jQuery

1 Einführung

Die Trennung von HTML (Struktur), Javascript (Verhalten) und CSS (Aussehen) in einer Webseite wird durch die Javascript-Bibliothek jQuery erleichtert. jQuery hat sich praktisch zu einem Standard in der clientseitigen Webprogrammierung entwickelt, wenn man kein Framework benutzen möchte. Außerdem bietet diese Bibliothek eine große Auswahl an zusätzlicher Funktionalität, um vielfältige Effekte mit einfachen Mitteln in eine Webseite einzubauen.
Weiters muss durch jQuery auf etwaige Browser-spezifische Lösungen nicht mehr Rücksicht genommen werden.
Zu den wichtigsten Funktionen gehören:

- Elementselektion
- DOM-Manipulation
- Events
- Effekte
- Animationen
- Ajax

Die Homepage lautet: https://jquery.com/. Dort findet man auch die Downloads.
Die aktuelle Version (Stand: 2022-04-20) ist 3.6.0. Dieses Dokument basiert auf dieser Version.

2 Visual Studio Code

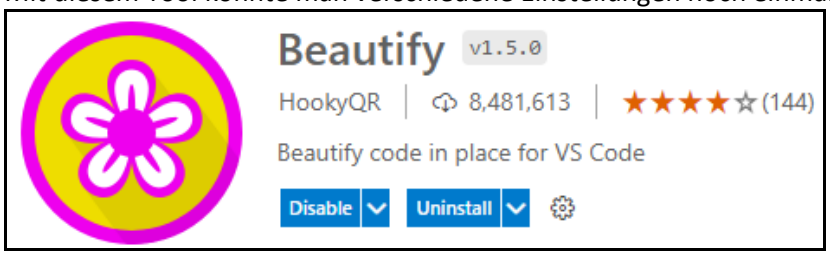
Zu Entwickeln von Web Projekten bietet sich **Visual Studio Code** an, das eine leichtgewichtige Variante von Visual Studio ist. Wie werden sie später noch bei der Entwicklung von Angular/Ionic benutzen.
Download: https://code.visualstudio.com/
Visual Studio Code kann man durch viele PlugIns entsprechend seinen Vorlieben erweitern, indem man diese mit dem „Extensions“-Symbol installiert.

2.1 Settings

Die Editor-Einstellungen können global bzw. pro Projekt eingestellt werden. Die Beschreibung dazu findet man unter https://code.visualstudio.com/docs/getstarted/settings.
Die globalen Einstellungen werden gespeichert in: **`C:\Users\Besitzer\AppData\Roaming\Code\User`**.
Diese Datei also bei Laptop-Wechsel berücksichtigen.
Es gibt eine Vielzahl an Plugins für die Entwicklung von HTML/Javascript, einige davon seien hier erwähnt.

2.1.1 Beautify

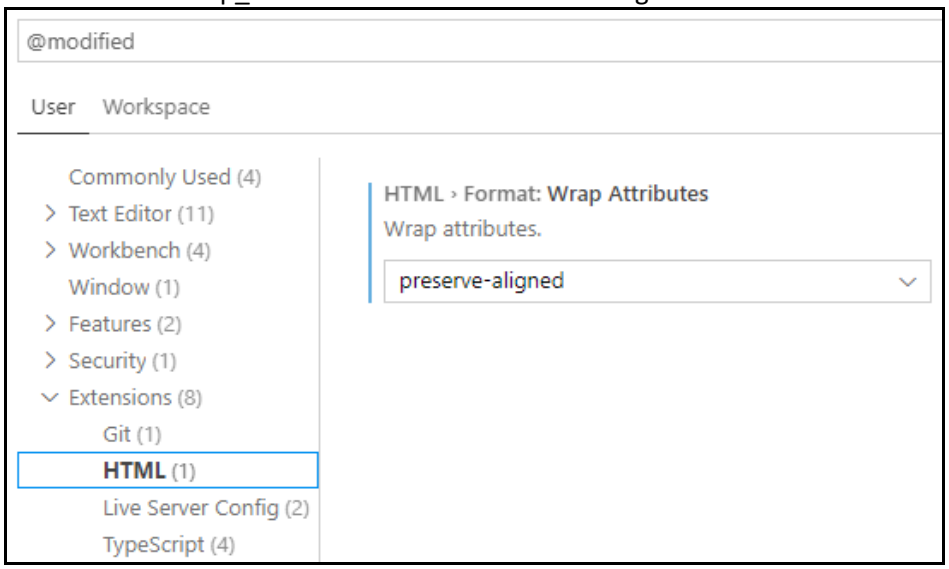
Mit diesem Tool könnte man verschiedene Einstellungen noch einmal zusammenfassen.



Beeinflusst die Formattierung von Javascript. Die Regeln werden in die Datei **`.jsbeautifyrc`** geschrieben.
Beispiel:

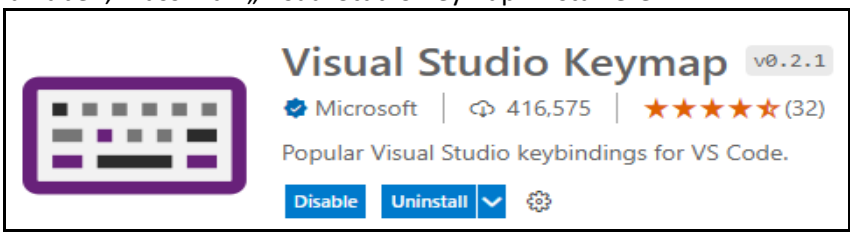
```
{
  "indent_size": 2,
  "indent_char": " ",
  "css": {
    "indent_size": 2
  },
  "wrap_attributes": "preserve-aligned",
  "brace_style": "end-expand,preserve-inline"
}
```

So steht etwa wrap_attributes auch in den Einstellungen von Visual Studio Code:



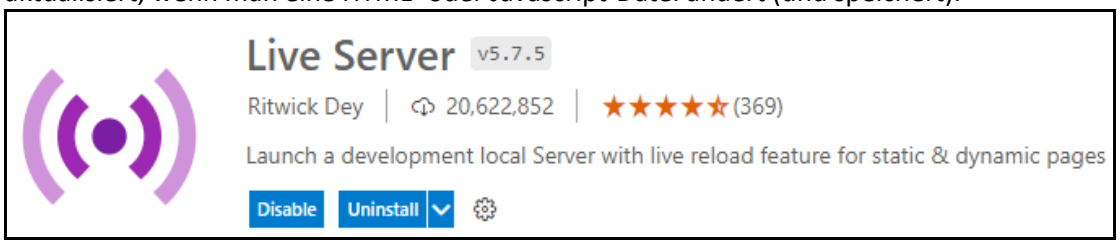
2.2 Keyboard shortcuts

Man kann sich verschiedenste Keyboard-Shortcuts einstellen. Um die von Visual Studio gewohnten Tastaturkürzel zu haben, muss man „Visual Studio Keymap“ installieren:



2.3 Live Server

Ähnlich wie mit Grunt oder Gulp (die jeweils unabhängig von der Entwicklungsplattform funktionieren) gibt es für Visual Studio Code eine Extension, mit der man einen Server starten kann, der den Browser automatisch aktualisiert, wenn man eine HTML- oder Javascript-Datei ändert (und speichert).



Wenn man Visual Studio Code danach neu startet, hat man **bei einer HTML**-Datei in der Statusleiste unten dann einen Eintrag „Go Live“. Ein Klick darauf öffnet die Seite im Standardbrowser.

2.4 Auto Add Brackets

3 Hello World

Zuerst muss jQuery installiert werden.

3.1 Installation mit npm/yarn

Es gibt grundsätzlich zwei Möglichkeiten, wie man jQuery installiert (siehe https://jquery.com/download/). Entweder man kopiert die entsprechende Datei oder man installiert sie mit **npm** (Node Package Manager) bzw. **yarn**, was letztlich auch nur ein lokales Kopieren ist.

Downloading jQuery using npm or Yarn

jQuery is registered as a package on npm. You can install the latest version of jQuery with the npm CLI command:

```
npm install jquery
```

As an alternative you can use the Yarn CLI command:

```
yarn add jquery
```

This will install jQuery in the `node_modules` directory. Within `node_modules/jquery/dist/` you will find an uncompressed release, a compressed release, and a map file.

3.1.1 package.json

Durch die Installation entsteht ein entsprechender Eintrag in package.json:

package.json > ...

```
{
  "dependencies": {
    "jquery": "^3.6.0"
  }
}
```

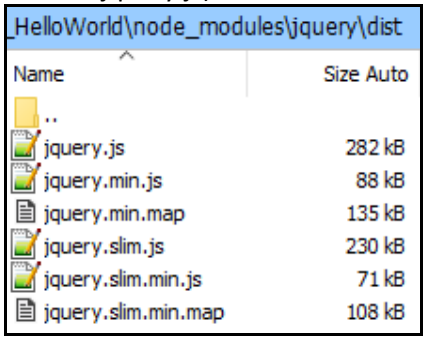
Allgemein werden die Version in der Form `major.minor.patch` angegeben. Das nennt man semantic versioning.
Der String „^3.6.0“ heißt dabei: installiere die aktuellste Version mit Major Version 3.
Allgemein (https://medium.com/att-israel/npm-versions-explained-60e4d6b9920f):

- **Caret (^)** — a caret is the default prefix you get from npm after installing a new package. It gives you the highest minor version available with its highest patch version.
- **Tilde (~)** — a tilde prefix will only promote patch versions, meaning that you'll get the highest patch version for your current minor.

Zum Ausprobieren: „semantic version calculator“ unter https://semver.npmjs.com/

3.1.2 node_modules

Dabei ist dann die Verzeichnisstruktur folgende (wie man sieht fehlt die Versionsnummer, sie findet man ganz oben in jquery.js):



Um jQuery verwenden zu können, muss eine der js-Dateien **`jquery[.slim][.min].js`** referenziert werden.

- Die **.min**-Variante ist eine kompakte Version von jQuery - mit voller Funktionalität aber deutlich kleiner.
- Die **.slim**-Variante enthält die Module für Ajax und Effekte nicht und ist daher auch etwas kleiner.

3.2 Html

Eine HTML-Seite mit jQuery sieht so aus und unterscheidet sich praktisch nicht von einer normalen Webseite:

```
<html>

<head>
  <link rel=stylesheet type="text/css" href="testerl.css">
  <meta charset="utf-8" />
  <title>Hello jQuery</title>
  <script src="node_modules/jquery/dist/jquery.js"></script>
  <script src="testerl.js"></script>
</head>

<body>
  <h1>jQuery</h1>
</body>

</html>
```

Es muss lediglich das jQuery-File referenziert werden, und zwar vor dem eigenen Javascript-Code.
Wichtig:

- Die Webseite enthält keinerlei Javascript-Code mehr (weder Implementierung noch Registrierung von Eventhandlern). Dieser wurde auf die Datei testerl.js ausgelagert, die über einen script-Tag eingebunden wird. **Davor** muss die jQuery-library eingebunden werden.
- Ob die Scripts am Ende des <head> oder am Ende von <body> referenziert werden, ist grundsätzlich egal.
- <style>-Tags kommen im HTML nicht vor, sie werden in CSS-Files ausgelagert. Diese Stylesheets werden **vor** den script-Tags notiert.

Es muss also eine vollständige Trennung von Programmcode, Styles und HTML erreicht werden.

3.3 Javascript

Ähnlich dem **window.onload**-Event von Javascript muss auf ein Event gewartet werden, das ausgelöst wird, wenn der DOM vollständig geladen und jQuery initialisiert wurde. Dieses Event heißt **$(document).ready()**, als Parameter muss eine Funktion angegeben werden, die bei Auftreten des Events ausgelöst wird:

```
$(document).ready(() => {
    console.log('jQuery ready');
});
```

Man könnt das auch kürzer schreiben, das wird aber eher nicht empfohlen:

```
$(() => {
    console.log('jQuery ready');
});
```

Wie man sieht, ruft man mit dem Symbol **$** jQuery-Funktionen auf.

4 ESLint

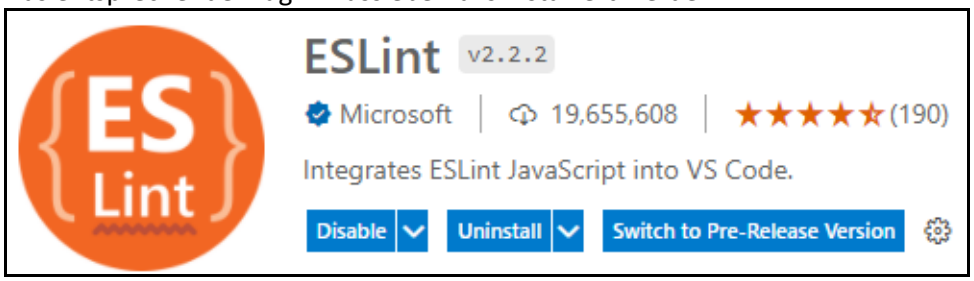
Definition von Lint (https://stackoverflow.com/questions/8503559/what-is-linting):
„**Linting** is the process of checking the source code for Programmatic as well as Stylistic errors. This is most helpful in identifying some common and uncommon mistakes that are made during coding.“.
Für Javascript gibt es für Linting das Node-Modul **`eslint`**. Am besten global installieren, also **`npm install -g eslint@7`**.
Achtung: am besten eslint in der Version 7 installieren, damit globale Rule-Files funktionieren (siehe weiter unten)!
Überprüfen der Installation:

```
C:\Users\Besitzer>eslint -v
v7.32.0
```

Ab der Version 8 scheinen gloable Konfigurations-Files in Visual Studio Code nicht mehr zu funktionieren (siehe weiter unten). Bei mir funktioniert es zumindest mit Version 7 noch.

4.1 Plugin

Das entsprechende Plugin muss ebenfalls installiert werden:



4.2 Regeln

Die Einstellungen werden in einem File gespeichert und können unter http://eslint.org/docs/rules/ nachgeschlagen werden.

4.2.1 .eslintrc.json

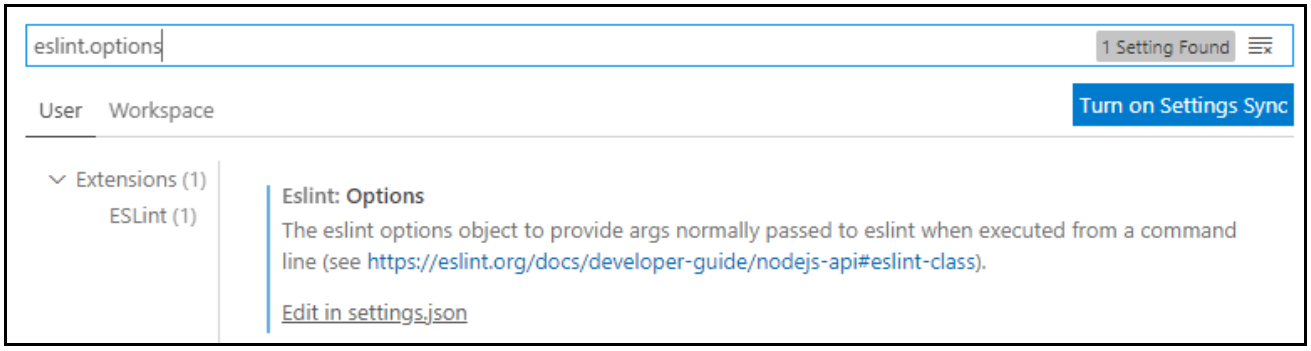
Die Datei, in dem die Regeln gesucht werden, muss **`.eslintrc.json`** heißen.
Diese Datei mit jenen Regeln, die ich benutze, findet ihr in eduvidual.
Man kann sich aber auch ein neues Rule-File mit **`eslint --init`** bzw. **`npm init @eslint/config`** erstellen lassen – dabei muss man ein paar Optionen auswählen.
Sowohl von der Commandline als auch in Visual Studio Code wird jenes Rule-File verwendet, das im lokalen Verzeichnis des Projekts liegt.
Ist kein lokales Rule-File vorhanden, wird das auf der Commandline und in Visual Studio Code unterschiedlich behandelt.

4.2.1.1 Commandline

Wird lokal kein Rule-File gefunden, wird im Elternverzeichnis gesucht, dann in dessen Elternverzeichnis usw. Das erste auf dem Weg zum root-Verzeichnis wird dann angewandt. Da z.B. bei mir alle Projekte unterhalb von C:_PR\CSharp gespeichert sind, habe ich die Regeln in der Datei **`C:\_PR\CSharp\.eslintrc.json`** gelagert. Das gilt aber leider nur, wenn man eslint von der Konsole ausführt.

4.2.1.2 Visual Studio Code

Für Visual Studio Code ist dieses Feature leider ab eslint v8.x nicht mehr verfügbar (daher habe ich oben auch eslint 7 installiert). Man muss also die Datei grundsätzlich beim Projekt speichern.
Man kann bei File → Preferences → Settings aber eine globales Rule-File angeben, indem man die Datei **`C:\Users\`*`{myUser}`*`\settings.json`** editiert:



Dort am Ende folgende Zeile (mit geändertem Pfad) einfügen:

```
"eslint.options": { "configFile": "C:\\_PR\\CSharp\\.eslintrc.json" }
```

Ändert man in der angegebenen global Datei .eslintrc.json Regeln, werden diese erst nach erneutem Öffnen einer Javascript-Datei übernommen.

4.2.2 rules

Die Regeln sehen z.B. so aus:

```
"rules": {
  "eqeqeq": 2,
  "comma-dangle": [2, {
    "arrays": "only-multiline",
    "objects": "only-multiline"
  }],
  "no-console": 0,
  "no-debugger": 1,
  "no-extra-semi": 1,
  "no-extra-parens": 1,
  "no-irregular-whitespace": 0,
  "no-undef": 2,
  "no-unused-vars": 0,
  "semi": 1
```

Die Konfiguration ist in viele Einzelprüfungen aufgeteilt, die mit Werten 0, 1 oder 2 bzw. „off“, „warn“ oder „err“ eingestellt werden:

0	"off"	Regel ignorieren
1	"warn"	Regel liefert eine Warnung
2	"err"	Regel liefert einen entsprechenden Fehler

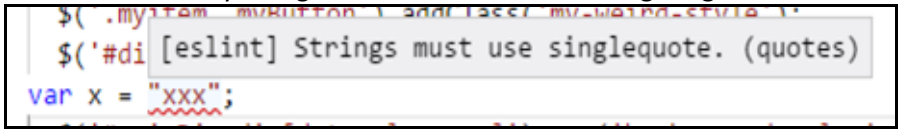
Fügt man z.B. in .eslintrc.json folgende Zeile für „quotes“ ein:

```
"no-unused-vars": 0,
"quotes": [ 2, "single" ],
"semi": 1,
```

so werden Strings mit doppelten Anführungszeichen als Fehler angezeigt.

4.3 Fehleranzeige

Danach werden Style-Regeln in Visual Studio Code angezeigt:



Auch von der Commandline kann man die Überprüfung ausführen:

```
C:\_PR\JS\jQuery_Ajax\01_HelloWorld>eslint tester1.js

C:\_PR\JS\jQuery_Ajax\01_HelloWorld\tester1.js
  4:3   error  Unexpected var, use let or const instead  no-var
  4:13  error  Strings must use singlequote              quotes

✖ 2 problems (2 errors, 0 warnings)
  2 errors, 0 warnings potentially fixable with the `--fix` option.

(node:14672) DeprecationWarning: [eslint] The 'ecmaFeatures' config
(node:14672) DeprecationWarning: [eslint] The 'ecmaFeatures' config

C:\_PR\JS\jQuery_Ajax\01_HelloWorld>
```

Oder evtl. auch mit absolutem Pfad oder Flag **`--debug`**:

```
C:\_PR\CSharp\Dummies\jQueryDemo>eslint tester1.js --debug
  eslint:cli CLI args: [ 'tester1.js', '--debug' ] +0ms
  eslint:cli Running on files +3ms
  eslintrc:config-array-factory Loading JSON config file: C:\_PR\CSharp\Dummies\jQueryDemo\package.json +0ms
  eslintrc:ignore-pattern Create with: [ IgnorePattern { patterns: [ '/**/node_modules/*' ], basePath: 'C:\\_PR\\CSharp\\Dummies\\jQuery
  eslintrc:ignore-pattern   processed: { basePath: 'C:\\_PR\\CSharp\\Dummies\\jQueryDemo', patterns: [ '/**/node_modules/*' ] } +2ms
  eslintrc:ignore-pattern Create with: [ IgnorePattern { patterns: [ '/**/node_modules/*' ], basePath: 'C:\\_PR\\CSharp\\Dummies\\jQuery
  eslintrc:ignore-pattern   processed: { basePath: 'C:\\_PR\\CSharp\\Dummies\\jQueryDemo', patterns: [ '/**/node_modules/*' ] } +1ms
  eslint:file-enumerator Start to iterate files: [ 'tester1.js' ] +0ms
  eslint:file-enumerator File: C:\_PR\CSharp\Dummies\jQueryDemo\tester1.js +0ms
  eslintrc:cascading-config-array-factory Load config files for C:\_PR\CSharp\Dummies\jQueryDemo. +0ms
  eslintrc:cascading-config-array-factory No cache found: C:\_PR\CSharp\Dummies\jQueryDemo. +1ms
  eslintrc:config-array-factory Loading JSON config file: C:\_PR\CSharp\Dummies\jQueryDemo\.eslintrc.json +8ms
  eslintrc:config-array-factory Config file found: C:\_PR\CSharp\Dummies\jQueryDemo\.eslintrc.json +1ms
  eslintrc:config-array-factory Loading {extends:"eslint:recommended"} relative to C:\_PR\CSharp\Dummies\jQueryDemo\.eslintrc.json +0ms
  eslintrc:cascading-config-array-factory No cache found: C:\_PR\CSharp\Dummies. +3ms
  eslintrc:config-array-factory Config file not found on C:\_PR\CSharp\Dummies +1ms
  eslintrc:cascading-config-array-factory No cache found: C:\_PR\CSharp. +1ms
  eslintrc:config-array-factory Loading JSON config file: C:\_PR\CSharp\.eslintrc.json +1ms
  eslintrc:config-array-factory Config file found: C:\_PR\CSharp\.eslintrc.json +1ms
  eslintrc:config-array-factory Loading {extends:"eslint:recommended"} relative to C:\_PR\CSharp\.eslintrc.json +0ms
  eslintrc:cascading-config-array-factory No cache found: C:\_PR. +1ms
  eslintrc:cascading-config-array-factory Config file not found on C:\_PR +1ms
```

Hier sieht man auch, dass die Datei .eslintrc.json zuerst lokal gesucht wird und dann im Pfad nach oben gefunden wird.

4.4 fix

Praktisch ist auch die Option **`--fix`**, die automatisch alle Änderung entsprechend den eingestellten Regeln ausführt.

```
C:\_PR\JS\jQuery_Ajax\01_HelloWorld>eslint tester1.js --fix
(node:13548) DeprecationWarning: [eslint] The 'ecmaFeatures' c
(node:13548) DeprecationWarning: [eslint] The 'ecmaFeatures' c

C:\_PR\JS\jQuery_Ajax\01_HelloWorld>
```

5 Element-Auswahl

In Javascript gibt es bereits einige Selektoren wie **`document.getElementById()`**. In jQuery gibt es eine Vielzahl von Selektoren, die eines oder mehrere HTML-Elemente auswählen. Diese werden als jQuery-Objekte zurückgegeben und können bearbeitet werden. Die Selektoren sind dabei stark an die CSS-Selektoren angelehnt und sind in jQuery sind so aufgebaut, dass **automatisch** über alle betroffenen Elemente **iteriert** wird, wenn der Ausdruck mehrere HTML-Elemente betrifft. Damit kann man z.B. sehr einfach alle Elemente mit demselben Klassen-Attribut in einer einzigen Codezeile bearbeiten.

5.1 Einführungsbeispiel

Das soll mit folgender Seite veranschaulicht werden:

```
<h1>jQuery</h1>
<p>Message: <span id="msg">Logs come here</span></p>
<input type="button" id="btnA" value="button A" />
<input type="button" class="myButton" value="button B" />
<input type="button" class="myButton" value="button C" />
<input type="button" id="btnOneClickOnly" value="One Click Only" />
<hr />
<ul>
    <li>Item A</li>
    <li class="myItem">Item B</li>
    <li class="myItem">Item C</li>
    <li>Item D</li>
</ul>
<hr />
<div id="divSpans">
    Das ist ein <span>Text</span> mit einem <span>Span</span> und noch einem <span>Span</span>.
</div>
```

Folgende Styles sind in **`testerl.css`** definiert:

```
.border {
    border-style: solid;
    border-color: black;
    border-width: 1px;
    background-color: lightgray;
}

.mySpan {
    background-color: blue;
    color: white;
}
```

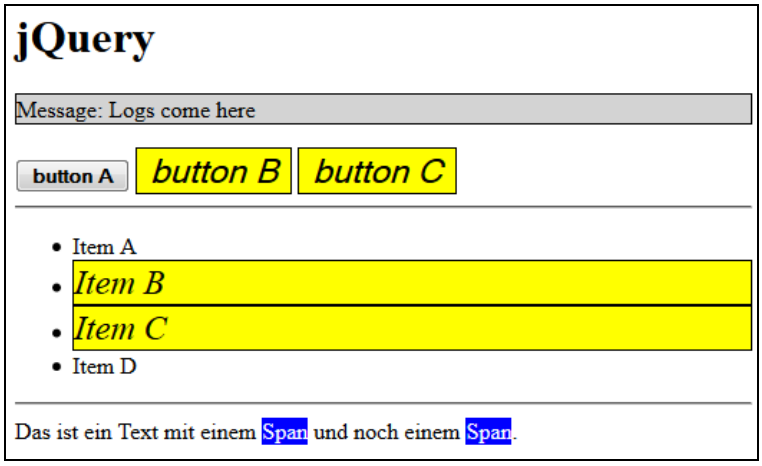
```
.special-button {
    font-weight: bold;
}

.my-weird-style {
    background-color: yellow;
    font-style: italic;
    font-size: 1.5em;
    border: 1px solid black;
}
```

Mit jQuery kann man jetzt auf vielfältige Weise einzelnen oder mehrere Element auswählen:

```
$(document).ready(() => {
    console.log('jQuery ready');
    $('#btnA').addClass('special-button');
    $('p').addClass('border');
    $('.myitem,.myButton').addClass('my-weird-style');
    $('#divSpans span').addClass('mySpan');
});
```

Damit sieht die Seite so aus:



Also:

`#btnA`	Wählt ein einzelnes Element mit ID **`btnA`** aus
`p`	Wählt alle **<p>**-Elemente aus
`.myitem,.myButton`	Wählt alle Elemente aus, die als Klassenattribut entweder **`myitem`** oder **`myButton`** haben (ist nicht case-sensitiv)
`#divSpans span`	Selektiert alle -Elemente, die unterhalb eines Elements mit ID **`divSpans`** im DOM hängen

Nachdem über die **Selektoren** festgelegt wurde, welche Elemente betroffen sind, werden diese über eine der vielen „DOM manipulation methods“ (siehe weiter unten) verändert. Ein der wichtigsten dieser Methoden ist **`addClass`**, die dem Element das entsprechende Klassenattribut zuweist. Als Folge davon wird dieses Element dann mit den Einstellungen dargestellt, die im Stylesheet für diese Klasse konfiguriert wurden. Mit **`removeClass`** werden diese Einstellungen wieder vom Element entfernt.

5.2 Simple CSS

Die wichtigsten CSS-Regeln sind vermutlich bekannt:

`*`	All elements.
`#id`	The element with the given ID.
`element`	All elements of the given type.
`.class`	All elements with the given class.
`a, b`	Elements that are matched by a or b.
`a b`	Elements b that are descendants of a.
`a > b`	Elements b that are children of a.
`a + b`	Elements b that immediately follow a.
`a ~ b`	Elements b that are siblings of a and follow a.

Neben den obigen Beispielen bietet jQuery noch viele weitere Möglichkeiten, bestimmte Elemente auszuwählen. So kann man z.B. auf das erste -Element im DOM so zugreifen: **`$('li:eq(0)').addClass('mySpan');`**

5.3 Eventhandler

Die erste Funktion, die bei jQuery aufgerufen wird, ist immer **`$(document).ready()`**. Sie wird aufgerufen, wenn der komplette DOM-Tree geladen wurde und daher Javascript alle HTML-Elemente zur Verfügung stehen.

5.3.1 ready()

Diese Funktion entspricht in etwa der Methode ***Loaded*** in WPF-Applikationen.
Als Argument wird der ready-Funktion eine Funktion übergeben, die üblicherweise als **anonyme Methode** realisiert wird. Die Syntax lautet:
`function() { … }` bzw. **`() =>{…}`**
Innerhalb dieser Funktion sollte man den gesamten jQuery-Code der Webseite schreiben. Aus Gründen der Übersichtlichkeit sollte man aber auch hier umfangreicheren Code in einzelne Methoden auslagern, aber eben innerhalb dieser ready-Funktion.

5.3.2 Registrierung

Die Registrierung hat dabei folgende Syntax:
`$(selector).on('eventname',function);`

selector	Bestimmt, welches HTML-Element an ein Event gebunden werden soll. Die Angabe erfolgt dabei über einen CSS-Selektor-Ausdruck, der in Anführungszeichen stehen muss.
eventname	Gibt an, welches Event behandelt werden soll. Der Eventname entspricht dabei jenem aus normalem Javascript, jedoch ohne einleitendes „on“. Das „onchange“-Event wird also mit Eventname `'change'` gebunden.
function	Funktion, die beim Auftreten des Events aufgerufen wird. Auch hier erfolgt die Implementierung wieder meist als anonyme Methode.

Beispiele:

```
const lblMsg = $('#msg');
$('#btnA').on('click', ev => lblMsg.html('Button A clicked'));
$('.myButton').on('click', _ => lblMsg.html('Some Button clicked'));
$('.myButton:last').on('click', event => {
  console.dir(event);
  lblMsg.html(`Button clicked - ${event.target.value}`);
});
```

5.3.3 target

Der Callback-Funktion wird ein Event-Objekt mitgegeben, in dem viele Infos gespeichert werden, eines davon ist **`target`**, das dem auslösenden HTML-Element entspricht. Alle Properties:
http://api.jquery.com/category/events/event-object/
Für andere Events (Maus, Keyboard,…) funktioniert es analog:

```
$('span').on('mouseenter', ev => $(ev.target).removeClass('mySpan'))
         .on('mouseleave', ev => $(ev.target).addClass('mySpan'))
```

Hinweise:

- wie man sieht, ist bei den meisten Funktionen in jQuery MethodChaining implementiert, d.h. die Funktionen geben das jQuery-Objekt (also **`this`**) wieder zurück.
- Aus einem beliebigen HTML-Element kann man mit **`$(…)`** ein jQuery-Objekt erzeugen

5.3.4 Eventhandler entfernen

Mit der Funktion **`off()`** kann man einen Eventhandler wieder entfernen.

```
$('#btnOneClickOnly').on('click',
  ev => {
    const btn = $(ev.target);
    alert(`${btn.val()} clicked`);
    btn.off('click');
  });
```

6 Wichtige Funktionen

6.1 val/html/text

Oft muss man mit jQuery auf den Inhalt von Elementen zugreifen. Daher werden folgende Funktionen sehr oft benötigt:

.val()	Inhalt eines <input>, also das `value`-Attribut
.html()	Text-Inhalt eines beliebigen Elements (entspricht also `innerHTML` von Javascript)
.text()	Entspricht praktisch html()

Beispiel:

```
<input type="text" id="txtA" value="Hallo Hansi" />
<input type="button" id="btnX" value="Click Me" />
<select id="cboName">
    <option>Susi</option>
    <option>Fritzi</option>
    <option>Pauli</option>
</select>
<div id="divX">Das ist mein DIV</div>
```

Zugriff auf die Elemente:

```
console.log('txtA          = ' + $('#txtA').val());
$('#txtA').val('Quaxi');
console.log('txtA          = ' + $('#txtA').val());
console.log('btnX          = ' + $('#btnX').val());
console.log('cboName       = ' + $('#cboName').val());
console.log('allOptionText = ' + $('option').text());
console.log('divX          = ' + $('#divX').html());
```

```
txtA          = Hallo Hansi
txtA          = Quaxi
btnX          = Click Me
cboName       = Pauli
allOptionText = SusiFritziPauli
divX          = Das ist mein DIV
```

6.2 Ein- u. Ausblenden

Möchte man die einzelnen Elemente ein- bzw. ausblenden, gibt es auch entsprechende Funktionen in jQuery:

- **show()**/**hide()**
- **fadeIn()**/**fadeOut()**
- **slideDown()**/**slideUp()**

Alle diese Funktionen können auf einzelne oder mehrere jQuery-Objekte angewendet werden (also auf das Ergebnis eines Selektors). Wahlweise kann ein Parameter mitgegeben werden, der die Dauer des Effekts in Millisekunden beschreibt (es gibt auch zwei String-Konstanten: "fast" für 200ms, "slow" für 600ms).
Wichtig: dabei muss das HTML-File nicht mehr verändert werden, es wird alles innerhalb des Javascript-Files programmiert.

```
<button id="btnHide">hide</button>
<ul>
  <li>aaa</li>
  <li class="toHide">bbb</li>
  <li>ccc</li>
  <li>ddd</li>
  <li class="toHide">eee</li>
  <li>fff</li>
</ul>
```

```
$('#btnHide').on('click', ev => $('.toHide').hide());
```

- aaa
- bbb
- ccc
- ddd
- eee
- fff

- aaa
- ccc
- ddd
- fff

6.3 css(): Properties setzen

Anstatt Elementen eine neue Klasse zuzuordnen, kann man auch direkt Properties auf bestimmte Werte setzen. Die Methode dazu heißt **`css`** und ist ähnlich anzuwenden wie **`addClass`**, nur dass die Properties und deren Werte als Parameter übergeben werden müssen.
Bei der Funktion **`css()`** gibt es zwei Varianten:

- Man führt die Funktion mehrmals aus (ein Mal pro Property), property und Wert werden dabei als String durch Beistrich getrennt angeführt

```
$('#mainDiv div:nth-child(3)').css('background-color', 'red');
```

- Man übergibt eine JSON-Map innerhalb geschwungener Klammern Parameter/Wert-Paare an. Parameter und Wert sind durch Doppelpunkt getrennt, diese Paare dann wieder durch Beistrich.

```
$('#mainDiv div:nth-child(3)').css({ backgroundColor: 'red', color: 'white', fontWeight: 'bold' });
```

6.3.1 animate: Properties animieren

`animate()` funktioniert praktisch wie **`css()`**, jedoch erfolgt die Anwendung der Properties nicht sofort, sondern wird innerhalb einer bestimmten Zeitdauer durchgeführt.

```
$('#btnAnimate').on('click',
    ev => $('#lblAnimate').animate({
        fontSize: '48px'
    }, 4000));
```

Der zweite Parameter gibt die Dauer in Millisekunden an, in der die neuen Eigenschaften angewandt werden.

7 jQuery DOM

Mit jQuery können Elemente sehr einfach innerhalb des DOM-Trees

- verschoben,
- kopiert,
- eingefügt oder
- entfernt

werden. Das ist mir reinem Javascript ja eher umständlich. Mit den unten angeführten Funktionen geht es deutlich einfacher. Die Beschreibungen zu den folgenden Funktionen sind aus „jQuery Pocket Reference“ entnommen.

7.1 Erzeugen

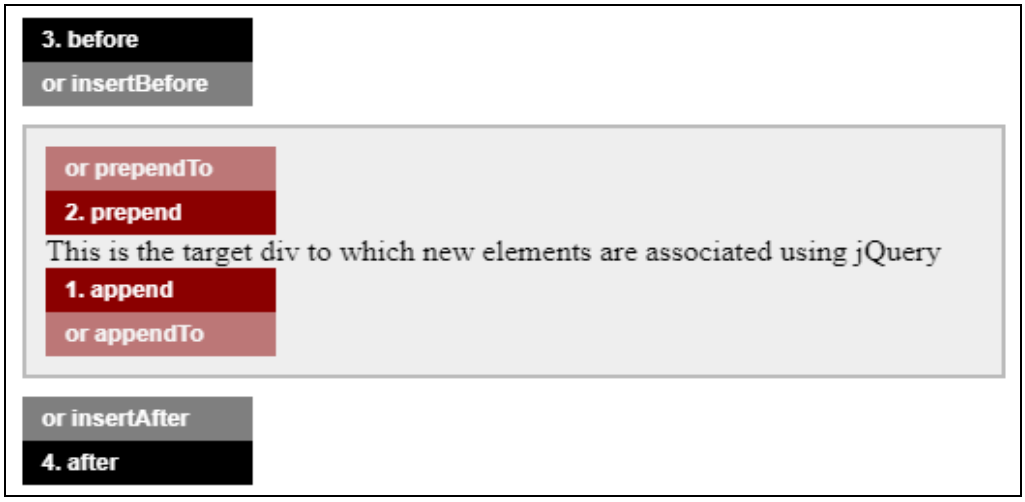
Mit `$()` können neue Elemente erzeugt werden.
Beispiel: `$('<td>')`
Häufiger Fehler: Man vergisst die spitzen Klammern, also z.B. `$('td')`. Damit werden aber alle bereits existierenden <td>-Elemente ausgewählt. `$('td').html('xxx')` erzeugt also KEIN neues <td> sondern verändert alle bestehenden <td>s.

7.2 Einfügen

Zum Einhängen von neuen Elementen in den DOM gibt es viele Möglichkeiten:

append(*content*)	Append *content* to each selected element, or invoke *f* as a method of—and append its return value to—each selected element.
appendTo(*target*)	Appends the selected elements to the end of each specified *target* element, cloning them as necessary if there is more than one target.
prepend(*content*)	Like append(), but insert content at the beginning of each selected element.
prependTo(*target*)	Like appendTo(), except that the selected elements are inserted at the beginning of the target elements.
after(*content*)	Insert *content* after each selected element.
insertAfter(*target*)	Inserts the selected elements after each *target* element, cloning them as necessary if there is more than one target.
before(*content*)	Like after(), but make insertions before the selected elements.
insertBefore(*target*)	Inserts the selected elements before each *target* element, cloning them as necessary if there is more than one target.
wrap(*wrapper*)	Wrap *wrapper* around each selected element, cloning as needed if there is more than one selected element. The *wrapper* may be an element, a jQuery object, a selector, or a string of HTML, but it must have a single innermost element.
wrapAll(*wrapper*)	Wraps *wrapper* around the selected elements as a group by inserting wrapper at the location of the first selected element and then copying all selected elements into the innermost element of *wrapper*.
wrapInner(*wrapper*)	Like wrap(), but inserts *wrapper* around the content of each selected element rather than around the elements themselves.
clone([*data*=false])	Makes a deep copy of each of the selected elements and returns a new jQuery object representing the cloned elements. If *data* is true, also clones the data (including event handlers) associated with the selected elements.

Den Unterschied zw. append/prepend bzw. before/after sieht kann man sich so vorstellen (siehe http://stackoverflow.com/questions/14846506/append-prepend-after-and-before):



So könnte man z.B. eine Liste relativ einfach erzeugen:

```
const ul = $('<ul>').appendTo('body');
for (let i = 1; i <= 5; i++) {
  $('<li>').html(`item_${i}`).appendTo(ul);
}
```

7.3 Ersetzen

`html():string` **`html(`*`htmlText`*`)`**	With **no arguments**, **return** the content of the first selected element as an HTML-formatted string. With **one argument**, **set** the content of all selected elements to the specified *htmlText*.
`text():string` **`text(`*`plainText`*`)`**	With no arguments, return the content of the first selected element as a plain-text string. With one argument, set the content of all selected elements to the specified *plainText*.
`replaceAll(`*`target`*`)`	Inserts the selected elements into the document so that they replace each *target* element, cloning the selected elements as needed if there is more than one target.
`replaceWith(`*`content`*`)`	Replace each selected element with *content*.

7.4 Entfernen

`remove([`*`sel`*`])`	Removes all selected elements, or all selected elements that also match *sel*, from the document, as well as any data (including event handlers) associated with them. Note that the removed elements are no longer part of the document.
`empty()`	Deletes the content of all selected elements.

8 Ajax

Ajax steht für **A**synchronous **J**avaScript **A**nd **X**ML.

Asynchron	Der Client wartet nicht auf die Antwort vom Server sondern setzt nur den Request ab und lässt sich mit einem Callback über die Antwort benachrichtigen
JavaScript	Clientseitig wird Ajax mit JavaScript programmiert.
XML	Die Antwort muss in einem bestimmten Format übermittelt werden. Üblicherweise ist das XML, eine andere Möglichkeit ist JSON. AJAX könnte als auch AJAJ heißen… Das Format der Antwort beeinflusst aber das grundlegende Prinzip nicht.

Ajax ist eine Form der Kommunikation zw. einer Webseite (Client) und einem WebServer.
Man möchte vermeiden, dass bei kleinen Änderungen in einer Webseite diese wieder vollständig geladen werden muss. Besser wäre es, nur jene **Teile neu zu laden**, die sich geändert haben.
Wie bei Client-Server-Anwendungen üblich wird ein Request vom Client an den Server geschickt, der diesen bearbeitet. Wenn der Server mit der Bearbeitung fertig ist, wird ein Response vom Server an den Client geschickt, der diesen in seine HTML-Seite „einbaut“ und entsprechend anzeigt.
Da die Bearbeitung der Anfrage teilweise lange dauern kann (viele DB-Zugriffe etc.), ist es erstrebenswert, dass der Client während der Serveraktivität **nicht blockiert**, sondern normal weiterarbeitet und die Antwort genau dann behandelt, wenn er sie vom Server erhalten hat. Mit anderen Worten: die Kommunikation läuft **asynchron**.
Wichtig: Ajax beschreibt **nur die Kommunikation**. Es spielt also keine Rolle, ob die Server-Anwendung mit php, Asp.NET, Java EE,… programmiert wird.

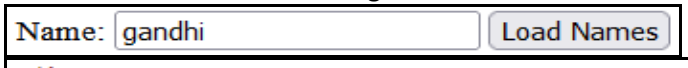
8.1 Plain Javascript – fetch API

Es gibt viele Möglichkeiten, Ajax in Javascript umzusetzen. Die ursprüngliche Variante der Ajax-Kommunikation mit Javascript ist aber ziemlich umständlich (man braucht eine Referenz auf ein **`HttpRequest`**-Objekt und muss im Event **`onreadystatechange`** auf einen bestimmten Status warten).
Da es mehrere Möglichkeiten gibt, diese Vorgehensweise mit anderen Interfaces zu umgehen, wird das nicht mehr weiter besprochen.
Ajax ist jedoch über das fetch-API jetzt deutlich einfacher zu verwenden. Es besteht aus der Methode **`fetch`**(), die ein Promise zurückgibt. Dabei ist es egal ob man Daten mit GET liest oder mit POST/PUT/... verändert.
Das API wird auf der Homepage https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch relativ gut beschrieben.

8.2 jQuery - $.getJSON()

Aber auch in jQuery ist Ajax sehr gut integriert. Es gibt mehrere Funktionen, mit denen man Ajax-Calls absetzen kann. Der Unterschied der verschiedenen Funktionen besteht hauptsächlich darin, in welcher Form die Daten zurückgeliefert und damit auch wie diese am Client verarbeitet werden.
Zuerst soll ein reiner GET-Request in seiner einfachsten Form besprochen werden.
Die meisten dieser Funktionen sind **globale Funktionen**, beziehen sich also nicht auf ein bestimmtes jQuery-Objekt. Daher ist die Schreibweise wie bei statischen C#-Funktionen, nämlich z.B. **`$.getJSON()`**.

Die Funktionsweise soll mit folgender kleiner Webseite veranschaulicht werden:

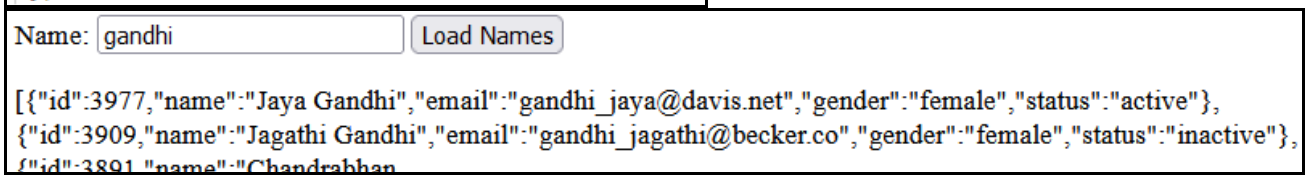



```
<div>
  Name: <input type="text" id="txtName" value="gandhi" />
  <button id="btnLoadNames">Load Names</button>
  <ul id="lstNames">
  </ul>
  <div id="divResult" />
</div>
```

In der einfachsten Variante eines Ajax-Calls gibt man nur das Ziel der Anfrage an und hängt die Parameter als Querystring an. Der Aufruf erfolgt durch die Funktion **`$.getJSON()`**.

Diese benötigt also die URL des Ziels des Aufrufs (incl. Querystring) und **`then()`** mit dem Delegate, das die Antwort behandeln soll. Diese Funktion bekommt dann das erhaltene JSON-Objekt als Parameter. Der JavaScript-Teil könnte dann etwa so programmiert werden:

```
const baseUrl = 'https://gorest.co.in/public/v2';
const name = $('#txtName').val();
$.getJSON(`${baseUrl}/users?name=${name}`)
 .then(data => {
   console.table(data);
   $('#divResult').html(JSON.stringify(data));
 });
```



8.2.1 Promises

Promises sind eine Möglichkeit, nicht blockierenden Code zu schreiben. Tiefergehende Details werden hier nicht besprochen, sondern nur die grundlegendste Vorgangsweise.
Obiger Aufruf liefert ein sogenanntes Promise zurück. Promises ersetzen Callback-Parameter direkt im Funktionsaufruf (also **`$.getJSON(url, obj=>{…})`**). Man kann dabei mit Method-Chaining im Erfolgsfall bzw. auf einen Fehler reagieren. Außerdem wird damit das Einrücken nach rechts der Callbacks vermieden („Callback hell“, „Christmas tree coding“, „Pyramid of doom“).
Es gibt dabei drei Funktionen, um auf ein Promise zu reagieren:

```
$.getJSON(`${baseUrl}/users?name=${name}`)
  .then(data => {
    console.table(data);
    $('#divResult').html(JSON.stringify(data));
  })
  .fail(err => console.error('Ajax error', err))
  .always(_ => console.log('Ajax request finished'));
```

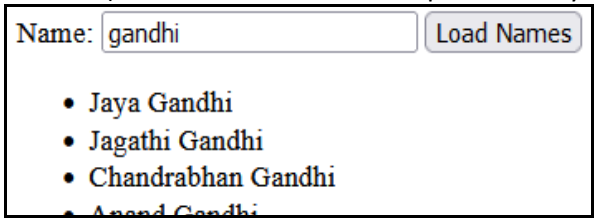
Die Funktionen haben folgende Bedeutung:

- **`always`**: wird sowohl im Erfolgs- als auch im Fehlerfall aufgerufen, wenn der Ajax-Call beendet wurde
- **`then/done`**: wird aufgerufen, wenn kein Fehler aufgetreten ist
- **`catch/fail`**: wird aufgerufen, wenn der Call nicht erfolgreich war

Wie man richtig vermuten würde (bzw. wie man oben gesehen hat), müssen nicht alle diese Funktionen immer implementiert werden.
Hinweis: Ein Fehler „404 – Not Found“ wird bei $.getJSON() nicht erkannt – siehe z.B. https://stackoverflow.com/questions/1002367/jquery-ajax-jsonp-ignores-a-timeout-and-doesnt-fire-the-error-event/5121811#5121811

8.2.2 DOM-Manipulation

Interessant ist Ajax vor allem dann, wenn man aus den JSON-Daten am Client eine vernünftige HTML-Struktur aufbaut (und evtl. noch mit CSS ansprechend stylt).



```
const lst = $('#lstNames');
lst.empty();
$.getJSON(`${baseUrl}/users?name=${name}`)
  .then(data => {
    console.table(data);
    data.forEach(x => $('<li>').html(x.name).appendTo(lst));
  });
```

Die Variable **`data`** enthält dabei schon das erhaltene JSON-Objekt.
In weiterer Folge werden verschiedene Elemente in den DOM-Tree eingehängt. Dazu werden Elemente erzeugt, indem mit der **`$('')`**-Funktion ein jQuery-Objekt erzeugt wird, das dann mit **`appendTo()`** an ein anderes jQuery-Objekt angehängt wird.
Der folgende Screenshot zeigt die HTML-Tabelle in der Firefox-Developerkonsole nach Einbau des JSON-Objekts in den DOM: