-ID nicht leer ist, die empfangene Broadcastnachricht dieselbe ist, wie die zufällig generierte, der Sender der

Nachricht dieselbe Benutzer-ID hat, wie die empfangene Benutzer-ID des zweiten Clients, der erste Client mit der richtigen Benutzer-ID kontaktiert wurde, als der zweite Benutzer sich verbunden und dann die Verbindung getrennt hat und ob das gesendete Datum dem empfangenen Datum entspricht und ob der empfangene Server-Datum gültig ist.

JsUnit hat einen eingebauten Timeout, falls die Variable, die JsUnit aktivieren soll, zu lange nicht auf „complete“ gesetzt wird. Aus dem Grund, dass der Timeout von JsUnit zu lange dauert und kein Protokoll bietet wurde ein eigener Timeout entwickelt. Falls also nach drei Sekunden die Anzahl der erwarteten Antworten noch immer nicht erhalten wurden wird eine eigene Fehlermeldung geworfen, welches die bereits empfangenen Daten auflistet. Auf diese Weise kann festgestellt werden, aus welchem Grund Antworten nicht erhalten wurden (falls der zweite Client beispielsweise einem falschen Raum beigetreten ist).

Ergebnis

Beim Aufruf der Whiteboard-Seite sieht der Benutzer links eine Werkzeugleiste und unten rechts einen Chat, der zuerst minimiert ist und mit einem Klick aufgemacht werden kann.

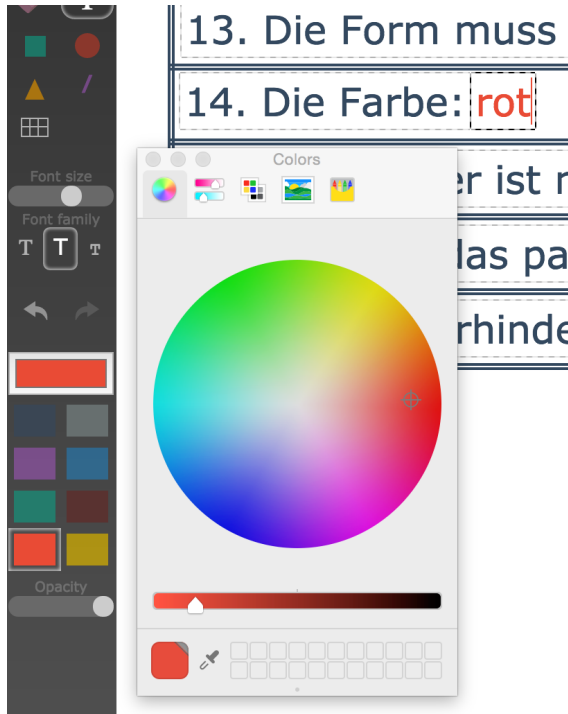


Die Werkzeugleiste hat zurzeit neun Werkzeuge und die beiden Funktionen „Rückgängig“ und „Wiederherstellen“. Eine Auflistung aller Werkzeuge mit einer kurzen Beschreibung der Funktion und der Quelle des Symbols ist in der folgenden Tabelle aufzufinden:

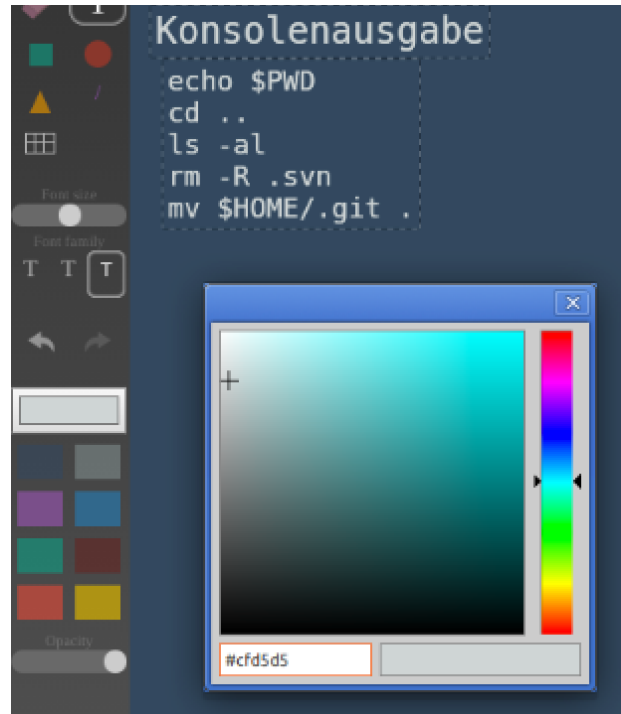
Symbol	Funktionen, Einstellungen und Symbolquelle
	<p>Stiftwerkzeug: Dient zum Zeichnen von feinen Linien. Enthält keine Einstellungen.</p> <p>Quelle: Eine leicht modifizierte Version von: https://www.iconfinder.com/icons/381620/pencil_icon</p>
	<p>Pinselfwerkzeug: Dient zum Zeichnen von dickeren Linien. Die Pinseldichte ist einstellbar zwischen 10px, 20px und 30px.</p> <p>Quelle: https://www.iconfinder.com/icons/416388/artist_brush_paintPainter_tool_icon</p>
	<p>Radierwerkzeug: Dient zum Radieren. Dicker als der Pinsel. Die Radierdicke ist einstellbar zwischen 20px, 50px und 80px.</p> <p>Quelle: https://upload.wikimedia.org/wikipedia/commons/d/dd/Pink-eraser.svg</p>
	<p>Textwerkzeug: Ein Werkzeug zum Erzeugen von Textelementen. Solange dieses Werkzeug gewählt ist können die Textelemente weiterhin bearbeitet werden. Es kann zwischen drei Schriftarten (Times New Roman, Verdana und Monospace) gewählt werden und die Schriftgröße kann zwischen 8px und 40px eingestellt werden.</p> <p>Symbol: Ein großes „T“ in der Schriftart „Times New Roman“ (kein Bild).</p>
	<p>Rechteckwerkzeug: Ermöglicht das Erzeugen von Rechtecken. Es kann eine Abrundung zwischen 0px und 200px gewählt werden.</p> <p>Symbol: Ein HTML-Element mit einer festgelegten Größe und grünen Hintergrundfarbe (kein Bild).</p>
	<p>Ellipsenwerkzeug: Ermöglicht das Erzeugen einer Ellipse. Enthält keine Einstellungen.</p> <p>Symbol: Ein HTML-Element mit einer festgelegten Größe, roten Hintergrundfarbe und als Umrandungsradius die Hälfte der Größe (kein Bild).</p>

Symbol	Funktionen, Einstellungen und Symbolquelle
	<p>Dreieckswerkzeug: Ermöglicht das Erzeugen von Dreiecken mit der Spitze nach oben oder nach unten zeigend. Es kann eine Abrundung zwischen 0% und 10% der Größe des Dreiecks eingestellt werden.</p> <p>Symbol: Ein HTML-Element mit einer Größe von 0px x 0px und einer sichtbaren unteren Umrandung mit einer festgelegten Umrandungsdichte (kein Bild).</p>
	<p>Linienwerkzeug: Ermöglicht das Zeichnen von geraden Linien. Die Linienbreite (zwischen 1px und 31px) und die Linienenden (eckig oder abgerundet) können eingestellt werden.</p> <p>Symbol: Ein Schrägstrich (Slash) in der Schriftart „Verdana“ und in „Fett“ (kein Bild).</p>
	<p>Tabellenwerkzeug: Ermöglicht es Tabellen mit einer benutzerdefinierten Anzahl an gleich hohen Zeilen und gleich breiten Spalten zu erzeugen. Enthält keine Einstellungen.</p> <p>Symbol: Eine Tabelle (HTML-Element <table>) mit zwei Zeilen und jeweils drei Spalten (kein Bild).</p>
	<p>Rückgängig-Funktion: Widerruft die letzte eigene Aktion. Kann keine Einstellungen haben, da dies eine Funktion ist und kein Werkzeug (kann also nicht ausgewählt werden).</p> <p>Quelle: Eine leicht modifizierte Version von: https://www.iconfinder.com/icons/308958/arrow_left_icon</p>
	<p>Wiederherstellen-Funktion: Stellt die zuletzt widerrufene, eigene Aktion wieder her. Keine Einstellungen (siehe „Rückgängig-Funktion“).</p> <p>Quelle: Eine spiegelverkehrte Version vom „Rückgängig“-Symbol (siehe „Rückgängig-Funktion“)</p>

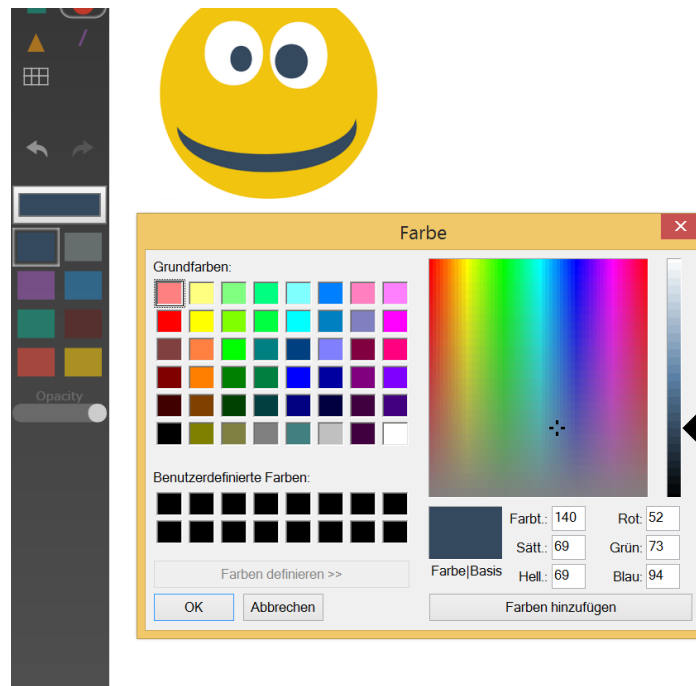
Unter den Werkzeugen, den Einstellungen und den Funktionsschaltflächen gibt es eine Farbauswahlschaltfläche^[19], eine Farbpalette und eine Deckkrafteinstellung zwischen 0% und 100%. Die Farbpalette enthält ausschließlich ausgewählte Farben von der Seite FlatUIColors (<https://flatuicolors.com>). Die Farbauswahlschaltfläche erlaubt das Auswählen von Farben im betriebssystemeigenen (nativen) Farbauswahlfenster.



Mac OS X 10.10.5

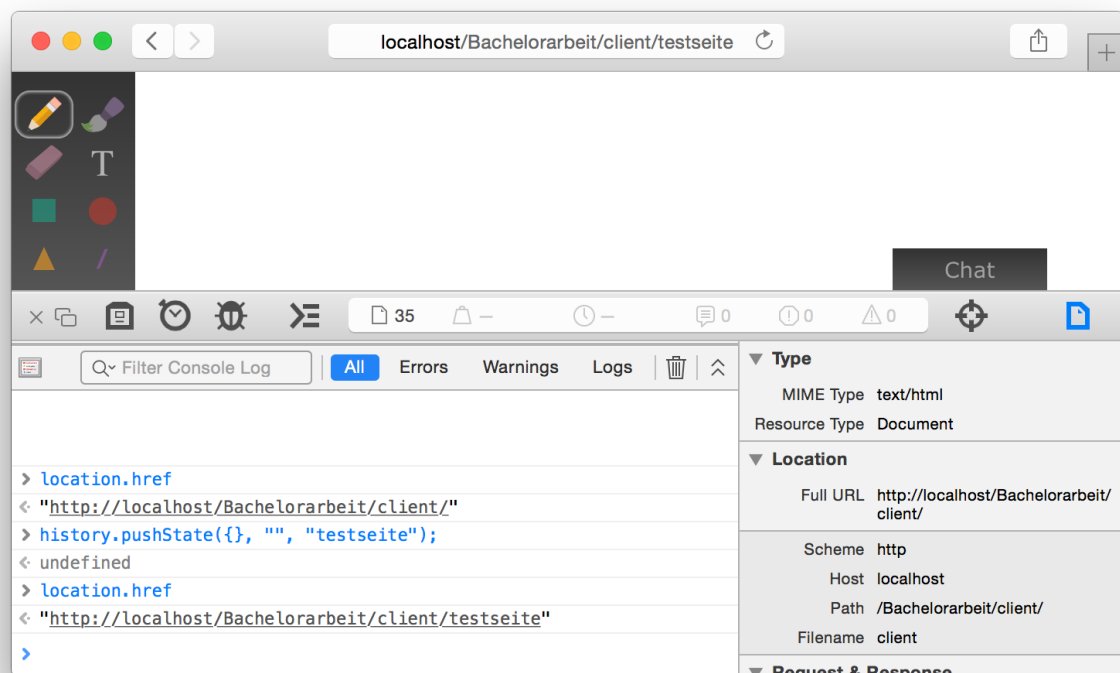


Ubuntu Linux 14.04



Windows 8.1

Falls der Benutzer zum Aufrufen der Seite einen Link von einem Partner aufgerufen hat, wo bereits eine Raum-ID angegeben ist, so kann er nun mit diesem Partner und anderen im selben Raum kommunizieren und gemeinsam zeichnen. Falls die Startseite der Whiteboard-Seite aufgerufen wurde (ohne Raum-ID), so ändert sich die Adresse in der Adressleiste zu einer neu generierten Raum-ID, die er seinen Partnern zusenden kann. Die Adresse wird dank der „history.pushState“^[20] JS-Funktion geändert, ohne eine Weiterleitung durchzuführen. Die Seite muss also nicht neu geladen werden und der Benutzer kriegt die Änderung kaum mit. Wenn der Benutzer nun die Adresse kopiert (um sie ihren Partnern zuzuschicken), oder die Seite aktualisiert oder favorisiert, so wird der richtige Raum ausgewählt.



In der Konsolenausgabe ist zu sehen, dass die Adresse mit der „pushState“-Funktion geändert wurde, ohne dass die Seite neu geladen werden musste.

Literatur

- [1] Google Docs (früher Google Text und Tabellen) ist eine Online Textverarbeitung. Aug. 2015
(<http://docs.google.com>)
- [2] Cloud9 ist eine Online Entwicklungsumgebung. Mit dem Editor ist es möglich, Aug. 2015
gemeinsam an Projekten zu arbeiten. (<https://c9.io>)
- [3] Coda ist ein Texteditor mit farblicher Hervorhebung. Er stellt einen FTP-Client, Aug. 2015
Datenbankclient, Terminal, Versionsverwaltung und mehr in einem einzigen Fenster
zur Verfügung. (<https://panic.com/coda/>)
- [4] Tower stellt ein GUI zur fortgeschrittenen Versionsverwaltung zur Verfügung Aug. 2015
(<http://www.git-tower.com>)
- [5] JavaScript Prototypen Aug. 2015
(http://www.w3schools.com/js/js_object_prototypes.asp)

[6]	new Operator (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new)	Aug. 2015
[7]	jQuery eines der beliebtesten Bibliotheken (https://www.javascripting.com/?sort=rating)	Aug. 2015
[8]	jQuery (https://jquery.com/)	Aug. 2015
[9]	NodeJS Module (https://nodejs.org/api/modules.html#modules_modules)	
[10]	Räume in socket.io (http://socket.io/docs/rooms-and-namespaces/#rooms)	Aug. 2015
[11]	KeyDown Event (https://developer.mozilla.org/en-US/docs/Web/Events/keydown)	Sep. 2015
[12]	MouseMove Event (https://developer.mozilla.org/en-US/docs/Web/Events/mousemove)	Aug. 2015
[13]	Coda SASS Plugin generiert beim Abspeichern einer .sass Datei die jeweilige .css Datei. (https://github.com/keegnotrub/coda-sass-plugin)	Aug. 2015
[14]	socketio/socket.io-protocol · GitHub (https://github.com/socketio/socket.io-protocol)	Aug. 2015
[15]	https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/getTime	Aug. 2015
[16]	Kreisförmig löschen (http://stackoverflow.com/questions/10396991/clearing-circular-regions-from-html5-canvas)	Aug. 2015
[17]	JsUnit (http://www.jsunit.net)	Sep. 2015
[18]	JsUnit Assert Dokumentation (http://www.jsunit.net/documentation/assertions/)	Sep. 2015
[19]	Farbauswahlschaltfläche (colorpicker) in HTML. Unterstützt von Mozilla Firefox (ab 38), Google Chrome (ab 31), Opera (ab 31) und den Androidbrowsern (ab Android 4.4.4+) (http://caniuse.com/#search=color). Alternativ (falls nicht unterstützt) wird ein Eingabefeld mit einem Farb-Hexwert angezeigt.	Sep. 2015
[20]	pushState (https://developer.mozilla.org/en-US/docs/Web/API/History_API#Adding_and_modifying_history_entries)	Sep. 2015
[21]	Echtzeit (Realtime) im Duden (http://www.duden.de/rechtschreibung/Realtime)	Aug. 2015
[22]	Definition von Groupware (http://searchdomino.techtarget.com/definition/groupware)	Aug. 2015

Verzeichnisse

Glossar

Echtzeit	Die Echtzeit ist die tatsächlich benötigte Zeit für die sofortige und unmittelbare Verarbeitung der Daten eines Rechners. ^[21]
Client	(hier) Ein Client ist eine Software, die sich über das Netzwerk zu einem Server verbindet und mit ihm kommuniziert.
Server	(hier) Ein (Web-)Server ist eine Software, mit der sich Clients verbinden können. Der Server verwaltet die Clients und verarbeitet Anfragen oder leitet sie weiter an andere verbundene Clients.
Groupware	Diese Art von Programmen ermöglichen es, gemeinsam kollektiv aus der Ferne in Echtzeit zu arbeiten. ^[22]

Objekt	(in JS) Objekte bestehen aus einem Schlüssel (einem Namen) und einem Wert beliebigen Typs.
Konstruktor	Eine Methode, die einmalig beim Instanzieren ausgeführt wird.
Instanziierung	Das Erzeugen eines Objektes in der objektorientierten Programmierung.
Inkludieren	Bei der Programmierung: Das Einbinden von Code von anderen Dateien in die eigene Datei.
Broadcast	Das Senden von Informationen an alle anderen Mitglieder.
Websocket	Ein bidirektionaler Kommunikationskanal.
Modul	Durch Aufteilen von Code in kleinere „Module“ ist es möglich, diese unabhängig aufzurufen und in andere Systeme zu implementieren.
Initialisieren	Herstellung eines Anfangszustands, z. B. durch das Ausführen des Konstruktors oder durch eine Initialisierungsfunktion.
Deklarieren	(in JS) Das Erstellen einer Variable. Ohne die Initialisierung hat die Variable den Typ „undefined“ (undefiniert).
Injizieren	Das Einschleusen von eigenem Code.
Array	Ein Container, der eine bestimmte Anzahl an Werten vom selben Datentyp enthält. Die Werte können durch den Index (von 0 bis Elementanzahl - 1) ausgelesen oder bearbeitet werden. In JS können Arrays außerdem während der Laufzeit frei erweitert oder verkleinert werden und multiple Datentypen enthalten.
JSON	Repräsentiert ein Objekt oder Array als Zeichenkette. Die Zeichenkette kann dann in ein Objekt oder Array mit Beibehaltung der Datentypen (integer, float, string) zurückverarbeitet werden. Das Objekt oder das Array kann multidimensional (verschachtelt) sein.
Web Inspector	(auch Developertools) dient zum Analysieren des Quelltextes, des DOM, der CSS Regeln, Anfragen und Antworten, zum Auslesen der JS-Konsole und mehr.
enum	Aufzählungstypen. Da JS keine enums hat, werden Objekte mit einem Schlüssel und einem Integer als Wert angelegt.

Abkürzungen

HTML	Hypertext Markup Language
JS	JavaScript
CSS	Cascading Stylesheet
Node	NodeJS
DOM	Document Object Model
JSON	JavaScript Object Notation
dpi	Dots per inch
AJAX	Asynchronous JavaScript and XML

Anhang

In der Anhang-CD befinden sich der gesamte Quellcode des Servers und des Clients mit den benötigten Ressourcen (Bilder, Node-Module und weitere) und eine Dokumentation des Quellcodes des Servers und des Clients (separiert in zwei verschiedene Verzeichnisse).⁴

⁴ Der Anhang ist ebenfalls online archiviert: <https://goo.gl/dA47OD>

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.