



Arbeitsaufwand: ca. 90 Minuten

Zusammenfassung

Wir lernen die wichtigsten Entwicklerwerkzeuge kennen, installieren benötigte Tools und machen uns ein wenig mit JavaScript-Engines vertraut. **Bei diesem Aufgabenblatt wird nichts abgegeben und auch nichts bewertet.**

Inhaltsverzeichnis

1	Einleitung	1
1.1	Lernziele	1
2	Laufzeitumgebungen	2
2.1	Browser als Entwicklungsumgebung	2
2.1.1	Developer Tools	3
2.1.2	HTML-Inspektor	3
2.1.3	JavaScript-Debugger	4
2.1.4	JavaScript-Console	5
2.2	Node.js	5
3	Visual Studio Code	6
3.1	Ausführen eines Beispielprogramms	6
3.2	Ausführen eines Skripts in HTML	7
	Abkürzungsverzeichnis	8

1 Einleitung

1.1 Lernziele

Sie können grundsätzlich JavaScript- und TypeScript-Projekte bearbeiten
indem Sie die Developer-Tools der Browser verwenden, eine Entwicklungsumgebung mit Extensions aufsetzen, sowie Node.js installieren und starten können
damit Sie sich in Zukunft auf die eigentlichen Inhalte konzentrieren können.

2 Laufzeitumgebungen



Arbeitsaufwand: 30 Minuten (Lesen, Nachvollziehen)

Der Grund für den Einsatz von JavaScript ist ganz einfach: Es ist die einzige Programmiersprache, die direkt von Web-Browsern ausgeführt werden kann. Im Browser ist dafür eine JavaScript-Engine (bzw. -Interpreter) zuständig. Je nach Browser hören diese auf unterschiedliche Namen: In Chromium (sowie darauf aufbauenden Browsern wie Chrome oder Edge) ist dies V8, in Firefox ist es SpiderMonkey und bei Safari (bzw. auf WebKit-basierenden Browsern) JavaScriptCore.

Um JavaScript ohne Browser auszuführen, brauchen wir ebenfalls eine JavaScript-Engine. Dafür gibt es Node.js: Node.js verpackt V8 mit weiteren praktischen Dingen (etwa einem umfangreichen Application Programming Interface (API)), so dass wir JavaScript von der Kommandozeile während der Entwicklung oder später auf dem Server laufen lassen können.

2.1 Browser als Entwicklungsumgebung

Quasi alle Browser bieten weitgehende Entwicklerwerkzeuge an:

- eine Read-Eval-Print Loop (REPL), also eine Konsole in der wir JavaScript eingeben und ausführen können und in der auch die Ausgabe über `console.log` erscheint
- einen JavaScript-Debugger
- einen HTML-Inspektor mit der Möglichkeit, Werte zu ändern (quasi ein Debugger für HTML und CSS)
- weitere Ansichten mit Informationen zu Netzwerk-Verkehr, Speicherverbrauch, Persistenz (Cookies, Datenbank etc.) und anderes










Sie können prinzipiell (fast) jeden Browser Ihrer Wahl verwenden, also Chrome, Firefox, Safari, oder Edge. In den Folien und Aufgabenblättern wird Chrome (auf Englisch) verwendet.¹⁾

¹⁾Wir verwenden Englisch, da die meisten Dokumente im Internet ebenfalls englische Versionen verwenden.

2.1.1 Developer Tools

Öffnen Sie die Seite

`https://www.bht-berlin.de/vi`

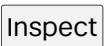
im Browser. Sie können die folgenden Schritte natürlich auch an einer anderen Seite testen! Öffnen Sie nun die sogenannten Developer Tools mit    (macos) oder    (Windows/Linux). Alternativ kann man die Ansicht über das Menü    öffnen.

Schauen Sie sich die einzelnen Funktionen an, die Sie in den Tabs (den Reitern) der Developer Tools sehen (vgl. Abbildung 1). Im Laufe der nächsten beiden Semester werden Sie immer besser verstehen, was man da alles sehen (und manipulieren) kann!





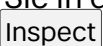
Abbildung 1: Developer-Tools Tabs

2.1.2 HTML-Inspektor

Gehen Sie in der Webseite auf ein beliebiges Element (Text, Bild oder einfach eine freie Fläche) und öffnen Sie das Kontextmenü (in Sie die rechte Maustaste klicken). Wählen Sie dort den Eintrag .

In den Developer-Tools ist nun der Reiter “Elements” aktiv. Links sehen Sie den Quellcode (HTML), rechts eine Ansicht mit vielen neuen Reitern.

Fahren Sie links über den Quellcode mit der Maus. Was können Sie beobachten? Man kann auch Codeteile über die Dreiecke  ein- und  ausklappen.

Selektieren Sie in der Webseite die Überschrift “Fachbereich VI” und wählen Sie wieder  im Kontextmenü. In der rechten Ansicht der Elements-Ansicht sehen Sie die “Styles” (vgl. Abbildung 2). Hier kann man sogar editieren! Gehen Sie auf einen beliebigen Eintrag, vielleicht auf die Farbe (color) und geben Sie dort einen neuen Wert ein (beim Klick auf das Farbfeld geht ein kleiner Dialog auf, der dabei hilft). Was passiert auf der Webseite? Spielen Sie auch mit anderen Werten herum.

Keine Angst – man kann hier nichts kaputt machen. Sie verändern ja nur die Ansicht im Browser.

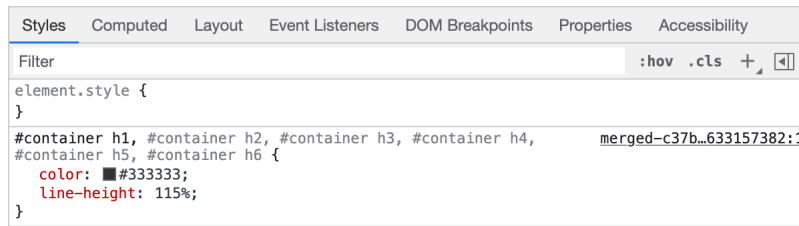


Abbildung 2: Styles (der HTML-Elemente)

2.1.3 JavaScript-Debugger

Wählen Sie nun (in der Developer-Tools-Ansicht) den Reiter "Sources". Dort sehen Sie ganz links eine Baum-Ansicht (unter dem Reiter "Page"), ganz oben steht "top". Darunter finden Sie eine Datei, die mit "jquery.tablesorter" beginnt. Wählen Sie diese Datei aus. Auch wenn Sie noch nie JavaScript gesehen haben, werden Sie die Kommentare vom Code unterscheiden können. Der Code fängt mit einer `function` an (ca. Zeile 100). Setzen Sie einen Breakpoint in dieser Funktion, indem Sie links bei den Zeilennummern klicken (etwa vor der Zeile `tablesorter: new`). Der Breakpoint wird blau dargestellt (vgl. Abbildung 3). (Wenn man in einer Zeile klickt, wo kein Breakpoint möglich ist, passiert auch nichts).

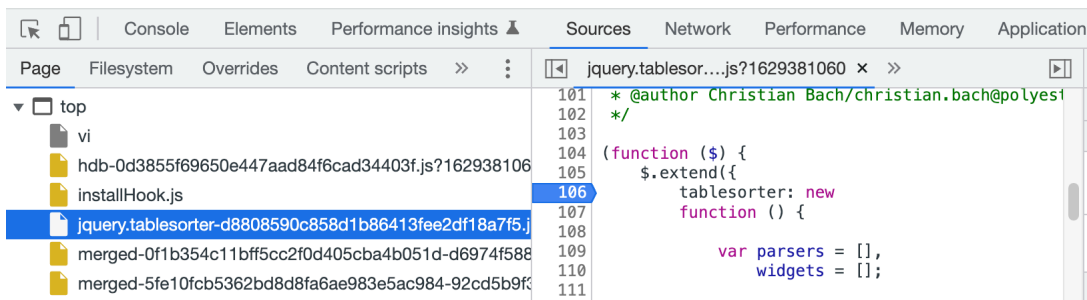


Abbildung 3: Sources mit Breakpoint

Laden Sie jetzt die Seite neu, etwa mit `⌘R` (macos) oder `Ctrl R` (Windows/Linux). Nun sollte der Breakpoint das Laden anhalten. Im Quellcode wird die aktuelle Zeile blau markiert. Links sind die Debug-Befehle zu sehen (Abbildung 4). Spielen Sie hiermit einfach mal rum – Sie kennen ja Debugger aus Eclipse oder IntelliJ.



Abbildung 4: Steuerbefehle des Debuggers

2.1.4 JavaScript-Console

Öffnen Sie nun den Tab "Console". In dieser sogenannten REPL können Sie interaktiv JavaScript programmieren. Probieren Sie doch einfach mal folgenden Code aus:

```
for (let i=0;i<10;i++) { console.log(i); }
```

Oder definieren Sie eine Variable wie folgt:

```
let hello = "Guten Tag";
```

Geben Sie dann einfach

```
hello
```

ein. Was wird ausgegeben?

Laden Sie die Seite neu. Was wird ausgegeben, wenn man erneut `hello` eingibt?

2.2 Node.js

Wir werden jetzt Node.js installieren. Laden Sie die Version 22 (22.x.x) unter

<https://nodejs.org/>

für Ihr Betriebssystem herunter, falls Sie noch kein Node.js installiert haben, und installieren Sie es.



Um mit verschiedenen Node-Versionen zu arbeiten und um einfach neue Versionen zu installieren, wird das Tool **nvm!** (**nvm!**) empfohlen. Wie Sie dieses Tool installieren und verwenden, ist auf der Projektseite des Tools unter <https://github.com/nvm-sh/nvm> beschrieben.

Gehen Sie jetzt auf die Kommandozeile (Terminal bei macOS/Linux oder PowerShell bei Windows). Sie sollten dort nun die Node.js-REPL mit `node` starten können und dann in etwa Folgendes sehen:

```
~$ node
Welcome to Node.js v22.14.0.
Type ".help" for more information.
>
```

Sie können die gleichen Beispiele wie unter Abschnitt 2.1.4 testen.

Um die REPL zu beenden, drücken Sie `ctrl` `D` oder geben Sie `.exit` (mit führendem Punkt!) ein.

3 Visual Studio Code



Arbeitsaufwand: 1 Stunde (Lesen, Nachvollziehen)

Zur Entwicklung werden wir Visual Studio Code, kurz VSCode, verwenden. Gehen Sie zu

<https://code.visualstudio.com/>

und installieren Sie die aktuelle Version. VSCode gibt es für Windows, Linux und macOS. Genauere Hinweise sind unter <https://code.visualstudio.com/docs/setup/setup-overview> zu finden.

Für die Programmierung von JavaScript und TypeScript ist an sich alles vorbereitet. Für ein paar Spezialdinge werden wir später Extensions installieren. Diese finden Sie in der Seitenleiste unter dem Symbol

Normalerweise werden Sie dort den Explorer aktivieren, um die Dateien zu sehen. Um Code nach Git (bzw. Gitlab) einzuspielen, können Sie entweder die Kommandozeile verwenden oder die Source-Control-Dialoge unter . Den Debugger finden Sie unter .

Die vermutlich wichtigste Funktion von VSCode ist die sogenannte “**Command Palette**”. Diese finden Sie im Menü unter oder über (macOS) oder (Windows/Linux).

Eine andere Ansicht, die Sie unbedingt brauchen, ist das “Terminal”. Damit können Sie innerhalb von VSCode auf die Kommandozeile (Bash o.ä. unter macOS/Linux bzw. PowerShell unter Windows) zugreifen. Die Terminal-View können Sie unter dem Menü öffnen. Wenn Sie später ein Projekt bearbeiten wird direkt das richtige Verzeichnis gewählt.

3.1 Ausführen eines Beispielprogramms

Erstellen Sie eine neue Datei mittels . Am besten speichern Sie die (noch leere) Datei gleich mal ab, bspw. unter `beispiel.js`. Jetzt weiß VSCode, dass es sich um eine JavaScript-Datei handelt und wird die entsprechenden Tools aktivieren.

Hier ist mal ein ganz einfaches Beispiel (die Zeilenumbrüche sind mit Absicht so gesetzt, um das Debuggen zu erleichtern):

```
for (let i=0;
  i < 10;
  i++) {
  console.log(i);
}
```

Speichern Sie ab und wählen Sie im Menü . Es geht dann (beim ersten Mal) ein Dialog auf, in dem Sie das “Environment” wählen müssen. Wir wählen hier “Node.js” (vgl. Abbildung 5). **Achtung:** Diese Einstellung können wir später nicht mehr ändern.



Falls es zu Problemen kommt: Achten Sie darauf, dass die Datei tatsächlich die Endung “.js” hat! Falls Sie Node.js quasi gleichzeitig mit VSCode installiert haben, kann es sein, dass Sie VSCode noch einmal beenden und neu starten musst, damit Node.js von VSCode auch wirklich gefunden wird.

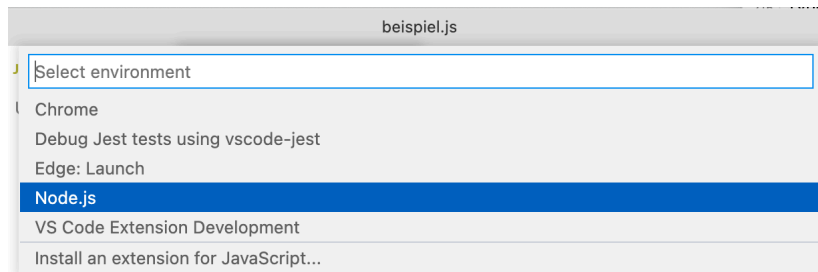


Abbildung 5: Auswahl des Environments zum Starten des JavaScript-Codes

In der Debug-Console sehen Sie dann die Ausgabe.

Setzen Sie jetzt einen Breakpoint, in dem Sie links neben den Zeilennummern einen roten Knopf setzen. Starten Sie erneut und spielen Sie ein wenig mit dem Debugger herum.



Wir werden später Programme niemals so starten, sondern meist Tests schreiben und diese über Jest ausführen. Das ist dann wesentlich komfortabler.

3.2 Ausführen eines Skripts in HTML

Erstellen Sie eine weitere Datei `sample.html`. Geben Sie dort folgenden Code ein:

```
<html>
<body>
<h1>Hello</h1>
<script>
  alert("Hello");
  alert("World");
</script>
</body>
</html>
```

Wir werden wieder (nach dem Speichern) das “Programm” (also den Code, der im HTML-Dokument unter dem sogenannten Script-Tag versteckt ist) ausführen. Wieder machen wir das mit `Run` `Start Debugging`. Diesmal wählen wir allerdings “Chrome” aus.

Wenn Sie alles richtig gemacht haben, startet nun Chrome und öffnet die Seite. Es werden zwei Dialoge gezeigt, danach sieht man einfach nur die Seite (mit einem großem “Hello”).

Setzen Sie nun einen Breakpoint in Zeile 6 und starte erneut. Sie müssen den ersten Dialog mit "OK" bestätigen. Jetzt wird der Debugger (in VSCode) stehen bleiben. In Chrome wird angezeigt, dass die Seite noch lädt.

Wir haben hier eine sehr komfortable andere Möglichkeit kennen gelernt, um Web-Seiten zu debuggen. Statt den Code in Chrome zu debuggen, verwenden wir VSCode, das den Debugger innerhalb von Chrome fernsteuert. Man kann diese Fernsteuerung auch mit anderen Tools (mehr oder weniger manuell) einrichten, aber mit VSCode ist es extrem einfach!

Probieren Sie doch mal aus, den Code in Chrome direkt zu debuggen!

Natürlich werden die Projekte später wesentlich komplexer und wir können häufig nicht ganz so einfach den Code starten und debuggen. Wir müssen in Zukunft viel mehr definieren, bspw. woher Code aus Bibliotheken geladen wird und wie Node.js bzw. der Browser diesen Code findet. Dazu werden dann Build-Skripte und vor allem `npm` verwendet. Das sehen wir uns im nächsten Aufgabenblatt an.

Abkürzungsverzeichnis

API Application Programming Interface
REPL Read-Eval-Print Loop