

I/O-Devices für Webstebs

Projektnname: I/O-Devices für Webstebs
Autor: Stefan Wohlgensinger und Adrian Bürki
Semester: HS17
Version: finale Version
Zeitraum: 18.09.2017 – 19.01.2018
Verteiler: Ruedi Müller

1 Abstract

Diese Arbeit befasst sich mit der Verbesserung des grafischen Userinterfaces der Applikation Webstebs. Webstebs ist ein Mikrorechnersimulator mit Webfrontend, der durch verschiedene Studierende im Rahmen der Projektschiene entwickelt wurde.

Die Usability sollte durch verschiedene Veränderungen verbessert werden. Dazu wurden die meisten GUI - Elemente von Webstebs überarbeitet.

Zuerst wurden das bestehende Programm und vor allem dessen GUI analysiert um Schwachstellen in der Bedienbarkeit zu finden. Dazu wurden auch Usability – Tests mit Studierenden der FHNW durchgeführt. Anhand der daraus gewonnenen Erkenntnisse, wurden verschiedene Lösungen ausgearbeitet. Diese Lösungen wurden in einen Papier-Prototyp umgesetzt, damit sie in weiteren Usability - Tests getestet werden konnten.

Die besten Lösungen wurden mit dem Kunden priorisiert und mit einer agilen Vorgehensweise in Webstebs implementiert.

Ob mit den implementierten Veränderungen auch tatsächlich einen Mehrwert generiert werden konnte, wurde in einer dritten Iteration an Usability - Tests überprüft.

Ein weiteres Ziel dieses Projekts war es, neue I/O Devices für Webstebs zu entwickeln. Es konnten fünf neue Devices implementiert werden. Dies soll die Vielfalt an Aufgaben, die den Studierenden gestellt werden können, vergrössern.

Inhaltsverzeichnis

1 Abstract	2
2 Einleitung	5
2.1 Rahmenbedingungen.....	5
2.2 Problemstellung	6
2.2.1 GUI	6
2.2.2 Devices.....	7
2.2.3 Portvergabekonzept.....	8
2.3 Vision, Strategische Ziele.....	8
2.4 Generierter Mehrwert.....	8
2.5 Umsetzung	8
3 Technologien	10
3.1 jQuery Validation Plugin 1.11.1	10
3.2 FontAwesome	11
3.3 JS - und CSS - Libraries einbinden	11
4 GUI.....	13
4.1 Bekannte Probleme	13
4.2 Untersuchungen und Erkenntnisse	13
4.2.1 System- und Dokumentationsanalyse	13
4.2.2 Erster Usabilitytest: IST-Zustand (UX-Test)	14
4.2.3 Benchmarking: Der Einfluss bekannter IDEs.....	20
4.2.4 Prototyping & Idea Testing (UX-Test)	22
5 Umsetzung	34
5.1 Durchgeführte Tests: SOLL - Messung.....	34
5.2 Design	34
5.3 Source Code.....	35
5.3.1 Views	35
5.3.2 Klassen.....	35
5.4 Umgesetzte UI-Komponenten.....	37
5.4.1 Simulation Control	37
5.4.2 Menubar	43
5.4.3 Seitenpanels	46
5.4.4 Filemanager.....	55
5.4.5 Output / List-File - Panel	60
5.5	61
5.5.1 RunAndDebug entfernt	61
5.5.2 StatusMessage Pop-Up	63
6 Devices	64
6.1 Umgesetzte Devices	64
6.1.1 Keyboard Device	64
6.1.2 Text Console.....	68
6.1.3 Character Viewer	69
6.1.4 Number Display	71
6.1.5 7-Segment Display.....	73

6.2	How to develop a device	76
6.2.1	Plugin erstellen Schritt für Schritt (mit eigenen Ergänzungen).....	76
6.2.2	IN und OUT - Zusammenspiel Plugin mit Stebs Backend	77
6.2.3	Objektorientiertes Programmieren in Javascript.....	81
6.2.4	Existierendes Plugin integrieren.....	82
7	Portvergabe.....	84
7.1	Problemstellung	84
7.2	Umgesetzte Lösung	84
7.3	Source Code.....	84
8	Testen	85
8.1	Verweise auf die Testfälle	86
9	Known Issues.....	87
9.1	Filemanager.....	87
9.2	Architektur-Panel	88
9.3	Architektur in separatem Fenster	88
9.4	Step into	89
11	Verbesserungsvorschläge.....	90
11.1	Filemanager.....	90
11.2	Keyboard-Shortcuts	90
11.2.1	Steuerung mit der ctrl. - Taste.....	90
11.2.2	Steuerung mit der Alt - Taste	90
11.3	List File Bereich	90
11.4	Status – Pop-up	91
11.5	Seitenpanels	91
11.6	Menübar	91
11.7	Breakpoints.....	91
12	mögliche Erweiterungen	93
12.1	Visual Design: Color with Meaning.....	93
12.2	Hilfeseite	93
12.3	Add Device	93
12.4	Deviceliste	93
12.5	Portnummern nachträglich verändern	93
12.6	Register	94
12.7	Device mit mehreren Ports.....	94
12.8	externes RAM	94
13	Fazit, Reflexion	95
14	Glossar.....	96
15	Abbildungsverzeichnis	97
16	Ehrlichkeitserklärung	99

2 Einleitung

In diesem Kapitel werden die Rahmenbedingungen sowie die Problemstellungen des Projektes dargestellt. Weiter werden die daraus erstellten Visionen und Ziele dieses Projekts erläutert. Weiter wird der, aus diesem Projekt generierte Mehrwert behandelt.

2.1 Rahmenbedingungen

Der Webrechner Webstebs baut auf der Desktop-Applikation „Stebs“ auf, welche im Rahmen einer Bachelorthesis im Jahr 2005 für das Unterrichtsmodul „Computer Hardware und Programmierung“ erstellt und seitdem mehrere Male weiterentwickelt wurde. Die Desktopapplikation bietet eine Entwicklungsumgebung für Assembler, sowie die Möglichkeit den erstellten Code auf einem simulierten Rechner auszuführen und zu testen. Zusätzlich kann der Status des Prozessors in einem Architekturfenster überwacht werden. Die Aufgabe der Web-Applikation ist es, diese Desktop-Applikation vollständig als Unterrichtsmittel abzulösen.¹

Im Rahmen des Projekts „Webrechner“ im Jahre 2015 entstand diese Web-Applikation. Diese Version enthielt eine bereits vollständig funktionierende Mikrorechner-Simulation, sowie eine Client-Server-Architektur, auf welcher die vollständige Simulation auf dem Server stattfindet.

Im darauffolgenden Jahr führte das Projekt „Webstebs 2“ neue Features und ein überarbeitetes User Interface ein. So wurde eine dynamische Visualisierung des Mikrocodes, Backstepping und Historisierung der Simulation sowie Breakpoints für den CodeEditor eingeführt.

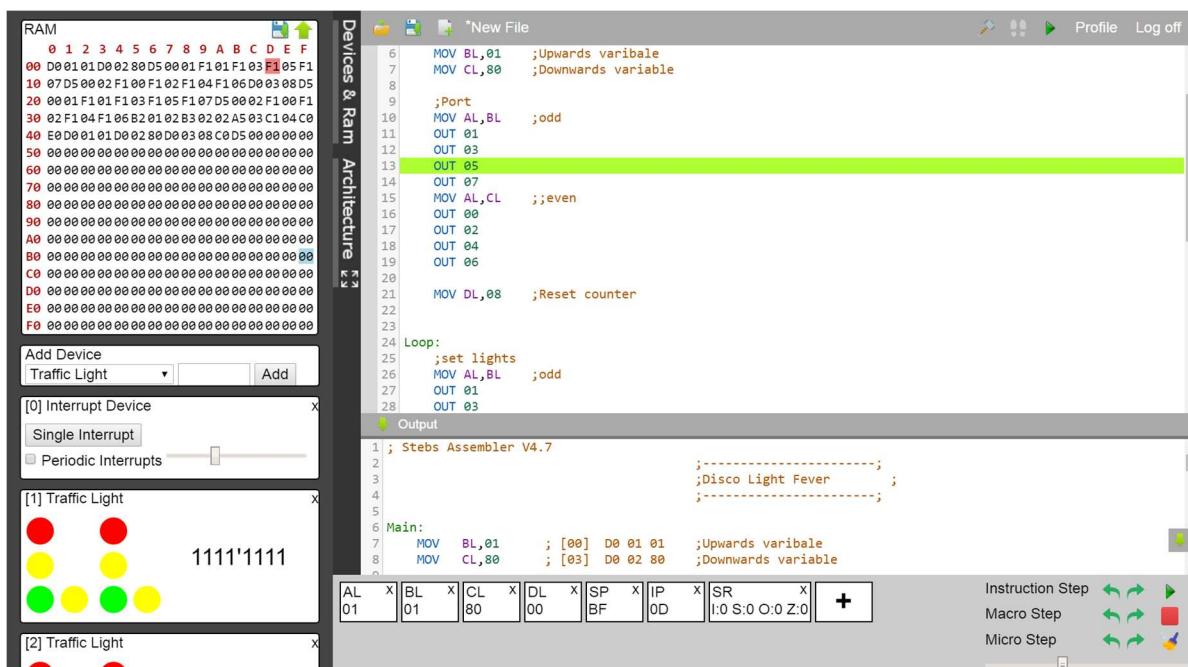


Abbildung 1 Stand der Applikation zum Start des Projekts

¹ IP5 - Webstebs2 (2016) - Projektbericht

Im Architecture - Panel ist zusätzlich eine Ansicht der CPU - Architektur ersichtlich.

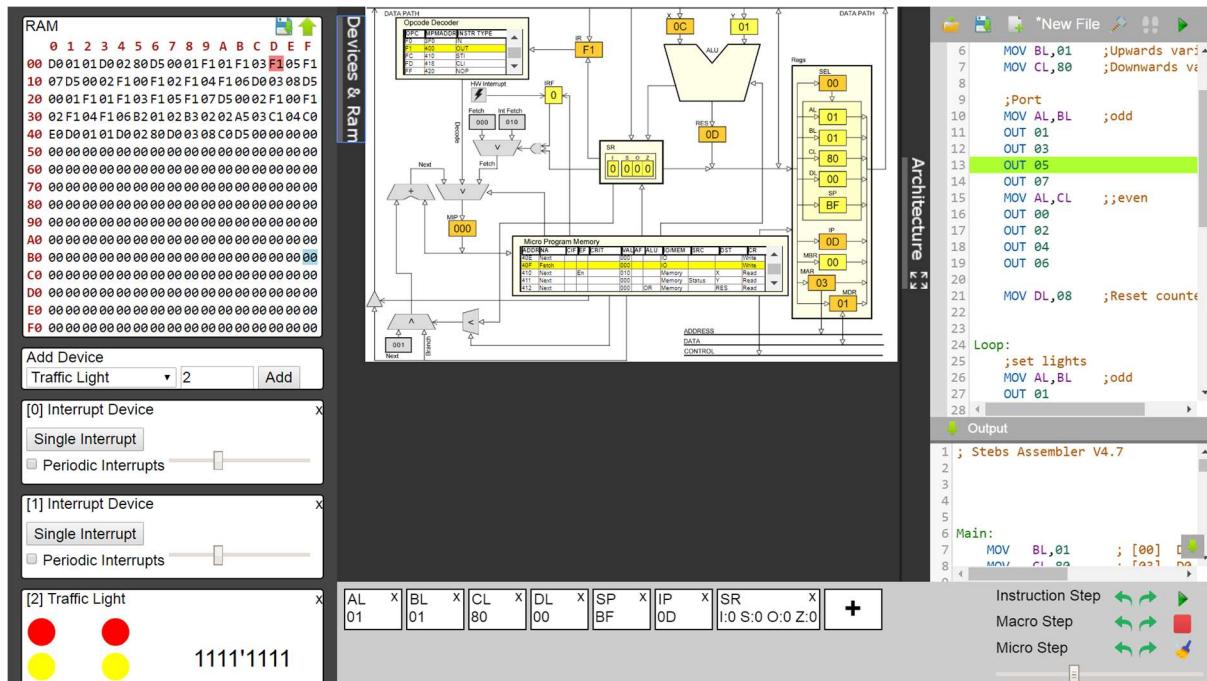


Abbildung 2 Stand der Applikation zum Start des Projekts - Architekturansicht

Die Aufgabe von "Webstebs 3 - IO Devices for Webstebs" bestand darin, die Usability von Webstebs zu verbessern sowie neue Devices für Unterrichtsaufgaben zu entwickeln.

2.2 Problemstellung

Im Projekt "I/O - Devices für Websteps" wurden folgende Problemschwerpunkte erkannt und behandelt: GUI, Devices und das Portvergabekonzept. Nachfolgend werden diese Schwerpunkte erläutert.

2.2.1 GUI

Devices werden in der unteren Hälfte des "Devices & RAM" - Fensters hinzugefügt. Werden mehrere Instanzen eines oder verschiedener Devices hinzugefügt, entstehen Probleme bei der Bedienbarkeit: Ab der vierten Instanz ragen die darunter aufgelisteten Devices aus dem Sichtbereich heraus. Alternativ kann sich der Endnutzer damit behelfen, die "Verkleinern" - Funktion des Browsers zu nutzen. Jedoch wird dabei die gesamte Applikation zunehmend schlechter lesbar. Damit diese Probleme gelöst werden können, müssen die Darstellung und die Administration der Devices angepasst werden.

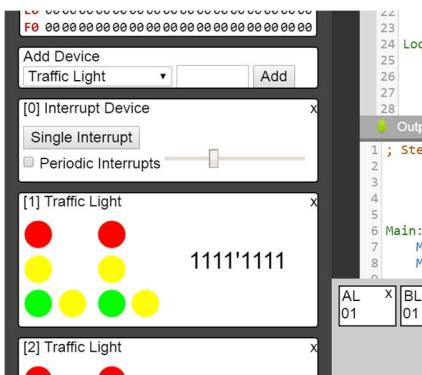


Abbildung 3 Devices in der alten Webstebs Version

Heutzutage kann angenommen werden, dass Studierende der Informatik zu Hause an zwei Monitoren arbeiten. Für diese Untergruppe soll Webstebs ein GUI anbieten, das ein Arbeiten an mehreren Monitoren zulässt und unterstützt.

Auch ist eine generelle Überarbeitung bzw. Verbesserung des GUI erwünscht: Dabei gilt es, die Usability zu verbessern und sicher zu stellen, dass alle GUI-Komponenten einem einheitlichen Design (Look and Feel) folgen.

2.2.2 Devices

Zurzeit existieren zu wenige Devices und somit mangelt es den Studierenden an spannenden und lehrreichen Aufgabenstellungen. Es sollen somit weitere Devices implementiert und dazu passenden Übungsaufgaben entworfen werden.

Die bereits bestehenden Devices verfügen über ein noch simples, noch nicht verfeinertes Aussehen. Diese sollen intuitiver bedienbar, ansprechender und aussagekräftiger gestaltet werden. Weiter soll das Aussehen der Devices einheitlicher sein.

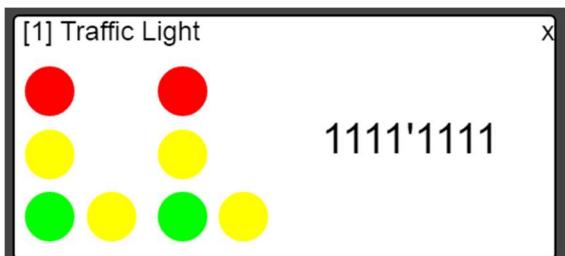


Abbildung 4 Traffic Light Device

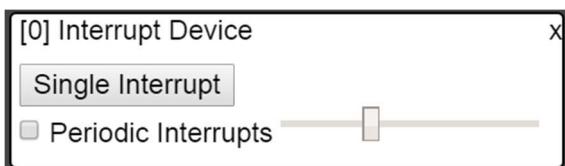


Abbildung 5 Interrupt Device

2.2.3 Portvergabekonzept

Den Devices werden Ports zugeordnet, über welche die CPU mit den Devices kommuniziert. Zurzeit geschieht die Portvergabe inkrementell, d.h. für jedes im GUI neu hinzugefügte Device wird die Portnummer um eins erhöht.

Werden zum Beispiel drei Devices hinzugefügt und löscht davon das dazwischen, wird der Port des entfernten Devices künftig nicht mehr automatisch vergeben. Stattdessen wird die Portnummer weiterhin inkrementiert.

Aus Sicht der Studenten besteht zusätzlich die Unannehmlichkeit, die Ports jedes Mal von Hand vergeben zu müssen. Besonders ins Gewicht fällt dies bei bereits geschriebenen Assemblercode, der bestimmte Devices über im Code vordefinierte Ports ansprechen soll.

2.3 Vision, Strategische Ziele

Die strategischen Ziele des Systems Webstebs lauten wie folgt:

- Webstebs kann im kommenden Semester FS18 in den chp - Vorlesungen eingesetzt werden
- Den Studierenden können spannende Aufgabenstellungen geboten und allenfalls fortlaufend neue kreiert werden.
- Studenten nutzen die Applikation im Unterricht als auch von zu Hause aus auf Windows, Mac und Linux (Desktop).
- Das GUI unterstützt die Studierenden bei ihrer Arbeit mit Webstebs.

2.4 Generierter Mehrwert

Durch die Verbesserung der Usability bietet Webstebs neu ein intuitives, semantisch strukturiertes User Interface, wodurch Studierende sich leichter zurechtfinden. Unsere Usabilitytests haben gezeigt, dass Probanden 40% weniger Zeit benötigten, um unsere Testaufgaben zu lösen. Das neue Design bietet Neuanwendern und Einsteigern der Thematik Mikroprozessor eine gute Unterstützung: Bei der Entwicklung des neuen User Interfaces wurde darauf geachtet, dass die Studierenden weniger Fehler machen können. Dies erleichtert ihnen das Arbeiten mit Webstebs.

Das Design und die Menüführung wurden vereinheitlicht, damit sich die Studierenden besser zurechtfinden können.

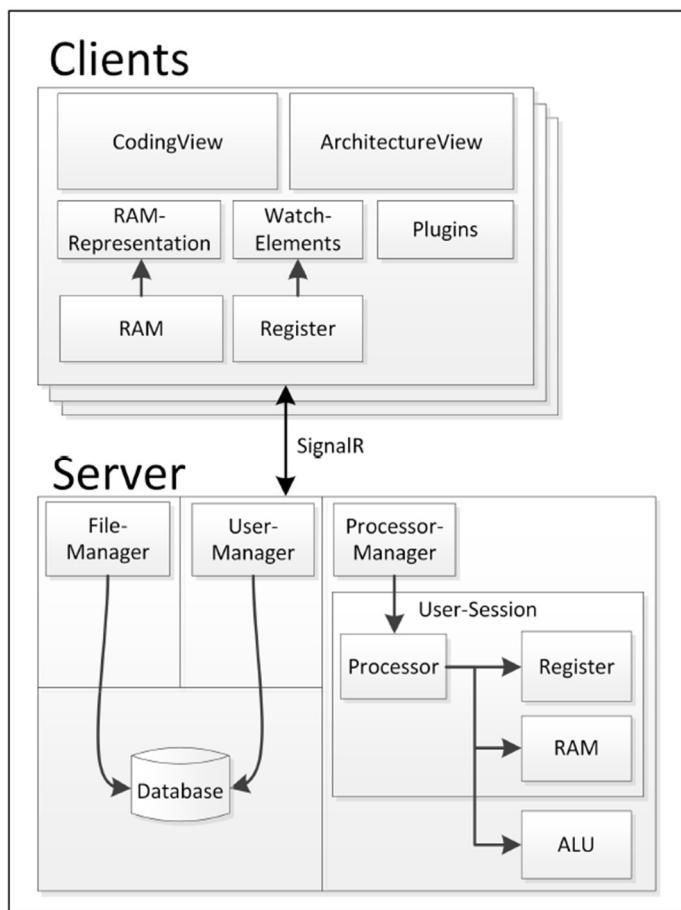
Zudem ermöglichen die von uns neu entwickelten Devices, den Dozierenden für die Studierenden essentielle und lehrreiche Aufgaben zum Thema Input und Output zu entwerfen.

Ebenfalls ist es nun möglich, eine viel grössere Anzahl an Devices auf dem Bildschirm darzustellen und zu bedienen.

2.5 Umsetzung

Bei Webstebs 3 handelt es sich aus technischer Sicht um ein reines Frontendprojekt. Das Backend blieb somit unberührt und die Architektur (Client-Server) wurde in seiner bestehenden Form übernommen.

Die verwendeten Technologien werden im gleichnamigen Kapitel aufgelistet und erläutert. Nachfolgend zeigen die Kapitel "GUI", "Devices" und "Portvergabe" auf, welche Usabilityprobleme vorgefunden und wie diese behoben wurden.

**Abbildung 6** Grobarchitektur

3 Technologien

Webstebs verwendet folgende Technologien:

- .NET Framework 4.6
- SignalR 2.2.0
- jQuery 2.1.4
- Typescript 2.0
- CodeMirror 5.9

Die hier aufgelisteten Technologien werden in Webstebs2_Projektbericht.pdf genauer erläutert.

Hinweis: Zusätzlich wurden Bootstrap 3.0.0 sowie diverse jQuery - Plugins in Stebs5 verwendet. Diese wurden zwar in Webstebs3 nicht weiterverwendet, könnten aber künftig von Nutzen sein.

Bootstrap 3.0.0

Bootstrap ist ein CSS - Framework, welches bekannt für sein mobile-first Responsive Grid sowie vorgefertigte UI - Controls und jQueryplugins bekannt ist. Letztere beiden Punkte können für Webstebs von Nutzen sein.

Mehr Informationen unter: getbootstrap.com

Razor Syntax

Razor ist eine Markupsyntax, der es dem Server erlaubt, in einer HTML - Datei dynamisch Code auszuführen, bevor diese an den Client geschickt wird. Er ist Bestandteil der von Webstebs verwendeten Microsofttechnologien und kommt in Stebs5 bereits zum Einsatz.

Mehr Informationen unter: w3schools.com/asp/razor_intro.asp

Mousetrap.js

Eine leicht zu bedienende JavaScript - Library, die es erlaubt, Tastatur - Shortcuts zu definieren und zu behandeln.

Mehr Informationen unter: craig.is/killing/mice

3.1 jQuery Validation Plugin 1.11.1

Ein Plugin, welches Validierung von Formulareingaben erlaubt. Bemerkung: Kann als Alternative zu selbst geschriebener Validierung verwendet werden.

Mehr Informationen unter: jqueryvalidation.org

3.2 FontAwesome

Fontawesome bietet über 600 aufeinander abgestimmte, vektorbasierte Icons, welche mit CSS gestylt werden können. Macht das aufwändige Bearbeiten von bildbasierten Icons überflüssig.

Mehr Informationen unter: fontawesome.io/

3.3 JS - und CSS - Libraries einbinden

Webstebs gibt basierend auf .NET 4.6 eine klare Struktur vor. Scripts und CSS - Files müssen in vorgegebenen Ordnern hinzugefügt werden. Diese werden dann mittels Razor Syntax dynamisch in das <header> - Element eingefügt. Damit dies mit neuen Plugins und Frameworks funktioniert, muss die Konfigurationsdatei BundleConfig.cs angepasst werden.

Datei/Ordner in Stebs5	Beschreibung
Stebs5/Content	Hier werden .css - Files abgelegt. <i>Bsp: font-awesome.css</i>
Stebs5/Scripts	JQuery - und JavaScript - Code von Plugins und Frameworks werden hier untergebracht. <i>Bsp: mousetrap-1.5.3.min.js</i>
Stebs5/Views/Shared/_StebsLayout.cshtml Stebs5/Views/Shared/_Layout.cshtml	Beinhaltet das <header> - Element der Webstebs Webseite. Mit @Styles.Render() werden die Referenzen auf Stylesheets und mit @Scripts.Render() die Referenzen auf Scripts dynamisch eingebettet. <i>Bsp: siehe nachfolgendes Code Snippet.</i>
Stebs5/App_Start/BundleConfig.cs	Dies ist das Konfigurationsfile, das als Brücke für den dynamischen Import der Dateireferenzen dient. Hier werden Script- und Stylesheetbundles definiert. Die Bundles enthalten die Information für Render(), welche Files eingebettet werden müssen. <i>Bsp: siehe zweites Code Snippet.</i>

Stebs5/Views/Shared/_Layout.cshtml (Beispielcode)

Hier ist ersichtlich, dass Styles.Render() das Bundle namens "css" konsumiert und Scripts.Render() die Referenzen des modernizr - Bundles einfügt. Wer Schwierigkeiten hat, den dynamischen Import zu konfigurieren, der kann auch mittels [<link>](#) - Element und einem relativen Pfad auf ein File verweisen. Relative Pfade beginnen mit einem ~.

```
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - Stebs</title>
    <link rel="shortcut icon" href="~/Icons/favicon.png" />
    <link href="~/Content/font-awesome.min.css" rel="stylesheet" />
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
```

Stebs5/App_Start/BundleConfig.cs (Beispielcode)

Hier wird ein Scriptbundle definiert. Dessen Name lautet "stebs" und deren *inkludierte Files* werden in der gleichnamigen Methode aufgelistet.

```
bundles.Add(new ScriptBundle("~/bundles/stebs").Include(
    "~/Scripts/jquery-{version}.js",
    "~/Scripts/jquery.signalR-2.2.0.min.js",
    "~/Scripts/codemirror-5.9.min.js",
    "~/Scripts/mode assembler.js",
    "~/Scripts/mousetrap-1.5.3.min.js",
    "~/Scripts/mousetrap-global-bind.min.js",
    "~/Scripts/main.js"));
```

4 GUI

Das User Interface ist das erste, womit ein Endnutzer interagiert. Es entscheidet darüber, wie gut er sich zurechtfindet. Zu Beginn wurde der IST-Zustand von Webstebs ermittelt, um bekannte Mängel in der Usability zu überprüfen und unbekannte aufzudecken. Anschließend wurden Lösungsideen entwickelt und davon die sinnvollsten aus Endnutzer, Kunden und technischer Sicht implementiert. Eine SOLL-Messung mittels Usabilitytests überprüfte die Wirksamkeit des neuen Designs und erhob zugleich Feedback für weitere Verbesserungsideen. Diese flossen ebenfalls ins Finale Design ein.

4.1 Bekannte Probleme

Gemäss Kapitel Einleitung > Problemstellung > GUI.

4.2 Untersuchungen und Erkenntnisse

Zu Beginn des Projektes lag der Fokus darauf, zu verstehen, welche Probleme existieren und gelöst werden müssen. Hierfür verwendeten wir die im Verlaufe des Kapitels erwähnten Methoden aus dem Usability Engineering. Die daraus gewonnenn Erkenntnisse und Lösungsvorschläge werden aufgezeigt.

4.2.1 System- und Dokumentationsanalyse

Durch die Analyse der Dokumentation und Ausprobieren der Applikation "Webstebs" konnten erste Vermutungen angestellt werden, was verbessert werden muss. Am auffälligsten war, wie die Buttons für das Ausführen und Steuern von Assembly Code in zwei Gruppen aufgeteilt und voneinander getrennt positioniert sind (**Abbildung X**). Dies hat zur Folge, dass der Endnutzer einen langen Weg mit Maus und Blick zurücklegen muss.

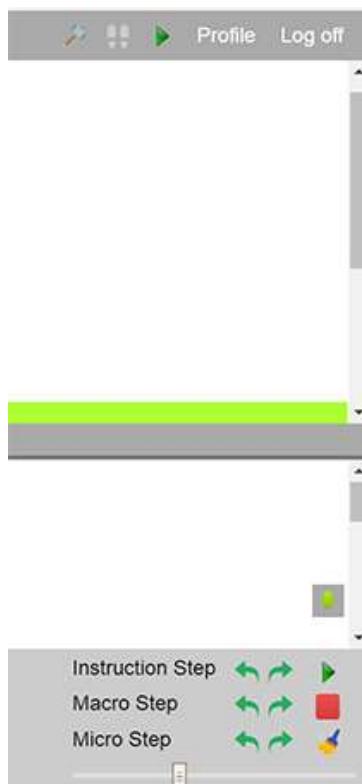


Abbildung 7 Simulation Control in der alten Webstebs Version

Auch fiel auf, dass der Run - Button (▶) als auch die doStep- (⟳) und undoStep - Buttons (⟲) redundant auf dem GUI vorkamen. Diese können jeweils mit nur einer Instanz pro Button repräsentiert werden. Die dadurch theoretisch wegfallende Unterscheidung zwischen Instruction-, Macro- und Microstep kann auf einem anderen Wege im GUI weitergeführt und dem Endnutzer angeboten werden.

Beim Durchspielen unterschiedlicher Use Cases wurde zusätzlich bemerkbar, dass der step into - Button (⇨⇨) nur direkt nach dem Assemblieren zum Einsatz kam. Danach fand er keine weitere Verwendung im Workflow...

Assemblieren (🔧) → Step into (⇨⇨) → run code (▶) oder doStep (⟳).

Diese Vermutungen wurden in der ersten Testiteration mit Probanden überprüft.

4.2.2 Erster Usabilitytest: IST-Zustand (UX-Test)

In diesem UX - Testlauf wurde das GUI von Webstebs anhand der aktuellen Live - Version (Version 5.0.6546.26001) getestet. Dabei wurden im ersten Teil des Tests den Probanden vier Aufgaben und anschliessend im zweiten Teil Fragen gestellt.

Folgendes Template wurde für diesen Test erstellt. Dieses enthält ausgewählte Aufgabenstellungen:

[Usability IST-Zustandsmessung: Testbogen TEMPLATE](#)

Durchgeführte Tests

Um die Zielgruppe angemessen zu vertreten, wurden Studierende und Absolventen der Informatik eingeladen, an den Tests teilzunehmen.

Individuelle Testberichte:

- [IP5_IODW -UX IST Test-Francois Martin.pdf](#)
- [IP5_IODW -UX IST Test-Jalil Hashemi.pdf](#)
- [IP5_IODW -UX IST Test-Dominic Huggenberger.pdf](#)
- [IP5_IODW -UX IST Test-Lukas Buerki.pdf](#)

Wichtige Befunde & Verbesserungsvorschläge

Basierend auf den gemachten Beobachtungen und den gestellten Fragen stachen folgende Befunde besonders hervor:

Befund-ID	Befund	Verbesserungsvorschläge	Bemerkung
UX01	Output verwirrt: Ergebnisse von ADD, SUB usw. werden instinktiv im Outputfenster gesucht.	Der Name "Output" irritiert und sollte geändert werden.	Gewohnheit in Java: Syso-Output geht in Konsole
UX02	Geschlossene Panels fallen nicht auf. Suche nach RAM in Aufgabe 01 fehlgeschlagen, d.h. niemand kam auf die Idee, danach zu suchen.	Sobald der Endnutzer eingeloggt ist, könnte das Panel im geöffneten Zustand angezeigt werden.	Abzuklären, ob technisch machbar und sinnvoll..
UX03	Breakpoints fallen nicht auf bzw. liegen nicht auf einem intuitiven Workflow.	Dem Endnutzer könnte per Popup-Fenster ein Hinweis gegeben werden.	
UX04	Das "Step Into" - Icon (脚步) nicht aussagekräftig.	Ein Debugbutton einführen, der mit einem Käfericon dargestellt und als Debugging bezeichnet wird. Alternativ: Icon ersetzen.	Wie in IntelliJ.
UX05	Assembliericon (螺丝刀): <ul style="list-style-type: none"> - Icon zu wenig aussagekräftig. - fällt nicht auf, da der farbliche Kontrast zum Hintergrund zu gering ist. - Endnutzer klicken intuitiv zuerst auf Run (wie IntelliJ) 	Lösungsvorschläge: <ul style="list-style-type: none"> - Anderes Icon wählen. - Hinweis in einem Popupfenster: "Assemblieren bitte". - Assemblieren als Button mit ausgeschriebenem Label 	
UX06	Das Textfeld, in welchem die Portnummer eines Devices festgelegt wird, erwies sich als missverständlich: Von einigen Endnutzern wird der Wert zum Teil als "Anzahl hinzuzufügender Devices" verstanden.	Verwendungszweck mit einem Label betiteln.	

UX07	Werden mehrere Devices hinzugefügt, erscheint ab dem vierten jedes weitere ausserhalb vom sichtbaren Bereich des Webbrowsers. Dem Endnutzer ist es nicht möglich, per Scrollen diese sich anzusehen.	Die Devicesliste muss scrollbar sein. Allenfalls könnte das Problem auch behoben werden, indem sie einen eigenen, auf Devices zugeschnittenen Bereich im GUI erhält.	
UX08	Die User haben verschiedene Bedürfnisse, wenn die Panels umplatziert werden können, können alle abgedeckt werden.	Panels Drag & Drop - fähig machen	techn. machbarkeit prüfen.
UX09	Für Anfänger erwies sich der Einstieg in Assembly Code als tendenziell schwierig. Befehle wie z.B. SUB oder PUSH waren nicht auf anhieb verständlich.	Eine Hilfestellung direkt in Webstebs einbauen. Dies könnte ein Seitenpanel mit wichtigen Tipps sein oder ein Link auf ein separates Wiki.	Alternativ auch als PDF-Download. Erzeugt funktionale / emotionalen Mehrwert f. Studenten, da der Einstieg erleichtert wird.
UX10	Icons von Continue, Stop und Reset an ungünstiger Stelle. 	Wie in IntelliJ: Alle zur Steuerung und Ausführung von Code relevanten Buttons eher oben rechts, horizontal anordnen.	Instinktiv wird an der Stelle nach dem Stop-Button gesucht, an welcher sich der Start-Button befindet.
UX11	Reset - Icon (👉) nicht aussagekräftig.	Durch ein aussagekräftiges Icon ersetzen.	
UX12	Temposlider: <ul style="list-style-type: none">• Beschriftung fehlt (Titel / Label)• Geschwindigkeitsangabe fehlt• Semantisch ungünstig positioniert (Hat nichts mit Debugging zu tun). Es wirkt so, als wussten die Entwickler nicht, wohin mit dem Slider.	<ul style="list-style-type: none">• Beschriften• Geschw.anzeige einführen: z.B. 1x, 2x, 5x oder x%• An semantisch sinnvolleren Ort im GUI positionieren. Evtl eigenen Raum geben.	
UX13	Filenamelabel zu kurz	Mehr Platz geben.	

UX14	<p>Inkonsistentes Visual Design für Aktion "Auf-/Zuklappen".</p> <p>Das Auf- und Zuklappen von Panels wurde uneinheitlich gestaltet. Das Output - Fenster öffnet sich durch einen Klick auf seinen horizontalen Titelbalken, während das RunAndDebug - Panel direkt darunter sich per einen Button am rechten Bildschirmrand öffnen lässt.</p>	<p>eine einheitliche Design Language implementieren: "Wenn du GUI-Element X siehst, kannst du immer Aktion Y tun."</p> <p>D.h. eine vom Endnutzer an mehreren Stellen ausführbare Aktion muss immer mit einem einheitlich aussehenden GUI-Element repräsentiert werden.</p>	
UX15	<p>Wenn...</p> <ol style="list-style-type: none"> 1. Code ausgeführt wird und... 2. während dem ein neues File geöffnet wird, dann... 3. wird die ausführung des alten Codes nicht unterbrochen. Der gelbe Balken bewegt sich weiter. 	<p>Beim Öffnen eines Files wird automatisch "Stop" gedrückt. Evtl. auch "Reset".</p>	
UX16	<p>Icons oberhalb von RAM:</p> <ul style="list-style-type: none"> - Das Exporticon (Export icon) direkt neben dem Pfeil icon suggeriert, dass der Pfeil für "Upload" da ist. Seine eigentliche Funktion ist jedoch das Auf- und Zuklappen des RAMs. - Fallen zu wenig auf. 	<p>Die Icons bzw. dessen ausgeführte Funktionalität im GUI so positionieren, dass sie semantisch korrekt verstanden werden.</p> <p>Icons mehr Weissraum geben (sichtbarer machen).</p> <p>Evtl mit Farbkontrast und Hovereffekten arbeiten.</p>	
UX17	<p>Das RunAndDebug - Panel nimmt viel Raum in Anspruch. Das gleiche gilt für die grafische Darstellung der Register.</p>	<p>Da davon ausgegangen werden kann, dass die Buttons rechts vom RunAndDebug-Panel umpositioniert werden, bleiben die Register alleine zurück.</p>	

		<p>So könnten diese im RAM- oder einem eigenen Panel platziert werden, allenfalls sogar in tabellarischer Darstellung.</p> <p>Bsp:</p> <table border="1"> <thead> <tr> <th>Register</th><th>Wert</th></tr> </thead> <tbody> <tr> <td>AL</td><td>E4</td></tr> <tr> <td>BL</td><td>C2</td></tr> </tbody> </table>	Register	Wert	AL	E4	BL	C2	
Register	Wert								
AL	E4								
BL	C2								
UX18	<p>Wird Assembly Code ausgeführt, erscheinen drei Checkboxen, die das Schrittempo beeinflussen: Instruction-, Macro- und Microstep.</p>  <p>Hierbei sind nur dessen Label anklickbar, die Checkboxen selbst jedoch nicht.</p>	<p>Checkboxen ebenfalls klickbar machen, sofern diese im GUI enthalten bleiben.</p>							
UX19	<p>Probanden versuchten mit den Pfeiltasten die Aktionen doStep (→) und undoStep (←) auszuführen.</p>	<p>Steuerung über Pfeiltasten implementieren.</p>							
UX20	<p>Nachdem eine Datei gespeichert wurde, gibt es kein Feedback für den Benutzer.</p>	<p>Meldung anzeigen z.B: "Datei XY wurde erfolgreich gespeichert."</p>							
UX21	<p>Es ist unklar wie der Name einer Datei geändert werden kann.</p>	<p>Den Namen einer Datei beim Speichern ändern, wie bei "Speichern unter" unter Windows.</p>							
UX22	<p>Mangelhafte automatische Portvergabe: Werden beispielsweise vier Devices hinzugefügt, so sind die Ports 0, 1, 2 und 3 vergeben. Wird nun das Device mit dem Port 1 gelöscht und an-</p>	<p>Portvergabealgorithmus verbessern. Automatisch den nächsten freien Port ins Inputfeld schreiben.</p>							

	<p>schliessend ein neues hinzugefügt, so wird dieses automatisch auf den Port 4 gelegt. Die Devices liegen nun auf den Ports 0, 2, 3 und 4.</p> <p>Zusätzlich wird die nächste freie Portnummer nicht im dafür vorgesehenen Textfeld angezeigt.</p>		
UX23	Bei einigen Buttons und Menüs ist unklar was sie machen.	Mit Tooltips können kurze Beschreibungen angezeigt werden.	
UX24	Für den Endnutzer bestehen Unklarheiten, wann assembled werden muss und ob das Assemblieren erfolgreich war.	<p>Der Run - Button soll deaktiviert und somit nicht klickbar bleiben, bis zu dem Moment, an welchem assembled wurde. Der enabled - und der disabled - Zustand des Buttons sollten visuell voneinander unterscheidbar sein.</p> <p>Wenn das Assemblieren erfolgreich war, den Endnutzer per z.B. Popup informieren.</p>	

Beliebte und unbeliebte IDEs / Software

Nach den Testaufgaben wurden die Probanden gebeten, Softwarelösungen zu nennen, die als Inspiration dienen können oder als solches gemieden werden sollten.

positiv aufgefallen	negativ aufgefallen
<p>Die Einfachheit von...</p> <ul style="list-style-type: none"> • Arduino IDE • Processing <p>Die Usability und Featurerichness von...</p> <ul style="list-style-type: none"> • IntelliJ <p>Die Hilfestellung(en) von...</p> <ul style="list-style-type: none"> • https://regexp.com/ 	<p>Die Komplexität von...</p> <ul style="list-style-type: none"> • Visual Studio • Photoshop (für Anfänger überforderndes GUI) <p>...vermeiden.</p>

Fazit

In diesem Testdurchgang konnte validiert werden, dass...

- das Hinausragen der Devices ausserhalb des Browserfensters ein Hauptproblem darstellt.
- Endnutzer nicht verstanden, zuerst den Code assemblieren zu müssen, bevor sie ihn ausführen können. Dies liess sich teilweise auf den nächsten Punkt zurückführen:
- die Mehrheit der verwendeten Icons sich als missverständlich, fehlleitend oder unberührbar erwiesen und somit die Funktion der Buttons fehlinterpretiert wurde.
- es den Endnutzern schwer fiel, auf Hilfsmittel wie den Breakpoints oder Anzeigen wie den RAM und den Architecture Canvas zu stossen. Hierfür fehlte visuelles Feedback.
- über Slidern und einigen Textfeldern Labels fehlten. So blieb der Endnutzer anfangs im Unklaren, welchen Zweck diese GUI-Elemente erfüllen.

Nebst den Befunden ergaben sich auch wertvolle Ideen, wie Webstebs verbessert werden kann. Unter anderem zeigte sich, dass die meisten Probanden bereits über Erfahrungen mit bekannten IDEs aus der Java - Welt verfügen.

4.2.3 Benchmarking: Der Einfluss bekannter IDEs

Der Usabilitytest auf dem Live-Server hatte gezeigt, dass GUI-Elemente neu positioniert bzw. miteinander gruppiert und im UI leichter auffindbar sein müssen. Da die meisten Probanden mit bekannten IDEs wie Eclipse oder IntelliJ vertraut sind, beherbergen diese eine gewisse Erwartungshaltung: Per Klick auf Run kann man Code ausführen und per Klick auf den Käfer kann man Code debuggen. Es ist daher sinnvoll diese Semantik im neuen Design von Webstebs zu berücksichtigen.

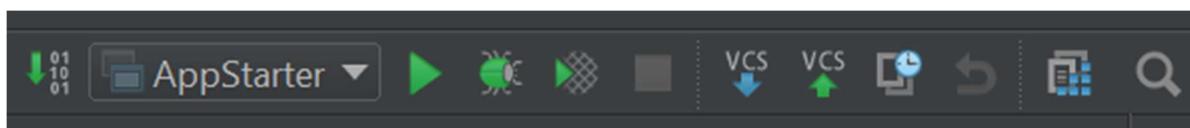
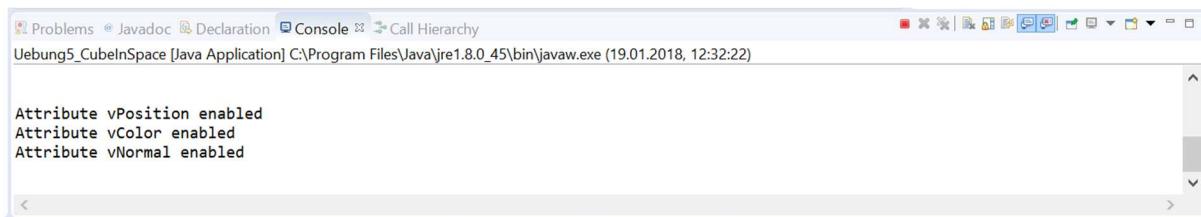


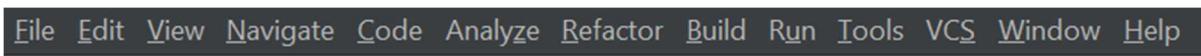
Abbildung 8 IntelliJ Werkzeugkasten

Eine weitere Herausforderung besteht darin, mit dem im Browserfenster zur Verfügung stehenden Platz auszukommen. IntelliJ und Eclipse sind zwei gute Beispiele dafür, wie viele Features platzsparend untergebracht werden können. Dies wird bewerkstelligt durch diverse Seitenpanels, Tabpanels und einer ausführlichen Menübar.

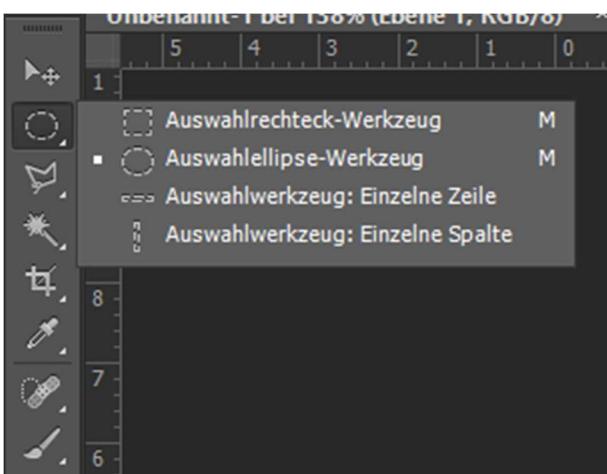
**Abbildung 9 Eclipse Tabs**

Tabs bieten sich an, um viele, thematisch verwandte Funktionen platzsparend unterzubringen. Jedoch muss berücksichtigt werden, dass inaktive Bereiche nicht ersichtlich sind. Vorsichtiges Planen des Inhaltes ist somit Voraussetzung für gute Nutzerführung.

Das Menubar - Pattern hat den Vorteil, seit Jahrzehnten im Einsatz und weit verbreitet zu sein. Das heisst, viele Nutzer wissen, wie damit umzugehen ist. Es eignet sich ideal dafür, um übliche Aktionen unterzubringen. Wichtig hier ist eine für den Endnutzer einleuchtende Gruppierung der Menüitems mit dessen Dropdownitems. Beispielsweise sind I/O - Operationen üblicherweise unter "File" vorzufinden.

**Abbildung 10 Photoshop Menubar**

Zwar wurden Photoshop CS6, Adruino IDE und Processing ebenfalls näher untersucht, doch wuchs daraus keinen grösseren Einfluss auf das Gesamtdesign. Das Werkzeugseitenpanel von Photoshop erwies sich dank den Icons zwar als platzsparend, jedoch auch als einsteigerunfreundlich, da man sich mit zahlreichen Icons und Werkzeugen zuerst vertraut machen muss.

**Abbildung 11 Photoshop Werkzeugkasten**

Arduino IDE und Processing sind zweifellos Paradebeispiele für schlankes Design, welche nur das Nötigste beinhalten und somit leicht zu bedienen sind. In ihrer vertikalen Struktur reihen sich von oben nach unten gesehen eine Menübar, Steuerungsknöpfe für Assemblieren, Code ausführen und I/O - Operationen, einen tab-orientierten Codeeditor und zu unterst die Ausgabekonsole ein. Auf das Design von Webstebs hatte schlussendlich nur die semantische Trennung der Buttons einen Einfluss, nämlich auf die Form des Assemble - Buttons. Denn dieser kann auch während dem Debuggen weiterhin betätigt werden und verlangt ein Gruppen unabhängiges Aussehen.

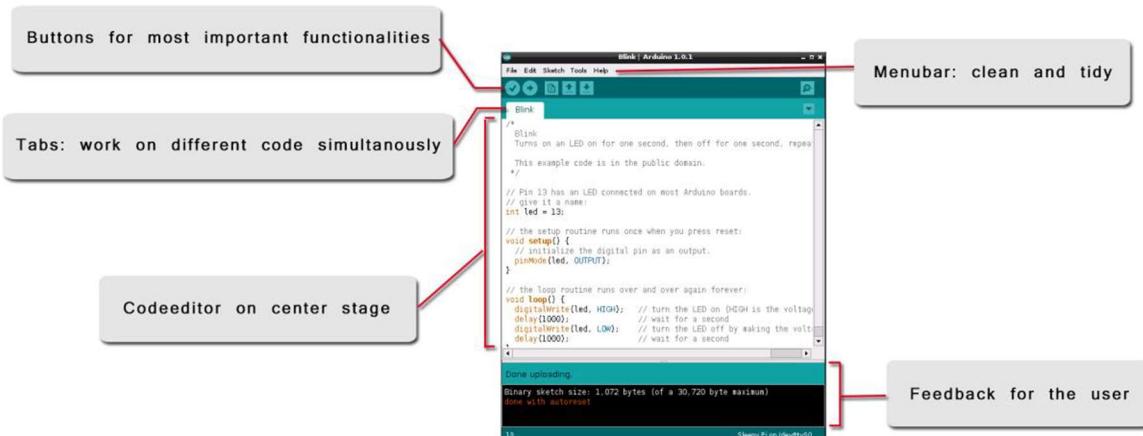


Abbildung 12 Arduino IDE

4.2.4 Prototyping & Idea Testing (UX-Test)

In diesem Testdurchlauf wurden erste Verbesserungsideen für das GUI getestet. Hierfür wurde ein Papierprototyp erstellt, an dem Probanden die einzelnen Aufgabenstellungen aus dem Fragebogen durchgespielten.



Abbildung 13 Papier-Prototyp erstellen

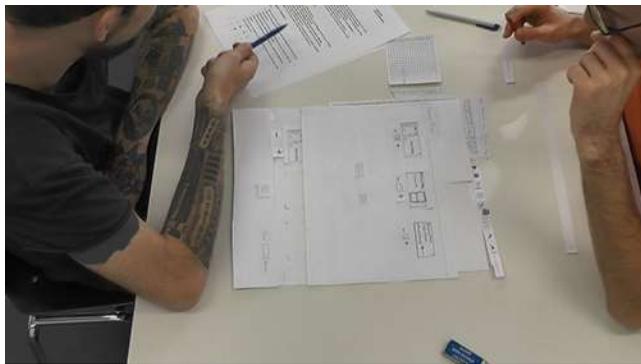


Abbildung 14 Papier-Prototyp UX-Test

Folgendes Template wurde für diesen Test erstellt.

[Usability Papierprototyp Iteration 01: Testbogen TEMPLATE](#)

Durchgeführte Tests

Individuelle Testberichte:

- [IP5 IODW -UX IST Test-Valentino Rugolo.pdf](#)
- [IP5 IODW -UX IST Test-Nikko Purbu.pdf](#)
- [IP5 IODW -UX IST Test-Dominique Dubler.pdf](#)

Prototyp Setups, Getestete Design - Variationen

Hinweis: Der auf den Fotos gezeigte Papierprototyp wurde basierend auf Probandenfeedback von Test zu Test verbessert, sodass er in seiner aktuellsten Form zu sehen ist. Hinweise darauf sind unter **Was wurde optimiert / verändert** aufgelistet.

01 Device hinzufügen

Inspiriert durch Eclipse, wird eine tab-orientierte Lösung ausprobiert. Nebst der Deviceanzeige am unteren Bildschirmrand beinhaltet diese das Hinzufügen mehrerer Instanzen eines Devices sowie die direkte und nachträgliche Portvergabe unterhalb des Devices.

01-1 Einstiegsbildschirm

Dieser Bildschirm zeigt die Ausgangslage nach dem Einloggen. Die Steuerungsbuttons befinden sich neu oben rechts und die Register links davon. Darüber eine Menübar und am unteren Bildschirmrand zwei auswählbare Tabs: OpCode und Devices (aka Plugins).

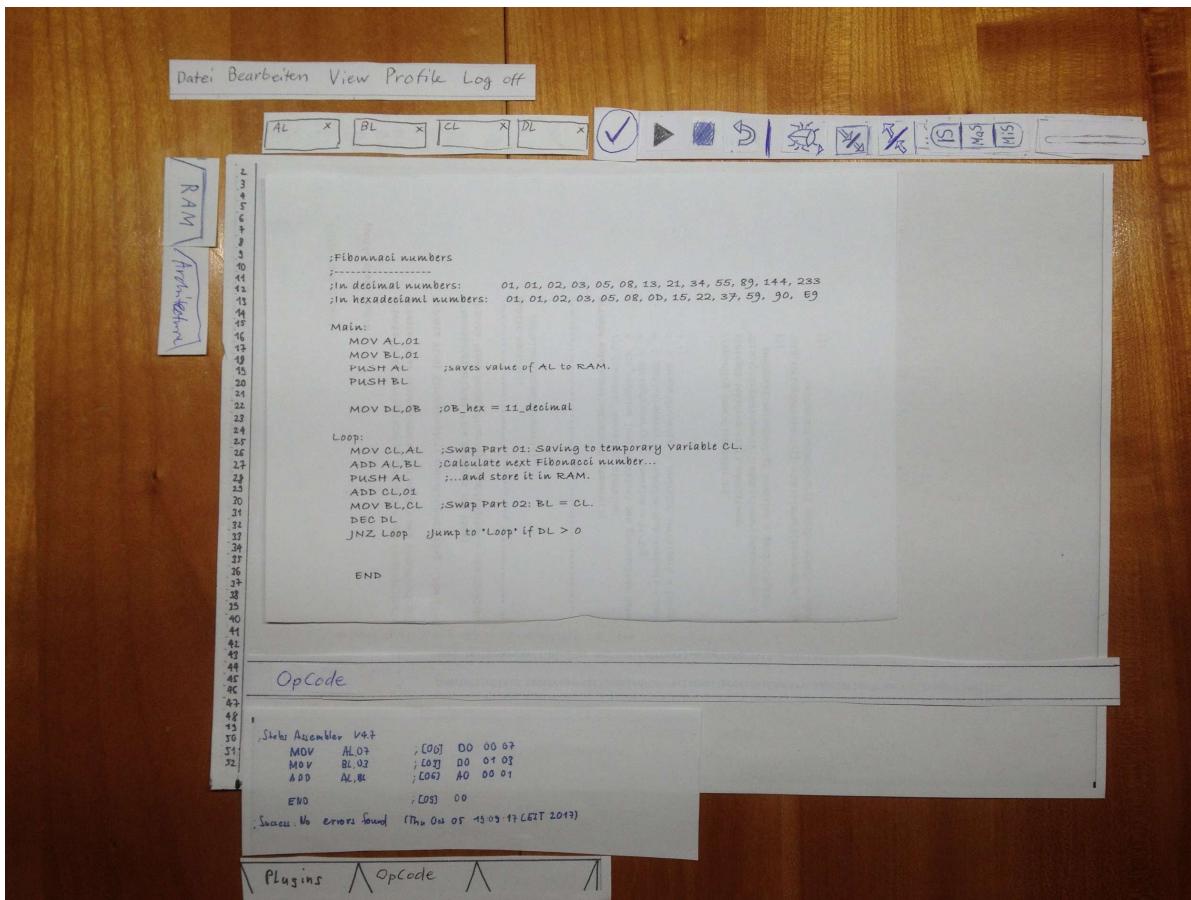


Abbildung 15 Einstiegsbildschirm

Erkenntnisse

positiv	neutral	negativ
<ul style="list-style-type: none"> Mehrere Plugin-instanzen leicht hinzufügbar. Menübar oben links wurde sofort verstanden. Register prominent, gut ersichtlich positioniert. Buttons oben rechts gut ersichtlich. 	<ul style="list-style-type: none"> Wichtig: Tooltippunterstützung bei Buttons wie Run, Assembly, Single Step etc. von essentieller Bedeutung. 	<ul style="list-style-type: none"> Nicht ausgeklappter "RAM" - Seitenpanel führte dazu, dass RAM praktisch nie beachtet wurde.

Was wurde optimiert / verändert

- Output wurde zu OpCode umbenannt.

01-2 Via Tab zur Device - Übersicht

Hier ist angedacht, die Devices wie Fotos einer Fotogallerie anzugeordnen. Durch Klick auf den + - Button gelangt man zum Hinzufügebildschirm. Dies könnte entweder durch Öffnen eines neuen Fensters oder durch eine Scrollanimation vollzogen werden.

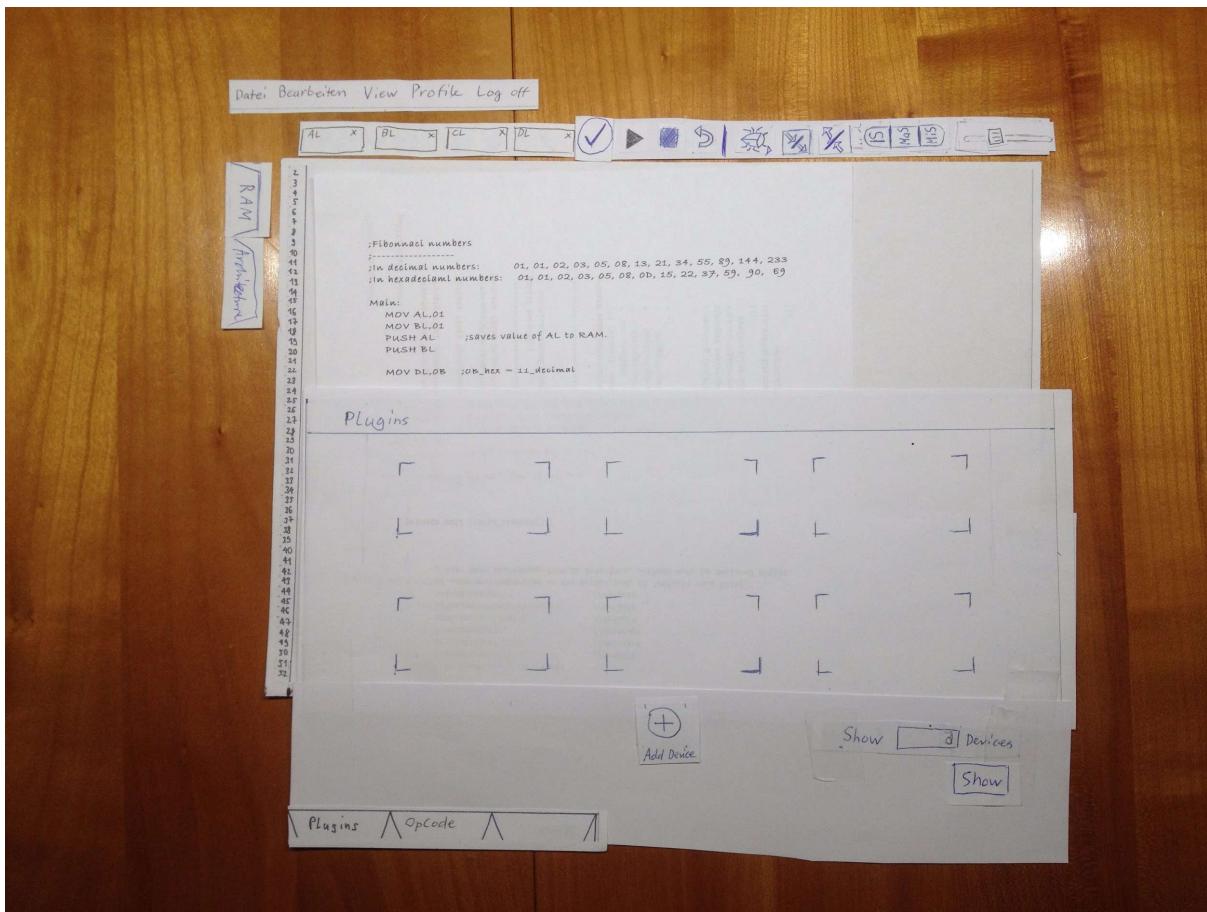


Abbildung 16 Device - Übersicht

Erkenntnisse

positiv	neutral	negativ
<ul style="list-style-type: none"> Positionierung unten wirkt übersichtlich. 		<ul style="list-style-type: none"> Abzuklären: Wie mit unterschiedlichen Bildschirmgrößen umgehen.

Was wurde optimiert / verändert

- Unten rechts wurde ein "Show" - Button hinzugefügt, um bestätigen zu können, dass z.B. drei Devices gleichzeitig angezeigt werden sollen.

01-3 Via “Add Device” - Button zum “Choose Plugin” - Menü

Die zur Auswahl stehenden Devices werden grafisch repräsentiert. Der Endnutzer kann null bis n Exemplare eines Devicetyps hinzufügen.

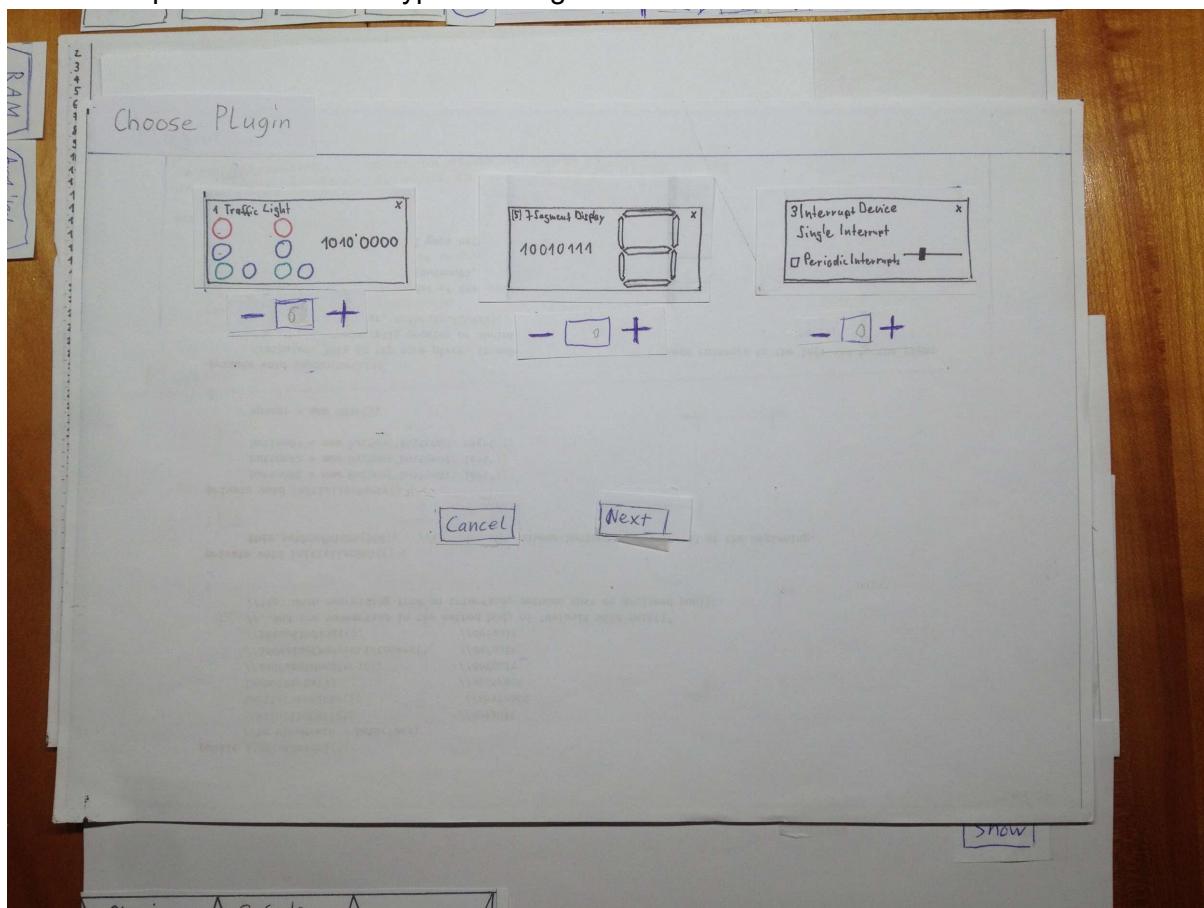


Abbildung 17 Add Device

Erkenntnisse

positiv	neutral	negativ
<ul style="list-style-type: none"> Mehrere Plugin-instanzen leicht hinzufügbar. + und - semantisch verstanden. Pluginauswahl in separatem Fenster wirkt übersichtlich. 		

Was wurde optimiert / verändert

- In der oberen, rechten Ecke jedes angezeigten Plugins wurde ursprünglich eine Checkbox angezeigt. Diese war dazu da, um eine Auswahl zu de-/aktivieren. Dies wurde als redundant befunden und somit entfernt.

01-4 Ports auswählen (Variante 1) [NICHT empfohlen]

Hier wurde getestet, ob die Portvergabe in tabellarischer Darstellung Sinn ergäbe. Jedoch erwies sich dies als unnötiger Zwischenschritt.

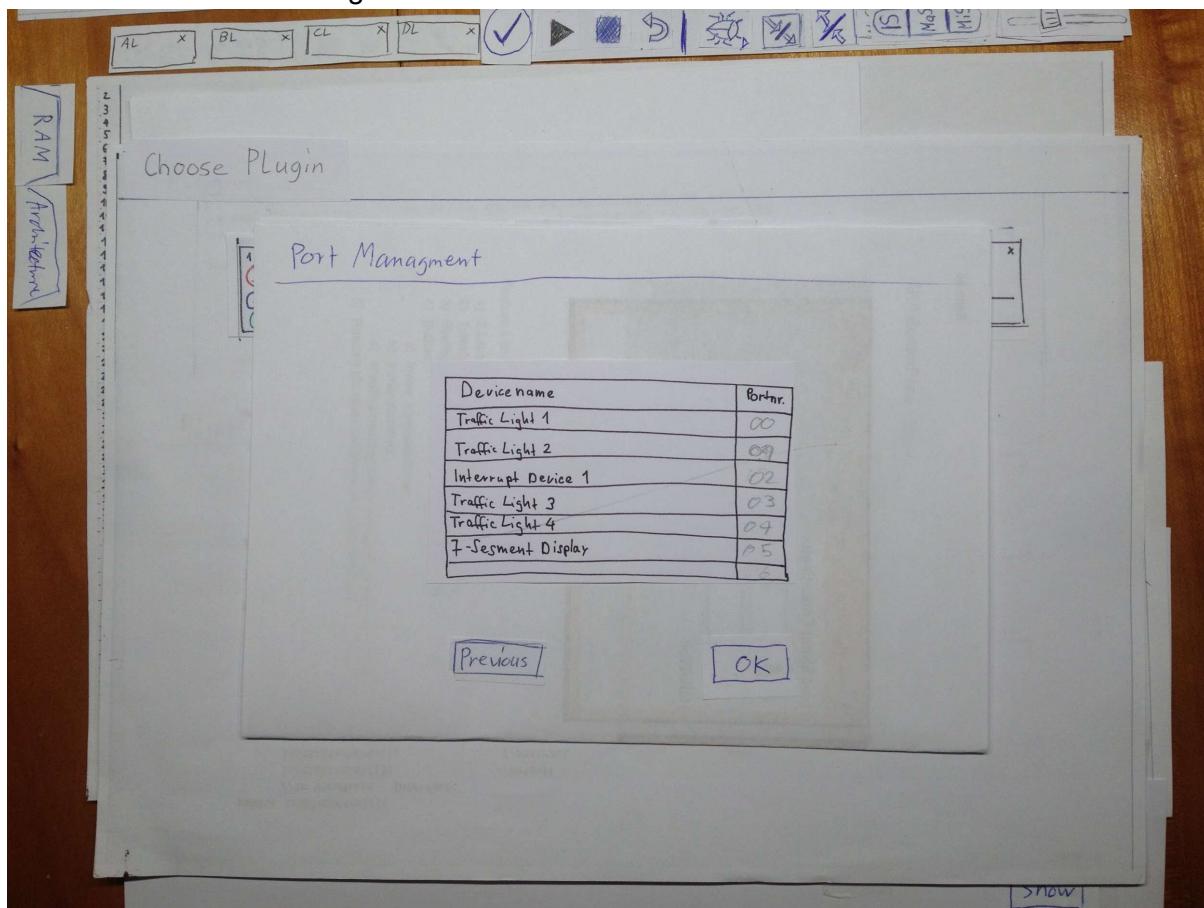


Abbildung 18 Portmanagement

Erkenntnisse

positiv	neutral	negativ
		<ul style="list-style-type: none">Dieser Zwischenschritt brachte keinen zusätzlichen Nutzen und kann weggelassen werden.

01-4 Ports auswählen (Variante 2) [empfohlen]

Standardmässig wird Devices automatisch eine Portnummer zugeteilt. Diese kann direkt im Device - Panel editiert werden.

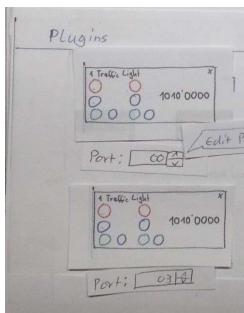


Abbildung 19 Portnummer editieren

Erkenntnisse

positiv	neutral	negativ
<ul style="list-style-type: none">Wurde von Probanden gut verstanden.Wurde einem Device ein unerwünschter Port zugewiesen, entfällt nun das Löschen und mit neuem Port wieder Hinzufügen.		

Was wurde optimiert / verändert

- Inkrementiere- und Dekrementier-Buttonicons ersetzt: Von + / - zu ^/ v.
- Label hinzugefügt.

01-5 Devices zur Device-Übersicht hinzugefügt

Die hinzugefügten Devices sind nun unterhalb des Codeeditors ersichtlich. Die Anzahl gleichzeitig angezeigter Devices kann unten rechts bei "Show n devices" angepasst werden. Jedoch ist noch unklar, wieviele maximal angezeigt werden sollten, um das User Interface nach wie vor übersichtlich zu halten.

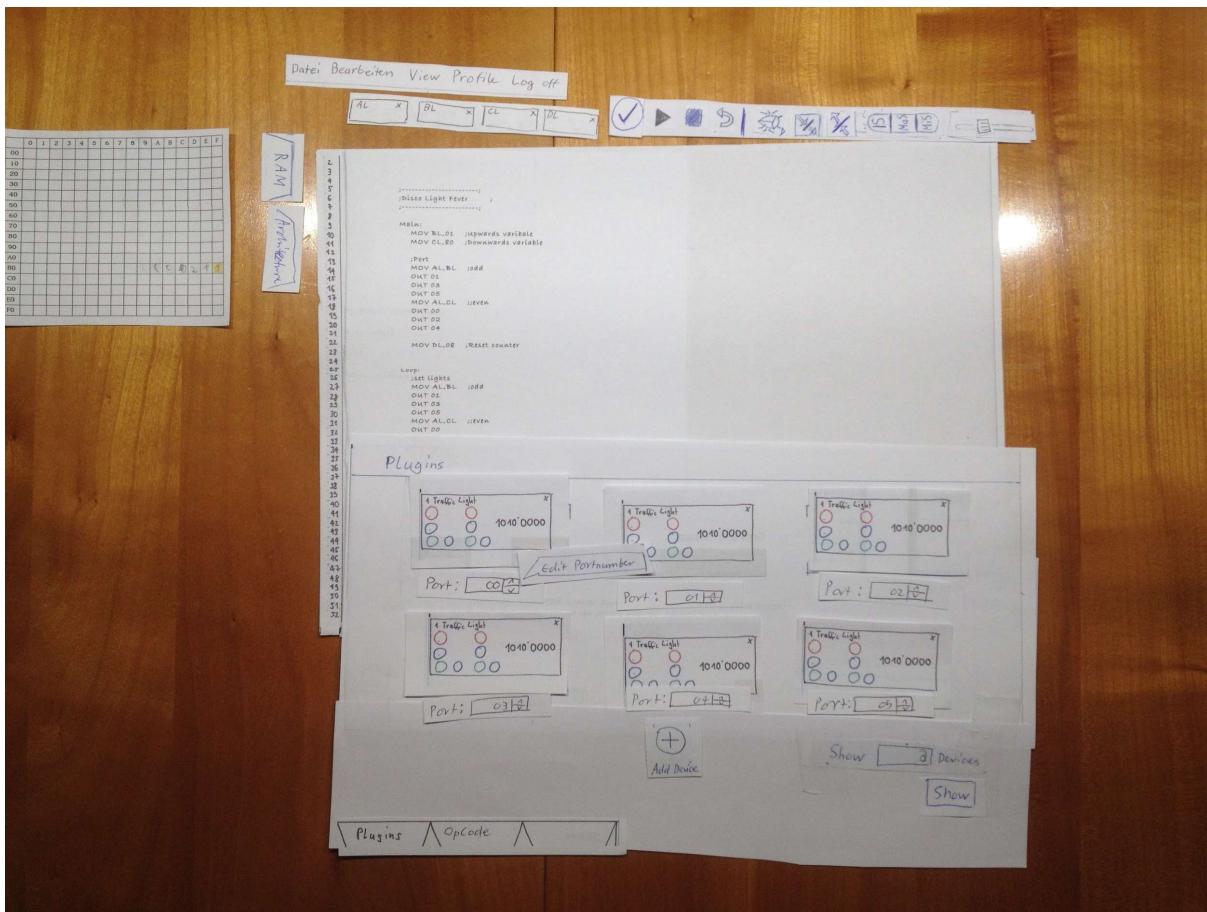


Abbildung 20 Device-Übersicht Variante 1

Erkenntnisse

positiv	neutral	negativ
<ul style="list-style-type: none"> Mehrere Plugin-instanzen leicht hinzufügbar. 	<ul style="list-style-type: none"> Wenn Code assembling wird, darf Tab nicht auf "OpCode" wechseln. Besser: Pop-up - Meldung à la "Erfolgreich assembled" (vgl. m. Processing) 	<ul style="list-style-type: none"> technisch aufwendige Implementierung. Bedeutet grösere Veränderung im Source Code.

Was wurde optimiert / verändert

- Zur Erhöhung/Verringerung der Portnummer wurden + und - Symbole verwendet. Dies überlagerte sich semantisch mit der Erhöhung/Verringerung der Anzahl zu instanzierender Plugins im Pluginauswahlfenster.

02 Devices: Alternatives Layout

Hier wurde eine vertikale Anordnung der Devices getestet. Die Liste lässt sich bequem mit dem Mausrad auf und ab scrollen.

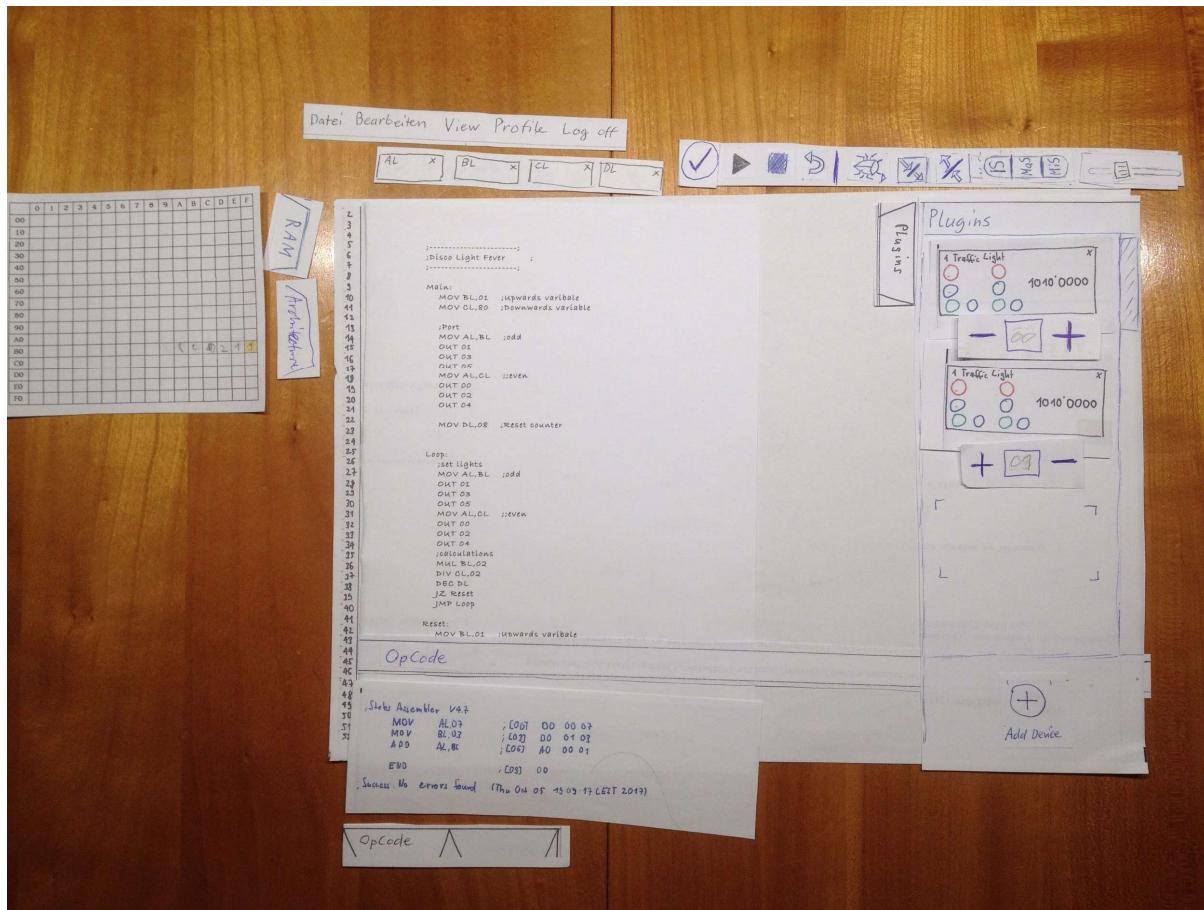


Abbildung 21 Device-Übersicht Variante 2

Erkenntnisse

positiv	neutral	negativ
<ul style="list-style-type: none"> Deviceübersicht: leicht zu verstehen, dass bei mehr als drei Devices nach unten gescrollt werden kann. Könnte als linksbündiges Seitenpanel implementiert werden. Technisch mit weniger Aufwand umsetzbar. 		<ul style="list-style-type: none"> Wenn links und rechts Panels sowie unten ein Tab geöffnet sind, wird der Nutzer kognitiv überfordert - GUI wirkt überladen.

03 Alternative Registerposition

Alternativ zur Position unterhalb der Menübar wurde getestet, wie eine Platzierung der Register innerhalb des RAM - Seitenpanels sich verhält.

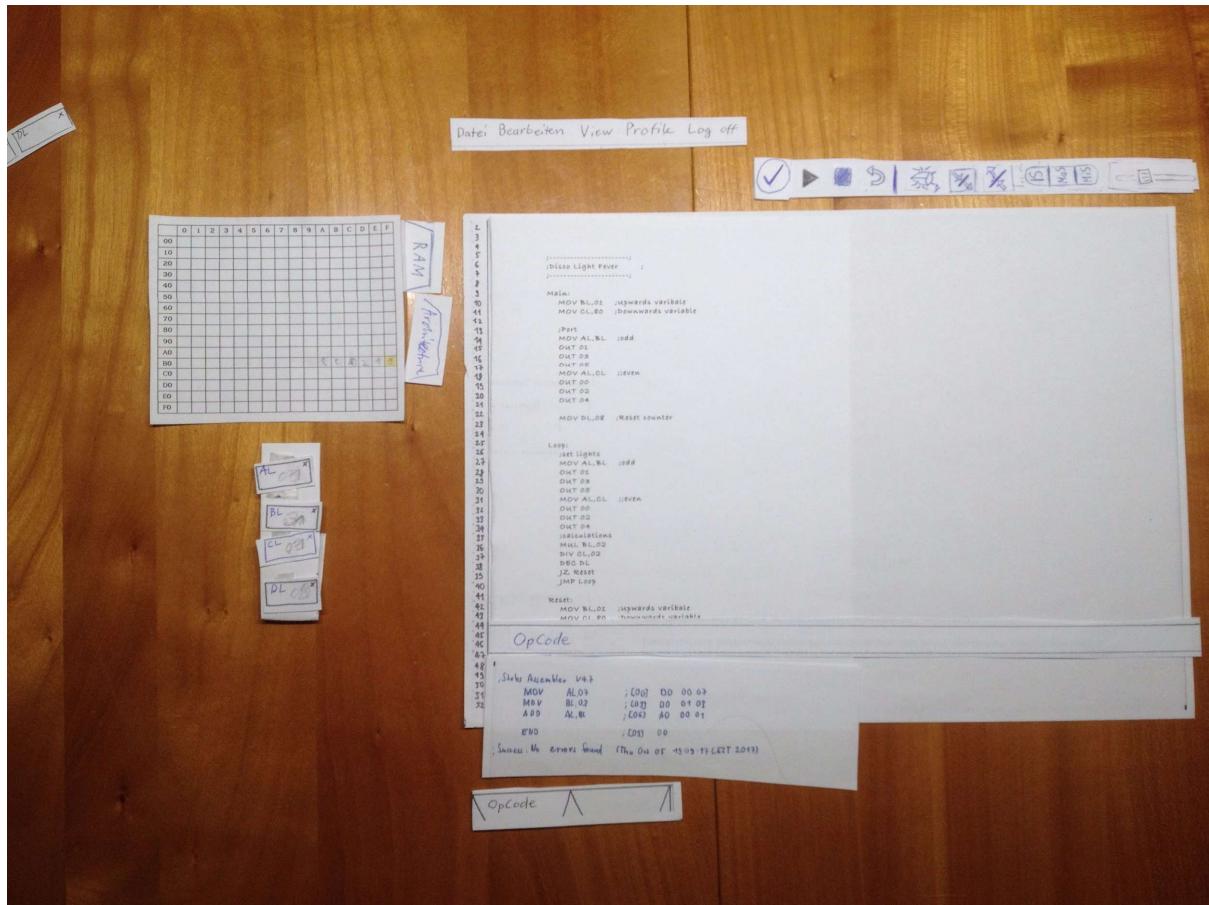


Abbildung 22 Alternative Registerposition

Erkenntnisse

positiv	neutral	negativ
	<ul style="list-style-type: none"> Für die Testpersonen machte es keinen merkbaren Unterschied, ob die Register unterhalb der Menübar oder im RAM - Panel zu finden sind. Voraussetzung ist, dass das besagte Panel von Anfang an aufgeklappt ist. 	

04 Hinweise: "Assemblieren nicht vergessen" und "Es gibt Breakpoints"

Während den Tests fiel immer wieder auf, dass Probanden direkt auf "Run" klickten und beim Debuggen nicht an die Breakpoints dachten. Um dieses Problem zu lösen, wurde eine Pop-Up - Meldung eingeführt und getestet.

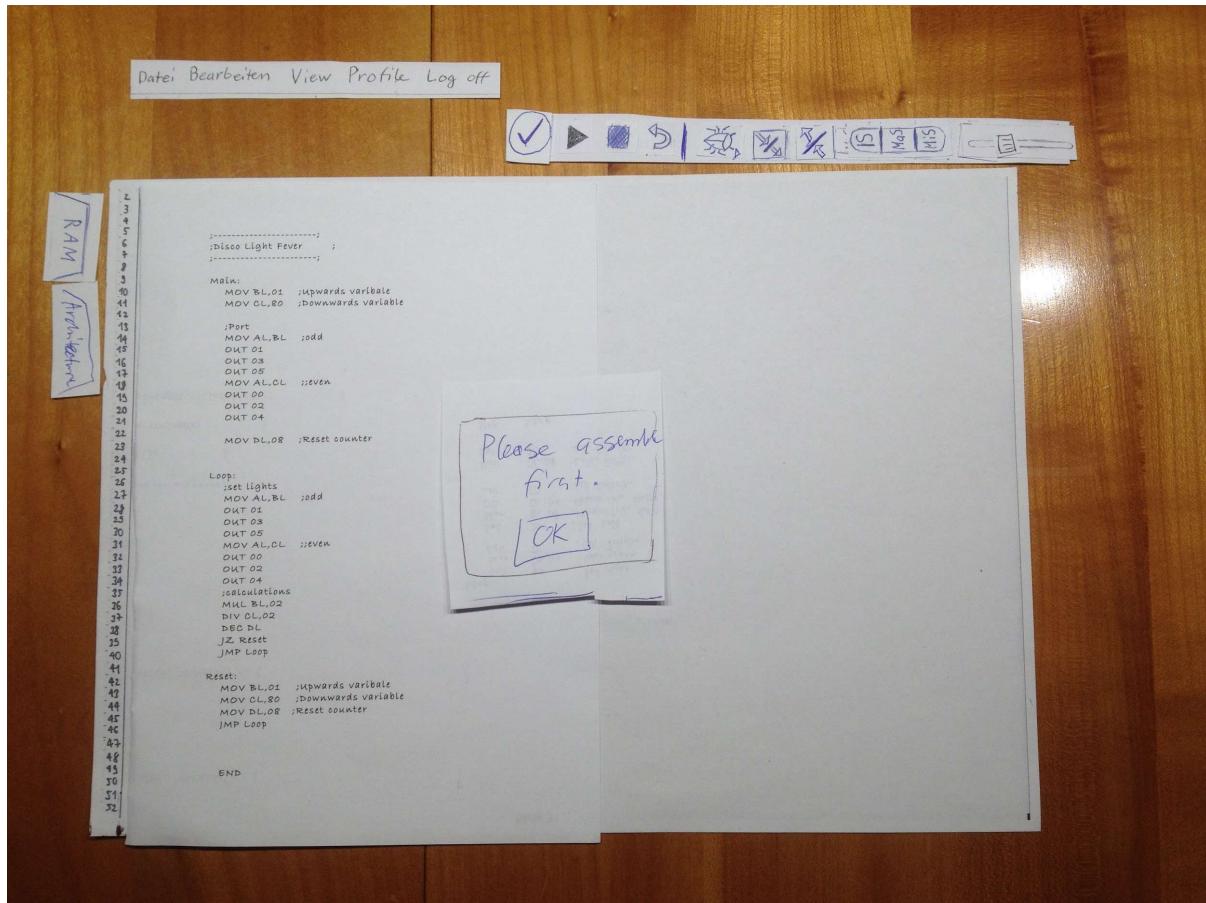


Abbildung 23 Hinweise: "Assemblieren nicht vergessen"

Erkenntnisse

positiv	neutral	negativ
<ul style="list-style-type: none"> Probanden erkannten ihren Fehler umgehend und korrigierten ihn. Den Endnutzern war bewusst, dass es Breakpoints gibt und man zuerst auf Assemble klicken muss. 		<ul style="list-style-type: none"> Da Assemble jedes Mal geklickt werden muss, erhob sich die Frage, ob dies sinnvoll sei. Zum Beispiel könnte man das Debugging von diesem Zwischen-schritt befreien.

05 Simulation Control

Die Simulation Control dient zum Ausführen und Debuggen von Code.



Abbildung 24 Simulation Control

Der Test mit dem Papier-Prototypen zeigte, dass...

- die semantische Trennung in "Code ausführen" und "Code debuggen" verstanden wird.
- der Assemble - Button nur durch Ausschlussverfahren entdeckt wurde.
- das Icon für den Reset - Button mehr an "Undo" erinnerte.
- der Käfer mit einem kleinen "play" unterhalb verstanden wird.
- die Fusion von "Step into" mit "doStep" sinnvoll ist.
- die im Test verwendeten Icons für "doStep" und "undoStep" unklar sind. Es müssen leichter verständliche dafür gefunden werden.
- der Wechsel zwischen Instruction-, Macro- und Microstep und der SpeedSlider selten gebraucht wird. Diese müssen dem zu Folge nicht immer ersichtlich sein.

Aus dem Feedback wurden folgende Veränderungen abgeleitet:

- Der Assemble - Button wird neu mit einem Label gekennzeichnet. Das Icon fällt weg.
- Der SpeedSlider und die StepModi werden jeweils in einem Dropdown untergebracht. SpeedSlider wird der Buttongruppe "Code ausführen" und die StepModi der "Code debuggen" zugeordnet.
- Die im Test verwendeten Icons für "doStep" und "undoStep" werden ersetzt.

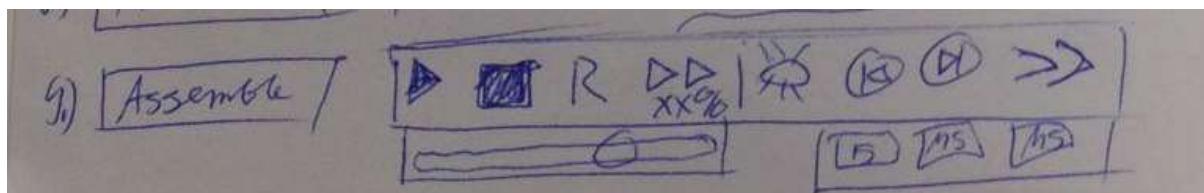


Abbildung 25 Nach der 9. Skizze stand das Layout fest. Mit den Icons wurde hingegen weiterexperimentiert.

5 Umsetzung

Das User Interface wurde basierend auf den gewonnenen Erkenntnissen aus Systemanalyse, Benchmarking und Usabilitytests wesentlich umstrukturiert. Weitere Entscheidungsfaktoren, die zum neuen Design beigetragen haben, sind die Wünsche des Kunden und der Probanden.

5.1 Durchgeführte Tests: SOLL - Messung

Das neue GUI-Design wurde nach seiner Implementierung mit Probanden getestet. Ziel war es zu prüfen, ob die in der Analyse erhobenen Usabilitymängel behoben wurden und das User Interface intuitiver nutzbar ist.

Individuelle Testberichte:

- [IP5_IODW-UX_SOLL_Test-Jan Müller.pdf](#)
- [IP5_IODW-UX_SOLL_Test-Tom Ohme.pdf](#)
- IP5_IODW-UX_SOLL_Test-Ivana Stoilova.pdf

Die Tests bestätigten eine verbesserte Nutzerführung. Dies zeigte sich darin, wie spielend leicht sich die Probanden im User Interface zurechtfanden, sowie in der für die gestellten Aufgaben benötigten Zeit. Es wurde circa 40% weniger Zeit, als in den vorangegangenen Usabilitytests der IST - Messung benötigt.

Des Weiteren wurden auch neue Anforderungen und Verbesserungsmöglichkeiten entdeckt, die ebenfalls ins Finale Design einflossen.

5.2 Design

Folgende Veränderungen wurden im GUI vollzogen und werden im Kapitel "Umgesetzte UI-Komponenten" näher aus Design- und Codeperspektive erläutert:

1. Simulation Control
2. Menubar
3. Filemanager
4. Sidepanels
5. RAM
6. Architecture
7. Register
8. RunAndDebug (entfernt)

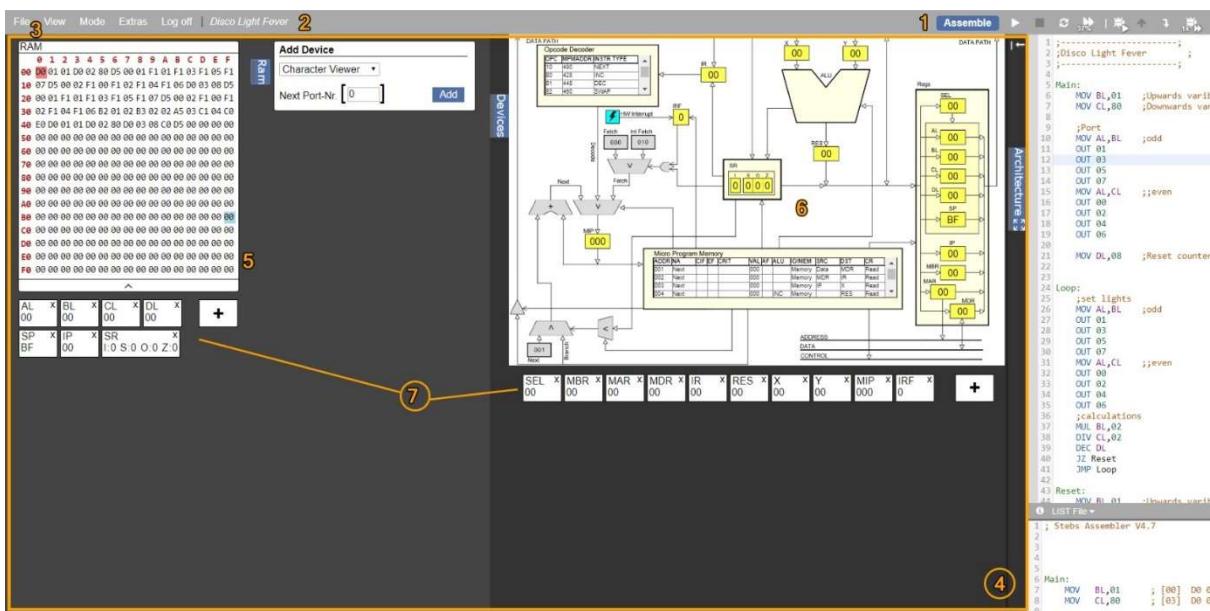


Abbildung 26 Übersicht Webstebs

5.3 Source Code

Die Architektur von Webstebs wurde in ihrer ursprünglichen Form belassen. Hingegen wurden bestehende Klassen und Views ergänzt, oder wo nötig, verändert.

5.3.1 Views

Hierbei handelt es sich um den HTML und CSS - Code von Webstebs. Folgende Dateien wurden im Projekt "Stebs5" verändert:

- Stebs5/Content/stebs.css
- Stebs5/Views/Home/Index.cshtml

Zusätzlich wurde die FontAwesome eingeführt. Diese bietet über 600 optisch einheitlich wirkende Icons, die spielend leicht mit CSS gestylt werden können. Die hierfür relevanten Dateien sind abgelegt in....

- Stebs5/Content/font-awesome.css
- Stebs5/Content/font-awesome.min.css
- Stebs5/fonts/FontAwesome.otf
- Stebs5/fonts/font-awesome-webfont.* (.eot, .svg, .ttf, .woff, .woff2)

5.3.2 Klassen

Hierbei handelt es sich um die Logik des Frontends in Form von TypeScript und JavaScript. Das aktualisierte Klassendiagramm enthält die wichtigsten, vollzogenen Veränderungen im Source Code. Zwar wurden keine neuen Klassen erstellt, jedoch entstanden neue Methoden oder es wurden diverse Eventhandler und neue Variablen eingeführt. Dies, um das neue Design und die damit verbundene Interaktion mit dem Endnutzer zu ermöglichen.

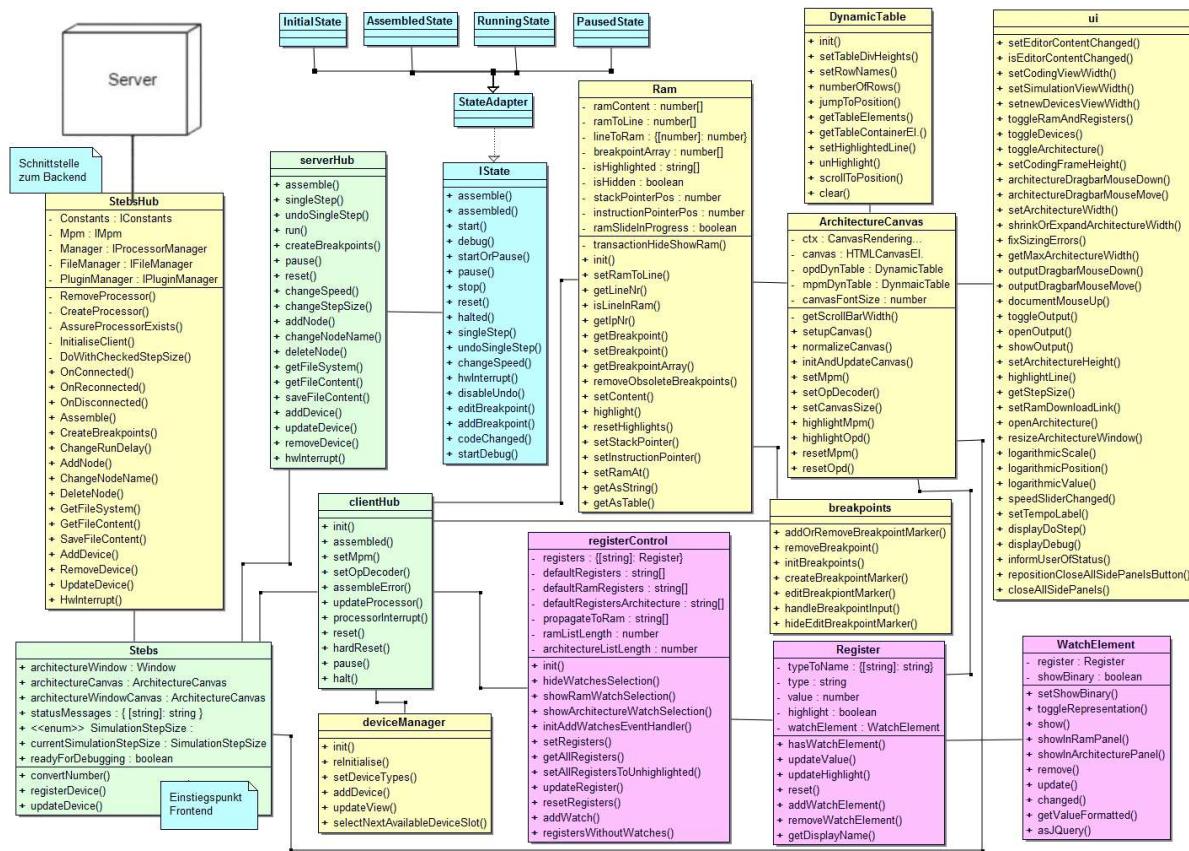


Abbildung 27 Klassendiagramm

Die Veränderungen im Source Code werden im nachfolgenden Abschnitt detailliert, im Kontext des dazugehörigen UI-Elements erläutert. Alle zuvor bereits existierten Codeelemente sind bereits in Webstebs2_Projektbericht.pdf dokumentiert. Damit der Einstieg in das Projekt Stebs5 leichter fällt, wird hier eine ergänzende Einleitung gegeben. Diese Orientiert sich anhand der in der Solution vorfindbaren Files und hebt die von uns gemachten Erfahrungen hervor.

main.ts

Diese Datei enthält das Modul "Stebs", in welcher der serverHub und der clientHub vorzufinden sind. Diese beiden Hubs enthalten Methoden, die zur Kommunikation mit dem Server dienen. Dabei wird der serverHub verwendet, um Befehle an den Server zu senden, während der clientHub Informationen und Updates vom Server empfängt.

Ebenfalls wichtig zu erwähnen ist, dass Stebs das Root - Modul des gesamten Frontends ist. Alle sich in diesem Modul befindlichen Variablen und Funktionen, die public sind, können gemäss nachfolgenden Beispielen aufgerufen werden:

- Variable in main.ts aufrufen: `Stebs.currentSimulationStepSize`
- Das Attribut eines Objektes in main.ts aufrufen: `Stebs.visible.architecture`
- Methode in serverHub aufrufen: `Stebs.serverHub.changeStepSize()`
- Eine Funktion aus ui.ts aufrufen: `Stebs.ui.toggleArchitecture()`

So werden Stebs.Widths und Stebs.Heights für das dynamische Resizing von UI-Elementen verwendet, während Stebs.visible auskunft darüber gibt, ob ein Element gerade sichtbar ist.

Ein weiterer Hinweis: Um sicherzustellen, dass Code nicht ausgeführt wird, bevor die Webseite geladen ist, bietet JQuery die `$(document).ready()` - Funktion an. Diese befindet sich in main.ts. In dessen Funktionskörper werden Initialisierungen durchgeführt und (einen Teil der) Eventhandler definiert.

executionStateMachine.ts

Die hier implementierte State Machine wird für die Simulation Control benötigt. Diese regelt, wann der Endnutzer welche Aktionen durchführen darf. Buttons werden dementsprechend deaktiviert oder aktiviert. Das Interface IState garantiert, dass alle States dieselben Zustandsübergänge anbieten. Ob diese dann wirklich Logik auslösen, wird in den einzelnen, konkreten Klassen InitialState, AssembledState, RunningState und PausedState definiert. Der StateAdaptor dient dazu, grundlegende Implementierungen anzubieten, die an die konkreten States vererbt werden. So müssen nur noch Zustandsübergänge implementiert werden, die eine spezifische Logik erfordern.

Da die StateMachine verändert werden musste, erfolgt eine genauere Erläuterung im Kapitel Umgesetzte UI-Komponenten > Simulation Control.

ui.ts

Stebs.ui stellt Funktionen zur Verfügung, die das User Interface steuern. Dazu gehören beispielsweise das dynamische Resizing der Panels, das kreieren von Pop-Ups und das Ein- und Ausblenden von UI-Elementen.

registers.ts

Die Register unterliegen konzeptionell einer MVC - Struktur, deren Komponenten im File registers.ts vorzufinden sind:

- Model: Register
- View: WatchElement
- Control: registerControl

5.4 Umgesetzte UI-Komponenten

5.4.1 Simulation Control

Unter Simulation Control sind alle UI - Elemente zu verstehen, die für das Assemblieren, Ausführen und Debuggen von Code relevant sind.

Design und gelöste Probleme

Im überarbeiteten Design befinden sich alle Elemente der Simulation Control nun im oberen, rechten Bildschirmrand. Dies behebt das in der Analyse markant aufgefallene Problem der langen Maus- und Blickwege zwischen zwei voneinander getrennten Elementgruppen.

Ebenfalls wurde die Semantik aus bekannten Java - IDEs eingeführt. Es gibt nun eine Buttongruppe "Code ausführen" (links) und eine -gruppe "Code debuggen" (rechts).

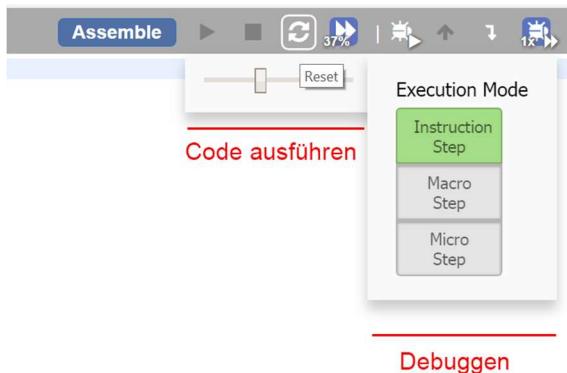


Abbildung 28 Pop-ups Simulation Control

Dropdown - Menüs für sekundäre Controls

Da der SpeedSlider und die StepTempi (auch Execution Modi genannt) seltener verwendet wurden als andere Controls, fanden diese in Dropdownmenüs ihren neuen Platz. So nehmen sie weniger Raum im UI ein, sind aber nach wie vor leicht zugänglich im Falle, dass sie gebraucht werden. Und zwar genau dann, wenn sie gebraucht werden.

Buttons und verbesserte Nutzerführung

Der Assemble - Button wurde hingegen stärker ins Rampenlicht geführt. Durch die klare Bezeichnung mittels Textlabel und einer auffallenden Call-to-action - Farbe werden Endnutzer nun animiert, von Anfang an den Code zu assemblyieren. Die Nähe zu den anderen Controls stellt sicher, dass dieser Button nach wie vor als Teil der Simulation Control verstanden wird.

Intuitiv verständliche Icons einzusetzen war ein immer wieder auftauchendes Thema im Rahmen des Projekts. Diese Problematik wurde gelöst, indem FontAwesome eingeführt wurde. Dabei handelt es sich um eine Sammlung bewährter, optisch aufeinander abgestimmte Icons. Zusätzlich sei erwähnt, dass sich damit auch Icons basteln lassen, die aus mehreren einzelnen bestehen. Hier das Mapping der Icons von alt zu neu:

Code Ausführen			Code Debuggen		
Button	alt	neu	Button	alt	neu
Assemble			Start Debug		
Run			Step into		
Stop			undoStep		
Reset			doStep		
SpeedSlider			Execution Modi		

Um den Nutzer zusätzlich zu führen, werden...

- zum aktuellen Zeitpunkt nicht benutzbare Buttons deaktiviert.
- aktive Buttons hingegen verfügen über einen Hovereffekt, der das Icon einerseits vergrößert und andererseits umrahmt.
- die Buttons, an welche die Dropdowns gebunden sind, erfüllen eine zusätzliche Funktion: Ihr Label dient dazu, den Endnutzer darüber informiert zu halten, wie schnell der Code bzw in welchem Modus dieser ausgeführt wird.



Abbildung 29 Geschwindigkeits-Slider

Der neue "Run Debug" - Button () wurde aus Usabilitygründen eingeführt:

- In IntelliJ wird ein Käfericon () verwendet, um dessen Debugmodus zu starten. Diese semantische Belegung wird hiermit übernommen und der Erwartungshaltung von Javaentwicklern dadurch Rechnung getragen.
- Funktional ist er eine Kombination aus den Buttons "Assemble" und "Run". Das heisst, dem Nutzer wird unter Umständen ein Klick erspart.
- Falls noch keine Breakpoints erstellt wurden, wird zuerst assembliert und dann ein Pop-Up - Fenster angezeigt. Auf diesem wird der Endnutzer informiert, dass ihm Breakpoints fürs Debuggen zur Verfügung stehen.

Redundante und selten benutzte Controls eliminieren

Redundante Controls war eines der Probleme, die behoben wurden. Zuvor gab es “doStep” () und “undoStep” () in drei Geschmacksrichtungen: Instructionstep, Macrostep und Microstep.

Die Wahl des StepTempo wurde für die ganze Applikation vereinheitlicht und in das Execution Mode - Dropdown ausgelagert. Dies ermöglichte die Anzahl der doStep und undoStep - Buttons auf jeweils einen zu verringern.

Ein weiterer Kandidat war der “Run” - Button. Dieser kam zuvor zweimal vor. Nun wurde auf eine Instanz reduziert. Das Besondere daran ist, dass dieser sich zu einem “Pause” - Button verwandelt, wenn Code ausgeführt wird.

Ähnlich erging es auch dem “Step into” - Button (), der sich nun den Platz mit “doStep” () teilt. Sobald “Run” oder er selbst gedrückt wird, erscheint “doStep” an seiner Stelle. So wird vermieden, dass ein selten benutztes Control unnötig Platz im User Interface einnimmt und den Endnutzer ablenkt.

Source Code

Dieser Abschnitt führt auf, welche Änderungen im Source Code durchgeführt wurden.

executionStateMachine

Damit die Semantik des “Code debuggen” eingeführt werden konnte, mussten Änderungen an der executionStateMachine.ts durchgeführt werden.

Für den neu eingeführten Button “Run Debug” musste ein neuer Zustandsübergang namens “startDebug()” eingeführt werden. Falls Breakpoints gesetzt wurden, wechselt die StateMachine direkt in den RunningState, ansonsten in den AssembledState.

Für den neu interpretierten “Step into” - Button wurde der zuvor schon vorhandene “debug()” - Übergang so umfunktioniert, sodass dem Endnutzer ein Klick auf Assemble erspart bleibt. Dieser führt nun direkt vom Initial- in den PausedState.

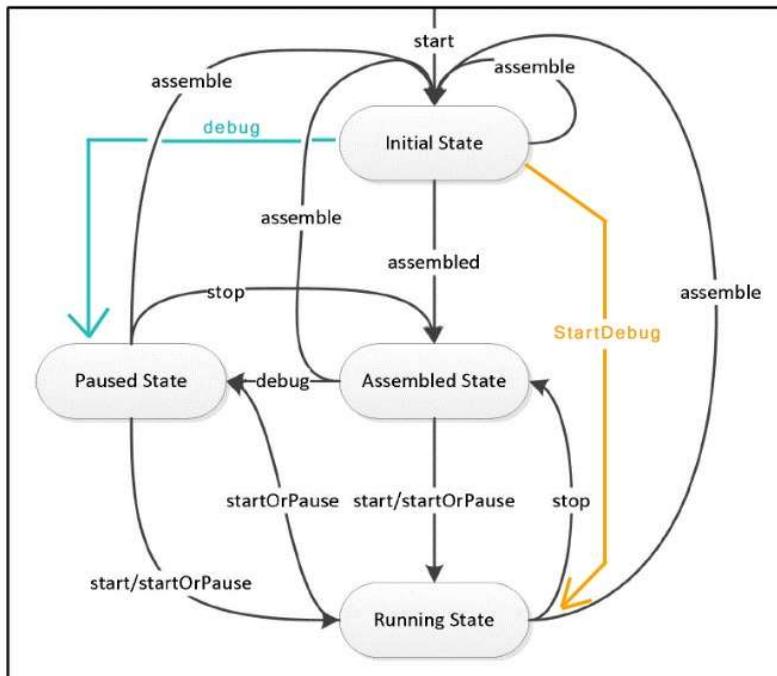


Abbildung 30 neues State Diagramm

Für den Wechsel zwischen den beiden Buttons "Step into" (↓) und "doStep" (↓) werden die beiden Funktionen `Stebs.ui.displayDoStep()` und `Stebs.ui.displayDebug()` angeboten. Diese werden jeweils zum richtigen Zeitpunkt von der StateMachine aufgerufen.

States lassen nur gewisse Aktionen zu. Welche erlaubt sind, wird in dessen Konstruktoren festgelegt. Zuvor gab es die `actions` `instructionStep`, `macroStep`, `microStep` und ihre `undo`-Gegenstücke. Diese wurden parallel zur Vereinfachung des User Interfaces zu `doStep` und `undoStep` vereinheitlicht.

```

var actions = {
    assemble: 'assemble',
    start: 'start',
    debug: 'debug',
    pause: 'pause',
    continue: 'continue',
    stop: 'stop',
    reset: 'reset',
    doStep: 'doStep',
    undoStep: 'undoStep',
    hwInterrupt: 'hwInterrupt'
};
    
```

Damit doStep und undoStep weiterhin den richtigen ExecutionStep vollziehen, wurde in main.ts die Variable `currentSimulationStepSize` eingeführt. Dessen Wert wird über die mit dem UI verknüpften Eventhandler gesetzt. Die dazugehörige...

```
enum SimulationStepSize { Micro = 0, Macro = 1, Instruction = 2 };
```

...stellt sicher, dass nur gültige Schritttempi gespeichert werden. In der StateMachine wird der aktuelle Wert via `Stebs.ui.getStepSize()` aufgerufen.

Die Deaktivier- und Aktiviermechanik der Buttons wurde vom Vorgängerprojekt übernommen. Die Konstruktoren der States merken sich, welche `actions` sie ausführen dürfen. Alle nicht erlaubten `actions` führen dazu, dass deren Buttons deaktiviert werden.

Dropdownmenü für SpeedSlider und ExecutionModi

Zwar bietet das ins Projekt eingebundene Bootstrap bereits Dropdownmenüs an, doch wurde auf mehr gestalterische Freiheiten Wert gelegt. Der CSS - Code basiert auf dem Tutorial von w3schools und kommt auch bei den Registern zum Einsatz:

https://www.w3schools.com/css/css_dropdowns.asp .

Zu finden ist der Code in Stebs5/Content/Stebs.css.

Beim für Execution Mode eingesetzte Switch-3 - Control handelt es sich ebenfalls um ein reines HTML/CSS-Control. Der hierfür grundlegende Code wurde auf dem Blog [theStizMedia](#) entdeckt und kann in Stebs für weitere Zwecke verwendet werden. Für eine horizontale Implementierung muss der CSS-Code wohlmöglich in angepasster Form dupliziert werden.

FontAwesome und Compound Icons

Die Grundlegende Anwendung der FontAwesome - Icons wird auf <http://fontawesome.io/examples/> erklärt. In Stebs5/Content/Stebs.css steht der für Compound Icons notwendige CSS-Code im Kapitel "1.2 Font-Awesome (icons)" zur Verfügung und kann somit weiterverwendet werden.



Der Dropdownbutton für die Execution Modi () zeigt alle relevanten CSS-Klassen im Einsatz.

```
<button id="debugTempo" title="Debug Tempo">
    <span class="stebs-fa-stack">
        <i class="fa fa-bug fa-rotate-90 fa-stack-2x awesomeEntity"
            aria-hidden="true"></i>
        <i class="fa fa-forward fa-stack-1x awesomeAction" aria-hidden="true"></i>
        <span id="debugTempo-label" class="fa-stack-2x fa-stack-label
            awesomeAction">1x</span>
    </span>
</button>
```

stebs-fa-stack:	Gibt an, dass ein Compound Icon erzeugt werden soll.
fa-stack-label:	Positioniert ein Statuslabel in die linke, untere Ecke des Icons.
fa-stack-1x, fa-stack-2x:	Legt fest, auf welchem Layer sich das Icon befindet.
awesomeEntity:	Das grosse Haupticon.
awesomeAction:	Das kleine Icon in der unteren, rechte Ecke.

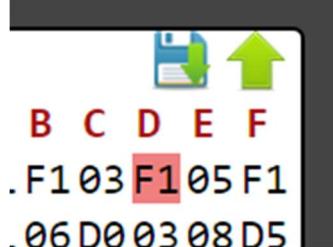
FontAwesome bietet auch eine eigene fa-stack - Klasse an. Jedoch erwies sie sich für die beiden Dropdownbuttons als ungeeignet. Diese müssen das aktuelle Slidertempo bzw. den aktuellen ExecutionModi in einem kleinen Label, der Bestandteil des Icons ist, angeben.

5.4.2 Menübar

Die Hauptaufgabe der neu eingeführten Menübar besteht darin, dem Endnutzer einen Orientierungspunkt zu geben. Menübars sind ein seit Jahrzehnten eingesetztes UI - Pattern und dadurch bekannt dafür, die wichtigsten Features zusammenfassend zu beinhalten.

Design

In der Menübar wurden folgende Operationen untergebracht:

Operation	Gelöstes Problem	Bemerkung
File I/O	Die zuvor genutzten Icons fielen zu wenig auf.	Zusätzlich war ungewohnt seltsam, dass bei geöffnetem Seitenpanel die Menübar sich nicht über die gesamte Bildschirmlänge hinwegzog. 
Export, z.B. des RAMs	Das zuvor verwendete Icon fiel zu wenig auf und war zudem irreführend.	Nebeneffekt: Die RAM - Tabelle sieht nun eleganter aus. 
View: Öffnen und schliessen diverser Panels und UI - Elementen	Verbessert Nutzerführung: Probanden kennen diesen Menüpunkt aus z.B. IntelliJ und Photoshop.	Den Endnutzern wird ein Mittel gegeben, das UI besser kennen zu lernen.
Mode: Wechseln zwischen Instruction-, Macro- und Microstep.	Ermöglicht es dem Endnutzer, über dieses Feature "zu stolpern" und es somit kennen zu lernen.	Wurde hinzugefügt wegen "Menübar als Featurezusammenfassung"
Extras: Für übriggebliebenes.	Beherbergt neu "Profil", welches sich zuvor neben Control Buttons befand, obwohl dieser nichts mit dem Ausführen von Code zu tun hat. 	Zusätzlich könnte fürs Erste ein "Hilfe" - Link unter diesem Menüpunkt untergebracht werden. Dieser verweist auf das Benutzerhandbuch von Webstebs.
Log off	"Log off" befand sich zuvor neben Control Buttons, obwohl dieser nichts mit dem Ausführen von Code zu tun hat.	

Für gewöhnlich befinden sich Menübars am oberen (oder auch unteren) Bildschirmrand und erstrecken sich über die gesamte Bildschirmbreite hinweg. In der Vorgängerversion von Webstebs war dies merkwürdigerweise nicht der Fall. Dies wurde nun verbessert.

Eine Unteraufgabe der Menübar ist es, den Namen des aktuell geöffneten Files anzuzeigen. In der früheren Version wurde dieser jedoch ab einer Länge von fünf Zeichen gekürzt und mit '...' konkateniert. Die maximale Zeichenlänge für die Namensanzeige wurde neu auf 20 Zeichen erhöht. Dadurch erhalten Endnutzer ein aussagekräftigeres Feedback darüber, an welchem File sie gerade arbeiten.



Abbildung 31 Filename in der alten Webstebs Version



Abbildung 32 Filename in der neuen Webstebs Version

Source Code

Die Menübar konnte in wenigen Schritten umgesetzt werden:

1. HTML + CSS: Gemäss w3schools.com/css/css_dropdowns.asp die Dropdown - Menüs erstellen.
2. Danach in main.ts die Menüitems mit JQuery - Klickeventhandlern verknüpfen. Die Eventhandler befinden sich innerhalb der \$(document).ready().
3. Mit den Eventhandlern die passende Logik auslösen (siehe main.ts).

Zum Schluss wurde das <div id="codingTopBar">, welches die Menübar beinhaltet, zu einem direkten Kind des <body> - Elements befördert. Dadurch erstreckt es sich nun über die volle Bildschirmbreite hinweg.

5.4.3 Seitenpanels

Als Seitenpanels werden die Bereiche am linken Bildschirmrand bezeichnet. Sie können bei Bedarf ein und ausgeklappt werden. Auf der folgenden Abbildung sind die beiden ausgeklappten Seitenpanels der alten Webstebs Version zu sehen.

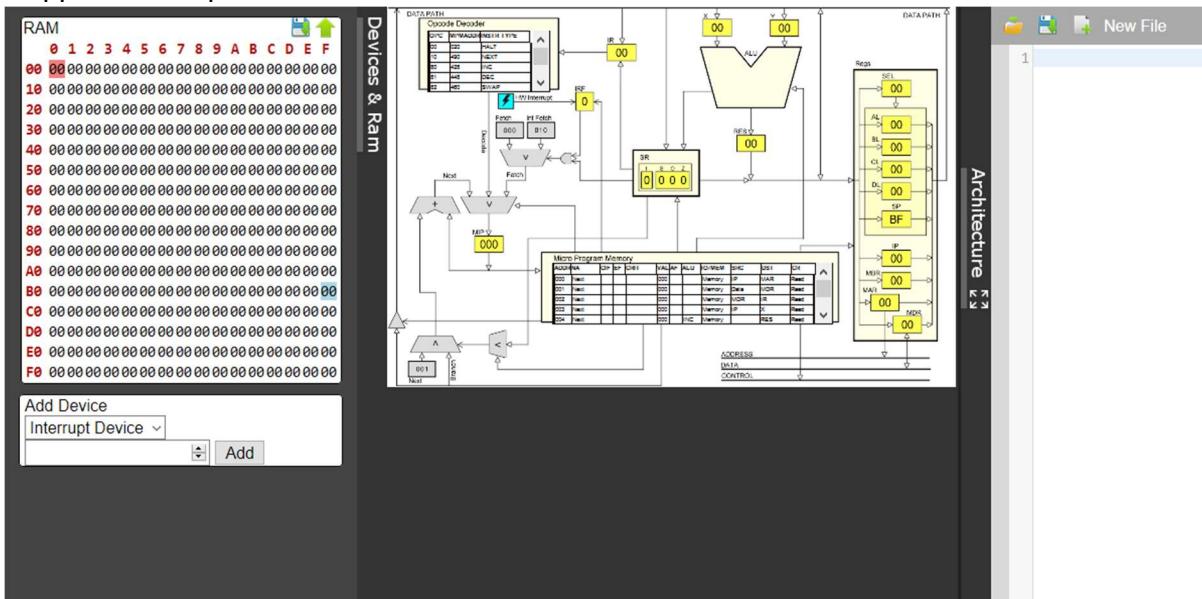


Abbildung 33 Seitenpanels in der alten Webstebs Version

Neu gibt es drei verschiedene Seitenpanels. Im linken Seitenpanel ist das RAM und die dazugehörigen Register zu finden. Im mittleren Seitenpanel können Devices hinzugefügt werden. Im Seitenpanel ganz rechts ist die Architektur-Ansicht zu finden. Die Endnutzer können frei diejenigen Seitenpanels ausklappen, die sie im Moment gerade benötigen.

Problematik Add Devices

Im linken Seitenpanel «Devices und Ram» konnten unterhalb des RAMs die Devices hinzugefügt werden. Die ersten drei hinzugefügten Devices werden korrekt angezeigt. Wird ein vierter Devices hinzugefügt, ragt dieses unten über das Browserfenster hinaus. Werden noch weitere Devices hinzugefügt, sind diese gar nicht mehr zu sehen. In den meisten Browsern kann man hinaus zoomen, so können die weiteren Devices auch angezeigt werden. Eine befriedigende Lösung ist das aber nicht.

Die Devices mussten an einem anderen Ort platziert werden. Es gab verschiedene Ideen wie das gelöst werden kann. Zwei Varianten wurde genauer ausgearbeitet und untersucht. Eine Variante war es, die Devices unterhalb des Codeeditors in einem Tab zu platzieren.

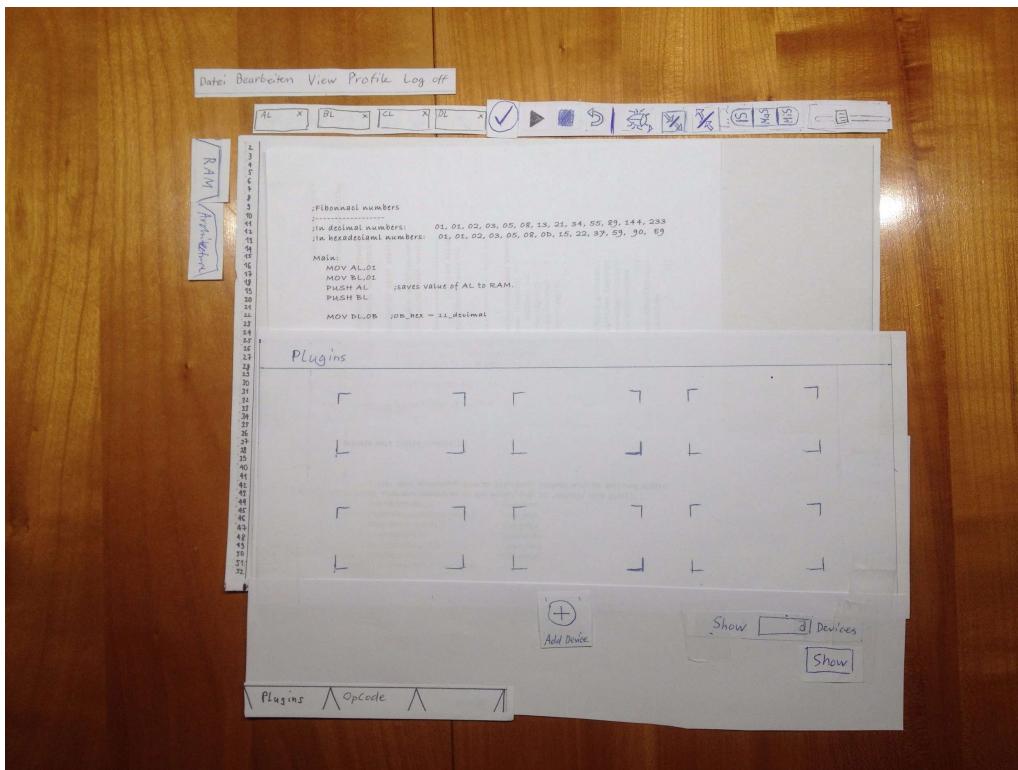


Abbildung 34 Devices unterhalb des Codeeditors

Die andere Variante war es, sie in einem Seitenpanel unterzubringen, entweder auf der linken oder rechten Seite.

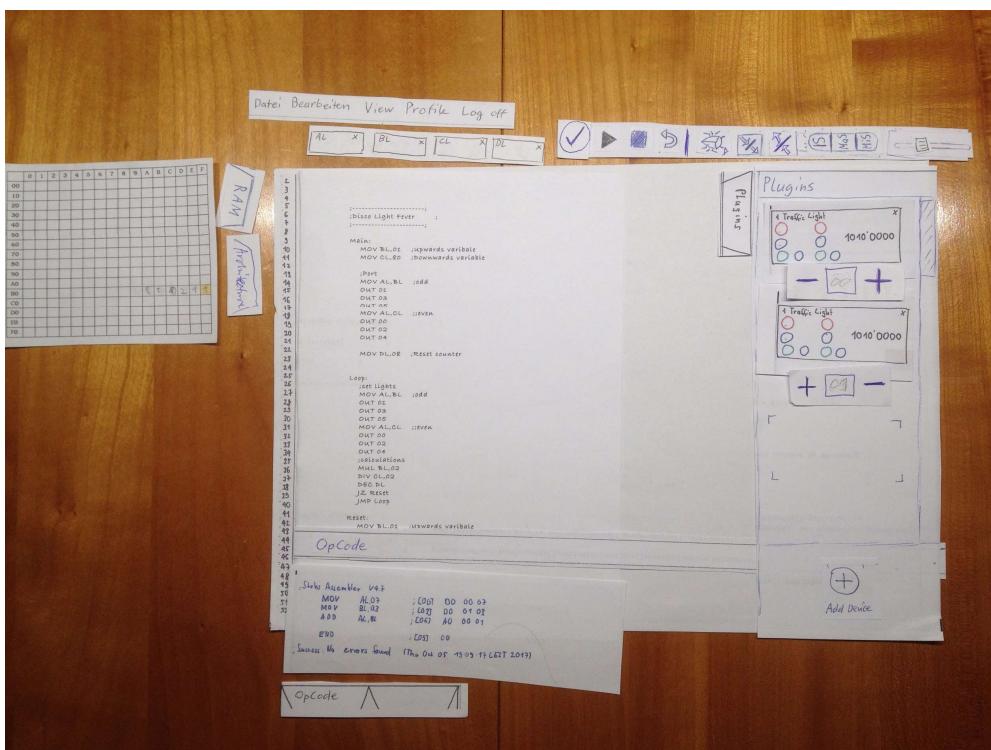


Abbildung 35 Devices in einem Seitenpanel

Diese beiden Optionen wurden mit dem Kunden besprochen und die Vor- und Nachteile miteinander verglichen. Der Nutzen der beiden Varianten wurde sehr ähnlich eingeschätzt, dies haben auch die zuvor gemachten Usability-Tests gezeigt.

Da mit einer agilen Vorgehensweise gearbeitet wird, hat man sich dazu entschieden ein minimal valuable product der zweiten Variante zu implementieren. Bei dieser Variante konnte mit dem geringsten Aufwand ein erster Entwurf umgesetzt werden kann. Dazu hat man in einer Iteration ein zusätzliches Seitenpanel entwickelt. Das Seitenpanel sollte möglichst ähnlich gemacht werden, wie die bereits bestehenden. So konnte der Aufwand klein gehalten werden. Deswegen hat man sich auch für die linke Bildschirmseite entschieden.

Das neue Seitenpanel wurde zwischen den bereits bestehenden Seitenpanels implementiert. Die Breite des Seitenpanels wird durch die Breite der Devices bestimmt. Die beiden bereits implantierten Devices «Interrupt Device» und «Traffic Light Device» sind beide 322 Pixel breit. Damit diese Devices nicht verändert werden müssen wurden alle neuen Devices mit derselben Breite entwickelt. Es gab keinen Grund daran etwas zu ändern.

Damit zwischen dem Rand und den hinzugefügten Devices ein bisschen Platz vorhanden ist, wurde die Breite des neuen Seitenpanels auf 380 Pixel festgelegt.

Bereits nach einer Iteration konnte dem Kunde ein noch nicht fertiges aber bereits benutzbares Seitenpanel präsentiert werden. Damit war es auch schon möglich Devices hinzuzufügen. Das Aussehen des Add-Device Bereichs wurde noch nicht verändert.

Die Umsetzung hat den Kunden sowie das Projektteam überzeugt, so wurde dieser Ansatz weiterverfolgt.

In einem zweiten Schritt wurde der Bereich auch scrollbar gemacht.

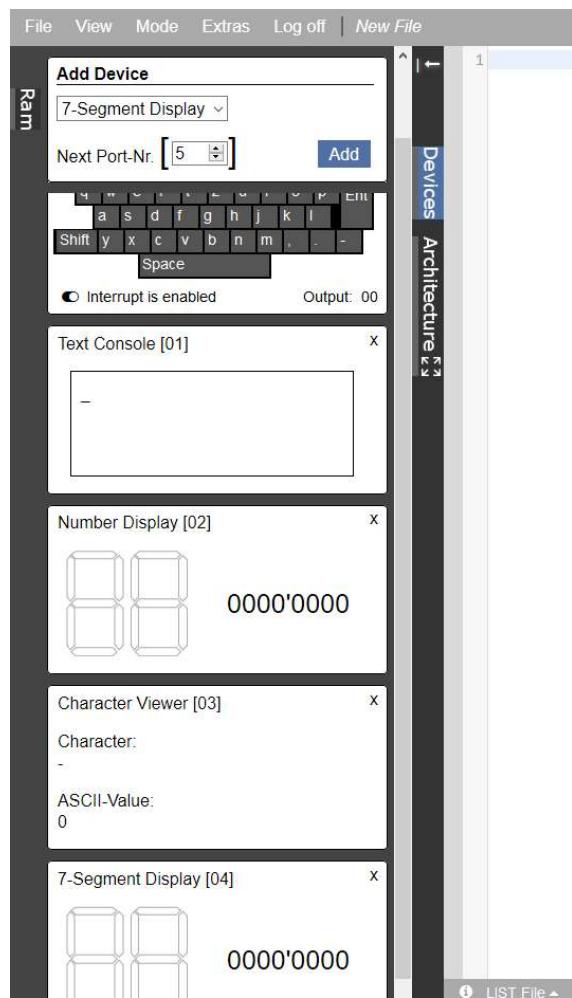


Abbildung 36 scrollbare Device Ansicht

Nun können fast beliebig viele Devices angezeigt werden.

Der Add-Device Bereich wurde so implementiert, dass er immer sichtbar ist, auch wenn gesrollt wird. So ist es immer möglich weitere Devices hinzuzufügen.

Design

Das Design dieses Bereichs wurde in dieser Iteration erneuert. Anfänglich waren nur die HTML-Elemente vorhanden, die benötigt werden ein neues Devices hinzuzufügen. Nun sollte das Aussehen und die Usability verbessert werden. Da die Entwicklungsphase schon fast zu Ende war, hat man sich entschieden das Design zu verändern aber nicht den gesamten Aufbau und Ablauf zu verbessern.

Das Inputfeld, in dem man die Portnummer eingibt, war nicht beschriftet. Dadurch wussten die Endnutzer nicht, was damit gemacht werden kann. Um dies verständlicher zu gestalten wurde eine ein Label mit der Bezeichnung «Next Port-No.» hinzugefügt. Damit der Endnutzer eine visuelle Verbindung zwischen diesem Inputfeld und der Portnummer eines Devices herstellen kann, wurden eckige Klammern vor und nach dem Inputfeld hinzugefügt. Dieses Aussehen soll an die Portnummernanzeige der Devices erinnern.

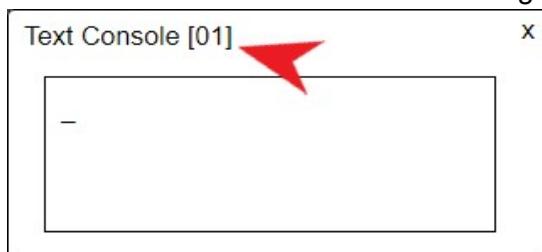


Abbildung 37 Device mit hervorgehobener Portnummernanzeige

Die eckigen Klammer sind etwas grösser und fett formatiert, als die der Devices, damit sie vom Endnutzer einfacher wahrgenommen werden können.

Der Abstand zwischen den HTML-Elementen wurde vergrössert. Dadurch wirkt das Design klarer und es ist für die Endnutzer angenehmer zu bedienen.

Auch der Button «Add» wurde erneuert. Der Button ist in die untere rechte Ecke verschoben worden. Die User sind sich gewohnt eine Aktion in der unteren rechten Ecke eines Fensters oder des Menüs zu beenden. Der Button war aber mitten im Add-Device Bereich platziert, dies wurde nun korrigiert. Auch das Aussehen wurde verbessert. Zuvor wurde das Standarddesign eines Buttons verwendet. Nun wurden die Farben und die Form, an die der anderen GUI-Elementen angepasst. Der Button verändert sein Aussehen jetzt, wenn sich der Mauszeiger über ihm befindet.

visuelle Anpassungen

Bei den Usability-Tests haben sich die Probanden beschwert, dass nicht immer klar ersichtlich ist, welche Seitenpanels geöffnet sind und welche nicht. Damit den Endnutzern dies einfacher fällt, werden die geöffneten Seitenpanels farblich markiert. Genauer gesagt erhält der dazugehörige vertikale Button einen blauen Hintergrund.

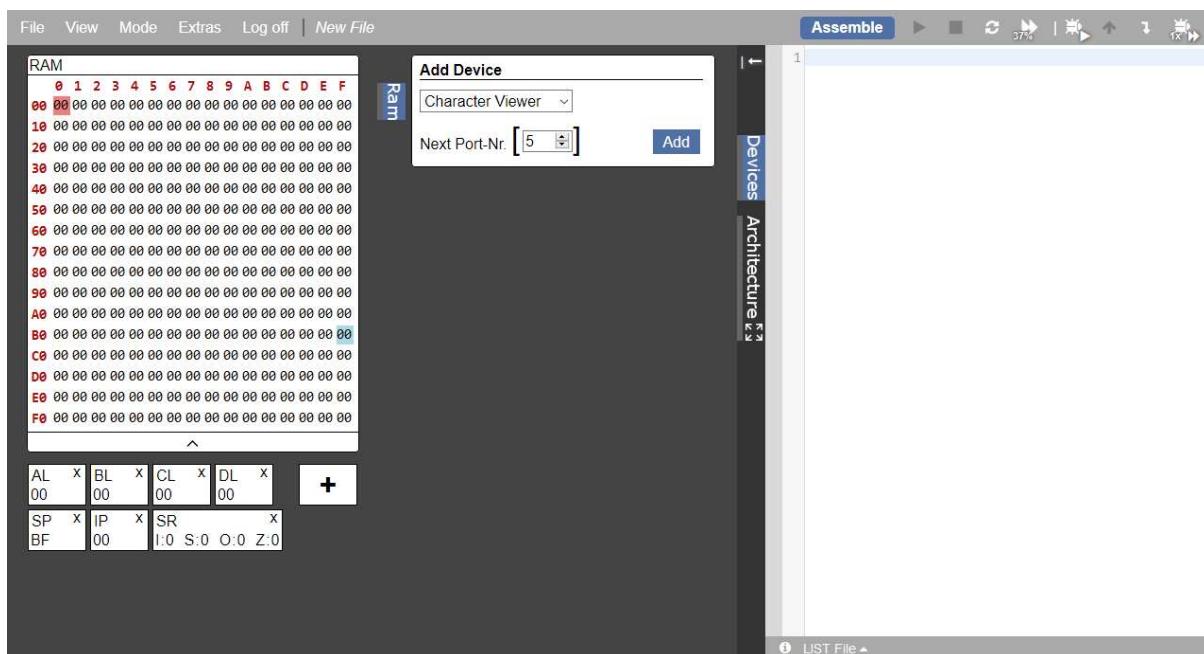


Abbildung 38 vertikale Buttons geöffnet/geschlossen

Bei der alten Webstebs Version wurden die Button grün, wenn man mit der Maus über sie gefahren ist. Diese Farbe passt jetzt aber nicht mehr zum blauen Hintergrund bei ausgeklapptem Seitenpanel. Aus diesem Grund wurde die Farbe des Hovereffekts auch auf blau geändert. Dabei wurde die beste Kombination zwischen den beiden Blautönen gesucht. Das Blau bei ausgeklappten Zustand war bereits definiert. Es ist dasselbe Blau, welches unter anderem auch bei den Simulation Control Buttons verwendet wird. Das Blau für den Hoverefekt musste nun so gewählt werden, dass ein möglichst guter Kontrast zwischen dem grauen Hintergrund bei eingeklapptem Seitenpanel und dem blauen Hintergrund bei ausgeklapptem Seitenpanel besteht. Die finale Version ist auf der letzten Abbildung zu sehen.

Register

Die Register waren in der alten Webstebs Version im gleichen Bereich, wie die Simulation Control untergebracht. Dieser Bereich befindet sich unten rechts und wurde automatisch ausgeklappt, wenn die Ausführung des Codes gestartet wurde. Er kann auch manuell über einen Button ausgeklappt werden. Die Simulation Control wurde zusammengeführt und befindet sich nun oben rechts in der Menübar. Mehr Informationen dazu befinden sich im Kapitel 5.4.1 Simulation Control. Der Bereich unten ist nun viel zu gross, um nur die Register anzuzeigen. Er muss überarbeitet oder entfernt werden.

Die Register an einen Ort zu verschieben wurde im Projektteam, wie auch mit dem Kunden, bereits diskutiert. So war schnell klar, dass es sinnvoller ist die Register zu verschieben und somit den Bereich unten rechts zu entfernen. Es wurden zwei konkrete Lösungsvorschläge gefunden. Die Register könnten oben in der Menübar in der Mitte des Browserfensters platziert werden. Die andere Möglichkeit wäre es die Register unter das RAM in dessen Seitenpanel zu verschieben.

Bei den Papierprototyp – Tests wurden beide Optionen getestet. Für die Probanden gab es keinen nennenswerten Unterschied zwischen den beiden Positionen. Im Kapitel Alternative Regiserposition ist dazu mehr zu finden.

Das Projektteam hat intern und mit dem Kunden die Vor- und Nachteile miteinander verglichen. Dabei hat man sich entschieden, dass die Platzierung im Seitenpanel des RAMs besser geeignet ist. Dies wurde aufgrund von mehreren Faktoren entschieden. Der Platz in der Menübar ist eher knapp. Unterhalb des RAMs ist genügend Platz frei geworden, nachdem man die Devices um platziert hat. Die Register werden meistens in Kombination mit dem RAM verwendet. Deswegen ist es besser, wenn sie in der Nähe positioniert sind.

Einige Register zeigen Informationen an, die nur benötigt werden, wenn man mit der Architekturansicht arbeitet. Für diese Register ist es besser, wenn sie unterhalb der Architekturansicht zu finden sind. Aus diesem Grund wurden die Register in zwei Gruppen aufgeteilt. Die erste Gruppe wird im Seitenpanel RAM angezeigt, die zweite im Seitenpanel Architecture.

Im Seitenpanel RAM:

- AL
- BL
- CL
- DL
- SP
- IP
- SR

Im Seitenpanel Architecture:

- SEL
- MBR
- MAR
- MDR
- IR
- RES
- X
- Y
- MIP
- IRF

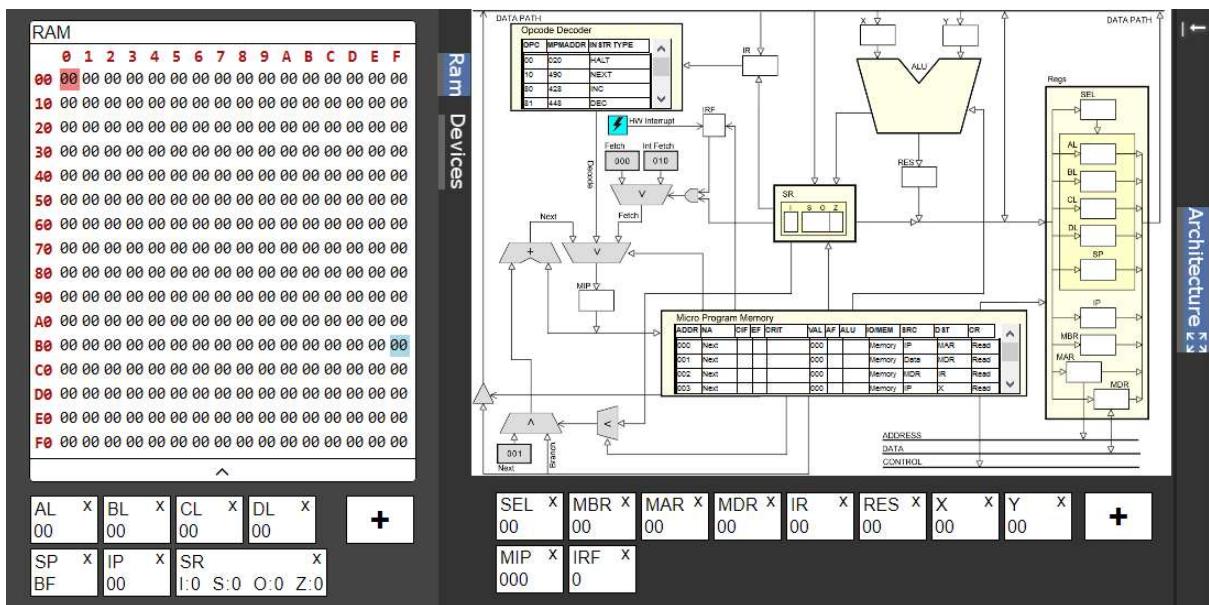


Abbildung 39 Register in den beiden Seitenpanels

Register können durch das x in der rechten oberen Ecke entfernt werden. Um sie wieder hinzuzufügen, muss auf das Plus rechts neben den Registern geklickt werden. Dieser Mechanismus war zuvor bereits implementiert. Da sich die Register nun aber an zwei Orten befinden, musste er überarbeitet werden. Der Button und das dazugehörige Dropdown wurden auch visuell überarbeitet. Wenn das Dropdown geöffnet ist, wird der Button blau eingefärbt. Dies zeigt dem Nutzer an, dass dieses Element zurzeit aktiv ist. Das Dropdown basiert auf dem gleichen Aufbau, wie es auch das Dropdown in der Menübar verwendet. Das Aussehen wurde jedoch angepasst. Der Kontrast wurde deutlich erhöht. Im Gegensatz zu dem Dropdown, welches in der alten Webstebs Version verwendet wurde.

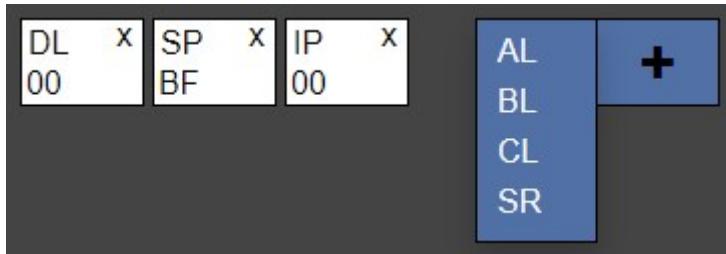


Abbildung 40 ausgeklapptes Dropdown "Register hinzufügen"

RAM

Auch das RAM selbst wurde überarbeitet.

RAM	
0 1 2 3 4 5 6 7 8 9 A B C D E F	
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

Abbildung 41 RAM in der alten Webstebs Version

Die Abstände zwischen den zweistelligen hexadezimalen Werten im RAM wurden vergrössert. Dadurch wird die Lesbarkeit der einzelnen Werte deutlich erhöht und es ist für die Endnutzer angenehmer damit zu arbeiten. Gerade bei Computern mit einer grossen Auflösung hilft dies enorm.

In der alten Webstebs Version, gab es einen Button, mit dem alle Werte im RAM in eine für Logisim lesbare Datei exportiert werden konnten. Dieser Button war nicht gerade benutzerfreundlich. Das verwendete Symbol  , repräsentiert normalerweise eine Speicherfunktion und es deshalb als Export-Symbol ungeeignet. Die Positionierung neben dem RAM, zeigte einen gewissen Zusammenhang zwischen den beiden Elementen. Der Nutzen des Buttons wurde daraus aber nicht ersichtlich. Einen Tooltip gab es für den Button auch nicht. Deswegen entschied man sich, diesen Button zu entfernen. Der Endnutzer kann nun über die Menübar diese Datei exportieren. Die Endnutzer sind sich durch viele andere Programme gewohnt, solche Aktionen über die Menübar durchführen zu können.

Neben dem RAM befand sich ein zweiter Button. Damit konnte das RAM ein- oder ausgeklappt werden. Diese Funktionalität kann in der neuen Webstebs Version auch über die Menübar ausgeführt werden. Um den Endnutzern mehr Möglichkeiten anzubieten, wurde der Button neben dem RAM belassen. Das Aussehen wurde aber angepasst und verbessert. Der zuvor benutzte grüne Pfeil  wurde entfernt. Die Probanden der Usability - Tests konnten keinen Zusammenhang zwischen diesem Button und dem auf- und zuklappen des RAMs herstellen.

Aus diesem Grund wurde der Pfeil - Button durch ein anderes Symbol und einen anderen Aufbau ersetzt. Jetzt ist der Button unterhalb des RAMs angebracht und erstreckt sich über die gesamte Breite.

D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00


Abbildung 42 Collape RAM

Endnutzer kennen die Symbole \wedge und \vee aus anderen Softwarelösungen. Gerade bei Touch - Oberflächen von Smartphones, ist dieses Symbol zum auf- und zuklappen von GUI-Elementen, weit verbreitet. Somit ist gewährleistet, dass die Endnutzer nun schneller und besser erkennen, dass das RAM ein- und aufgeklappt werden kann.

Closeall-Button

In den UX-Tests wurde von den Probanden der Wunsch geäussert, alle ausgeklappten Seitenpanels auf einmal einklappen zu können. Dies erhöht die Usability ungemein, darum wurde dieser Vorschlag umgesetzt. Es wurde ein neuer Button eingeführt mit dem genau, diese Aktion ausgeführt werden kann. Der Button befindet sich am gleichen Ort, wie die vertikalen Buttons, mit denen die Seitenpanels auf- und zugeklappt werden können.



Abbildung 43 Closeall-Button

Code

Die HTML – Elemente der Seitenpanels, auch die des Neuen, befinden sich in der Datei Stebs5/Views/Manage/Index.cshtml. Die Steuerung der Ein- und Ausklapp - Animationen wird mit Typescript umgesetzt. Sie ist in der Datei Stebs5/Typecripts/ui.ts implementiert. In der Datei Stebs5/Typecripts/main.ts werden Informationen gespeichert, die für die Animationen der Seitenpanels genutzt werden können. Diese Datei enthält zum einen die vordefinierten Breiten der Seitenpanels, zum anderen wird der aktuelle Zustand, ob das Seitenpanel geöffnet oder geschlossen ist, dort gespeichert.

5.4.4 Filemanager

Webstebs bietet die Möglichkeit, Assemblyfiles auf dem Server zu speichern. Hierfür kommt das Menü namens Filemanager zum Einsatz, in dem die Dateien und Ordner von Webstebs verwaltet werden. Damit können neue Files erstellt, gespeichert, geöffnet und gelöscht werden. Es ist auch möglich Ordner zu erstellen um die Dateien darin zu ordnen.



Abbildung 44 Filemanager in der alten Webstebs Version

Mangel

Bei den Tests mit Webstebs wurde ein Mangel an der Menüführung des Filemanagers gefunden. Wenn man dabei war den Namen eines neuen Files einzugeben und dann den Filemanager geschlossen hat, wurde der Vorgang nicht abgebrochen.

Der genaue Ablauf war wie folgt:

1. auf den Button "new File" klicken
 - a. danach ist folgendes zu sehen:

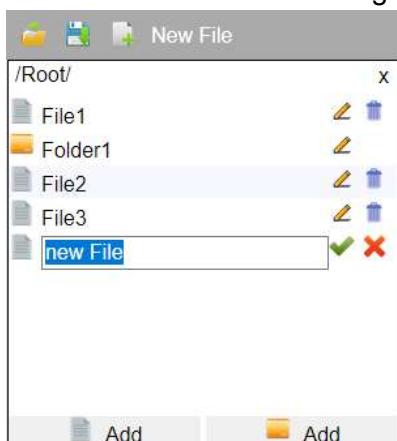


Abbildung 45 Filemanager - neues File erstellen 1

2. auf den Button "open File" klicken
 - a. danach ist folgendes zu sehen:

**Abbildung 46 Filemanager - neues File erstellen 2**

Nach erneutem Öffnen des Filemanagers war man am gleichen Punkt, wie vor dem Schliessen. Der Vorgang ein neues File zu erstellen wurde nicht abgebrochen. Dies kann die User verwirren und verunsichern. Das gleiche Problem gab es auch beim Ändern des Filenamens. Für Ordner bestand das Problem bei beiden Fällen ebenfalls.

Dieses Problem konnte behoben werden. Zuvor wurde der Filemanager nur ausgeblendet, nun wird zusätzlich der Inhalt des Filemanagers aktualisiert. Dadurch werden jegliche Vorgänge abgebrochen.

```
serverHub.getFileSystem().then(fileManagement.reloadFileManagement);
```

Design

Das Design des Filemanagers wurde überarbeitet. Der grundlegende Aufbau wurde nicht verändert, nur die Icons wurden ersetzt. Dadurch wurde das Aussehen an das neu geplante, einheitliche Design von Webstebs angepasst.

Da die komplette Menübar überarbeitet wurde, wurden die alten Buttons des Filemanagers entfernt.

**Abbildung 47 Filemanager Buttons in der alten Webstebs Version**

Auf der Abbildung oben sind die Buttons zum Öffnen, Speichern und zum Erstellen eines neuen Files oberhalb des Filemanagers zu sehen. Die gesamte Funktionalität des Filemanagers kann nun über die Menübar erreicht werden.

Das Filemanager - Fenster wurde lediglich visuell verändert. Seine UI - Funktionalitäten blieben soweit erhalten.

Ursprünglich oben links positioniert, wurde der Filemanager in die Mitte des Code-Editors verschoben. Wenn eines der Dropdowns der Menübar und der Filemanager geöffnet wären, würden sich die beiden Elemente überlappen. Es ist angenehmer für die Endnutzer, wenn sich der Filemanager in der Mitte des Bildschirms befindet.

Alle auf Bilddateien basierten Icons wurden durch Fontawesome-Icons ersetzt. In der folgenden Tabelle sind die alten Icons und diejenigen, durch die sie ersetzt wurden, aufgelistet.

Beschreibung	altes Icon	neues Icon	Fontawesome-Bezeichnung
File			fa-file-text-o
Folder			fa-folder-open-o
Name ändern			fa-pencil-square-o
File/Folder löschen			fa-trash-o
bestätigen			fa-check
abbrechen			fa-times

Source Code

Die nachfolgend gezeigten Code Snippets wurden aus dem File "Stebs5/TypeScripts/fileManagement.ts" entnommen.

Fontawesome

Nachdem Fontawesome der Applikation hinzugefügt wurde (siehe Kapitel 3.2 FontAwesome), ist das Ersetzen der alten Icons kein Problem. Es muss lediglich eine andere CSS Klasse hinzugefügt werden. Nachfolgend ist der Code zu sehen, wie es zuvor umgesetzt wurde:

```
var imgSpan = $('')
    .addClass('icon')
    .addClass('fileIcon')
    (...)
```

Nun wird dem Icon eine andere Klasse und somit auch das Fontawesome-Icon hinzugefügt.

```
var imgSpan = $('')
    .addClass('icon')
    .addClass('fa fa-file-text-o')
    (...)
```

Um zwischen den passenden Icons, die die verschiedenen UI - Funktionalitäten des Filemanagers repräsentieren, zu wechseln, ist es notwendig die CSS - Klassen auszutauschen. Dies kann mit Hilfe von JQuery leicht bewerkstelligt werden. Zuerst muss das Element angesteuert (Zeile 1), danach die alte CSS-Klasse entfernt (Zeile 2), und zuletzt die neue Klasse hinzugefügt werden (Zeile 3).

```
1     $('#file-' + nummer + ' i.fa-square-o')
2             .removeClass('fa-square-o')
3             .addClass('fa-check-square-o')
```

Ein solcher Wechsel wird für drei verschiedene Vorgänge benötigt. Betroffen sind dabei immer

dieselben beiden Icons, die sich jeweils hinter den Filenamen befinden. Diese werden entweder zum Namen ändern und File löschen, zum Bestätigen und Abbrechen oder als Checkbox verwendet und brauchen jedes Mal ein anderes Icon.

Files löschen

Dem Filemanager wurden zwei zusätzliche Funktionen hinzugefügt. Zum einen ist es nun möglich mehrere Files auf einmal zu löschen. Zum anderen ist es möglich das derzeit geöffnete File ohne Umwege zu löschen.

Zuerst werden die Änderungen behandelt, die notwendig waren, um das derzeit geöffnete File zu löschen. Es ist bekannt welches File derzeit geöffnet ist.

```
var actualNode = fileManagement.openedFile
```

Falls kein File geöffnet ist, wird die Funktion bereits jetzt beendet. Wenn ein File geöffnet ist, wird dieses normal gelöscht. Nun muss ein neues noch nicht gespeichertes File geöffnet werden. Dazu wird der Inhalt des Code-Editors geleert:

```
codeEditor.getDoc().setValue("");
codeEditor.getDoc().clearHistory();
```

Der Dateiname sowie der Titel der Webseite werden auf den Default-Wert gesetzt.

```
$('#filename').text("new File");
document.title = 'Stebs';
```

Die Attribute des Filemanagers werden zurückgesetzt.

```
fileManagement.openedFile = null;
fileManagement.rootNode = <Folder>fileManagement.fileSystem.Root;
fileManagement.actualFolder = fileManagement.rootNode;
```

Weiter ist nun möglich mehrere Files in einem Vorgang zu löschen. Dabei kann mit Hilfe von Checkboxes ausgewählt werden, welche Files gelöscht werden sollen. Durch einen Klick auf OK werden diese Files entfernt. Der Vorgang um mehrere Files zu löschen, kann auch gestartet werden, wenn der Filemanager bereits geöffnet ist. Im nachfolgenden Bild ist der Filemanager während des Lösch-Vorgangs zu sehen:

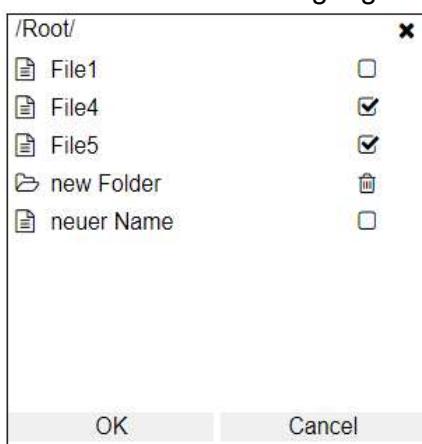


Abbildung 48 Filemanager - delete many

Um Löschvorgänge zu bestätigen oder abzubrechen, werden häufig ein OK - und ein Cancel - Button angeboten. Diese befinden sich üblicherweise am unteren Rand eines Pop-Up - Fens-ters.

Beim Filemanager nehmen an dieser Stelle die Buttons um Files oder Folders zu erstellen, bereits den Platz ein.



Abbildung 49 Filemanager - Add File/Folder

Diese werden jedoch während des Löschtorgangs nicht benötigt. Aus diesen Gründen werden diese Buttons durch OK und Cancel ersetzt.

Weiter muss für jedes File eine Checkbox angezeigt werden. Alle Files im aktuellen Folder befinden sich in einer Liste, auf die einfach zugegriffen werden kann.

```
fileManagement.actualFolder.Children
```

Danach werden, wie zuvor beschrieben, die Icons ersetzt. Bei diesem Vorgang wird jeder Checkbox noch ein Eventhandler hinzugefügt. Diese Funktion fügt, bei einem Klick auf die Checkbox, das jeweilige File einem Array hinzu.

Ein File kann aus diesem Array wieder entfernt werden. Dazu wird der ausgewählten Checkbox ebenfalls eine Funktion hinzugefügt, die auf Klicks reagiert.

Wenn auf OK geklickt wird, um den Vorgang abzuschliessen, wird die Funktion zum entfernen aller Nodes im Array aufgerufen. Darin wird der User zuerst gefragt ob er sich sicher ist die Files zu löschen. Dazu bietet HTML5 eine eigene Funktion namens «confirm», die bereits einen Bestätigungsvorgang enthält.²

```
confirm('Are you sure you want to delete these files?')
```

Falls keine Files dem Array hinzugefügt oder alle wieder entfernt wurden, wird der User darauf aufmerksam gemacht. Danach kann er wieder Files dem Array hinzufügen.

Wenn der User bestätigt, wird mit einer for-Schleife über das Array iteriert und alle darin enthaltenen Files gelöscht.

```
for (let node of Stebs.nodesToRemove) {  
    if (node != -1) {  
        serverHub.deleteNode(node,  
            fileManagement.nodeIsFolder(fileManagement.actualFolder))  
            .then(fileManagement.reloadFileManagement);  
    }  
}
```

Danach wird der Filemanager geschlossen und in den Ausgangszustand gesetzt. Das Array wird geleert. Wie beim Löschen des zurzeit geöffneten Files, wird ein neues noch nicht gespeichertes File geöffnet.

² https://www.w3schools.com/jsref/met_win_confirm.asp

Filelänge

Zuvor war es möglich, wie auf der Abbildung unten zu sehen, einen sehr langen Filenamen zu vergeben.

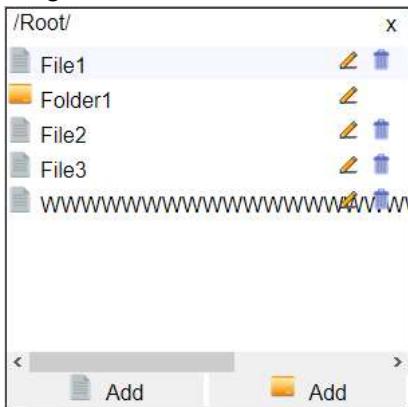


Abbildung 50 Sehr langer Filename

Um dies zu verhindern, wird die Länge bei der Namenseingabe überprüft. Wenn der String länger als 20 Zeichen ist, wird der User darauf aufmerksam gemacht. Weitere Eingaben werden danach unterbunden, der User wird dann aber nicht mehr informiert. Die gewünschte Länge wird über die folgenden Variable definiert.

```
export const MAXLENGTHFILENAME = 20;
```

In der Funktion checkLength() wird die Länge überprüft. Aufgerufen wird diese Funktion nach jeder Eingabe eines Zeichens.

5.4.5 Output / List-File - Panel

Das zuvor als "Output" bezeichnete Panel dient dazu, dem Endnutzer zu zeigen, wie der CPU den Assembly Code liest bzw. versteht. Zusätzlich werden vom CPU vollführte Arbeitsschritte angezeigt.

Design

In den Usabilitytests neigten die Probanden dazu, Feedback für ihren Code in diesem Panel zu suchen. Einerseits wegen der irreleitenden Namensgebung und andererseits aufgrund der Endnutzergewohnheiten: Die Ausgabekonsole in IntelliJ und Eclipse befindet sich ebenfalls am unteren Bildschirmrand. Um dem entgegenzuwirken, wurden folgende Schritte unternommen:

- Der Name wurde von "Output" auf "LIST File" geändert. Dies ist die im Fachjargon korrekte Bezeichnung für das, was in diesem Panel ausgegeben wird.
- Dem Panelheader wurde ein - icon hinzugefügt. Dies soll den Endnutzer dazu bewegen, mit der Maus den Tooltip auszulösen. Dieser enthält Hinweise darauf, wo sich das RAM, die Register und der ArchitectureCanvas befinden.

Um zu verdeutlichen, dass das LIST File - Panel durch einen Klick auf dessen Header geöffnet bzw geschlossen werden kann, wurde ein weiteres Symbol hinzugefügt.

bedeutet, das Panel kann geöffnet werden. Hingegen weist darauf hin, dass es geschlossen werden kann.



Abbildung 51 LIST File

Source Code

Der Code für dieses GUI - Element blieb weitgehend unverändert. Im Headerbalken befindet sich ein Button, der nun den Tooltip im title - Attribut trägt sowie den neuen Namen und die beiden Icons beinhaltet.

```
<button id="openOutput" title=" Click to toggle the LIST File - view.
  &#13;&#10;&#13;&#10; This view displays how the assembler reads and
  processes your code. &#13;&#10;&#13;&#10; ...">
  <i class="fa fa-info-circle" aria-hidden="true"></i>
  <span>LIST File</span>
  <i class="fa fa-caret-up" aria-hidden="true"></i>
</button>
```

Der mit diesem Button verknüpfte Eventhandler greift auf `Stebs.ui.toggleOutput()` zu.

Diese Funktion in `ui.ts` ist nun zusätzlich für den Wechsel der Icons und verantwortlich.

5.5

5.5.1 RunAndDebug entfernt

Ursprünglich beheimatete dieses Panel alle Register sowie einen Teil der Simulation Control. Da nun beide Komponenten ausserhalb von RunAndDebug positioniert wurden, verlor es seinen Stellenwert und konnte somit komplett entfernt werden.

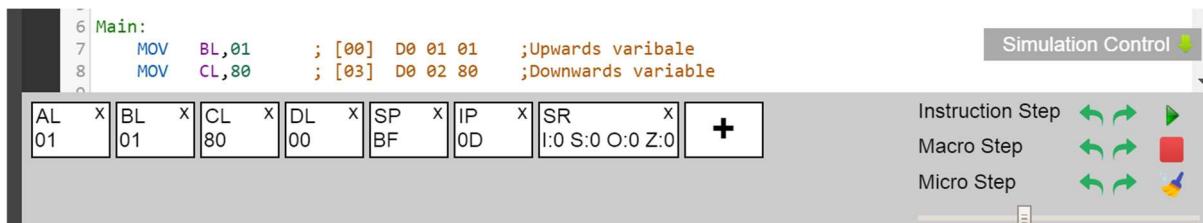


Abbildung 52 Register und Simulation Control in der alten Webstebs Version

Design

Nachdem RunAndDebug eliminiert wurde, entstand ein angenehmer Nebeneffekt für den Endnutzer: Die CodingView erhielt mehr vertikalen Freiraum. Jedoch musste zuerst der Source Code angepasst werden.

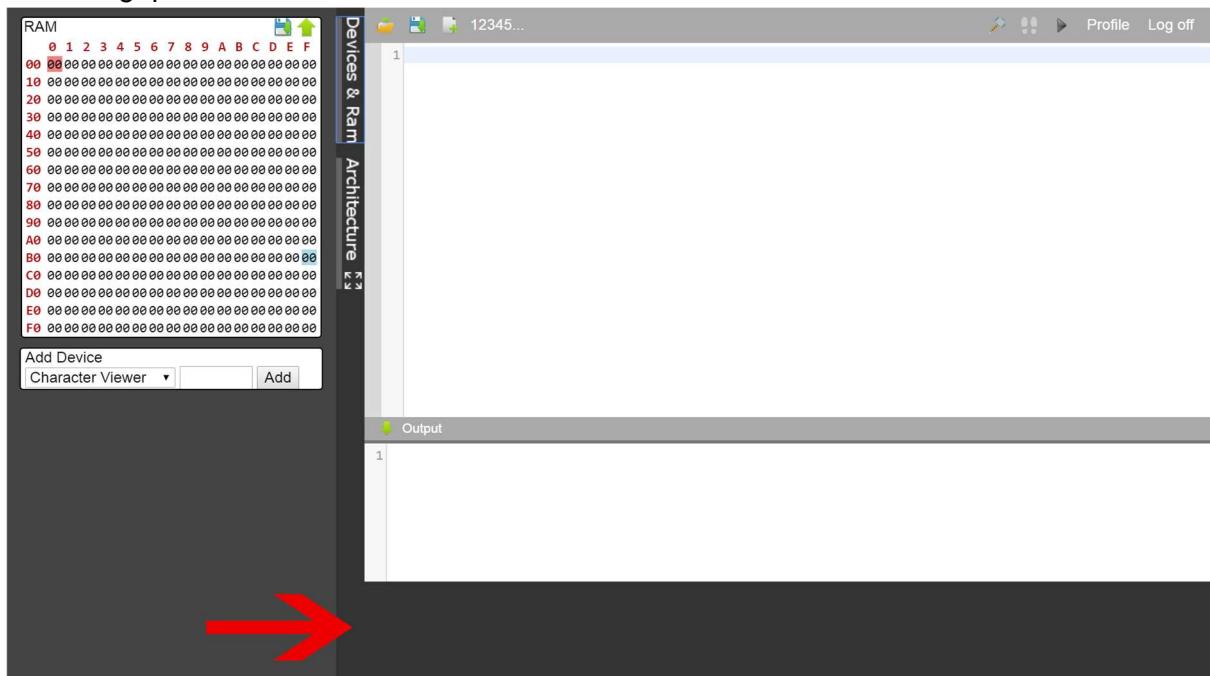


Abbildung 53 vor der Anpassung des Codes

Source Code

Nachdem RunAndDebug aus dem User Interface verschwand, blieb eine klaffende Lücke zwischen LINE File - Panel und dem unteren Bildschirmrand zurück. Im Code musste demzufolge das LINE File - Element an den unteren Rand um positioniert und das dynamische Resizing der CodingView angepasst werden.

main.ts beinhaltet jeweils die Dimensionen eines UI-Elementes als auch die Information, ob es gerade angezeigt wird. Die zu RunAndDebug gehörenden Werte konnten entfernt werden:

- Stebs.visibility.runAndDebug
- Stebs.Heights.runAndDebug

Das Resizing vertikaler Panels arbeitet basierend auf den Werten von Stebs.Heights. Da nun der Wert für RunAndDebug nicht mehr vorhanden ist, wurden folgende Funktionen in ui.ts angepasst. Dabei wurden alle zu RunAndDebug - relevanten Werte in dessen Funktionskörpern entfernt:

- setCodingFrameHeight ()
- outputDragbarMouseMove ()
- setArchitectureHeight ()

Nachfolgende Funktionen konnten gänzlich entfernt werden, da diese aufgerufen wurden, sobald RunAndDebug geöffnet bzw. wieder geschlossen wurde:

- Stebs.ui.toggleRunAndDebug ()
- Stebs.ui.openRunAndDebug ()
- Stebs.ui.repositionSimulationControl ()

Zu guter Letzt verabschiedete sich auch der Togglebutton "Simulation Control"

( und sein Eventhandler aus dem Frontend. Das spezielle an diesem Button war, dass er zu einem inkonsistenten Design beitrug: Seitenpanels verfügen über Tabs, horizontale Panels über Header zum Öffnen und Schliessen des Panels. Der Togglebutton widersprach jedoch dieser Konvention.

5.5.2 StatusMessage Pop-Up

Damit der Endnutzer weiss, dass er ein File gespeichert oder den Code erfolgreich assembliert hat, wurde ein halbtransparentes Pop-up eingeführt. Dieses kann dazu verwendet werden, um dem Endnutzer Feedback über erfolgreich oder nicht erfolgreich getätigte Aktionen zu geben.



Um eine neue Nachricht zu definieren, muss in Stebs.statusMessages eine neue Nachricht hinzugefügt werden.

```
export var statusMessages = {
    saved:      "<i class='fa fa-check-circle' aria-hidden='true'> </i>" +
                "<span>Saved!</span>",
    assembled: "<i class='fa fa-code' aria-hidden='true'> </i>" +
                "<span style='font-size:24px;'>Assembled!</span>"
};
```

Danach kann diese neue Nachricht in einem Eventhandler an `Stebs.ui.informUserOfStatus(msgAsHtml: string)` übergeben werden.

6 Devices

Devices erweitern die funktionalen Möglichkeiten von Webstebs. Dem Endnutzer können so fortlaufend neue, virtuelle Peripherie und Hardwarekomponenten angeboten werden. Aus didaktischer Sicht eröffnen Devices spannende Aufgabenstellungen, mit welchen Wissen zu Mikroprozessoren und der Hardwareprogrammierung vermittelt werden kann.



Abbildung 54 Device Ansicht

Damit Devices mit dem Server kommunizieren können, wird ihnen jeweils ein Port zugeordnet (Next Port-No.). Über diesen wird ein Datenbyte übermittelt. In der Abbildung 53 ist eine Tastatur und eine Textausgabekonsole ersichtlich. Wird auf eine Keyboardtaste gedrückt, übermittelt dieses Device den ASCII - Wert an den Server. Anschliessend, weil der Endnutzer es so mit Assembly - Code programmierte, wird dieser ASCII - Wert an das Textkonsolendevice gesendet. Dieses kann nun den korrekten Buchstaben ausgeben.

6.1 Umgesetzte Devices

Vor Webstebs3 verfügte die Applikation über zwei Devices: Das Interrupt Device und das Traffic Light. Ersteres unterbricht bei Betätigung die Ausführung des momentan laufenden Codes, letzteres simuliert eine Strassenampel, die programmatisch gesteuert werden kann.

Die nachfolgend aufgeführten Devices wurden als Teil dieses Projektes entwickelt, um Studierenden eine grössere Vielzahl an Aufgabenstellungen anbieten zu können. Dabei wurde gemäss Kundenwunsch der Fokus auf noch fehlende Peripherie gelegt.

6.1.1 Keyboard Device

Mit dem Keyboard Device wird, wie es der Name schon vermuten lässt, eine Tastatur simuliert. Die Bedienung erfolgt aber mit der Maus. Wird eine Taste angeklickt, wird der ASCII-Wert des jeweiligen Buchstabens oder Zeichens auf den IN-Port des Keyboard Device gelegt. Dieser Wert kann dann innerhalb von Webstebs weiterverwendet werden. Es sind die am meisten verwendeten Zeichen umgesetzt. Namentlich die Zahlen von 0 bis 9, alle Buchstaben, Punkt, Komma, Fragezeichen, Backspace, Enter, Space, und Bindestrich. Zusätzlich kann mit Hilfe der Shift-Taste Grossbuchstaben sowie Unterstrich, Doppelpunkt und Strichpunkt ausgegeben werden.

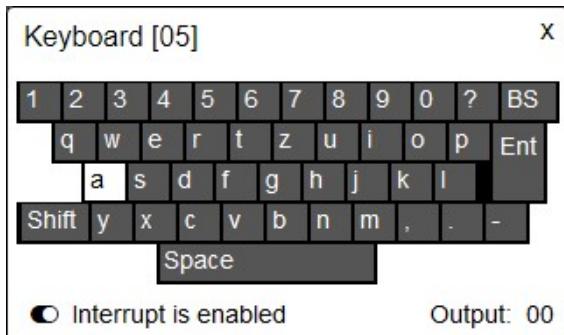


Abbildung 55 Keyboard Device

Durch einen Klick auf eine Zeichentaste wird automatisch ein Interrupt ausgelöst. Dies ist per Default eingeschaltet. Möchte man dies deaktivieren, kann auf den Schalter geklickt werden.

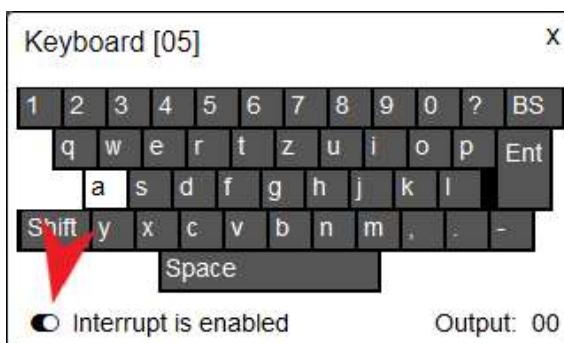


Abbildung 56 Keyboard Device - hervorgehobener Interrupt Schalter

Ein Interrupt ermöglicht es, während der Ausführung einer Programmroutine an eine bestimmte Stelle im Code zu springen. Diese Stelle muss zuvor definiert werden. Dort kann Code implementiert werden, der bei einem Interrupt ausgeführt werden soll. Wird also in Webstebs eine Programmroutine durch einen Interrupt unterbrochen, dann wird der aktuelle Stand zwischengespeichert. Danach wird an die Stelle im Programm gesprungen, die den Code enthält, der bei einem Interrupt ausgeführt werden soll. Nachdem dieser Code ausgeführt wurde, springt Webstebs wieder an die Stelle in der Programmroutine zurück, bei der die Ausführung zuvor unterbrochen wurde. Der Interrupt kann so genutzt werden, dass bei einem Interrupt die Tastatureingabe eingelesen wird und danach das Programm weiter ausgeführt wird.

In der rechten unteren Ecke wird der Output des Keyboard Devices angezeigt. Dieser Wert wird nach jedem Tastendruck aktualisiert. Dies muss jedoch nicht heißen, dass dieser Wert auch von Webstebs gelesen wurde.

Der strukturelle Aufbau des Tastaturdevices ist dem einer handelsüblichen QWERTZ-Tastatur nachempfunden. Das Design wurde an die Bildschirmtastatur von Microsoft Windows 10 angelehnt. Es sollte möglichst schlicht und einfach sein.

Wird die Shift-Taste aktiviert, wird sie weiss eingefärbt. Ebenfalls werden die Tastaturbezeichnungen gemäss der Shift-Taste angepasst. Befindet sich der Mauszeiger über einer Taste, wird der Hintergrund der Taste weiss eingefärbt. Dies erleichtert dem Benutzer die Eingabe.

In HTML entspricht jede Taste des Keyboard - Devices einem div - Element. Alle divs einer Reihe sind wiederum von einem div umschlossen. Da sich die Enter-Taste über zwei Reihen

erstreckt, wird sie nicht von einem anderen div umschlossen. Die Tasten Shift, Backspace, Enter und Space haben je eine eigene CSS-Klasse, da sie spezielle Größen haben. Alle anderen Tasten sind gleich formatiert.

JavaScript

Der JavaScript Teil des Keyboards Device wird anhand der Taste q genauer erläutert.

Für jede Taste wird ein Klickeventhandler definiert.

```
$('#key-q-{slot}').click(function(){{(...)};});
```

Zuerst wird überprüft, ob die Shift-Taste gedrückt wurde. Mit Hilfe einer Lambda-Funktion wird der entsprechende ASCII-Wert in einer Variable gespeichert. In diesem Fall ist dies 87 für Q und 119 für q.

```
var asciiValue = (shift == 0) ? 119 : 87;
```

Danach wird die Slot-Nummer, der eben definierte ASCII-Wert und true oder false an das Backend gesendet. Der ASCII-Wert wird so auf das IN-Port des Devices gelegt. Mit true oder false wird festgelegt, ob ein Interrupt ausgelöst werden soll oder nicht. Dies wird auch mit einer Lambda-Funktion gemacht, in der überprüft wird, in welcher Position sich der Interrupt-Schalter befindet.

```
Stebs.updateDeviceSecondUpdate({slot}, asciiValue,  
    $('#interrupt-enable-{slot}').is(':visible') ? 'true' : 'false');
```

Als letztes wird der ASCII-Wert in das Output-Feld geschrieben.

```
setOutput(asciiValue);
```

Shift-Taste

Beim Keyboard Device gibt es viel sich wiederholender Code. Für jede Taste muss einzeln ein Klickeventhandler definiert werden. Da es sich um 44 Tasten handelt, gibt es entsprechend viele Zeilen mit sehr ähnlichem Code. Diese Unschönheit gibt es auch bei der Umsetzung der Shift-Taste. Denn bei jedem Klick auf diese Taste, muss der Inhalt von fast allen Tasten angepasst werden. Bei einem erneuten Klick, muss alles wieder zurückgesetzt werden.

Überprüft, ob die Shift-Taste aktiviert ist oder nicht, wird anhand einer Variable. Diese wird jeweils nach einem Klick auf die Shift-Taste auf den aktuellen Status gesetzt. Danach wird der Inhalt der Tasten von Klein- auf Grossbuchstaben oder umgekehrt geändert.

```
$('#key-q-{slot}').text('"Q"');
```

Dieses Problem konnte nicht anders gelöst werden. Jede Taste muss einzeln angesteuert werden. Man könnte zwar mit einer Schleife über alle Tasten iterieren, da aber jede Taste einen anderen Inhalt hat, bringt dies kaum einen Nutzen. Denn über den Inhalt kann nicht vernünftig iteriert werden und er muss für jede Taste separat implementiert werden.

Backend

Für das Keyboard Device musste am Backend kleine Änderungen vorgenommen werden. In Stebs5/main.ts war bereits die Funktion «updateDevice» enthalten. Damit kann ein Device auf sein Model auf dem Server zugreifen.

Die Funktion updateDevice enthält zwei Argumente. Mit der Slot-Nummer wird das entsprechende Device identifiziert. Das zweite Argument ist ein String.

```
Stebs.updateDevice({slot}, String);
```

Damit können verschiedene Informationen ins Backend weitergegeben werden. Das Keyboard Device nutzt dies um den ASCII-Wert der gedrückten Taste zu versenden. Es muss aber zusätzlich noch die Information, ob ein Interrupt ausgelöst werden soll oder nicht, mitversendet werden. Man könnte den String dementsprechend erweitern, sodass er beide Informationen enthält. Der Code wird dadurch aber schlechter lesbar und mündet in keiner sauberen Lösung. Besser ist eine Trennung dieser beiden Informationen. Deswegen wurde eine weitere Funktion eingeführt mit der es möglich ist drei Argumente an das Backend zu senden. Nun können zwei Strings mitgegeben werden. Dazu waren mehrere Erweiterungen notwendig, diese werden jetzt genauer beschrieben.

In der Datei «main.ts» können die Daten vom Frontend an das Backend gesendet werden. Darin befindet sich auch die eben erwähnte Funktion updateDevice(). An der gleichen Stelle wurde nun eine weitere Funktion «updateDeviceSecondUpdate()» implementiert. Diese kann zwei Argumente aufnehmen.

Damit die Daten an das Backend versendet werden können, muss dieses für den Empfang eingerichtet werden. Dazu wurde im StebsHub eine weitere Funktion implementiert, welche zwei Argumente weiterleitet. Im ProcessorManager befindet sich die eigentliche UpadteDevice Funktion, die auf die Devices auch tatsächlich zugreift. Auch hier wurde eine weitere Funktion UpdateDevice eingefügt. Diese enthält ein zusätzliches Argument namens «secondInput». Dieses Argument wird direkt an das richtige Device weitergeleitet.

```
public void UpdateDevice(string clientId, byte slot, string input, string secondInput) =>
    ProcessorAction(clientId, session => {
        var devices = session.DeviceManager.Devices;
        if (devices.ContainsKey(slot)) {
            devices[slot].Update(input, secondInput);
        }
    });
}
```

Nun ist es möglich zwei String-Argumente an ein Device zu senden. Dies kann auch von neuen Devices verwendet werden.

Anhand dieses neuen zweiten Arguments wird schlussendlich entschieden, ob zum Setzen des neuen ASCII-Werts zusätzlich ein Interrupt ausgelöst werden soll. Zuerst wird der ASCII-Wert in die Variable «data» gespeichert. Danach wird je nachdem ein Interrupt ausgelöst oder nichts weiter gemacht.

```
public override void Update(string input, string interruptEnabled)
{
    data = Convert.ToByte(input);
    if (interruptEnabled == "true") {
        Interrupt();
    }
}
```

6.1.2 Text Console

Das Text Console Device erlaubt die Ausgabe von alphanumerischen Zeichen, die zu einem zusammenhängenden String konkateniert werden. Die maximale Länge dieses ausgegebenen Strings beträgt 64 Zeichen. Dabei wird er über vier Reihen à 16 Zeichen verteilt dargestellt wird.

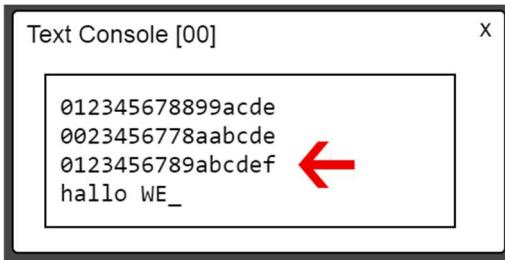


Abbildung 57 Text Console Device

Die Zeichen werden als ASCII-Code übermittelt. Das heisst, es werden die binären Werte 0000'0000 bis 0111'1111 als Zeichen erkannt. Davon werden alle nicht alphanumerische durch ein □ (HTML - Entitywert: ࡫) ersetzt.

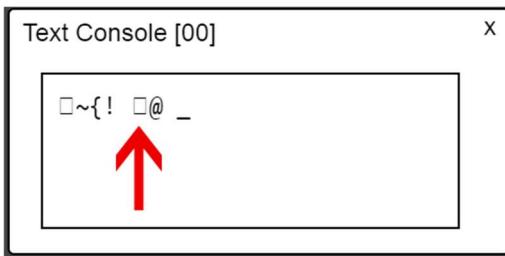


Abbildung 58 Text Console Device - hervorgehobenes invalid Symbol

Text verschiebbar

Dank der ASCII - Codierung kann das MSB verwendet werden, um Befehle an ein Device zu senden. In der jetzigen Version des Text Console Device wird der Wertebereich von binär 1000'000 bis 1111'1110 für das Verschieben des Textes verwendet.

Ist das MSB gesetzt, wird zunächst geprüft, ob in vertikaler oder horizontaler Richtung verschoben werden soll. Hierfür wird das Bit direkt rechts vom MSB geprüft:

- 11xx'xxxx führt zu einer vertikalen Verschiebung.
- 10xx'xxxx führt zu einer horizontalen Verschiebung.

Somit bleibt ein Wertebereich von 1x00'0000 und 1x11'1110 zurück, der den Verschiebungswert darstellt. **Hinweis:** 1111'1111 ist für den Reset des Devicezustandes reserviert.

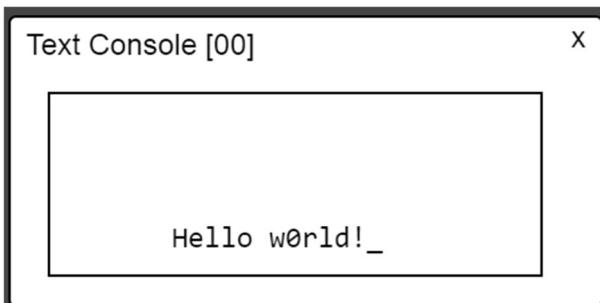


Abbildung 59 Text Console Device 1

HTML - Injectionssicher

Bei Webstebs handelt es sich um eine Webapplikation. Das Text Console Device stellt sicher, dass kein HTML, CSS oder JavaScript injiziert werden kann. Dies geschieht, indem die für HTML - Markup gebräulichen Zeichen durch HTML Entities ausgetauscht werden.

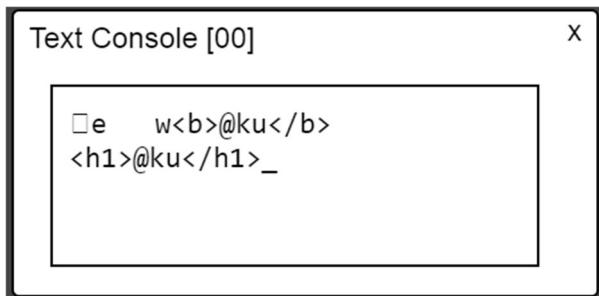


Abbildung 60 Text Console Device 2

Source Code

Stebs.registerDevice()

Ist vergleichbar mit der main() in Java und stellt somit den Entrypoint in den Code dar

str[slot][]

speichert die an das Device gesendete Zeichen.

prepareString[slot]()

Konvertiert str[] in einen browserkompatiblen und -sicheren String und gibt diesen zurück.

encodeToHtmlEntity(rawStr)

Tauscht alle nicht-alphanumerischen und Markupzeichen in rawStr mit Entity Codes aus.

6.1.3 Character Viewer

Ein einfaches Device für ASCII - unerfahrene. Mit ihm kann er sich das Zeichen (z.B. y) und dessen dezimaler Wert anzeigen lassen. Liegt der im AL - Register abgelegte Wert ausserhalb der ASCII - Tabelle, so wird "invalid" ausgegeben.

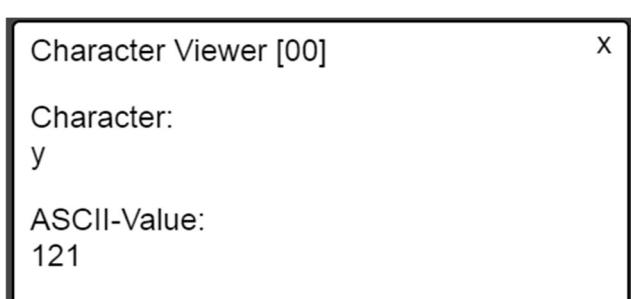


Abbildung 61 Character Viewer Device

Source Code

Der Code gestaltet sich rudimentär: Der in data.Data empfangene Wert wird darauf überprüft, ob er sich im Wertebereich der ASCII - Tabelle befindet. Anschliessend wird die entsprechend die adäquate Ausgabe vollzogen.

```
Stebs.registerDevice({slot}, function(data) {  
    $('#data-{slot}').text(data.Data);  
    //is it an ASCII - character?  
    if (data.Data > 0 && data.Data <= 127) {{  
        $('#character-{slot}').text( String.fromCharCode-  
Code(data.Data) );  
    } } else {{  
        $('#character-{slot}').text('invalid');  
    } }  
});
```

6.1.4 Number Display

Das Number Display verfügt über zwei 7-Segment Displays. Der Endnutzer kann diese beiden Displays ansteuern und Zahlen darauf anzeigen. Jedes dieser Displays kann alle einstelligen hexadezimale Zahlen anzeigen. Mit beiden zusammen ist es möglich zweistellige hexadezimale Werte zu visualisieren. Im Assembly Code kann der Endnutzer einen zweistelligen Wert auf das Out-Port des Number Display Device legen. Dieser Wert wird danach auf den beiden Displays angezeigt

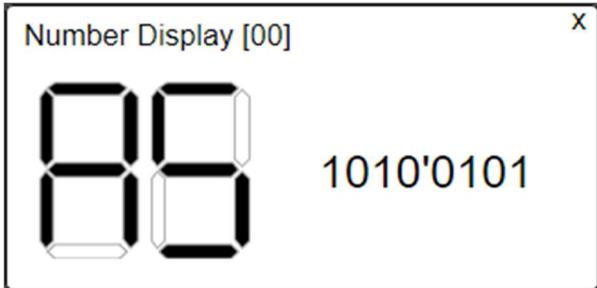


Abbildung 62 Number Display Device

Möchte man alle hexadezimalen Zahlen von 0 bis F auf einem 7-Segment Display anzeigen, kommt es zu Überschneidungen. Um den hexadezimalen Wert B auf einem Display anzuzeigen, werden alle Segmente eingeschaltet. Damit der hexadezimale Wert 8 angezeigt werden kann, müssten ebenfalls alle Segmente eingeschaltet werden. Damit dies verhindert werden kann, wird anstatt eines grossen B eine kleines b angezeigt. Dasselbe Problem besteht mit dem hexadezimalen Wert für D und einer Null. Dort wird anstatt eines grossen D ein kleines angezeigt.

Die beiden Displays können durch Webstebs über den OUT-Port angesteuert werden. Der OUT-Port entspricht einem zweistelligen hexadezimalen Wert. Die erste der beiden Stellen wird auf dem linken Display angezeigt, der zweite auf dem rechten Display.

Das Design entspricht einem typischen 7-Segment Displays, wie man es zum Beispiel von einem Wecker kennt. Rechts neben den beiden Displays wird derselbe Wert angezeigt aber in binärer Schreibweise.

SVG

Die beiden Displays wurden mit HTML umgesetzt. Die beiden Displays sind zwei Scalable Vector Graphics(SVG), die wiederum aus mehreren Polygonen bestehen. Jedes Segment ist ein solches Polygon. SVG-Bilder sind am besten geeignet um zweidimensionale Formen anzuzeigen. Man könnte auch ein Bild für jedes Segment verwenden. Mit SVG ist man jedoch viel flexibler bei der Gestaltung und kann diese auch nachträglich noch problemlos in der Grösse skalieren. Die Form der Polygone wird über ein Koordinatensystem festgelegt. Weiter ist es möglich, mit CSS das SVG so anzupassen wie man möchte. So kann zum Beispiel die Hintergrundfarbe oder die Grösse des SVGs verändert werden. Auch die Segmente können einzeln gestaltet werden. Dies wird verwendet um sie ein oder auszuschalten, indem die Füllfarbe geändert wird.

Backend

Das Number Display kann, wie die anderen Devices, über den OUT-Port angesteuert werden. Bei jedem Update dieses Ports werden die Segmente entsprechend dem neuen Wert eingeschaltet. Dies wird über das Javascript dieses Devices gesteuert. Bei jedem Update werden zuerst alle Segmente ausgeschaltet, also weiss eingefärbt.

Danach wird der neue Wert in das binäre Zahlensystem umgewandelt. Dadurch kann der Wert halbiert werden. Als Beispiel wird der binäre Wert der Zahl 165 verwendet. In binärer Schreibweise ist dies der Wert 1010'0101, in hexadezimaler ist es der Wert A5. Der linke Display zeigt den linken Teil mit dem Wert 1010 an. Das rechte Display zeigt den rechten Teil mit dem Wert 0101 an. Dies hat einen enormen Vorteil, da es genau der Aufteilung des zweistelligen hexadezimalen Werts in zwei Teile entspricht. Der Wert 1010 ist hexadezimal nämlich der Wert A und der Wert 0101 entspricht dem hexadezimalen Wert 5.

Der binäre Wert wird für die beiden Displays aufgeteilt.

```
var leftDisplay = binary.slice(0, 4);
var rightDisplay = binary.slice(4, 8);
```

Anschliessend wird der Wert in binärer Schreibweise angezeigt. Damit die Werte auf den Displays angezeigt werden können, ist für jede der 16 Zahlen von 0 bis F, festgelegt welche Segmente eingeschalten werden müssen. Es besteht keine Möglichkeit mit JavaScript auszurechnen, welche Segmente eingeschalten werden müssen. Deswegen muss dies für jede Zahl einzeln hinterlegt werden. Die Unterscheidung ist mit einem Switch-Case realisiert. Hier ist der Fall zu sehen, auf dem rechten Display den Wert 0 anzugeben. Wenn die Variable «rightDisplay» den binären Wert 0000 enthält, werden die Segmente eingeschaltet (schwarz eingefärbt), die notwendig sind eine Null anzuzeigen.

```
switch (rightDisplay) {
    case '0000':
        $('#poly21-{slot}').css("fill", "black");
        $('#poly22-{slot}').css("fill", "black");
        $('#poly24-{slot}').css("fill", "black");
        $('#poly25-{slot}').css("fill", "black");
        $('#poly26-{slot}').css("fill", "black");
        $('#poly27-{slot}').css("fill", "black");
    break;
}
(...)
```

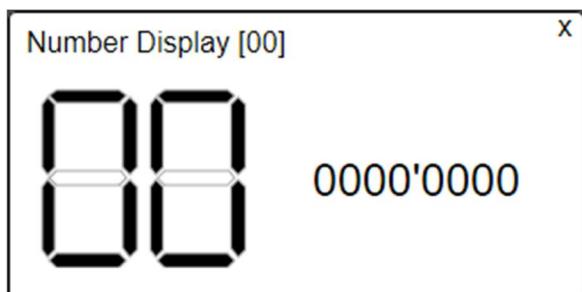


Abbildung 63 Number Display - Wert 00

Ein falscher Wert kann vom User nicht eingegeben werden. Beim Assemblieren wird bereits überprüft, dass nur zweistellige hexadezimale Werte vorkommen können. All diese Werte können von den Number Displays angezeigt werden. Weitere Überprüfungen sind, deswegen unnötig.

6.1.5 7-Segment Display

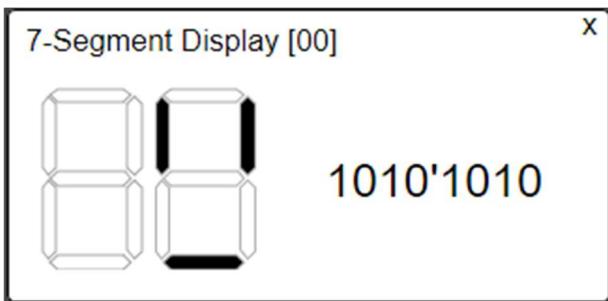


Abbildung 64 7-Segment Display

Das 7-Segment Display hat vieles mit dem Number Display gemeinsam. Der Aufbau der beiden Devices ist derselbe. Die beiden Displays sind auch hier zwei SVGs und es wird auch zusätzlich der Wert im binären Zahlensystem angezeigt. Der Unterschied der beiden Devices ist, dass beim 7-Segment Display die Segmente direkt vom User ein- oder ausgeschaltet werden können. So kann der User selber bestimmen was angezeigt werden soll. Der Zugriff erfolgt, wie beim Number Display, über den OUT-Port. Einschalten heisst, dass der Hintergrund des Segments schwarz eingefärbt wird. Beim Ausschalten wird die Hintergrundfarbe weiss.

Der Aufbau des 7-Segment Displays wird in diesem Kapitel nicht weiter beschreiben. Der Aufbau entspricht dem des Number Display und dieser ist bereits im Kapitel 6.1.4 Number Display erfasst.

Jedes Segment der beiden 7-Segment Displays kann angesteuert werden. Dies kann der Benutzer mit dem hexadezimalen Wert machen, den er auf den OUT-Port legt. Der zweistellige hexadezimale Wert entspricht binär einem 8-stelligen Wert. Mit der Stelle ganz links (MSB) kann bestimmt werden, welches der beiden Displays angesteuert werden soll. Ist das Most Significant Bit (MSB) eine 0 wird das linke Display angesteuert, ist es eine 1 wird das rechte angesteuert. Die anderen 7 Stellen können je genau einem Segment zugewiesen werden. Der Wert der jeweiligen Stelle bestimmt dann, ob ein Segment eingeschaltet (Wert = 1) oder ausgeschaltet (Wert = 0) werden soll. In der nachfolgenden Tabelle sind drei Beispiele mit dem jeweiligen Resultat zu sehen.

Stelle	Hex-Wert	8. (MSB)	7.	6.	5.	4.	3.	2.	1.	Resultat:
Eingabe	7F	0	1	1	1	1	1	1	1	
Eingabe	FF	1	1	1	1	1	1	1	1	
Eingabe	AA	1	0	1	0	1	0	1	0	

Der User kann nur einen zweistelligen hexadezimalen Wert auf das OUT-Port legen. So ist es nur möglich das linke oder das rechte Display zu verändern. Möchte man beide verändern, muss zwei Mal ein Wert auf das OUT-Port gelegt werden.

JavaScript

Die Hintergrundfarbe der Segmente wird mit JavaScript verändert. Der JavaScript Code dieses Devices ist um einiges kürzer als der des Number Displays oder des Keyboard Devices, da alle Elemente gleich verändert werden. Es kann mit Hilfe einer Schleife auf alle benötigten HTML - Elemente zugegriffen werden. Damit auf die Elemente zugegriffen werden kann, müssen diese zuerst identifiziert werden. Dazu hat jedes HTML – Element eine ID, diese sieht zum Beispiel wie folgt aus:

`id=""poly21-{slot}""`

Sie ist aus drei Teilen aufgebaut.

poly	Alle Polygone haben diese Bezeichnung in ihrem Namen, um Entwicklern anzuzeigen, um was für ein Element es sich handelt.
2	Alle Polygone sind durchnummeriert. Die erste Stelle bestimmt ob es sich um das linke oder das rechte Display handelt. 1 = linkes Display 2 = rechtes Display
1	Die zweite Stelle steht für die Nummer des jeweiligen Segments. Jedes Display hat sieben Segmente mit je einer Nummer zwischen 1 und 7.
-{slot}	Anhand der Slot Nummer wird das gesamte Device identifiziert.

In der zuvor erwähnten Schleife wird über alle Segmente eines Displays iteriert, damit der gewünschte Wert

In der nächsten Codezeile kann man sehen, wie mit Hilfe dieser ID auf ein Element zugegriffen und die Hintergrundfarbe mit CSS verändert wird.

```
$('#poly'+ MSB + i + "-{slot}").css("fill", "black");
```

Interessant ist hierbei, wie das HTML-Element identifiziert wird. Es wird ein String (fett markiert) aus fünf Teilen zusammengesetzt.

#	In JQuery gibt dieses Zeichen an, dass der nachfolgende String die ID eines Elements ist.
---	---

poly	Elementbezeichnung
MSB	Dieser Variable wurde zuvor bereits der Wert zugewiesen, welches Display angesteuert werden sollt. 1 = linkes Display 2 = rechtes Display
i	Dies ist die Index – Variable innerhalb der for – Schleife. Sie wird verwendet die einzelnen Segmente zu identifizieren.
-{slot}	Slotnummer

In der zuvor erwähnten Schleife, wird über alle Segmente eines Displays iteriert. In der Schleife wird entschieden, ob das Segment einen weissen oder schwarzen Hintergrund erhält.

```
for (var i = 1; i < 8; i++){  
    if(parseInt(binary.slice(i, i+1))==1){  
        $('#poly'+ MSB + i + "-{slot}").css("fill", "black");  
    }else {  
        $('#poly'+ MSB + i + "-{slot}").css("fill", "white");  
    }  
}
```

6.2 How to develop a device

6.2.1 Plugin erstellen Schritt für Schritt (mit eigenen Ergänzungen)

Aus IP5vt_Bericht_Webrechner_Anhang.pdf entnommen:

1. In Visual Studio ein neues C# Library Projekt erstellen. Die .NET Framework Version des Plugins und von "Stebs5" müssen dieselbe sein.
2. References hinzufügen: Projekte „PluginApi“ und „ProcessorSimulation“ (<https://msdn.microsoft.com/de-de/library/ez524kew.aspx>)
3. Eine Klasse erstellen, welche von DefaultDevice erbt, und "using ProcessorSimulation.Device" hinzufügen;
4. Die Methoden überschreiben, welche gebraucht werden
5. Eine Klasse erstellen, welche von IDevicePlugin erbt: Property Name: Menschenlesbarer Name des Plugins Property PluginId: Name des Plugins ohne Sonderzeichen, Spaces, ... (Möglichst nur die Zeichen: a-z und 0-9) Methode CreateDevice: Eine neue Instanz der Device Klasse (aus Schritt 3) erstellen und zurückgeben. Methode DeviceTemplate: Das Device Template ist Client Code, welcher pro Slot Nummer separat erstellt wird. Das Template kann css, html und javascript Elemente enthalten.
6. Das neue Plugin zu Stebs5 hinzufügen. 2 Varianten:
 - a. In Stebs5 eine Referenz auf das PluginProjekt erstellen (Ähnlich wie dies in Schritt 2 gemacht wurde)
 - b. Die resultierende .dll des Plugin Projekts in den Plugin Ordner einer Stebs Instanz kopieren (Danach Stebs neustarten)

Eigene Ergänzungen dazu:

1. Darauf achten, die richtige .NET - Version zu wählen. Siehe folgenden Screenshot:

Wähle...
Visual C# > .NET Framework 4.6 (Dropdown-Menü) > **Class Library (.NET Framework)**

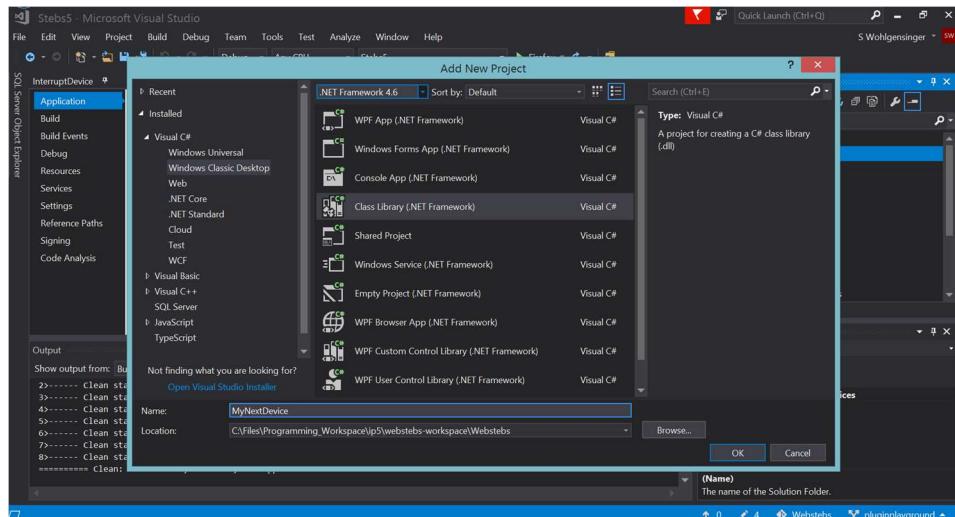


Abbildung 65 Visual Studio - Add New Project

6.2.2 IN und OUT - Zusammenspiel Plugin mit Stebs Backend

Werte im Device Empfangen

1. xxPlugin.cs öffnen
2. Im Methodenkörper von public string DeviceTemplate(byte slot) befindet sich der HTML, CSS und JavaScript - Code des Plugins.

```
public string DeviceTemplate(byte slot) =>
$@"<style>
</style>
<div class=""traffic-data"""
    <span id=""traffic-data-{slot}"">1111'1111</span>
</div>
<script>
    Stebs.registerDevice({slot}, function(data){
        console.log(data.Data);
    });
</script>";
```

3. Stebs.registerDevice ist von essentieller Bedeutung. Sie stellt die Einstiegsmethode des Device - Codes dar, ähnlich wie main() in Java.
4. Bei Assemblercode wie...

```
MOV AL,03
OUT 00
END
```

...ist der Wert 03 via data.Data abrufbar.

Werte ins Backend “senden” bzw fürs Backend aktualisieren

Um einen neuen Wert für den Assemblerbefehl “OUT xx” bereitzustellen, ist im Javascriptcode `Stebs.updateDevice({slot}, string message);` von Bedeutung. Dabei kann ein veränderter Wert wie...

```
data.Data = data.Data * 2;
```

...auf zwei Varianten für das Backend aktualisiert und sichtbar gemacht werden kann.

Variante 1: Rechnung im JS - Code und neues Resultat als String übergeben

1. Veränderter Wert als String übergeben:

```
Stebs.registerDevice({slot}, function(data){
    data.Data = data.Data * 2;
    console.log(data.Data);
    var send = data.Data;
    Stebs.updateDevice({slot}, send); //send wird autom. zu string konv.
});
```

2. In xxDevice.cs die Methode "Update" überschreiben.

```
//#IN xx: get values from plugin to backend
public override void Update(string input)    //experiments
{
    byte value = Convert.ToByte(input);
    data = value; //Wert aktualisieren. data ist für Backend immer
                  //ersichtlich.
}
```

Alternativ kann auch die Update() - Methode überladen werden, um einen Interruptbefehl auszulösen:

```
public override void Update(string input, string interruptEnabled)
{
    data = Convert.ToByte(input);
    if (interruptEnabled == "true")
    {
        Interrupt();
    }
}
```

Variante 2: Rechnung in xxDevice.cs (**Not tested yet**)

1. Keyword senden

```
Stebs.registerDevice({slot}, function(data){
    Stebs.updateDevice({slot}, 'multiplyBy(2)');
});
```

2. Keyword verarbeiten

Hinweis: Enthält Pseudocode

```
//#IN xx: get values from plugin to backend
public override void Update(string input)    //experiments
{
    //expected keyword: multiplyBy. received string: multiplyBy(2)
    if( input.substring(0, indexOf('(')).equals("multiplyBy") ) {
        byte value = Convert.toByte(
            input.substring(indexOf('('),indexOf(')'))
        ); //extract the argument, namely 2 and convert to byte val.
        data = data * value;
    }
    if( input.equals('interrupt') ) {
        Interrupt(); //method in xxDevice.cs
        //do other logic stuff
    }
}
```

Grafiken für PluginGUI zeichnen

SVG verwenden:

https://www.w3schools.com/graphics/svg_intro.asp

Vorteile:

SVG Advantages

Advantages of using SVG over other image formats (like JPEG and GIF) are:³

- *SVG images can be created and edited with any text editor*
- *SVG images can be searched, indexed, scripted, and compressed*
- *SVG images are scalable*
- *SVG images can be printed with high quality at any resolution*
- *SVG images are zoomable (and the image can be zoomed without degradation)*
- *SVG is an open standard*
- *SVG files are pure XML*

String Interpolation und Razor

Innerhalb der xxPlugin.cs - Files befindet sich der jeweilige JavaScript - Code für jenes Device. Dieser ist jedoch Teil eines Strings, der von DeviceTemplate() zurückgegeben wird. Da es sich um ein .NET - Projekt handelt, kann die String Interpolation von C# verwendet werden, um Werte von Variablen in den JS - Code einzubacken. Parallel dazu kann auch mit dem Razor Syntax serverseitiger Code ausgeführt werden, bevor der JS - Code an den Client geschickt wird. Folglich kann in...

```
public string DeviceTemplate(byte slot) =>

    $"<style>

    </style>
    ...

```

...eine C# - Variable wie folgt 'eingebacken' werden. Bei test handelt es sich um einen String:

```
<script>
    Stebs.registerDevice({slot}, function(data){
        ...
        console.log(`"${@test}"`);
    });
</script>"
```

³ https://www.w3schools.com/graphics/svg_intro.asp

Im zweiten Beispiel werden Integerwerte eingefügt, bevor der JS - Code an den Client geschickt wird:

```
$@"<style>
  #monitor-{slot} {{
    border: 1px solid black;
    padding-top: {@INITIAL_PADDING_TOP}px;
    padding-left: {@INITIAL_PADDING_LEFT}px;
  }}
</style>
```

6.2.3 Objektorientiertes Programmieren in Javascript

Während der Entwicklung des Textkonsolendevices fiel uns ein Bug auf, in welchem alle Deviceinstanzen auf einen einzelnen, globalen Zustand (globale Variablen im Webbrower) zugriffen. Dies hatte zur Folge, dass jede Instanz ein zusätzliches Zeichen mehr ausdruck als das vorherige. Das erwünschte Verhalten war jedoch so, dass alle Instanz das selbe ausgeben sollten, bevor der nächste MOV-Befehl ausgeführt wird. Im folgenden Screenshot ist dies anhand des Buchstabens 'x' zu erkennen.

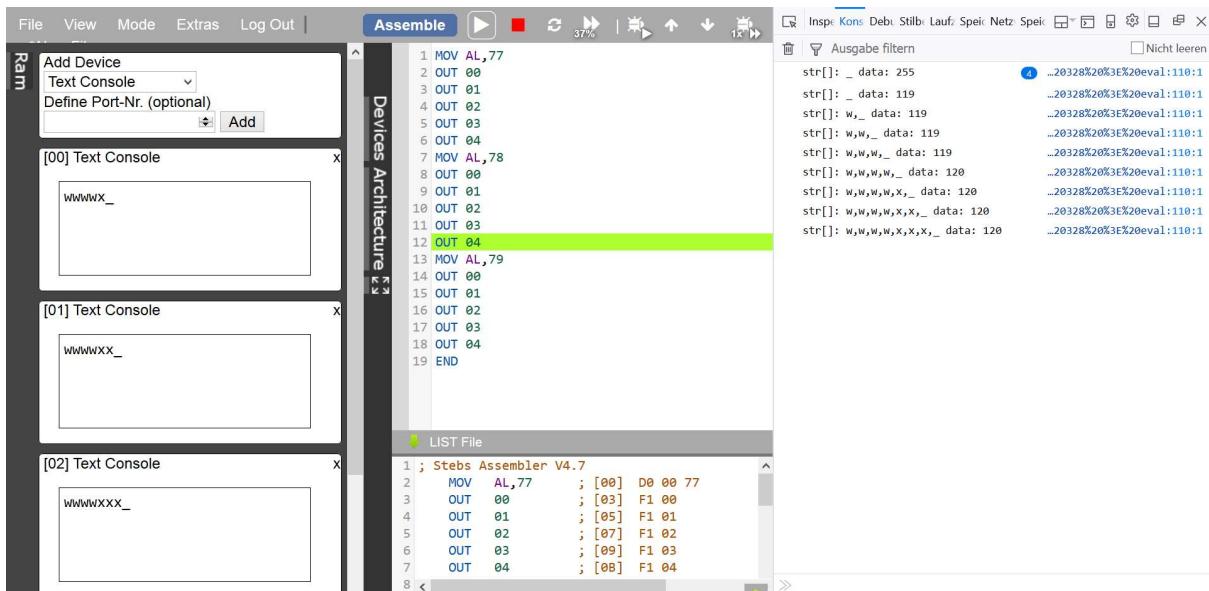


Abbildung 66 Webstebs - geöffnete Entwicklerhilfe

Zustände (Variablen)

Dieses Problem kann Umgangen werden, indem objektorientiertes Denken auf den Javascript-Code angewendet wird. D.h. jede globale Variable, die in Java ein Objektattribut wäre, ist mit einem {slot} zu versehen:

```
var array{slot} = [ 'a', 'b', 'c' ];
```

Möchte man jedoch eine Klassenvariable setzen, kann das {slot} weggelassen werden.

Methoden, Funktionen

Ebenso ist dies bei Methoden und Funktionen zu berücksichtigen. JavaScriptfunktionen, die auf Zustände einer Deviceinstanz zugreifen oder diese verändern, müssen wie eine Objektmethode behandelt und somit mit einem {slot} versehen werden.

Deklaration:

```
var prepareString{slot} = function() {
    //code
}
```

Aufruf:

```
prepareString{slot}()
```

Klassenfunktionen hingegen greifen auf keine globalen Javascriptvariablen zu und übernehmen stattdessen ein oder mehrere Parameter.

```
var isNotAlphaNumericalAscii = function(value) {
    return value < 32 || 126 < value; // [space, ~]
}
```

6.2.4 Existierendes Plugin integrieren

1. Ordner des existierenden Plugins kopieren und ins neue Repository einfügen.
2. Nun muss das Projekt der Projektmappe hinzugefügt werden. Dazu Rechtsklick auf den Ordner Devices, dann auf "Hinzufügen" und "Vorhandenes Projekt" klicken.

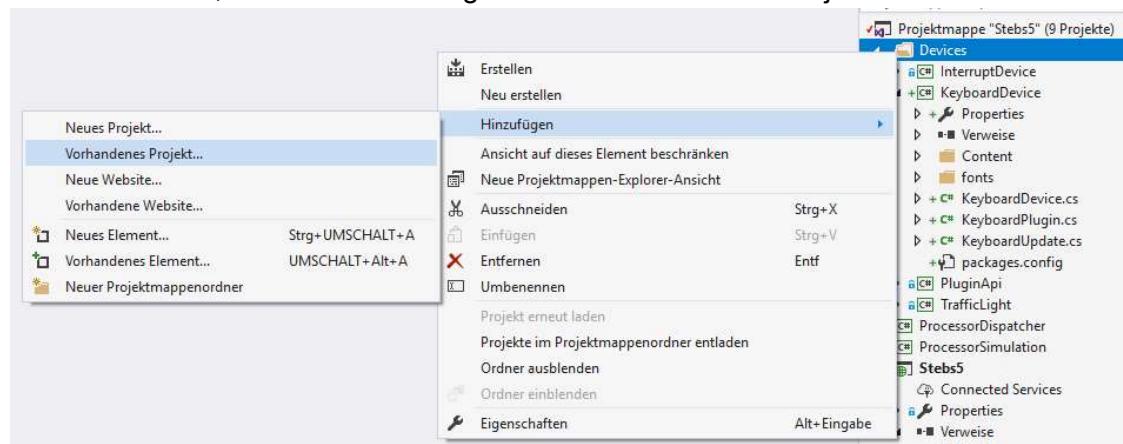


Abbildung 67 neues Projekt für neues Device hinzufügen

3. Jetzt muss der eben kopierte Ordner geöffnet werden und die Datei [Pluginname].csproj ausgewählt werden.

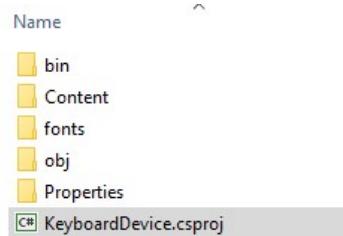


Abbildung 68 Ordnerinhalt

4. Weiter müssen die Referenzen in der Projektmappe kontrolliert werden. Zuerst die Referenzen des neuen Plugins. Dazu macht man im Visual Studio einen Rechtsklick auf "Verweise" im Ordner des neuen Plugins und drückt auf "Verweis hinzufügen".

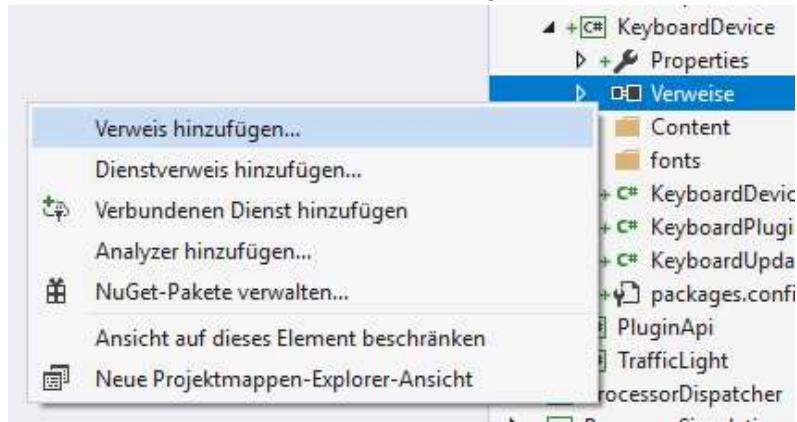


Abbildung 69 Verweise hinzufügen

5. Die Verweise auf "PluginAPI" und ProcessorSimulation sollten bereits vorhanden sein, falls nicht müssen diese hinzugefügt werden.
6. Die Verweise im Projekt Stebs5 müssen noch angepasst werden. Dazu klickt man im Projekt "Stebs5" mit einem Rechtsklick auf Verweise und drückt dann "Verweis hinzufügen". Nun kann dort das neue Plugin ausgewählt werden.
7. Als letzter Schritt muss die Projektmappe neu gebaut werden.

7 Portvergabe

Wird durch den Endnutzer ein Device hinzugefügt, erhält dieses eine Portnummer. Über diesen Port kommuniziert der Server mit ihm. Verwendet der Endnutzer den Assemblybefehl IN oder OUT, wird dabei ein Datenbyte übermittelt. Bei OUT gelangt dieses an das Device und wird von diesem verarbeitet. Bei IN wird das Datenbyte zum Server gesendet.

7.1 Problemstellung

Die Problemstellung wird im Kapitel Problemstellung > Portvergabekonzept beschrieben.

7.2 Umgesetzte Lösung

Bei der umgesetzten Lösung sucht sich Webstebs die in aufsteigender Reihenfolge nächst grösste, freie Portnummer. Werden zum Beispiel fünf Devices hinzugefügt und anschliessend das dritte von ihnen wieder entfernt, so bleiben die Portnummern 00, 01, 03 und 04 weiterhin besetzt. Es besteht nun eine Lücke, nämlich die Nr. 02. Entscheidet sich der Endnutzer, ohne manuelle Porteingabe ein Device hinzuzufügen, so werden die Ports 00, 01, 02, 03 und 04 vergeben sein. Beim nächsten Mal würde dann der Port mit der Nummer 05 belegt werden.

7.3 Source Code

Umgesetzt wurde die automatische Portvergabe mit einer sequentiellen Suche. Dazu mussten ein paar kleinere Änderungen im Stebs.deviceManager vollzogen werden.

Die Information, welche Portnummern bereits vergeben sind, werden im Array `devices` gespeichert. Doch damit dies korrekt verläuft, musste einerseits jedes durch `addDevice()` hinzugefügte Device auch im Array vermerkt werden:

```
deviceManager.devices.push({ slot: device.Slot, type: deviceType });
```

Andererseits müssen nicht mehr benötigte Devices auch aus dem `devices` - Array wieder entfernt werden. Hierfür wurde das Snippet...

```
let index = deviceManager.devices.map(function (e) { return e.slot; }).indexOf(device.Slot);  
if (index > -1) { deviceManager.devices.splice(index, 1); }
```

...dem Klickeventhandler des jeweiligen Devices hinzugefügt, welcher durch Drücken auf den x - Knopf an der oberen, rechten Ecke ausgelöst wird.

`selectNextAvailableDeviceSlot()` kann nun die bereits vergebenen Ports aus dem `devices` - Array gewinnen und aufsteigend sortieren, bevor die sequentielle Suche ausgeführt wird. Das Ergebnis der Suche wird schlussendlich ins Inputfeld geschrieben und somit im GUI angezeigt.

8 Testen

Die meisten Änderungen, die in diesem Projekt an Webstebs gemacht wurden, betreffen das GUI. Ein GUI lässt sich nur schlecht testen. Das Verfahren ist aufwendig und mühsam. Mit Software kann der Prozess nur teilweise automatisiert werden.

Die Tests in diesem Projekt mit einer Software durchzuführen, wurde in Betracht gezogen. Der Nutzen daraus wurde aber als zu gering angesehen. Der Initialaufwand alle Testfälle in die Software auf zu nehmen wäre zu gross gewesen. Wären die Tests später angepasst worden, hätten sie wiederum neu aufgenommen werden müssen. Aus diesem Grund wurden die Tests von Hand durchgeführt.

Damit das GUI besser getestet werden kann, wurde es in mehrere Bereiche unterteilt. So kann jeder Bereich unabhängig von den anderen getestet werden. Das GUI als gesamtes zu testen ist kaum möglich. Dies würde eine riesige Anzahl an Tests nach sich ziehen. Man müsste jede erdenkliche Kombination von Aktionen, die der Endnutzer auslösen kann, testen.

Deswegen hat man sich mit dem Kunden geeinigt jeden Bereich separat zu testen. Das GUI von Webstebs wurde in folgende Bereiche unterteilt:

- Menübar
- Simulation Controls
- Sidepanels
- Register
- Keyboard Device
- Text Console Device
- Number Display Device
- 7-Segment Display Device
- Character Viewer Device

Für jeden dieser Bereiche werden alle normalen Vorgänge und Aktionen getestet, die der User machen kann. Es wird nicht versucht alle Grenzen der Applikation auszureißen, es soll die normale Bedienung von Webstebs überprüft werden.

Getestet wurde mit den, von den Studierenden, am meisten verwendeten Browsern. Dies sind die folgenden Browser:

- Mozilla Firefox
- Google Chrome
- Apple Safari
- Microsoft Edge

Der Safari Browser ist für die Verwendung mit Apple MACs gedacht. Für Microsoft Windows wird er nicht mehr weiterentwickelt. Damit Webstebs auch in diesem Browser getestet werden kann, wurde MAC Computer eingesetzt.

Bei der Entwicklung wurde festgestellt, dass Webstebs bei unterschiedlichen Bildschirmgrössen anders reagieren kann. Aus diesem Grund wurden auch Tests mit verschiedenen Bildschirmgrössen gemacht. Dabei wurde festgestellt, dass Bildschirme mit einer Diagonale kleiner als 14 Zoll anders reagieren als grössere. Die Unterschiede betreffen nur den Bereich Seitenpanels, alle anderen funktionieren bei allen Bildschirmgrössen gleich.

Es wurden drei Testdurchgänge durchgeführt.

1. Im ersten Durchgang, wurden bei den Browsern Chrome, Safari und Edge Fehler gefunden, die beim Firefox Browser Firefox nicht vorhanden waren.
2. Im zweiten Durchgang, waren diese Fehler behoben. Es konnten keine neuen Probleme entdeckt werden,
3. Der letzte Durchgang wurde mit der finalen Version des Codes durchgeführt.

8.1 Verweise auf die Testfälle

Firefox:

- [Testfälle\Test_1_FF_03.01.18.docx](#)
- [Testfälle\Test_2_FF_08.01.18.docx](#)
- [Testfälle\Test_3_FF_18.01.18.docx](#)

Chrome:

- [Testfälle\Test_1_Chrome_05.01.18.docx](#)
- [Testfälle\Test_2_Chrome_07.01.18.docx](#)
- [Testfälle\Test_3_Chrome_18.01.18.docx](#)

Edge:

- [Testfälle\Test_1_Edge_5.1.18.docx](#)
- [Testfälle\Test_2_Edge_8.1.18.docx](#)
- [sTestfälle\Test_3_Edge_18.1.18.docx](#)

9 Known Issues

9.1 Filemanager

Beim Erstellen von neuen Files oder Foldern gibt es einen Bug. Wenn ein neues File erstellt werden soll, muss zuerst der Filename eingegeben werden. Danach klickt man auf bestätigen, um das neue File zu erstellen. Das neu erstellte File sollte jetzt in der Liste des Filemanagers zu sehen sein. Dies ist aber nicht der Fall. Das File wurde zwar erstellt, wird dem Endnutzer aber nicht angezeigt. Das File wird erst angezeigt, wenn der Filemanager geschlossen und wieder geöffnet wird. Eine andere Möglichkeit, dass File sichtbar zu machen, ist es ein weiteres neues File zu erstellen. Nachdem dies gemacht wurde, wird das zuerst erstellte File angezeigt, das zuletzt erstellte aber nicht.

Dieser Bug taucht nicht bei jedem Mal auf, wenn ein neues File erstellt wird. Die Endnutzer können durch diesen Bug verwirrt werden, da sie die Aktion durchgeführt haben aber keinerlei Feedback dazu sehen können. Auch ein öffnen der gerade erstellten Datei ist so nicht möglich. Beim Ändern eines Namens kann es teilweise zu Problemen kommen.

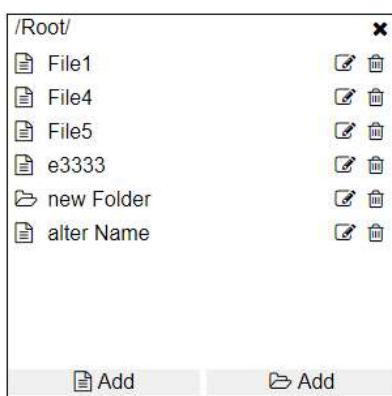


Abbildung 70 Filemanager zu Beginn

Wenn die Änderungen am Namen gespeichert werden sollen, klickt man auf bestätigen.

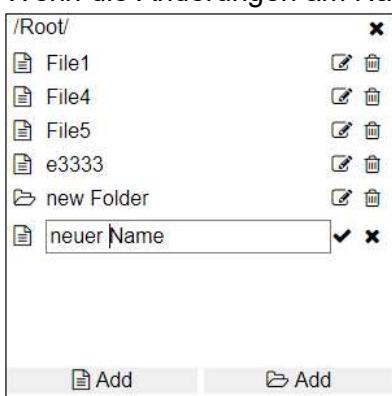


Abbildung 71 Filemanager - neues File erstellen

Danach wird nicht das File mit dem neuen Filenamen angezeigt. Sondern es ist das Eingabefeld mit dem alten Namen zu sehen, dieser könnte wieder bearbeitet werden.

Daneben sind immer noch die Icons zum Bestätigen oder zum Abbrechen zu sehen.



Abbildung 72 Filemanager - alter Name wird angezeigt

Wird nun erneut auf Bestätigen geklickt ist wieder der neue Name zu sehen.

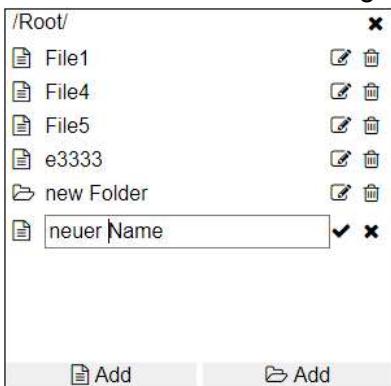


Abbildung 73 Filemanager - neuer Name wird angezeigt

Es gibt einen Workaround, damit der Filename trotzdem geändert werden kann. Nachdem man den Namen geändert hat und wieder der alte Name angezeigt wird, muss abgebrochen und nicht erneut bestätigt werden. Danach hat das File den neuen Namen erhalten.

9.2 Architektur-Panel

Die Grösse des Architektur-Canvas im Seitenpanel Architecture wird je nach Breite des Seitenpanels angepasst. Bei der Entwicklung wurde festgestellt, dass das Resizing bei unterschiedlichen Bildschirmgrössen anders reagieren kann. Bei Bildschirmen mit einer Diagonale grösser als 14 Zoll funktioniert es nicht richtig. Wenn das Architektur-Seitenpanel geöffnet ist und dann eines der anderen beiden Seitenpanels geöffnet oder geschlossen wird, tritt ein Fehler auf. Das Architektur-Canvas wird dann zu gross dargestellt. Wird jetzt die Grösse des Architektur-Seitenpanels durch Drag and Drop verändert, wird die Grösse des Architektur-Canvas wieder korrekt angepasst.

9.3 Architektur in separatem Fenster

Im Browser Edge von Microsoft ist es nicht möglich die Architekturansicht in einem separaten Fenster anzuzeigen. Das Fenster wird zwar geöffnet, anstatt des Architektur-Canvas ist aber nur ein schwarzer Hintergrund zu sehen.

Ein weiteres Problem stellen die Tabellen in der Architekturansicht dar. Wenn der Assembly-Code assembliert wird, verschwinden die beiden Tabellen in der Architekturansicht im separaten Fenster.

9.4 Step into

Vorbedingungen:

1. Frisch in Webstebs eingeloggt.
2. Code geschrieben oder eingefügt.

Reproduktionsschritte:

1. Per Mausklick Cursor des Texteditors auf eine beliebige Zeile Code setzen.
2. Nun auf "step into" klicken.

Effekt: Der grüne Balken, der die aktuell ausgeführte Codezeile hervorhebt, springt an die Stelle des Cursors. Erwartet wird eigentlich, dass der Balken bei der ersten Zeile Code startet.

11 Verbesserungsvorschläge

11.1 Filemanager

Im Filemanager ist es möglich, den selben Dateinamen mehrfach zu vergeben. Dies sollte unverbunden werden.

11.2 Keyboard-Shortcuts

In Webstebs sind bereits einige Keyboard-Shortcuts implementiert. Es gibt aber eine Vielzahl an Möglichkeiten weitere Shortcuts einzuführen. Nachfolgend sind einige mögliche Erweiterungen aufgelistet.

11.2.1 Steuerung mit der ctrl. - Taste

Simulation Control	Assemble	Play	Stop	Reset	Speed
Shortcut mit ctrl. +	g	h	j	k/r	Links, recht oder + , -

Simulation Control	Debug	Up	Down	Export	Hilfe
Shortcut mit ctrl. +	L	Up	Down	e	F1

11.2.2 Steuerung mit der Alt - Taste

alt + 1 = toggle RAM-Sidepanel
alt + 2 = toggle Device-Sidepanel
alt + 3 = toggle Architektur-Sidepanel

oder

alt + 1 = Instruction-Geschwindigkeit
alt + 2 = Macro-Geschwindigkeit
alt + 3 = Micro-Geschwindigkeit

11.3 List File Bereich

Der Bereich, der nach dem Assemblieren unterhalb des Codeeditors ausklappt, kann in eine Position gebracht werden, aus der er kaum mehr gelöst werden kann. Der Bereich lässt sich manuell ein- und ausklappen. Zusätzlich kann er im ausgeklappten Zustand auch per Drag and Drop vergrössert oder verkleinert werden. Dabei kann er so stark verkleinert werden, dass nur noch ein wenige Pixel hoher Balken zu sehen ist. Dieser kleine Balken kann vom Endnutzer übersehen werden. Hier muss eine Möglichkeit gefunden werden, dass der Endnutzer gar nicht in diese Situation kommt.

11.4 Status – Pop-up

Nach erfolgreichem Speichern oder Assemblieren wird dem Endnutzer dies in einem Pop-up angezeigt.



Abbildung 74 Status Pop-up

Dieses Pop-up befindet sich in der Mitte des Codeeditors. Wenn jedoch Seitenpanels geöffnet sind, rückt die Mitte des Codeeditors an die rechte Seite des Browserfensters. So ist es möglich, dass es von den Endnutzern nicht mehr richtig wahrgenommen wird. Besser wäre es, die Pop-up Nachricht in der Mitte des Browserfensters anzuzeigen.

11.5 Seitenpanels

Die vertikalen Buttons zum ein- und ausklappen der Seitenpanel sind zurzeit mit Bildern umgesetzt. Dies sollte grundsätzlich vermieden werden. Weiter sollten alle Buchstaben des Buttons «Ram» gross geschrieben werden.

11.6 Menübar

Es können nicht immer alle Aktionen der Menübar ausgeführt werden. Ist eine Aktion momentan nicht verfügbar, sollte ihr Button disabled werden. Normalerweise wird dies gemacht, indem der Text nicht mehr schwarz, sondern grau dargestellt wird. So weiss der Endnutzer, dass er diese Aktion derzeit nicht ausführen kann. Dies Methode ist bei Softwarelösungen weit verbreitet und die meisten Endnutzer kennen sie.

11.7 Breakpoints

Breakpoints können nur im AssembledState und PausedState gesetzt werden. Durch klicken auf den Resetknopf verlässt man diese wieder, sodass gesetzte Breakpoints nicht direkt entfernt werden können. Hierfür muss der Endnutzer zuerst wieder assemblieren. Dies wirkt aus Usabilitysicht unintuitiv.

Mit diesem Workflow kann die Situation reproduziert werden:

1. Assemblieren
2. Breakpoint setzen
3. Code ausführen
4. Reset klicken
5. Versuchen, den Breakpoint zu entfernen.

Ein zusätzliches Problem wird im Anschluss zu Schritt 5 klar. Wer wieder assembliert und den Breakpoint aus Schritt 2 erhalten bleiben lässt, der wird verblüfft sein. Obwohl der alte Breakpoint noch im UI angezeigt wird, wird die Ausführung des Codes nicht an dessen Stelle gestoppt.

12 mögliche Erweiterungen

12.1 Visual Design: Color with Meaning

In "WebstebsIO" wurde das Augenmerk vermehrt auf das Interaction Design und die Struktur des GUI gelegt. In einem weiteren Schritt könnte ein einheitlicher Style Guide erarbeitet und implementiert werden. Wünschenswert ist es, dass Farben klar definierte Bedeutungen erhalten und dadurch den Endnutzer zusätzlich führen.

12.2 Hilfeseite

In Webstebs könnte eine Hilfe - Webseite für die Endnutzer enthalten sein. Auf dieser Webseite werden die Möglichkeiten von Webstebs aufgezeigt und für Einsteiger verständlich erklärt. Darin finden sich auch Hilfestellungen zum Assemble – Code. Die Webseite könnte über die Menübar und über die Taste F1 erreichbar sein.

12.3 Add Device

Der gesamte Vorgang zum Hinzufügen eines Devices könnte überarbeitet werden. Zurzeit ist er noch sehr rudimentär umgesetzt. Man könnte einen Menüablauf einführen, bei dem die Endnutzer durch den Vorgang geführt werden. Dabei könnte es auch möglich sein, mehrere Devices auf einmal hinzuzufügen. Die Auswahl könnte auch anders als mit einem Dropdown gelöst werden. Am besten anbieten würde sich eine Auswahl mit Bildern der Devices. Bilder zeigen den Endnutzern auf eine einfache und schnelle Art, um welches Device es sich handelt.

12.4 Deviceliste

Es liegt noch viel Potential darin, wie die hinzugefügten Devices künftig dargestellt werden. Zurzeit wird dies durch eine vertikale Liste bewerkstelligt. Diese hat für den Endnutzer jedoch den Nachteil, nur eine bestimmte Anzahl Devices gleichzeitig betrachten zu können. Zwar ist dieser nicht gravierend, doch wäre wünschenswert, wenn eine Grossübersicht über alle Devices zur Verfügung stünde. Dabei könnte sich ähnlich wie beim Architecture - Canvas um ein sich neu öffnendes Browserfenster handeln. Oder ein Seitenpanel, dessen Anzeigefläche sich verändern kann.

Egal welcher Weg gegangen wird, ein sicherer Zusatznutzen würde sich ergeben, wenn Devices per Drag and Drop in ihrer Reihenfolge geändert werden könnten.

12.5 Portnummern nachträglich verändern

Es wäre sehr nützlich die Portnummer eines Devices ändern zu können, nachdem man ein Device hinzugefügt hat. Falls ein Endnutzer beim Hinzufügen eines Devices eine falsche Portnummer eingegeben hat, kann diese danach noch verändert werden.

Da die Portnummer eines Devices auch gleichzeitig die Slotnummer ist, gestaltet sich die Umsetzung schwierig. Der grundlegende Aufbau der Devices müsste angepasst werden. Dies kann viele weitere Anpassungen nach sich ziehen.

12.6 Register

Die Reihenfolge der Register nach Belieben zu verändern kann nützlich sein. Für die Endnutzer wäre es am einfachste, wenn sie die Register per Drag and Drop verschieben könnten. Derzeit kann die Reihenfolge der Register nur verändert werden, in dem die Register entfernt und dann in der gewünschten Reihenfolge wieder eingefügt werden.

12.7 Device mit mehreren Ports

In Webstebs können bis jetzt Devices nur über einen Port angesprochen werden. Über diesen Port kann ein Byte an Informationen versendet werden. Würde es auch Devices mit zwei oder mehr Ports geben, stünden ganz neue Möglichkeiten zur Verfügung. Es könnten neue, derzeit nicht umsetzbare, Devices erstellt werden. Es könnten auch neue Aufgabenstellungen erarbeitet werden, die in der jetzigen Webstebs Version einsetzbar gewesen wären.

Die beiden Ports könnten für ganz unterschiedliche Aufgaben verwendet werden. Zum Beispiel könnte das 7-Segment Display mit zwei Ports umgesetzt werden. Derzeit müssen die beiden Displays nacheinander angesprochen werden. Die Unterscheidung, welches Display angesprochen werden soll, findet über die erste Stelle im Byte statt. Genauere Informationen über diesen Vorgang ist im Kapitel 6.1.5 7-Segment Display zu finden. Dadurch, dass ein Bit des Bytes für die Definition des Displays benötigt wird, können weniger Informationen übertragen werden. Ein weiterer Nachteil ist, dass zweimal etwas über das Port versendet werden muss, wenn man auf beiden Displays etwas anzeigen möchte. Wenn zwei Ports vorhanden wären, könnte je ein Port für ein Display benutzt werden.

Könnten Devices mit mehr als zwei Ports entwickelt werden, gäbe es sogar noch mehr Möglichkeiten für Devices.

Damit Devices mit mehr als einem Port eingesetzt werden können, muss Webstebs stark überarbeitet werden. Die Portnummer stellt zurzeit die Identifizierung der Devices dar. Eine Änderung daran kann viele weitere Änderungen nach sich ziehen.

12.8 externes RAM

Ein solches Device mit mehreren Ports wäre ein externes RAM. Die Idee, dieses externen RAMs, wäre es, den Endnutzern die Möglichkeit zu bieten, Informationen zu speichern. Derzeit können hexadezimale Werte nur auf dem Stack speichern. Dort ist aber der Speicherplatz, gerade bei grösseren Programmen, sehr begrenzt. Deswegen wäre es für sie sehr nützlich, wenn sie weiteren Speicherplatz zur Verfügung hätten. Das externe RAM könnte ähnlich aufgebaut werden, wie das bereits bestehende. Über zwei Ports könnten Informationen abgespeichert und wieder ausgelesen werden. Über einen der beiden Ports könnte eine Speicheradresse versendet werden. Über den zweiten Port könnten die Informationen, die an dieser Speicheradresse gespeichert werden sollen, versendet werden.

13 Fazit, Reflexion

Das Hauptziel des Projektes war, die Usability von Webstebs zu verbessern. Dieses Ziel haben wir eindeutig erreicht. Die Applikation verfügt über ein überarbeitetes, benutzerfreundlicheres Design, wodurch Endnutzer sich leichter in Webstebs zurechtfinden. Dabei wurden UI - Elemente gruppiert, verschoben oder neu eingeführt. Die neue Menübar hilft Neuanwendern, sich in Webstebs zu orientieren, während das ebenfalls neue Devices - Seitenpanel erlaubt, mit einer beliebigen Anzahl an Devices zu arbeiten. Die Simulation Control befindet sich nun an einer einzelnen, gut zugänglichen Stelle des User Interfaces. Die ebenfalls zuvor am unteren Bildschirmrand beheimateten Register wurden den semantischen Gruppen "RAM" und "Architecture" zugeordnet und unter den gleichnamigen Seitenpanels untergebracht. Dadurch konnte Platz für die CodingView gewonnen und ein sauberes Gesamtbild erzeugt werden. Zudem wurde das Portvergabekonzept verbessert, um den Endnutzer das Hinzufügen von Devices angenehmer zu gestalten.

Trotz den zahlreichen Änderungen am Frontend konnte die Systemarchitektur problemlos übernommen werden.

Mit den neu entwickelten Devices wurde ausserdem das Spektrum der möglichen Übungsaufgaben erweitert. Theorie zur IO - Peripherie kann nun in Webstebs erprobt werden.

Als einiges zeitaufwändiger als anfangs erhofft, hat sich die Einarbeitung in den Source Code erwiesen. Zwar gab die Dokumentation einen guten, ersten Überblick, was zu erwarten war, doch handelt es sich bei Webstebs um ein etwas grösseres System. So dauerte es alleine schon zwei Wochen, bis unser erstes Device korrekt lief.

Danach nahm das Verständnis für den Source Code über die gesamte Projektdauer hinweg fortlaufend zu.

Parallel dazu wurden Untersuchungen zur Usability geführt, Verbesserungen an Webstebs entwickelt und schlussendlich getestet. Dies funktionierte soweit gut und bereitete uns viel Freude. Die bis zu Weihnachten gesammelten Endnutzer- und Kundenfeedbacks ergaben Aufschluss darüber, wie noch mehr aus Webstebs herausgeholt werden könnte. Dies führte dazu, dass noch mehr Zeit ins Polishing investiert wurde, aber das Projekt dadurch von seinem ursprünglichen Plan abwich. Für die Dokumentation waren ursprünglich zwei Sprints à je zwei Wochen geplant. Davon konnte jedoch nur eine zweckmässig genutzt werden.

Wir bedauern, dass das Projekt bereits nach einem Semester zu Ende ist. Kaum hat man sich in ein Projekt eingearbeitet, muss schon bald wieder die Bremse betätigt werden. Dennoch blicken wir mit etwas Stolz auf dieses Projekt zurück, welches uns viel Freude bereitete und wir neues dazulernen konnten - als Mensch und als Programmierer.

14 Glossar

15 Abbildungsverzeichnis

Abbildung 1 Stand der Applikation zum Start des Projekts	5
Abbildung 2 Stand der Applikation zum Start des Projekts - Architekturansicht	6
Abbildung 3 Devices in der alten Webstebs Version	7
Abbildung 4 Traffic Light Device	7
Abbildung 5 Interrupt Device	7
Abbildung 6 Grobarchitektur	9
Abbildung 7 Simulation Control in der alten Webstebs Version	13
Abbildung 8 IntelliJ Werkzeugkasten	20
Abbildung 9 Eclipse Tabs	21
Abbildung 10 Photoshop Menübar	21
Abbildung 11 Photoshop Werkzeugkasten	21
Abbildung 12 Arduino IDE	22
Abbildung 13 Papier-Prototyp erstellen	22
Abbildung 14 Papier-Prototyp UX-Test	23
Abbildung 15 Einstiegsbildschirm	24
Abbildung 16 Device - Übersicht	25
Abbildung 17 Add Device	26
Abbildung 18 Portmanagement	27
Abbildung 19 Portnummer editieren	28
Abbildung 20 Device-Übersicht Variante 1	29
Abbildung 21 Device-Übersicht Variante 2	30
Abbildung 22 Alternative Registerposition	31
Abbildung 23 Hinweise: "Assemblieren nicht vergessen"	32
Abbildung 24 Simulation Control	33
Abbildung 25 Nach der 9. Skizze stand das Layout fest. Mit den Icons wurde hingegen weiterexperimen-tiert.	33
Abbildung 26 Übersicht Webstebs	35
Abbildung 27 Klassendiagramm	36
Abbildung 28 Pop-ups Simulation Control	38
Abbildung 29 Geschwindigkeits-Slider	39
Abbildung 30 neues State Diagramm	41
Abbildung 31 Filename in der alten Webstebs Version	45
Abbildung 32 Filename in der neuen Webstebs Version	45
Abbildung 33 Seitenpanels in der alten Webstebs Version	46
Abbildung 34 Devices unterhalb des Codeeditors	47
Abbildung 35 Devices in einem Seitenpanel	47
Abbildung 36 scrollbare Device Ansicht	48
Abbildung 37 Device mit hervorgehobener Portnummeranzeige	49
Abbildung 38 vertikale Buttons geöffnet/geschlossen	50
Abbildung 39 Register in den beiden Seitenpanels	52
Abbildung 40 ausgeklapptes Dropdown "Register hinzufügen"	52
Abbildung 41 RAM in der alten Webstebs Version	53
Abbildung 42 Collape RAM	53
Abbildung 43 Closeall-Button	54
Abbildung 44 Filemanager in der alten Webstebs Version	55
Abbildung 45 Filemanager - neues File erstellen 1	55
Abbildung 46 Filemanager - neues File erstellen 2	56
Abbildung 47 Filemanager Buttons in der alten Webstebs Version	56

Abbildung 48 Filemanager - delete many	58
Abbildung 49 Filemanager - Add File/Folder	59
Abbildung 50 Sehr langer Filename	60
Abbildung 51 LIST File	61
Abbildung 52 Register und Simulation Control in der alten Webstebs Version	61
Abbildung 53 vor der Anpassung des Codes	62
Abbildung 54 Device Ansicht	64
Abbildung 55 Keyboard Device	65
Abbildung 56 Keyboard Device - hervorgehobener Interrupt Schalter	65
Abbildung 57 Text Console Device	68
Abbildung 58 Text Console Device - hervorgehobenes invalid Symbol	68
Abbildung 59 Text Console Device 1	68
Abbildung 60 Text Console Device 2	69
Abbildung 61 Character Viewer Device	69
Abbildung 62 Number Display Device	71
Abbildung 63 Number Display - Wert 00	72
Abbildung 64 7-Segment Display	73
Abbildung 65 Visual Studio - Add New Project	76
Abbildung 66 Webstebs - geöffnete Entwicklerhilfe	81
Abbildung 67 neues Projekt für neues Device hinzufügen	82
Abbildung 68 Ordnerinhalt	82
Abbildung 69 Verweise hinzufügen	83
Abbildung 70 Filemanager zu Beginn	87
Abbildung 71 Filemanager - neues File erstellen	87
Abbildung 72 Filemanager - alter Name wird angezeigt	88
Abbildung 73 Filemanager - neuer Name wird angezeigt	88
Abbildung 74 Status Pop-up	91

16 Ehrlichkeitserklärung

Hiermit erkläre ich, das vorliegende IP5-Projekt zum Thema “I/O-Devices für Webstebs” selbstständig, ohne Hilfe Dritter und nur unter Benutzung der angegebenen Quellen verfasst zu haben.

Stefan Wohlgensinger

Adrian Bürki

Windisch, 22. Januar 2018