

2024

Sportferienprojekt

Wetter-Website

Colin Piguet

Inhalt

Änderungstabelle.....	2
Information	2
Motivation.....	2
Meilensteine	3
Planung	4
Entscheidungen	4
Umsetzung.....	5
Startseite	5
Navigation	6
Anordnung	7
API.....	8
Stadtinformation API	8
Wetter API	9
Analoge Uhr.....	10
Wetter Animation.....	12
Testplan	14
Probleme.....	15
Fazit	16
Quellen	16
Anhang	16

Änderungstabelle

Datum	Was
05.02.2024	Dokument erstellt Startseite mit Bildern eingefügt
05.02.2024	Homebutton, Pfeil nach rechts und Pfeil nach links eingefügt
06.02.2024	Für jede Stadt eine Datei erstellt
06.02.2024	Bilder verlinkt zu Dateien Alle miteinander über Pfeile verbunden
07.02.2024	API für Infos der Städte eingefügt
07.02.2024	API für das Wetter eingefügt
08.02.2024	Analoge Uhr erstellt
08.02.2024	Alles auf die einzelnen Städte übertragen und angepasst
09.02.2024	Wetter Animation eingefügt
09.02.2024	Alles übertragen
21.02.2024	Gantt Diagramm, Testplan eingefügt Dokumentation bearbeitet
22.02.2024	Information, Motivation und Meilensteine geschrieben Einige Teile der Umsetzung geschrieben
23.02.2024	Umsetzung, Probleme, Fazit, Quellen und Anhang fertig PowerPoint fertig

Information

Dieses Dokument erklärt und zeigt einzelne Teile meines Wetter-Website-Codes. Auf der Webseite können verschiedene Informationen zu insgesamt Neun Städten angezeigt werden, wie beispielsweise die aktuelle Temperatur, Uhrzeit und zahlreiche andere Informationen zu den einzelnen Städten. Außerdem zeigen wir die aktuelle Zeit mit einer analogen Uhr an. Es gibt auch eine kleine Animation zum aktuellen Wetter. Man hatte insgesamt acht Tage Zeit für das Projekt und musste sich für alles selbst einplanen. Wir mussten ein Kanban Board erstellen und alle einzelnen Schritte dokumentieren.

Motivation

Da ich gerne reise und bereits viele Städte besucht habe, kam ich auf die Idee, eine Website zu erstellen, auf der Daten zu verschiedenen Städten angezeigt werden. Ich würde eine API verwenden, um die aktuelle Temperatur und das Wetter anzuzeigen. Ich wollte auch eine analoge Uhr verwenden, um die aktuelle Uhrzeit zu sehen. Daher war meine Motivation, genau diese Ziele zu erreichen und eine Website zu erstellen, die mir auch etwas bringt, da ich gerne reise. Ich finde es auch faszinierend zu sehen, wie das Wetter in anderen Städten ist, oder wie die Uhrzeit ist, ob es schon Nacht ist oder erst morgen.

Meilensteine

Meilenstein 1 = Schöne funktionierende Indexseite, mit verschiedenen Städten

Ich möchte eine funktionierende Indexseite erstellen, auf der man verschiedene Bilder von Städten auswählen kann und wenn man auf diese klickt, auf eine andere Seite weitergeleitet wird. Sie soll auch schön dargestellt sein.

Ich habe mir für diesen Meilenstein entschieden, da ich finde das es wichtig ist das man eine funktionierende und ansprechende Startseite hat. Weil wenn ein Benutzer auf diese Website gehen würde und eine nicht schön gestaltete Startseite sehen würde, wahrscheinlich nicht mehr weiter diese Website nutzen würde, da er keinen guten Eindruck hatte.

Meilenstein 2 =Mindestens 9 verschiedene Städte, die man auswählen kann und Daten sehen kann

Mindestens 9 verschiedene Städte, die man auswählen kann, möchte ich auf der Indexseite erstellen. Zu jeder Stadt soll man auf eine andere Seite weitergeleitet werden und dort die Daten schön dargestellt werden. Die jeweiligen Seiten sollten alle gleich aufgebaut sein.

Für diesen Meilenstein habe ich mich entschieden, weil ich finde, dass 9 Städte eine gute Anzahl für den Anfang sind. Es macht auch Sinn, dass man zu jeder Stadt eine einzelne Seite erstellt und diese dann miteinander verknüpft.

Meilenstein 3 = Wetterdaten mit API und genaue Uhrzeit der verschiedenen Städte anzeigen

Zu den jeweiligen Städten möchte ich verschiedene Wetterdaten, wie Temperatur oder ob es gerade sonnig oder bewölkt ist, anzeigen. Dies werde ich mit APIs machen. Ausserdem möchte ich noch die genaue Uhrzeit den jeweiligen Städten mit einer analogen Uhr anzeigen.

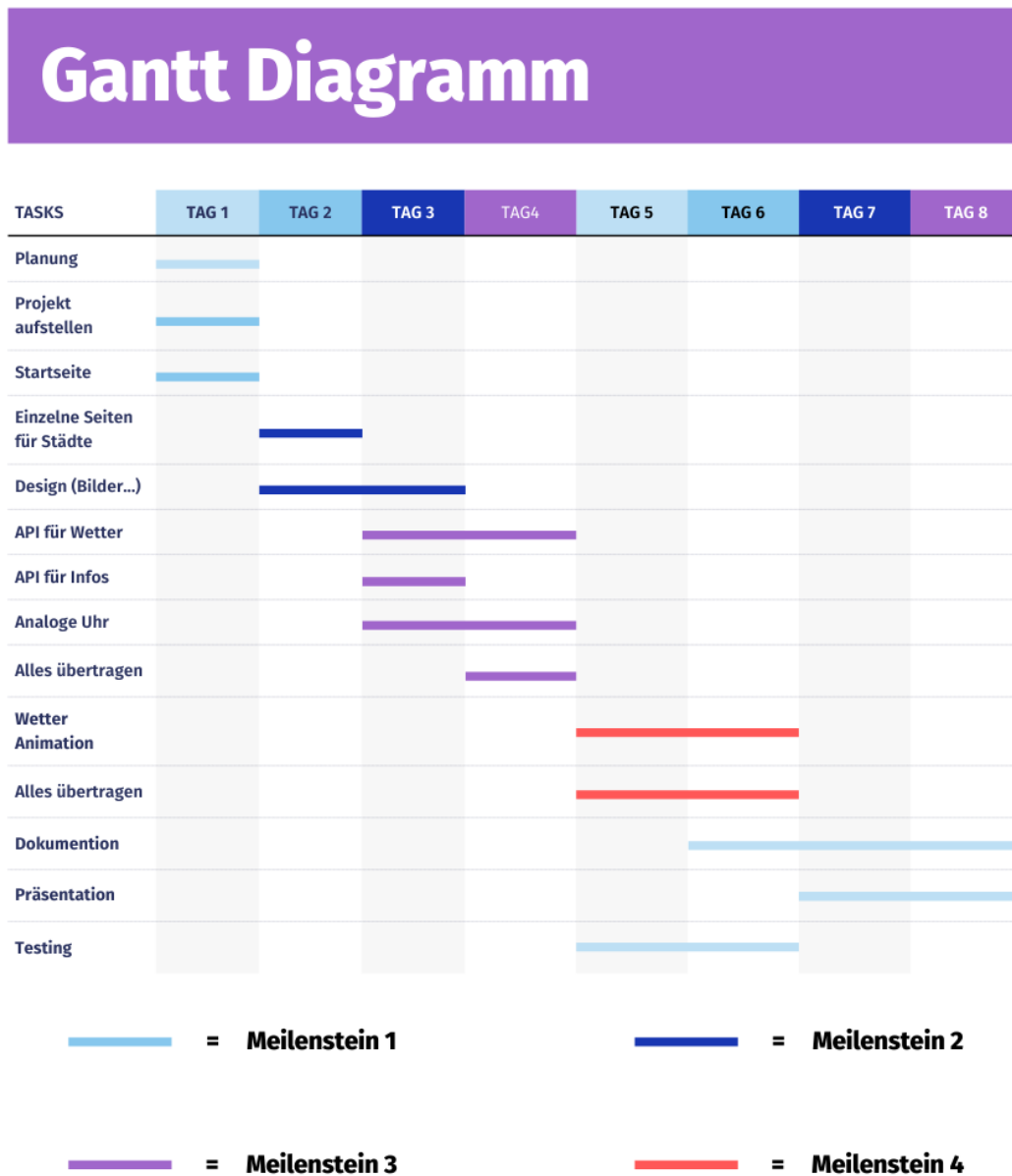
Bei diesem Meilenstein habe ich mir am meisten Zeit eingeplant, da er auch der grösste Teil meines Projektes beinhaltet. Wie vorher schon mal erwähnt, ist dieser Meilenstein eigentlich mein grösstes Ziel und Motivation. Ich wollte die einzelnen Daten auf den einzelnen Seiten der Städte anzeigen lassen.

Meilenstein 4 = Animation für Wetter programmieren

Für die einzelnen Wetterbedingungen möchte eine kleine Animation einfügen. Man sollte zum Beispiel drei Wolken sehen, welche sich bewegen, wenn es gerade bewölkt ist.

Ursprünglich plante ich nur drei Meilensteine, aber dann stellte ich fest, dass ich schneller fertig sein würde als gedacht. Deshalb entschied ich mich, einen vierten Meilenstein hinzuzufügen. Diesen plante ich nur für einen Tag, da ich auch noch Dokumentation und Präsentation erledigen musste. Bei diesem Meilenstein wollte ich noch eine kleine Animation einfügen, die das aktuelle Wetter in dieser Stadt darstellt.

Planung



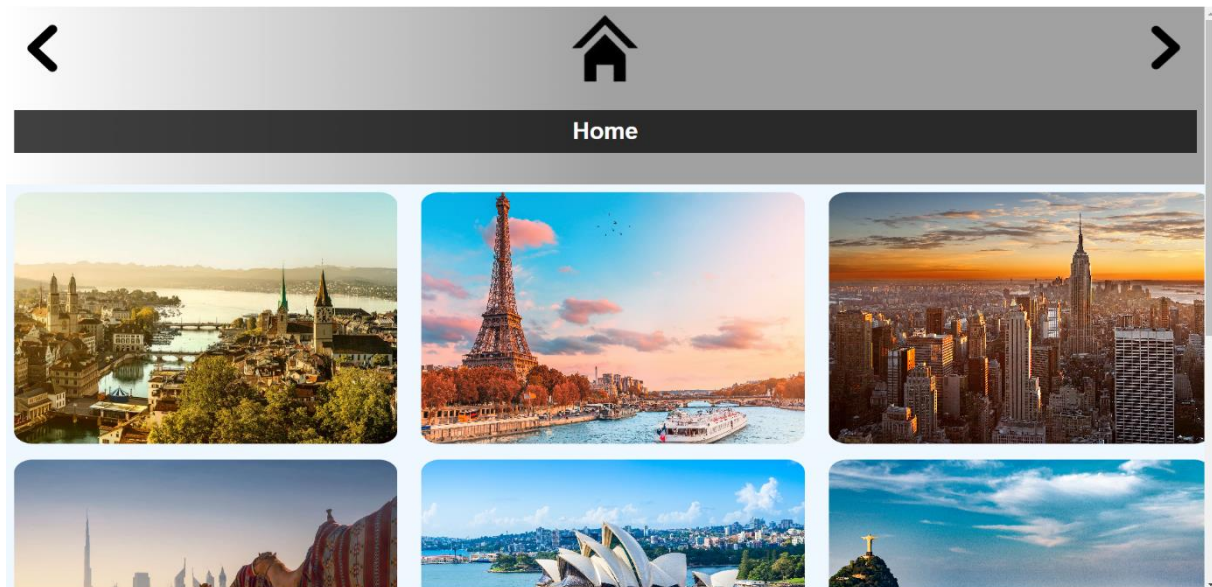
Entscheidungen

Ich habe mich entschieden das ich die verschiedenen API's, die Analoge Uhr und die Wetteranimation, alles mit JavaScript programmieren werde, da ich bereits ähnliche kleine Projekte mit diesen Programmiersprachen erstellt habe und so bereits ein wenig Erfahrung habe.

Umsetzung

Startseite

Als Erstes habe ich mit der Startseite oder Indexseite gestartet. Ich erstellte eine Kopf- und Fusszeile. Für die neun Städte, die ich ausgewählt habe, suchte ich ein schönes Bild heraus und machte diese in auch auf die Startseite.



Startseite der Website

Im gegebenen CSS-Code wird die Flexbox-Technik verwendet, um die Elemente mit der Klasse ".city" aufzuteilen und anzuzeigen. Die Eigenschaft "flex" wird verwendet, um die Flexibilität der Elemente zu definieren, und die Eigenschaft "margin" wird verwendet, um den Abstand zwischen den Elementen zu steuern. Man sieht auch, dass die Bilder eine leichte Hover-Animation haben, die sie beim Überfahren mit der Maus leicht vergrößert. Dies wird durch geschickte Übergänge und Transformationen im CSS erreicht. Genauer gesagt mit der Regel .city img:hover, die aktiv wird, wenn der Mauszeiger über ein Bild schwebt. Hier wird die Skalierung des Bildes leicht erhöht, um einen subtilen Vergrößerungseffekt zu erzeugen. Durch die zuvor definierte Übergangsanimation geschieht dies fließend.

```
.city {  
  flex: 0 1 calc(33.33% - 20px);  
  margin: 10px 0;  
  text-align: center;  
  text-decoration: none;  
}  
  
.city img {  
  width: 100%;  
  height: 100%;  
  object-fit: cover;  
  margin-left: 10px;  
  margin-right: 10px;  
  border-radius: 5%;  
  transition: transform 0.3s ease-in-out;  
}  
  
.city img:hover {  
  transform: scale(1.05);  
}
```

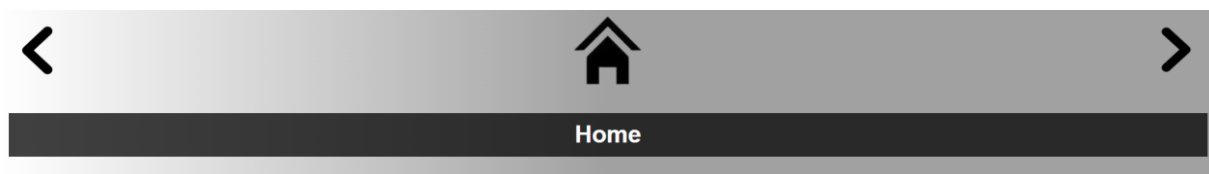
Code Ausschnitt CSS für Bilder

Im unteren HTML-Code sieht man eine `<section>`, innerhalb derer drei `<div>`-Elemente mit der Klasse "city" eingebunden sind. Jedes dieser `<div>`-Elemente enthält ein `<a>`-Element mit einem Bild, das als Link zu einer bestimmten Seite dient. Die Bilder repräsentieren jeweils Städte (Zürich, Paris, New York), und die Links führen zu den entsprechenden Seiten ("zuerich.html", "paris.html", "newYork.html"). Diese Struktur ermöglicht es, Bilder als Links darzustellen und die Benutzer zu den entsprechenden Seiten zu führen, wenn sie auf ein Bild klicken.

```
<section>
  <div class="city"><a href="zuerich.html"></a></div>
  <div class="city"><a href="paris.html"></a></div>
  <div class="city"><a href="newYork.html"></a></div>
</section>
```

Code Ausschnitt HTML für Startseite

Navigation



Navigation der Website

Um zwischen den einzelnen Städten zu wechseln und wieder auf die Startseite zu kommen, habe ich Pfeile nach rechts und links und einen Home Button eingefügt. Wenn man auf den Home Button klickt, kommt man wieder zurück auf die Startseite. Klickt man auf den Pfeil, der nach rechts zeigt, geht es eine Seite nach vorne und wenn man den Pfeil nach links anklickt, kommt man eine Seite zurück. Wenn man bei der letzten Stadt (Tokio) ist und den Pfeil nach vorne anklickt, kommt man wieder auf die Startseite. Dasselbe, wenn man auf der ersten Stadt (Zürich) ist und den Pfeil nach hinten anklickt.

```
<div class="navigation">
  <div class="links"><a href="tokio.html" title="Links"></a></div>
  <div class="home"><a href="index.html" title="Home"></a></div>
  <div class="rechts"><a href="zuerich.html" title="Rechts"></a></div>
</div>
```

Code Ausschnitt HTML für Navigation

Die Funktion, die man wieder auf die einzelnen Bilder klicken kann, ist gleich wie bei der Startseite, die Bilder aufgebaut. Die Bilder werden mit den jeweiligen Seiten verlinkt. Dies muss man aber immer je nach Seite anpassen. Die einzelnen Bilder hätte man alle in die gleichen Klassen machen können, was jedoch nicht ging, da man die Grösse mancher Bilder grösser oder kleiner machen müssen.

Im unteren CSS-Code sieht man, wie man dem Header und Footer diesen speziellen Farbverlauf machen kann. Dies kann man ganz einfach durch die CSS-Funktion `linear-gradient`. Hier muss man dann nur noch die Richtung, 2 Farben, an welcher Stelle (mit %) und wie flüssig der Übergang sein soll (0 % ist am flüssigsten).

```
header,
footer {
  text-align: center;
  background-image: linear-gradient(to right, white 0%, #a1a1a1 50%);
  color: rgb(0, 0, 0);
  padding: 10px;
}
```

Code Ausschnitt CSS für Header und Footer

Anordnung

Wie man im unterliegenden Bild sehen kann, sind die verschiedenen Daten in einzelnen Boxen angeordnet und abgebildet. Diese Art Boxen sind alle schön aufgeteilt und übersichtlich dargestellt.



Aufteilung der Daten von Website

Um diese Boxen zu erstellen und benutzen, kann man ganz einfach verschiedene Klassen erstellen und im CSS bearbeiten. Dies habe ich auch gemacht und im unterliegenden CSS-Code kann man das noch genauer sehen.

```
.container {
  display: flex;
  /* Flexbox-Container */
}

.box {
  flex: 1;
  /* Jede Box soll gleich viel Platz einnehmen */
  padding: 10px;
  background-image: linear-gradient(to top right, rgb(251, 198, 154) 0%, #ffdfbdea 20%);
  border: 20px solid;
  border-color: white;
  align-items: center;
  border-radius: 10px;
}
```

Code Ausschnitt CSS der Aufteilung

.container: Dieser Stil definiert einen Container, der die Flexbox-Technik verwendet. Das bedeutet, dass die Elemente innerhalb dieses Containers flexibel positioniert und ausgerichtet werden können.

.box: Dieser Stil definiert die Stile für die Boxen innerhalb des Containers.

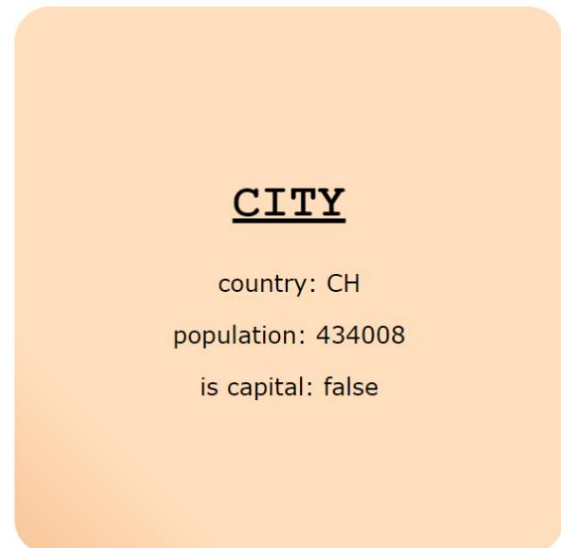
- **flex: 1;** Jede Box innerhalb des Containers soll gleich viel Platz einnehmen und sich somit gleichmäßig über den verfügbaren Platz im Container erstrecken.
- Im Bild sieht man, dass es auch noch andere Sachen, wie Bilderrand oder Farbe gibt, die man aber relativ einfach benutzen kann.

API

Um die unterschiedlichen Informationen anzuzeigen, musste ich auf verschiedene APIs zurückgreifen. Ich habe eine API für Wetterdaten und eine andere für Stadtdaten genutzt. Obwohl der Code für beide APIs recht ähnlich ist, hat es trotzdem Unterschiede, da sie von unterschiedlichen Websites stammen.

Stadtinformation API

Im Bild kann man sehen, dass ich verschiedene Daten wie das Land, die Bevölkerung und ob es die Hauptstadt ist, abgefragt habe. Die API, die ich dafür benutzt habe, ist von der Website API Ninjas. Die Website wurde mir von Adrian bekannt gemacht. Um eine API zu verwenden, musste man sich bei der Website kostenlos anmelden und seinen zugewiesenen API-Key abzurufen. Mit dem API-Key kann man jede API von dieser Website benutzen und verwenden. Jedoch kann man nur 1000 Anfragen pro Monat machen, wenn man die kostenlose Version benutzt. Ich habe mich für diese Website entschieden, da es kostenlos ist und alle Daten hat, die ich brauche. Ein Pluspunkt war auch,



dass es eine Codevorlage hatte, um die API abzurufen.

Stadinfo API von Website

Ich habe mit der Vorlage der Website gearbeitet. Zusammen mit Adrian haben wir eine gute Lösung zur Abrufung der Daten gefunden. Die Abrufung der Daten konnten wir 1:1 von der Website übernehmen, jedoch die Daten in das HTML zu schreiben war eine schwierige Aufgabe. Mithilfe einer Idee von ChatGPT kamen wir auf eine erfolgreiche Lösung, die man im unterliegenden Bild sehen kann.

```
<div class="box time">
  <div id="time">City information is loading...</div>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <script>
    var city = 'Zürich'
    $.ajax({
      method: 'GET',
      url: 'https://api.api-ninjas.com/v1/city?name=' + city,
      headers: { 'X-API-Key': 'XWcKrVsexY5n4D1jf+62wA==IcOG7a9RIhQimMoX' },
      contentType: 'application/json',
      success: function (result) {
        var timeHtml = '<h2>' + ' CITY </h2>';
        timeHtml += '<p>country: ' + result[0].country + '</p>';
        timeHtml += '<p>population: ' + result[0].population + '</p>';
        timeHtml += '<p>is capital: ' + result[0].is_capital + '</p>';
        $('#time').html(timeHtml);
      },
      error: function (jqXHR) {
        console.error('Error: ', jqXHR.responseText);
        $('#time').html('Error while loading the timeinformation');
      }
    });
  </script>
</div>
```

Code Ausschnitt der Stadinfo API Abrufung

AJAX

Ajax steht für "Asynchronous JavaScript and XML" und ist eine Technik, die es ermöglicht, Daten zwischen einem Webbrowser und einem Webserver auszutauschen, ohne dass die gesamte Seite neu geladen werden muss. Im Wesentlichen ermöglicht Ajax die Aktualisierung von Teilen einer Webseite, ohne dass die Benutzererfahrung unterbrochen wird.

- `src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>`: Dieser `<script>`-Tag lädt die jQuery-Bibliothek einer externen Quelle. jQuery ist eine JavaScript-Bibliothek, die die Arbeit mit dem DOM (Document Object Model) vereinfacht.
- `$.ajax({ ... })`: Ein AJAX-Aufruf wird gestartet, um die Informationen von einer externen API abzurufen. Dieser Aufruf verwendet die jQuery-Methode `$.ajax()`, um eine GET-Anfrage an die angegebene URL zu senden.

Diese Parameter definieren die AJAX-Anfrage:

- `method: 'GET'`: Es wird eine GET-Anfrage gesendet, um Daten von der API abzurufen.
- `url: 'https://api.api-ninjas.com/v1/city?name=' + city`: Die URL der API wird angegeben, einschließlich des Stadtnamens als Abfrageparameter, um spezifische Daten für die angegebene Stadt abzurufen.
- `headers: { 'X-API-Key': 'XWcKrVsexY5n4D1jf+62wA==lcOG7a9RlhQimMoX' }`: Ein benutzerdefinierter Header wird hinzugefügt, der den API-Schlüssel enthält, um auf die API zuzugreifen und die Authentifizierung zu gewährleisten.
- `contentType: 'application/json'`: Der Inhaltstyp der Anfrage wird auf JSON festgelegt, um sicherzustellen, dass die Daten im JSON-Format gesendet werden und die API sie korrekt verarbeiten kann.

Die `success`-Funktion wird nach einer erfolgreichen AJAX-Anfrage aufgerufen, um die erhaltenen Daten zu verarbeiten. Dabei wird ein HTML-String erstellt, der als Überschrift "CITY" für die Stadtinformationen dient. Die spezifischen Daten aus der API-Antwort (`result`), wie Land, Bevölkerung und Hauptstadtstatus, werden extrahiert und in den HTML-String eingefügt. Schliesslich wird der aktualisierte HTML-String verwendet, um die Stadtinformationen im Element mit der ID 'time' anzuzeigen, welche ganz oben im HTML-Code definiert wurde.

Wetter API

Die Wetter API habe ich von der Website OpenWeatherMap, welche ich bereits mal für ein anderes kleines Projekt benutzt habe. Um diese API abzurufen, muss man sich wieder nur einen Account erstellen und kann so wieder die API kostenlos abrufen. Bei dieser API habe ich insgesamt Fünf verschiedenen Daten abgerufen, nämlich das Aktuelle Wetter, das Wetter genauer beschrieben, die aktuelle Temperatur, die Wind Geschwindigkeit und die Luftfeuchtigkeit.

WEATHER

main weather: Rain
description: moderate rain
temperature: 7.80°C
wind speed: 4.12km/h
humidity: 93%

Wetter API von Website

Der Code, um die API abzurufen, sieht nicht ganz so aus wie bei der vorherigen API, das liegt daran, dass bei der Stadtinformation API bekommt man den API Key mit Namen (Key) und seinem Inhalt (Value). Bei der Wetter API bekommt man jedoch nur den Value des API Key. So konnte ich den Code nicht 1:1 übernehmen und musste noch einige Dinge ändern.

```
<div class="container">
  <!-- ----- Erste Spalte Start ----->
  <div class="box api">
    <div id="info">Weather information is loading...</div>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
    <script>
      $.ajax({
        method: 'GET',
        url: 'https://api.openweathermap.org/data/2.5/weather?q=Zurich&appid=f39c7ffc405fe7cfff9462730f188fb1',
        dataType: 'json',
        success: function (result) {
          console.log(result);
          var weatherHtml = '<h2>' + 'WEATHER</h2>';
          var temperatureCelsius = (result.main.temp - 273.15).toFixed(2);
          weatherHtml += '<p>main weather: ' + result.weather[0].main + '</p>';
          weatherHtml += '<p>description: ' + result.weather[0].description + '</p>';
          weatherHtml += '<p>temperature: ' + temperatureCelsius + '°C</p>';
          weatherHtml += '<p>wind speed: ' + result.wind.speed + 'km/h</p>';
          weatherHtml += '<p>humidity: ' + result.main.humidity + '%</p>';
          $('#info').html(weatherHtml);
        },
        error: function ajaxError(jqXHR) {
          console.error('Error: ', jqXHR.responseText);
          $('#info').html('Error while loading the weather information');
        }
      });
    </script>
  </div>
```

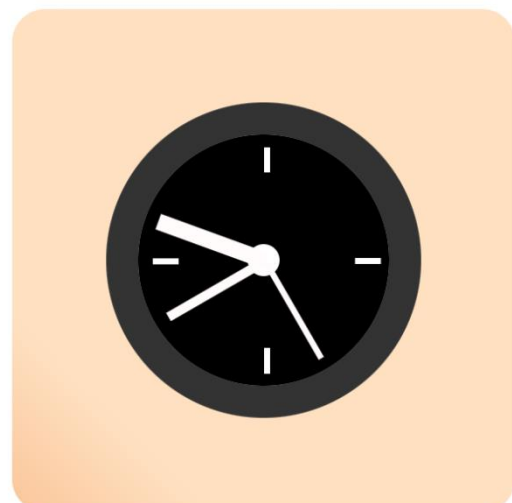
Code Ausschnitt der Wetter API Abrufung

Auf dem Code kann man sehen, dass der API-Key gleich im «url» mitgegeben wird. So braucht es nur das «Value». Sonst ist der Code gleich aufgebaut und nur bei der Ausgabe natürlich andere Daten, die man abrufen will. Weil die Temperatur eigentlich mit Fahrenheit angezeigt wird, musste ich noch die Temperatur minus 273.15 rechnen, damit man die Temperatur in Celsius hat. Es wird hier auch ein Ajax Aufruf gemacht und die Daten werden in den HTML-String eingefügt. Schliesslich wird der aktualisierte HTML-String verwendet, um die Wetterinformation im Element mit der ID "info" anzuzeigen, welche ganz oben im HTML-Code definiert wurde.

Man kann im Code auch noch sehen, dass alles in der Klasse "container" ist. Der Code gehört auch zu der Klasse "box". Mit diesen Klassen wurde vor allem das Design gemacht, wie vorhin bereits beschrieben.

Analoge Uhr

Um die aktuelle Uhrzeit anzuzeigen, habe ich mich entschieden, dass ich diese mit einer analogen Uhr darstellen werde. Da ich bereits eine analoge Uhr programmiert habe, konnte ich viel übernehmen. Ich verbesserte diese und musste sie auch abändern, damit ich sie gut darstellen. Die Uhr ist auch in einer diesen Boxen.



Analoge Uhr von der Website

```

<!-- START clock element with all utilities -->
<div class="box clock">
  <div class="outer-clock-face">
    <div class="marking marking-one"></div>
    <div class="marking marking-two"></div>
    <div class="marking marking-three"></div>
    <div class="marking marking-four"></div>
    <div class="inner-clock-face">
      <div class="point"></div>
      <div class="hand hour-hand"></div>
      <div class="hand min-hand"></div>
      <div class="hand second-hand"></div>
    </div>
  </div>
</div>
<!-- END clock element -->

<script src="clockZh.js"></script>

```

Code Ausschnitt der Uhr-Elemente

Die Funktion `updateClockHand(selector, rotation)` dient dazu, die Rotation der Uhrzeiger auf der Uhr zu aktualisieren. Über den Parameter `selector` wird das entsprechende Uhrzeiger-Element ausgewählt, während `rotation` den Rotationswinkel in Grad angibt, um den der Uhrzeiger gedreht werden soll.

Die Funktion `updateClock()` wird aufgerufen, um die Uhrzeit zu aktualisieren und die Position der Uhrzeiger entsprechend anzupassen. Dazu wird zuerst die aktuelle Zeit abgerufen und dann

die Stunden, Minuten und Sekunden extrahiert. Basierend

auf diesen Werten werden die Rotationswinkel für die Stunden-, Minuten- und Sekundenzeiger berechnet und mit Hilfe der `updateClockHand()`-Funktion aktualisiert.

Die Event-Handler-Funktion für das Fensterladen (`window.onload`) wird verwendet, um sicherzustellen, dass der Code ausgeführt wird, sobald das Fenster geladen ist. Anschliessend wird die `updateClock()`-Funktion aufgerufen, um die Uhr zu initialisieren und die Uhrzeiger entsprechend einzustellen. Schließlich wird `setInterval()` verwendet, um `updateClock()` alle Sekunde aufzurufen und die Uhrzeit kontinuierlich zu aktualisieren.

Da nicht jede Stadt die gleiche Zeitzone wie Zürich hat, musste man noch die Stunden ändern und eine Minuszahl oder eine grössere Zahl als 12 verhindern, da es sonst nicht mehr aufgehen würde. Wenn die Stadt in der Zeit zurück ist (z.B. New York), dann muss man sicher gehen, dass

```

let hours = now.getHours() - 6; // Minus 6 hours

// If the hours become negative, set them to 6 (for 12-hour format)
if (hours < 0) {
  hours = 6;
}

```

Code Ausschnitt für Zeit Berechnung

```

let hours = now.getHours() + 8; // Plus 8 hours

// If the hours exceed 12, adjust them to correspond to the 12-hour format by subtracting 12.
if (hours >= 12) {
  hours -= 12;
}

```

Code Ausschnitt für Zeit Berechnung

Auf dem Bild kann man den HTML-Code sehen, welchen alle Elemente der Uhr (das bedeutet Stundenzeiger, Minutenzeiger ...) definiert. Die einzelnen Teile der Uhr mussten im CSS auch noch positioniert und bearbeitet werden, damit verschiedene Dinge wie zum Beispiel die Grösse oder die Platzierung stimmt.

```

// Function to update the rotation of clock hands
function updateClockHand(selector, rotation) {
  const hand = document.querySelector(selector);
  hand.style.transform = `rotate(${rotation}deg)`;
}

// Function to update the clock hands based on the current time
function updateClock() {
  const now = new Date();
  const hours = now.getHours() % 12 || 12; // Convert to 12-hour format
  const minutes = now.getMinutes();
  const seconds = now.getSeconds();

  // Update rotation angles for each clock hand
  updateClockHand('.hour-hand', (360 / 12) * hours + (360 / 12) * (minutes / 60));
  updateClockHand('.min-hand', (360 / 60) * minutes);
  updateClockHand('.second-hand', (360 / 60) * seconds);
}

// Function to set a random background image based on the time of day
window.onload = function() {
  const now = new Date();
  const hours = now.getHours() % 24 || 24; // Convert to 24-hour format

  // Initial call to update the clock
  updateClock();

  // Update the clock every second
  setInterval(updateClock, 1000);
};

```

Code Ausschnitt von der Bewegung der Uhr

es keine Minuszahl wird, dass man ja subtrahieren muss. Auf der anderen Seite muss man bei Städten (z.B. Tokio) welche in der Zeit voraus sind, sichergehen, dass es keine grössere Zahl 12 gibt, da man

ja addieren muss. Wenn die Uhr 12 erreicht, wird sie wieder auf 0 gesetzt und beim oberen Beispiel auf die Differenz der Uhrzeiten.

Wetter Animation

Für die einzelnen Wetter Konditionen wollte ich noch eine kleine Animation einfügen, bei welcher zu der sich passende Bilder bewegen.

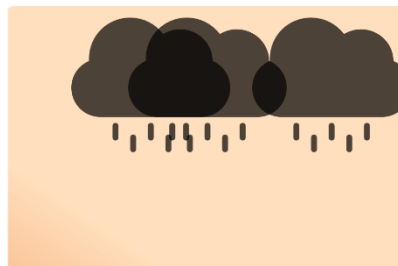
Damit sich die Bilder bewegen benutzte ich das CSS. Ich benutzte auch einige Beispiele im Internet und veränderte diese, damit es für mich passt.



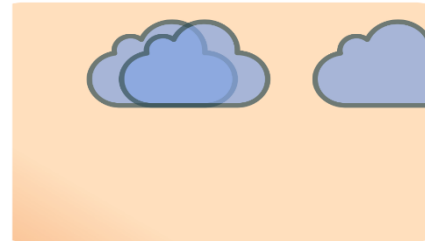
Wetter Animation Sonne



Wetter Animation Drizzle



Wetter Animation Regen



Wetter Animation Wolke

Die Animation moveSun wird mit @keyframes definiert, was bedeutet, dass eine Sequenz von Schlüsselbildern oder Zwischenzuständen definiert wird, die im Laufe der Animation durchlaufen werden.

```
@keyframes moveSun {
  0% {
    left: -100px;
  }

  /* Startposition */
  100% {
    left: calc(100% + 100px);
  }

  /* Endposition */
}

#sun {
  animation: moveSun 5s linear infinite;
  /* Animation starten */
}
```

CSS für Wetter Animation Sonne

Innerhalb des moveSun-Keyframes werden zwei Zustände definiert:

0%: Dies ist der Startzustand der Animation. Die Sonne wird anfangs um 100 Pixel nach links verschoben, außerhalb des sichtbaren Bereichs. Damit es aussieht, als würden sie von ausserhalb der Box kommen.

100%: Dies ist der Endzustand der Animation. Die Sonne wird nach rechts verschoben, sodass sie sich außerhalb des sichtbaren Bereichs befindet. Hier wird `calc(100% + 100px)` verwendet, um die Position der Sonne um 100 Pixel über die rechte Kante des sichtbaren Bereichs hinaus zu verschieben.

Das Element mit der ID sun wird dann animiert, indem ihm die Animation moveSun zugewiesen wird. Die Animation wird mit einer Dauer von 5 Sekunden (5s) ausgeführt, linear (linear) verlangsamt und unendlich (infinite) wiederholt. Das bedeutet, dass die Sonne alle 5 Sekunden linear von links nach rechts verschoben wird und die Animation unendlich oft wiederholt wird.

Im Bild kann man den Code sehen, damit die richtige Animation zu der entsprechenden Wetter Animation angezeigt wird.

```
const weatherCondition = result.weather[0].main;
let conditionHtml = '';

if (weatherCondition === "Rain") {
  conditionHtml += '';
  conditionHtml += '';
  conditionHtml += '';
} else if (weatherCondition === "Clouds") {
  conditionHtml += '';
  conditionHtml += '';
  conditionHtml += '';
} else if (weatherCondition === "Clear") {
  conditionHtml += '';
} else if (weatherCondition === "Drizzle") {
  conditionHtml += '';
  conditionHtml += '';
  conditionHtml += '';
} else if (weatherCondition === "Mist") {
  conditionHtml += '';
  conditionHtml += '';
  conditionHtml += '';
} else {
  conditionHtml += '';
}

$('#condition').html(conditionHtml);
```

JavaScript-Code für Wetter Animation

Zuerst wird die Hauptwetterbedingung (main) aus den empfangenen Daten extrahiert. Ich habe den gleichen Code wie bei der Wetter API benutzt, und musste so nur noch die success function ändern, wie man es im Bild sehen kann.

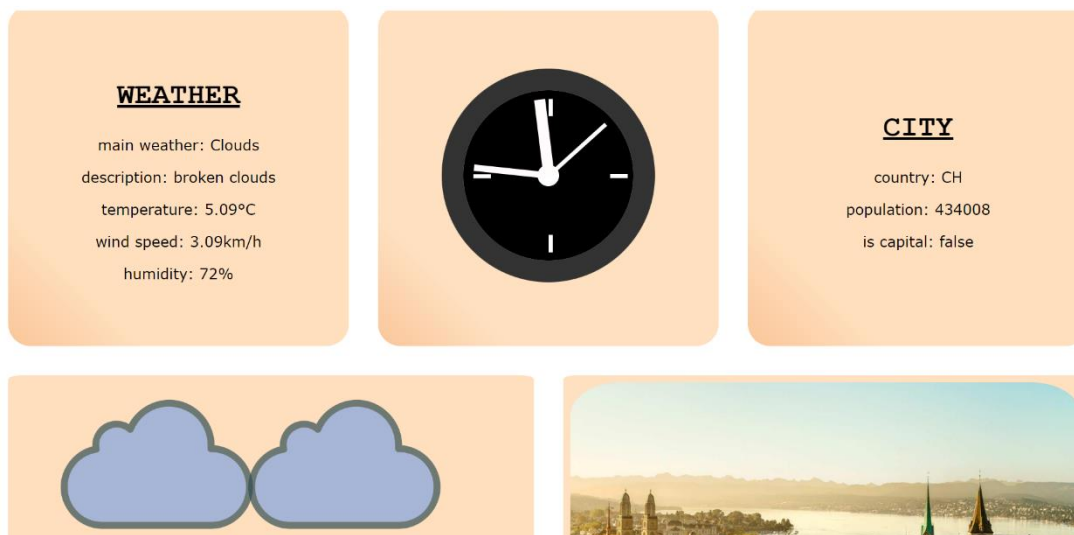
Anschliessend wird ein HTML-String (conditionHtml) erstellt, der Bilder entsprechend der Wetterbedingung enthält.

Es werden Bedingungen für verschiedene Wetterzustände überprüft:

- Bei Regen (Rain) werden drei Bilder von Regenwolken hinzugefügt.
- Bei Bewölkung (Clouds) werden drei Bilder von Wolken hinzugefügt.
- Bei klarem Himmel (Clear) wird ein Bild der Sonne hinzugefügt.
- Bei Nieselregen (Drizzle) werden Bilder von Nieselregenwolken hinzugefügt.
- Bei Dunst (Mist) werden ebenfalls Bilder von Wolken hinzugefügt.






Falls keine spezifische Bedingung zutrifft, wird standardmäßig ein Bild der Sonne hinzugefügt.



Schliesslich wird der erstellte HTML-String mit den entsprechenden Bildern in das Element mit der ID condition eingefügt, um die Wetterbedingungen anzuzeigen.



Ausschnitt aus der Seite von Zürich

Testplan

BESCHREIBUNG	WAS TESTEN	WIE TESTEN	ERGEBNIS
Buttons der Kopfzeile und Verknüpfungen	Testen ob alle Buttons funktionieren und alle Files miteinander und den entsprechenden Bildern verbunden sind.	Alle möglichen Wechsel und Verbindungen testen.	
API-Daten anzeigen	Schauen, ob alle Daten abgerufen werden und es angezeigt wird.	Zuerst im Postman den Link testen und danach auf meiner Website schauen, ob alles dort angezeigt wird.	Postman:  Website: Hat nicht funktioniert, da ich nicht die richtige Abfrage der API gemacht habe.
API-Abfragen testen	Schauen, ob ich dieses Mal die richtigen Abfragen gemacht habe.	Im Postman kontrolliere ich zuerst noch mal und schaue auch, was man alles Abfragen kann, und dann überprüfe ich ob auf meiner Website die richtigen Daten angezeigt werden.	
Überprüfung der analogen Uhr von Zürich	Schauen, ob die Uhrzeit mit der richtigen Zeit übereinstimmt.	Die aktuelle Uhrzeit mit der Uhrzeit, die auf der Uhr angezeigt wird, vergleichen.	
Die Uhrzeit aller anderen Städte überprüfen	Ob die Uhrzeit der einzelnen Städte stimmt.	Überprüfen, ob die aktuelle Uhrzeit dieser Zeitzone, gleich wie die angegebene Uhrzeit der Uhr ist.	 Hat nicht funktioniert, da die Uhrzeit je nach dem eine Minus Zahl oder eine Zahl über 12 Stunden sein konnte.

Überarbeitete Uhr mit richtiger Zeit	Uhrzeit sollte jetzt ohne Fehler für jede einzelne Stadt richtig sein.	Wieder vergleichen ob die aktuelle Uhrzeit mit der Uhrzeit der analogen Uhr übereinstimmt.	
Wetter Animation für einzelne Wetter Konditionen	Schauen, ob die richtige Animation für die einzelnen Wetter Bedingungen kommt.	Zuerst mit Postman schauen, ob ich für alle Wetter Bedingungen eine Animation habe. Und danach ob jeweils auch die richtige Animation auf meiner Website angezeigt wird.	

Probleme

Ich hatte eher weniger Probleme bei meinem Projekt. Mit der Wetter API hatte am meisten Probleme, da ich von dieser API keine Vorlage, wie bei der Stadtinfo API, gehabt habe. Ich habe viel Zeit aufgewendet mit dem API-Key, da wie schon vorher beschrieben, dieser ein wenig anders war als der von der Stadtinfo API.

```
method: 'GET',
url: 'https://api.openweathermap.org/data/2.5/weather?q=Zurich&appid=f39c7ffc405fe7cfff9462730f188fb1',
dataType: 'json',
success: function (result) {
```

Wie man im Bild oben sehen kann, musste ich einfach den API-Key gleich in den «url» mitgeben. Ich konnte ein Beispiel im Internet finden, bei dem jemand auch den API-Key gleich in den «url» mitgegeben hat. Ich probierte dies auch und war auch erfolgreich.

Bei der Analogen Uhr hatte ich auch ein Problem. Da nicht jede Stadt die gleiche Uhrzeit hat, musste ich die Stunde für die jeweilige Stadt ändern. Als ich dies aber machte, war es nicht ganz richtig und einige Uhren bewegten sich nicht mehr.

```
let hours = now.getHours() - 6; // Minus 6 hours

// If the hours become negative, set them to 6 (for 12-hour format)
if (hours < 0) {
  hours = 6;
}
```

Hier musste ich sichergehen, dass keine Minuszahl entsteht.

Hier musste ich sichergehen das keine grössere Zahl als 12 entsteht, da die Uhr ja im 12 Stunden Format ist.

```
let hours = now.getHours() + 8; // Plus 8 hours

// If the hours exceed 12, adjust them to correspond to the 12-hour format by subtracting 12.
if (hours >= 12) {
  hours -= 12;
}
```


Fazit

Ich denke das ich mit meinem Projekt erfolgreich war, da ich alle meine gesetzten Meilensteine erreicht habe und sogar noch einen vierten Meilenstein erreichen konnte. Ich hatte mich also nicht so gut eingeschätzt oder mein Projekt überschätzt, da ich relativ schnell war. Andere Meilensteine hätte ich mir aber setzten können, da diese nicht wirklich gut aufgeteilt waren. Dies wäre der einzige Punkt, den ich anders machen würde. Ich hatte auch nicht so viele Probleme, welche mich viel Zeit gekostet haben, was auch für den vierten Meilenstein verantwortlich war. Ich hätte auch noch mehr Städte hinzufügen könne und auf den vierten Meilenstein verzichten könne, jedoch fand ich es mit dem vierten Meilenstein spezieller und einzigartiger. Es war auch eine grössere und spannendere Aufgabe als nochmals das gleiche zu machen wie zuvor.

Insgesamt bin ich sehr stolz auf mich das ich sogar einen vierten Meilenstein erreichen konnte und auch mit dem Endergebnis sehr zufrieden, da es ja mein erstes richtiges Einzelprojekt ist.

Quellen

Wetter API	→	OpenWeatherMap.org
Wetter API Beispiel	→	Medium.com
Stadtinformation API	→	API-Ninjas.com
CSS-Beispiele und Infos	→	Mediaevent.de
Piktogramme für Wetter Animation	→	Icons.com
Hilfe und Ideen	→	Chat.openai.com
Rechtschreibung	→	Rechtschreibpruefung24.de

Anhang

[GitHub von meinem Projekt](#)