**Projektarbeit im Wahlpflichtfach Informatik**



**Name des Projekts: Das Computerlexikon**

**Name: Elwag**

**Klasse: 10C**

**Betreuende Lehrkraft: Herr \*\*\*\*\*\*\*\*\*\***

**Erreichbarkeit des Projektes:  
- In den beigefügten Dateien**

**- GitHub: https://github.com/Elwag/Computerlexikon**

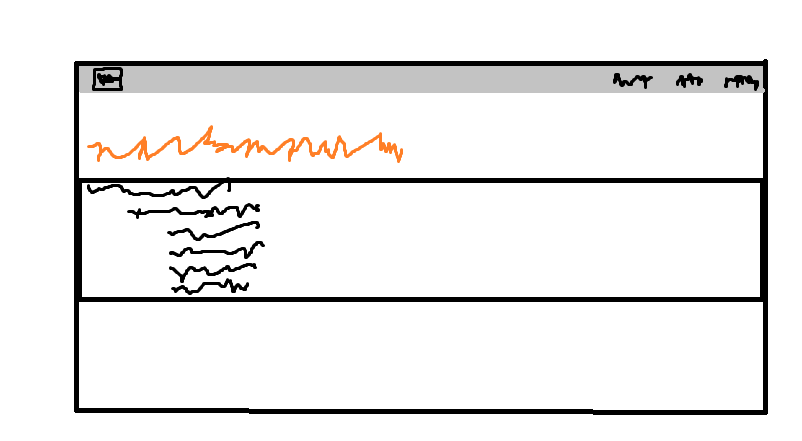
**1. Die erste Idee**

Die Idee für meine Projektarbeit entstand aus der Beschäftigung mit dem Thema Computerhardware im Unterricht und meinem eigenen Interesse daran.

Das Ziel ist eine Website, die mit Unterpunkten sinnvoll zusammenfasst, was zu so einem PC eigentlich dazu gehört.

Ich frage mich immer wieder warum es noch kein Computerlexikon im Internet zu finden gibt, abgesehen von Wikipedia und ähnlichen Seiten.  
Ich wollte mein Computerlexikon als vollständige Website anbieten, geschrieben in HTML und CSS.

Wichtig ist mir hier auch, dass lebensechte Bilder der einzelnen Komponenten zu finden sind. Es soll erreicht werden, dass man die Komponenten so anschauen kann, wie sie im echten Leben auch aussehen und wie man sie zu Gesicht bekommt.

****

**2. Umsetzung**

Zeitlich hatte ich mir vor genommen:  
1. Tag: Grundstruktur, eventuell erste Inhalte

2. Tag: CSS benutzen, um die Website sinnvoll zu gestalten

Alle anderen Tage: Alles was mir so einfällt einbinden

Ich konnte mich die ersten zwei Tage relativ gut an diesen Zeitplan halten, außerdem habe ich auch häufig zu Hause noch weiter gearbeitet.

Als erstes baute ich mir die Grundstruktur:  
- Überschrift mit <h1>

- ausklappbare Elemente mit Listen, eingepackt in einen div-Container und mit Klassen eingeordnet (<summary> - <details>, <div>, class=“Klasse“)

- Listenelemente ebenfalls ausklappbar mit Inhalt (<p>)

Darauf folgte ein wenig Design. Das CSS-Stylesheet wurde angelegt, Bilder eingefügt, erste Farben ins Spiel gebracht.

Die Überschrift wurde größer und Orange eingefärbt, die ausklappbaren Listenelemente erhielten eine weiße und fette Schriftart.

Danach lernte ich meinen ersten neuen Befehl kennen, ebenfalls in CSS:

background: linear-gradient(90deg, rgba(0,0,0,0) 0%, rgba(0,0,0,0) 100%);

Hiermit lässt sich ein Farbverlauf als Hintergrund einstellen.

Damit der Farbverlauf nicht Hochkant sondern Vertikal verläuft, drehte ich diesen mit dem Operator „90deg“ um 90°. Um die Farben auf der linken und auf der rechten Seite zu definieren, wird der Operator rgba(0,0,0,0) pro Seite jeweils ein Mal verwendet. RGBa steht für Red(Rot), Green(Grün), Blue(Blau) und alpha. Diese Werte werden jeweils anstelle der 0en eingesetzt. Hierbei werden Rot, Grün und Blau jeweils von 0-255 angegeben und miteinander vermischt. Alpha beschreibt die Opazität, auch genannt Transparenz oder Deckkraft. Dieser Wert wird von 0-100(%) angegeben. Je höher der Wert, desto mehr Deckkraft erhält dieser, bei 0 Prozent ist das Element vollständig Transparent, bei 100 vollständig Deckend.

Den nächsten Befehl lernte ich direkt danach bei dem Design der Liste kennen:

backdrop-filter: invert(100%)

Wendet man diesen Befehl auf ein Element in CSS an, so wird der Hintergrund dieses Elementes um 0-100% invertiert, je nach Angabe in den Klammern.

Um den Bereich der Listenelemente hervorzuheben, wollte ich den Hintergrund derer herausstechen lassen.

Da mich eine vollständige Invertierung am meisten ansprach, wählte ich 100% als Invertierung. Somit wurde der Hintergrund auf der einen Seite vollständig Schwarz, zu der anderen Seit hin von den Nuancen des Hintergrundbildes meiner Seite geprägt immer gräulicher.

Jetzt sprachen mich Animationen besonders an, denn interaktive und animierte Elemente können eine Seite wesentlich ansehnlicher machen.

Da ich allerdings noch nie mit Animationen in HTML und CSS gearbeitet hatte, war dies Neuland für mich. Da dies ein vollkommen neues Gebiet für mich war, benötigte ich hier auch sehr viel Zeit im Vergleich zu allen anderen Schritten.

Um Animationen zu realisieren, musste zuerst eingestellt werden, wann welche Animation abgespielt wird. Dies wird mit dem Befehl

animation: name duration timing-function delay iteration-count direction

definiert. Name ist der Name der Animation, die abgespielt werden soll. Duration die Dauer dieser, timing-function gibt an, ob am Anfang und/oder Ende der Animation diese langsamer abgespielt werden soll, iteration-count gibt an wie viel Mal die Animation abgespielt wird/werden darf und direction gibt die Richtung an, z.B. Vorwärts oder Rückwärts.

Eine Animation wird immer dann abgespielt, wenn das betroffene Element geladen wird oder bestimmte Bedingungen, wie zum Beispiel das klicken darauf, erfüllt sind.

Damit eine Animation abgespielt werden kann, muss diese aber noch definiert werden. Ein gutes Beispiel ist hier der color-change (so nicht in meinem Code zu finden, lediglich ein Beispiel):

@keyframes color-change {

0% { color: white; }

100% { color: black; }

}

Mit @keyframes wird die Definition der Keyframes der Animation, also die einzelnen Ankerpunkte dieser, gestartet. Dahinter folgt, von einem Leerzeichen getrennt, der Name der Animation. In geschweifte Klammern begrenzt werden die Keyframes dann angegeben. 0% ist hier der Ausgangszustand der Animation, 100% das Ende. Es können beliebig viele Keyframes von 0-100% angegeben werden, solange diese sich nicht doppeln.

Eine Zeile der Keyframedefinition fängt mit der Prozentzahl, gefolgt von einem Prozentzeichen, der Position des Keyframes innerhalb der Animation an.

Wieder in geschweiften Klammern begrenzt folgt hier nun die Definition der CSS-Parameter, die an dieser Stelle der Animation erfüllt sein sollen.

Gehen wir davon aus, dass in unserem Beispiel ein Text animiert sein soll, dann würde dieser, je nach Angabe im animation-Befehl (s.O.) von weiß zu schwarz übergehen.

Nach diesem Prinzip animierte ich dann auch alle anderen Animationen, u.A. dass Überschrift und Listenelemente bei Aufruf der Seite langsam sich auf ihre Positionen bewegen und erscheinen, oder dass wenn man den Mauszeiger über ein Bild bewegt, dass dieses dann geschmeidig größer wird.

Dies kann mit dem nächsten neuen CSS-Befehl realisiert werden, Beispiel:

p {

color:white;

}

------------------------------------

p:hover {

color:black;

}

Mit p:hover werden Parameter angegeben, die nur dann angewandt werden, wenn sich der Mauszeiger über dem betroffenen Element befindet.

Diese Parameter überschreiben dann auch, solange sich der Mauszeiger darüber befindet, die aktuellen Parameter, bis sich dieser wieder weg bewegt. Ist der Mauszeiger nicht mehr auf dem betroffenen Element, so werden die Parameter nicht mehr angewandt.

In unserem Beispiel ist beispielsweise ein Text also weiß, sobald sich der Mauszeiger darüber befindet schwarz, tut er dies nicht mehr, ist der Text wieder weiß.

Mit der Zeit fügte ich weitere Einzelheiten hinzu, die sich alle im Detail gar nicht lohnen zu erwähnen, denn das würde den Rahmen dieser Dokumentation sprengen.  
Jedoch kann ich auf einige wesentliche Punkte eingehen, die essentiell für die Website-Gestaltung waren.

Hierzu zählen z.B. die animierten Buttons der Navigationsleiste, die Startanimationen, der wechsel von Orange zu Grau der Listenelemente beim darüber gleiten, oder eben der Footer.

Der Footer sollte dauerhaft am unteren Rand der Website zu sehen sein, egal wie weit gescrollt wurde. Zudem sollte er sich über die ganze Seite von links nach rechts ziehen, während der Text darauf in der Mitte stand. Dies lies sich mit drei Variablen in CSS realisieren:  
  
position: fixed;

und:  
bottom: 0;

außerdem:  
text-align: center;

Der erste Operator beschreibt, dass die Position des Footers immer fest stehen und sich nicht verändern soll.

Der zweite beschreibt, dass der Footer von dem unteren Rand aus 0 Pixel weg sein soll, sprich: Ganz unten, direkt am unteren Rand.

Damit der Text auch zentriert in der Mitte des Footers steht, benutze ich auch noch den 3. Befehl. Er bestimmt, woran der Text ausgerichtet ist.

In unserem Fall an der Mitte („center“).

Damit der Footer ein wenig dunkler wird, wenn man über ihn mit der Maus gleitet, benutze ich hier wieder .footer:hover.

Innerhalb dieses Kontextes wird alles auf den Footer, den ich vorher mit der class=“footer“definiert habe, angewandt. Hier ändere ich nur die Hintergrundfarbe mit background-color: #dededee3; etwas mehr ins Gräuliche, sodass der Footer beim darüber gleiten etwas gräulicher wird. Dies hat keinen speziellen Nutzen und ist rein optischer Natur.

Alternativ könnte man hier, wie auch bei den Buttons in der Navigationsleiste, eine Animation einfügen, um den Wechsel zur dunkleren Farbe flüssiger zu gestalten.

Allerdings ist dies an dieser Stelle gar nicht nötig und würde gar nicht unbedingt einen Mehrwert bringen, eher den Code nur komplizierter machen. Daher habe ich an dieser Stelle keine Animation in Betracht gezogen.

Die Navigationsleiste wollte ich noch etwas interaktiver haben, daher wollte ich unbedingt einen Farbverlauf, der der Maus folgt.

Um der Maus zu folgen, brauchte ich JavaScript, also fragte ich ChatGPT, der mir dies ausgab:

document.addEventListener('mousemove', function(e) {

const x = e.clientX;

const y = e.clientY;

const width = window.innerWidth;

const height = window.innerHeight;

const percentX = x / width;

const percentY = y / height;

const centerX = percentX \* 100;

const centerY = percentY \* 100;

const gradient = `radial-gradient(circle at ${centerX}% ${centerY}%, #777777, #cccccc)`;

document.querySelector('.gradient-background').style.background = gradient;

});

Danach beschäftigte ich mich intensiv damit, wie der Code funktioniert.

Zuerst wird eine Funktion namens „mousemove“ definiert.

In dieser werden dann einige Variablen nacheinander definiert.

Die ersten beiden Befehle definieren die Variablen x und y, die mit e.clientX und e.clientY einfach die aktuelle Mausposition auf der Website in Pixeln angeben.

Die nächsten beiden Befehle definieren die Variablen width und height. Durch window.innerWidth und window.innerHeight wird jeweils die Breite und die Höhe der aktuellen Website abgefragt und in die Variablen geschrieben.

Die Variablen percentX und percentY, die in den zwei darauf folgenden Befehlen definiert werden, geben die vertikalen und senkrechten Koordinaten relativ zu dem aktuellen Fenster in einem Wert von 0-1 an.

Hierzu wird die jeweilige Achse durch die Höhe bzw. Breite der Seite geteilt.

Befindet sich der Mauszeiger z.B. auf der vertikalen Ebene genau in der Mitte der Seite, so liegt der Wert von percentX bei 0.5.  
Teilt man die Seite in 4 Einzelteile (entweder vertikal oder horizontal) und würde der Mauszeiger exakt zwischen dem ersten Viertel und dem zweiten liegen, so wäre percentX bzw. percentY exakt gleich 0.25.

Um nun aus den Werten percentX und percentY jeweils Prozentzahlen zu errechnen, werden die Werte, die ja gleich 0-1 sind, mit 100 multipliziert.

Heraus kommt ein Wert von 0-100%, je nach Position des Mauszeigers relativ zur Website.

Damit der Farbverlauf jetzt passend reagieren kann, wird die Variable gradient definiert. Sie gibt alle Werte an, die auf den Hintergrund der Navigationsleiste angewandt werden sollen.

Zuletzt wird der Farbverlauf passend zum Mauszeiger noch angewandt.

Dazu wird der letzte Befehl verwendet (s.O.):  
  
document.querySelector bestimmt, dass das Folgende auf die erste passende Sektion im html-Dokument angewandt werden soll.

Was eine passende Sektion ist, wird in den runden Klammern dahinter definiert. ‘.navbar’ beschreibt hier alle Elemente, die die Klasse „navbar“ besitzen. Der Punkt ist wichtig, er gibt an, dass es sich nicht um einen Tag sondern eine Klasse handelt.

.style.background = gradient beschreibt dann das, was angewandt werden soll: Der Operator style in css, in diesem Falle der Hintergrund, soll der vorher definierten Variable gradient entsprechen. Puh, geschafft ;-)

Eine wichtige Erkenntnis war, dass ein Button, der sich weg bewegt sobald der Mauszeiger über ihm gleitet, zwar lustig sein kann, jedoch absolut nicht sinnvoll ist.

Ursprünglich war die Idee, dass die Listenelemente sich etwas nach rechts bewegen, wenn man mit der Maus über sie gleitet. Dies hat sich jedoch als sehr viel nerviger herausgestellt als gedacht, denn häufig hat man das Element beim Klicken nicht erwischt, da es immer weg gerutscht ist.

Hin und wieder baute ich nun auch weitere Feinheiten ein. Hierzu zählt z.B. dass der Footer mittlerweile ebenfalls den gleichen Effekt wie die Navbar hat. Außerdem ist bei Navbar und Footer der Hintergrund etwas verschwommen. Ich verlinkte noch im Footer die Website von Jeremy und Yoon-Ho und die Website von Laura und Sarah.

Damit die Website auch aus dem gesamten Internet erreichbar ist, habe ich diese natürlich noch gehostet. Hierfür nahm ich eine der Domains meines Vaters (https://sörva.de) und richtete dahinter einen Server mit Ubuntu Server 24.04 LTS als OS und lighttpd als http-Server ein. Dieser lädt einfach die Website, die ich als Dateien hinterlegt habe, in den RAM des Servers und daraufhin schickt er diese Dateien an den Browser vom Client, der die Website aufruft. Als Server dient ein virtueller Ionos-Server mit einem Kern und 512MB RAM. SSL-Zertifikat, dns-caching, http(s) forwarding etc. würde den Rahmen dieser Dokumentation sprengen, ich möchte es hier nur kurz erwähnt haben.

**3. Rechtliche Aspekte**

Als Impressum nahm ich in diesem Falle einfach den Namen und Kontaktdaten des IKG. Außerdem nannte ich meinen Namen mit E-Mail Adresse als Verantwortlicher für den Inhalt.

Das Impressum ist insofern wichtig, dass wenn es ein Problem mit meiner Website gibt, dass jemand den Betreiber der Seite auf direktem Weg kontaktieren kann.

Gesetzlich ist vorgeschrieben, dass eine Website, die ausschließlich für private Zwecke genutzt wird, kein Impressum braucht.

Eine Website, die nicht ausschließlich privat ist, braucht aber sehr wohl eines.

Da allerdings sehr streng definiert ist, was eine private Website ist und was nicht, habe ich lieber ein Impressum angegeben, um kein Risiko einzugehen.

Es ist auch eigentlich nicht ganz richtig, das IKG als den Betreiber der Website anzugeben, denn faktisch gesehen ist das ja nicht der Fall.

Für eine so kleine Website die so oder so keinerlei rechtlich geschützte Bilder, Texte oder ähnliches verwendet und zudem noch nur für wenige Wochen, wenn überhaupt, online bleiben wird, ist das allerdings in diesem Fall weniger relevant. Hauptsache es gibt überhaupt ein Impressum

Da ich, bzw. meine Website, keinerlei Daten erhebe/erhebt, ein Cookie-Banner aber zumindest drin sein sollte, habe ich diesen mit einem einfachen div-Container und sich darin befindlichem Text und einem Button realisiert. Der div-Container und der Button erhielten jeweils ihre eigene ID (so ähnlich wie eine Klasse, wird aber z.B. mit einem # statt Punkt im CSS-Dokument aufgerufen), damit diese vom JavaScript-Code benutzt werden können:

document.getElementById("accept-button").onclick = function() {

document.getElementById("cookie-banner").style.display = "none";

};

Sobald der Button mit der ID „accept-button“ geklickt wird, soll eine Funktion aufgerufen werden.

In diesem Falle handelt es sich nur um eine einzelne Zeile, in der das Element mit der ID „cookie-banner“ (in diesem Fall der div-Container, der als Cookie-Banner dient) das CSS-Attribut „display = none“ erhält und somit nicht mehr angezeigt wird, sobald der Button gedrückt wurde.

**4. Fazit**

Abschließend bin ich mit meiner Website ziemlich zufrieden.  
Optisch ist sie zwar kein hochglanzpolierter Porsche GT3 (die Website hat kein herausragend schönes Design), doch das war auch gar nicht mein Ziel.

Die Website sollte vor allem viele Features mitbringen, viel mit Animationen spielen und nebenbei noch einige Informationen zu Computerteilen mitteilen.

Ursprünglich waren viele der kleinen Elemente/Features wie der Cookie-Banner, der Farbverlauf des Hintergrundes oder die vielen Animationen gar nicht eingeplant, daher wich ich in dieser Hinsicht sehr von meinem ursprünglichen Plan ab.

Ich bin jedoch bei meiner grundlegenden Idee geblieben und die Website sieht nicht komplett anders aus, als anfangs erdacht.

Dies hat den Grund, dass ich zwar die Idee sehr gut fand, jedoch es mich in den Fingern juckte immer mehr und mehr im Hintergrund einzubauen,

ein wenig JavaScript zu erlernen und die Website mit einigen Animationen interessanter und interaktiver zu machen.

Daher bin ich vollkommen zufrieden mit meiner Website und würde jederzeit eine weitere, noch bessere Website programmieren wollen. Es hat mir großen Spaß gemacht neue Dinge zu erlernen, die ich vorher noch gar nicht kannte.

PS (wenn nicht schon gesehen): Schonmal auf das Bild oben links geklickt? ;-)