JEGYZŐKÖNYV

Web technológia alapjai

WatchOut karóra ismertető oldal

Készítette: **Szalóczy Krisztián**

Neptunkód: **Y4O4X0**

Dátum: 2024. május 11

**Miskolc, 2024**

**Tartalomjegyzék**

[Bevezetés](#_Bevezetés:) 1

[1.Mappa struktúra](#_1._Mappa_struktúra) 2

[1.1 feladat (kezdőlap)](#_1.1_feladat_(kezdőlap):) 3

[1.2 Videó megjelenítése](#_1.2_Videó_megjelenítése) 4

[1.3 Képek és információk megjelenítése táblázatban](#_1.3_Képek_és) 5

[1.4 Űrlap elemek használata és validációja](#_1.4_Űrlap_elemek) 6

[1.5 Az űrlap validációja JQuery használatával](#_1.5_Az_űrlap) 7

[2. Új HTML elem készítése](#_2._Új_HTML) 8

[2.1 Rolex, Orient, Cartier, Baume oldalak ismertetése](#_2.1_Rolex,_Orient,) 9

[2.2 Navigációs sáv](#_2.2_Navigációs_sáv) 10

[2.3 Image-slider megvalósítása](#_2.3_Image-slider_megvalósítása) 11

[2.4 A slider JavaScript implementációja](#_2.4_A_slider) 12

[3. Json fájlok és megjelenítésük az oldalon](#_3._Json_fájlok) 13

[3.1 Ajax használata az adtok betöltéséhez](#_3.1_Ajax_használata) 14

[3.2 Szorgalmi : node.js inicializálása](#_3.2_Szorgalmi:_node.js) 15

## Bevezetés:

Az általam készített projektben a karórák világát bemutató weboldal létrehozását választottam. Ez a döntésem arra vezethető vissza, hogy érdeklődéssel figyelem a divat és technológiai fejlődés találkozását, ami a karórák világában különösen izgalmasnak ígérkezik.

A karórák többek mint pusztán időmérő eszközök: egyúttal stílust és személyiséget is kifejeznek. Évszázadok óta hű társai az embereknek, és a technológiai előre lépesekkel párhuzamosan folyamatosan alakultak és fejlődtek. Ez a weboldal lehetőséget kínál arra, hogy átfogó képet kapjunk erről a fascináló világról, beleértve a történetüket, technikai jellemzőiket és a híres gyártóikat.

Az oldal célja, hogy könnyed, szórakoztató és informatív módon mutassa be a karórák világát. Szándékom, hogy a látogatókat magával ragadó élménnyel ajándékozzam meg, és egyben lehetőséget biztosítsak számukra arra, hogy mélyebben elmerüljenek ebben a témában. Így nem csupán egy egyszerű ismertető oldalt készítek, hanem egy olyan platformot, ahol a karórák szerelmesei felfedezhetik az újabbnál újabb karórákat.

A weboldal tervezése és fejlesztése során számos különböző technológia és eszköz kerül alkalmazásra annak érdekében, hogy egy modern és dinamikus felületet hozzak létre. Ennek részeként HTML-t és CSS-t használok az oldal struktúrájának és stílusának kialakításához, hogy a látogatók könnyen navigálhassanak és kellemes élményben részesüljenek az oldal böngészése során.

A felhasználói élmény fokozása érdekében beépítem a JavaScriptet és jQuery-t, amelyek segítségével interaktív elemeket és animációkat hozok létre az oldalon. Ez lehetővé teszi például a dinamikus tartalom betöltését, az interaktív űrlapokat vagy akár az animált effekteket, amelyek felhasználóbarát élményt biztosítanak.

## 1. Mappa struktúra

Fő könyvtár: WebTechY4O4X0

* assets
* jquery
* watches
  + baume
  + cartier
  + orient
  + rolex
* index.html
* app.js
* server.js
* style.css

assest: A weboldalon felhasznált képeket és videókat tartalmazza.

jquery: A JQuery használatához szükséges JavaScript fájlt tartalmazza. A jelenlegi legfrissebb 3.7.1 verzió, amely letölthető az alábbi linken: <https://jquery.com/>

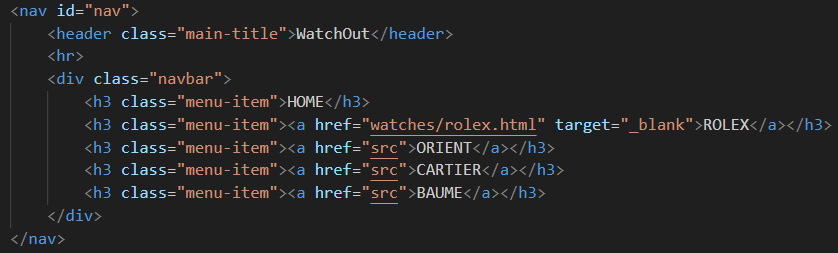
watches: További almappákat tartalmaz melyek a weboldalon megtekinthető html fájlok forrás kódját tartalmazzák , illetve a hozzájuk tartozó css, js, json fájlokat.

## 1.1 feladat (kezdőlap):

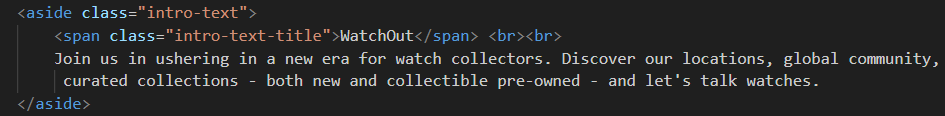
Általános weboldal célja, általános információk az órák típusairól. A weboldal szerkezete tartalmazza a következő HTML elemeket: *article, section, aside, nav, header, footer.* Az órák típusairól szóló információk táblázatba rendezve jelennek meg amit *internal CSS*-el formáztam.

<nav>- tag használata a navigációs sávhoz.

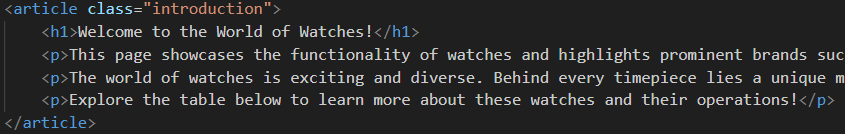
<header>- tag használata a fejléc megjelítéséhez



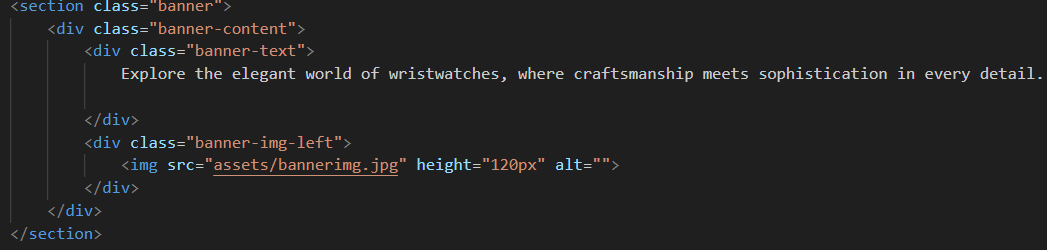
<aside>- tag használata a videón levő szöveg megjelenítéséhez



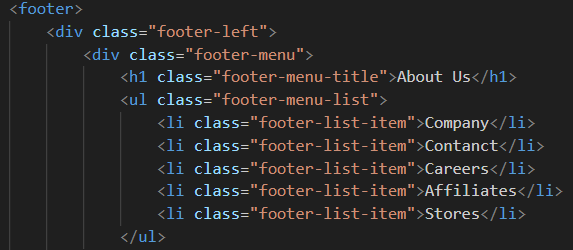
<article>-tag használat a bevezető szöveg megjelenítéséhez:



<section>-tag használata szövegek megjelenítésére:

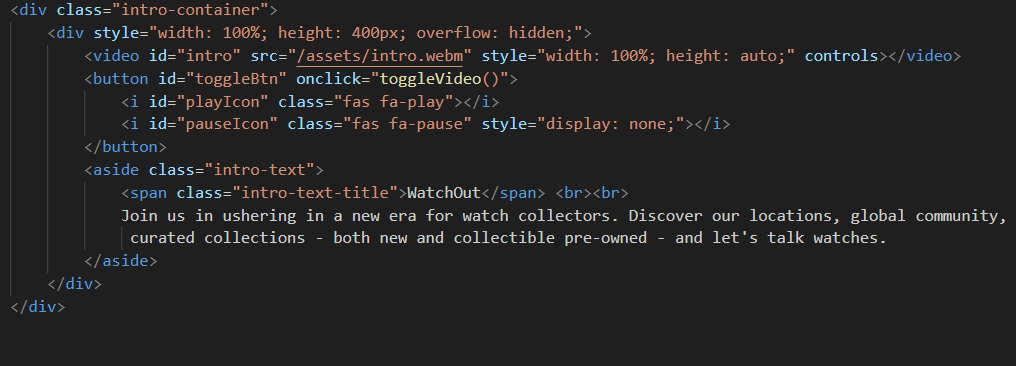


<footer>-tag használata a lábléc elkészítéséhez



## 1.2 Videó megjelenítése

A navigációs sáv után egy rövid videót helyeztem el, a <video>-tag használatával, melynek *inline CSS-el* adtam meg a méreteit. A videón elhelyeztem egy gombot <button> aminek az onclick attribútumát használva meghívásra kerül egy Javascript függvény, amit az app.js tartalmaz. Ez a függvény elindítja/megállítja a videó lejátszását illetve változtatja a megjelenített ikont.

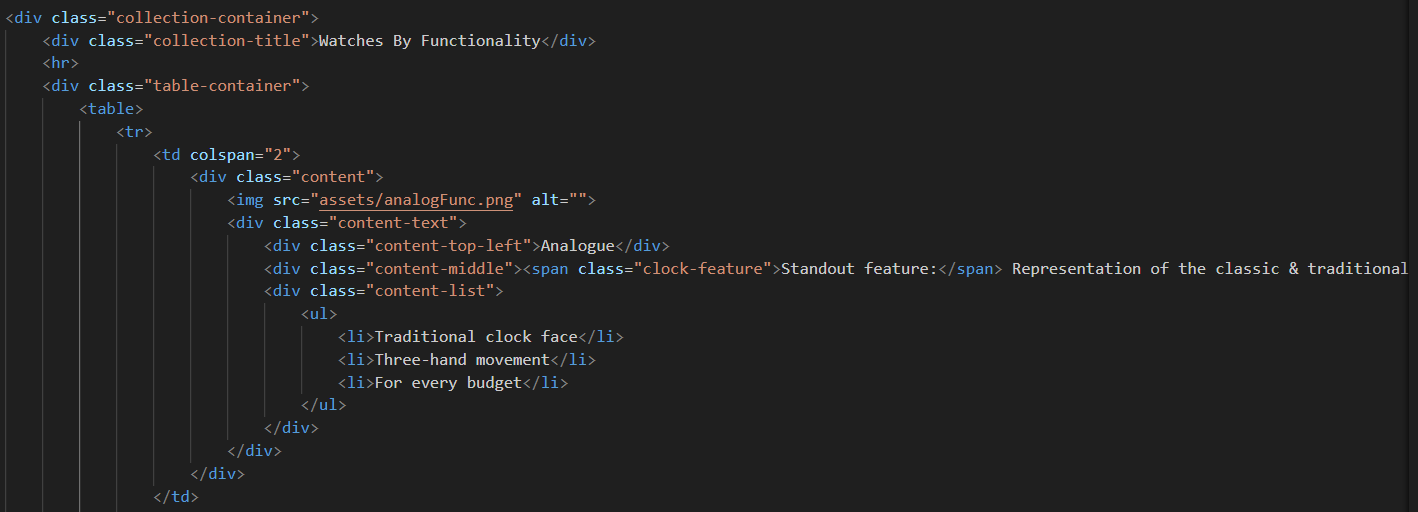


Az app.js *toggleVideo()* metódus implementációja:



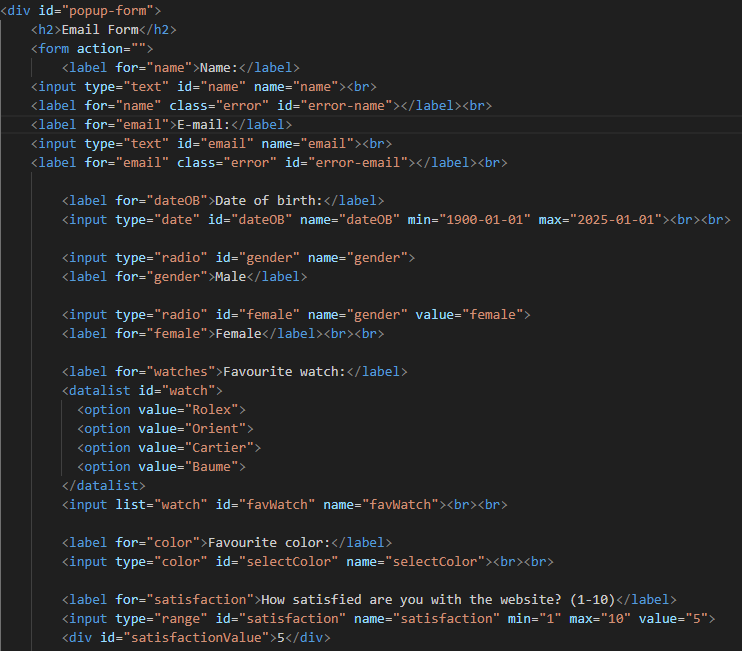
### 1.3 Képek és információk megjelenítése táblázatban

Az órák csoportosítását funkcionalitás alapján egy táblázatba rendezve jelenítettem meg. A táblázat 2 soros <tr> és egy sorban 2 cella <td> - van amin a colspan attribútumot használva annyi helyet foglal el mint 2 db cella. Egy cellában egy <div>-en belül van egy kép az óráról és egy másik <div>-ben az adott óra típus információ. A táblázatnak internal CSS-el tehát a <head>-tagen belül <style> páros tag-et használva adtam stílust.

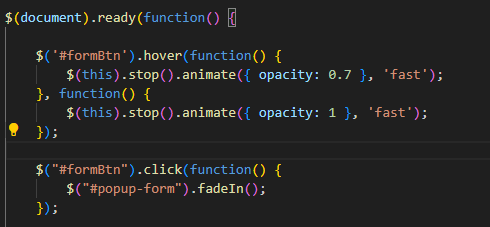
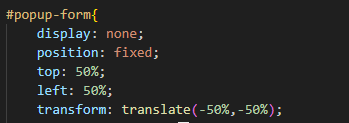


## 1.4 Űrlap elemek használata és validációja

Az űrlapot egy <div>-ben helyeztem el és egy azonosítóval id-val láttam el, amire későbbiekben fogok hivatkozni, mint css mint a JavaScript kódban. Az Űrlapon számos HTML elemet használtam a feladatnak megfelelően. A <form> -tagek között szerepelnek a feladatban megadott elemek, mint a szöveges beviteli mező egy/több soros, adatlista, jelölőnégyzet, rádió gomb, színválasztó, dátumválasztás, gombok.



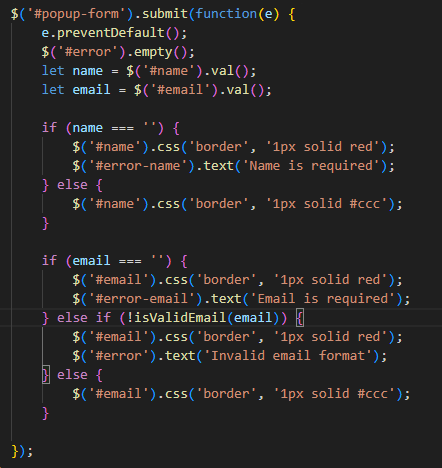
Alapvetően maga az ürlap nem jelenik meg (display:none) az oldalon. Az oldalon létrehoztam egy gombot aminek az id-ra hivatkozva JQuery-t használva jelenítem meg az oldalon az űrlapot. A click eseményt meghívva a gombra nyomva előtűnik (fade) az űrlap. Illeteve a gombra is raktam egy JQuery animációt, amelynél a hover függvényt használtam ezáltal az egeret a gomb fölé helyezve megváltozik az áttünése (opacity) a gombnak.



## 1.5 Az űrlap validációja JQuery használatával:

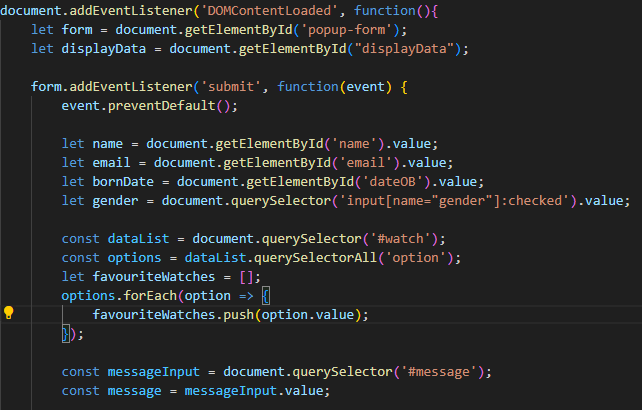
Az űrlap azonosítójára hivatkozva #popup-form JQuery-t használva megtudom hívni a submit esemény figyelőt. Amikor az űrlap beküldésre kerül, a **submit** eseményt megelőzően a kódrészlet meghívja az e.**preventDefault**() függvényt, hogy megakadályozza az alapértelmezett űrlap beküldési működését (oldal újratöltése). Ezután a kódban lévő input mezők értékeit ellenőrzi **(let name = $('#name').val();, let email = $('#email').val();).**

Ezután elvégzi a kötelező mezők (név és e-mail) validációját. Ha a név vagy az e-mail mező üres **(name === '', email === ''),** akkor piros szegéllyel jelez hibát **($('#name').css('border', '1px solid red');, $('#email').css('border', '1px solid red');),** és megjeleníti a megfelelő hibaüzenetet **($('#error-name').text('Name is required');, $('#error-email').text('Email is required');).** Ha az e-mail formátuma nem megfelelő (!isValidEmail(email)), szintén piros keretet és hibaüzenetet jelenít meg **($('#email').css('border', '1px solid red');, $('#error').text('Invalid email format');).**

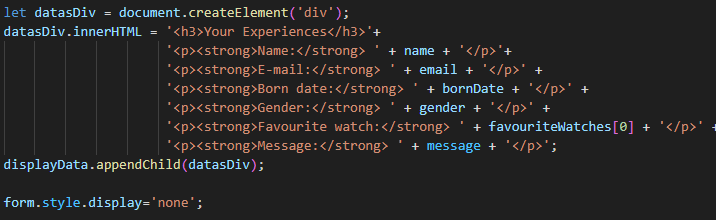


## 2. Új HTML elem készítése:

Az előzőekben bemutatott űrlapot kitöltve, a bevitt adatokat, ha azok megfelelőek akkor a submit gombra nyomva megjelenítem az oldalon. Amikor a dokumentum betöltődik (**'DOMContentLoaded'** esemény), az eseménykezelő függvény beállítja az űrlapot (**form**) és az adatok megjelenítésére szolgáló tartományt (**displayData**). Ezután hozzáad egy eseményfigyelőt az űrlap beküldéséhez (**form.addEventListener('submit', ...)**).

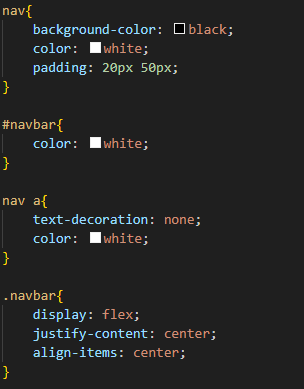
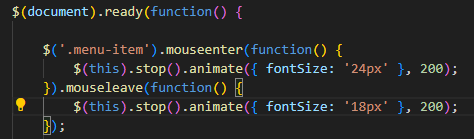


Ezután létrehozok egy új **div** elemet (**datasDiv**), amelyben összeállítom az adatokat egy HTML formában, majd ezt az elemet hozzáadom a **displayData** elemhez, hogy megjelenítse az összegyűjtött adatokat. Végül az űrlapot elrejti (**form.style.display='none';**). A kód feltételezi, hogy minden szükséges mezőt megfelelően kitöltöttek. Ha valamelyik mező nincs kitöltve, a kód hibákat okozhat.

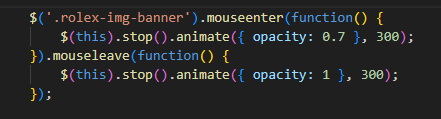
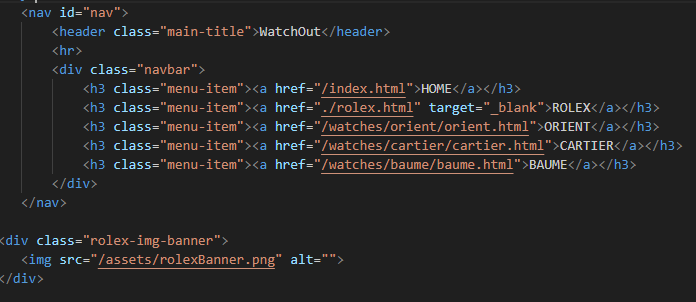


## 2.1 Rolex, Orient, Cartier, Baume oldalak ismertetése

Ezeknek a HTML oldalaknak a struktúrája egy forma. Mind az 5 oldalon látható a navigációs sáv, amin a linkek találhatóak amik a további oldalakra vezetnek. A navigációs sávhoz a <nav>-taget használtam. A stílus kialakításához a CSS-be különböző szelektorokat használtam, mint példáúl osztály, azonosító, elem kijelölő. Illetve jQuery-vel adtam hozzá egy animációt aminek hatásásra ha rávisszük az egeret az egyik opcióra annak megnövekszik a betűmérete.



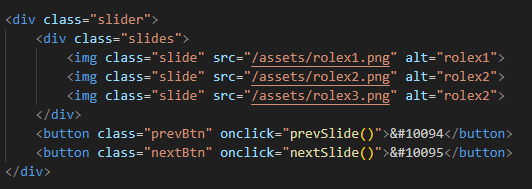
## 2.2 Navigációs sáv

A navigációs sávot követően egy képek szúrtam be ami az oldal teljes szélességében fedi le. Magára az <img>-tagre használtam egy osztály (class) jelölőt amit felhasználva a JavaScript kódban JQuery-vel könnyedén eltudtam érni és használni rá a mouseenter metódust. Ezzel egy animációt adtam hozzá amivel megváltoztatom az áttünését (opacity). Ha rávisszük az egeret akkor 0.7 változik 300 millisec alatt, majd ha levesszük róla akkor vissza áll 1-re.

A kép után egy szöveget helyeztem el egy div-ben, amit egyszerűen CSS- segítségével beállítottam az oldal közepére, sormagasságot, szint állítottam.

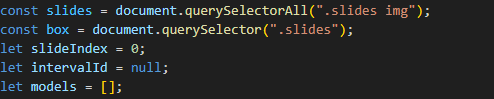
## 2.3 Image-slider megvalósítása

Az oldalon elhelyeztem egy kép csúsztatót, amin adott az oldal témájának megfelelő óra márkájának 3 modellje látható. Ezek automatikusan váltják egymást, illetve az oldalán elhelyeztem két gombot, amivel a felhasználó is tovább pörgetheti a képeket. A <div>-ek egymásba vannak ágyazva ezzel lehet elérni a megfelelő pozíciót az oldalon és maguknak a képeknek a pozícióját a <div>-ken belül.



## 2.4 **A slider JavaScript implementációja**

A JavaScript kódban létrehoztam két konstanst amiből az egyiknek értékül adtam querySelectorAll() metódusban megadott elem összes gyerekének a képeit, ez a függvény egy nodeList-et ad vissza ami az oldalon található összes diaképet tartalmazza. A másik konstansban eltároltam azt az elemet ami tartalmazza a csúszka összes képét ez az elem lesz a diavetítés konténere.



**let slideIndex = 0;:** Ez a változó tárolja az aktuálisan megjelenített kép indexét a diavetítés során. Kezdetben ez az érték 0.

**let intervalId = null;:** Ez a változó tárolja az időzítő (timer) azonosítóját, amely a diavetítés automatikus előre léptetéséért felelős. Kezdetben ez az érték null.

**let models = [];:** Ez egy üres tömb, amelyet később használhatunk a diavetítéshez kapcsolódó modelladatok tárolására, például képaláírások, linkek vagy egyéb információk. Ez a tömb azonban jelenleg üres, és nem tartalmaz semmilyen adatot.

**let choosenModel = models[0];**: Ez a változó tárolja a jelenleg kiválasztott modellt az alapértelmezett modellként. Kezdetben ez az első modell a **models** tömbből.

**const currentModelImg = document.querySelector(".model-img");**, **const currentModelTitle = document.querySelector(".model-title");**, **const currentRelease = document.querySelector(".model-release");**, **const currentDescription = document.querySelector(".model-description");**: Ezek a változók tárolják azokat a HTML elemeket, amelyek a jelenlegi modell képét, címét, kiadási dátumát és leírását jelenítik meg.

A **slides.forEach((item, index) => { ... });** ciklus bejárja az összes diavetítési képet, és mindegyikhez hozzárendel egy eseménykezelőt, amely a képre kattintáskor fut le. Ennek eredményeként a kiválasztott modell a kattintott képhez lesz rendelve, és az adatai megjelennek az oldalon.

Az **initializeSlider()** függvény meghívásra kerül, amely beállítja az alapértelmezett diavetítést és elindítja az automatikus előre haladást. A **DOMContentLoaded** esemény figyelője biztosítja, hogy a diavetítés csak akkor induljon, amikor az összes tartalom betöltődött az oldalon.

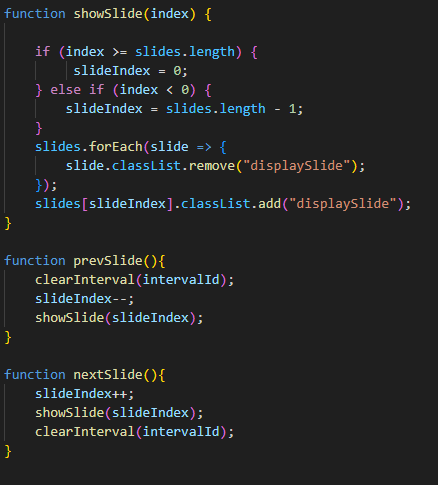


**showSlide(index)**: Ez a függvény megjeleníti a megadott indexű diavetítési képet. Ellenőrzi, hogy az index nem lépi-e túl a diavetítési képek számát, és visszaállítja, ha igen. Ha az index negatív, akkor a legutolsó diavetítési képre ugrik. Ezután eltávolítja az összes diavetítési képet tartalmazó **.displaySlide** osztályt és hozzáadja azt az aktuális diavetítési képhez.

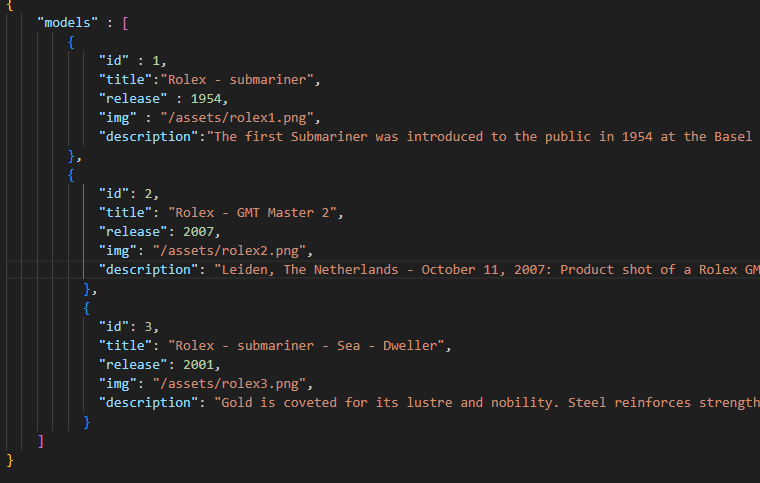
**prevSlide()**: Ez a függvény előreugrik egy diavetítési képpel az aktuális index alapján. Megállítja az automatikus előre haladást, csökkenti az indexet egyel, majd meghívja a **showSlide()** függvényt az új indexszel.

**nextSlide()**: Ez a függvény hátraugrik egy diavetítési képpel az aktuális index alapján. Növeli az indexet egyel, majd meghívja a **showSlide()** függvényt az új indexszel. Előtte azonban megszakítja az automatikus előre haladást, hogy ne zavarja a felhasználót.

A HTML-en belül létrehoztam két gombot amiknek az onclick attribútumába meghívom a prevSlide() és a nextSlide() JavaScript függvényeket a képek tovább pörgetésére.



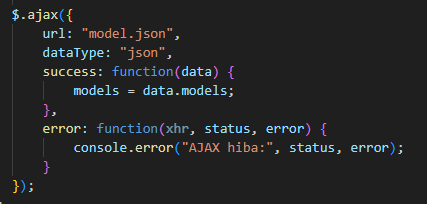
## 3. Json fájlok és megjelenítésük az oldalon



## 3.1 **Ajax használata az adtok betöltéséhez**

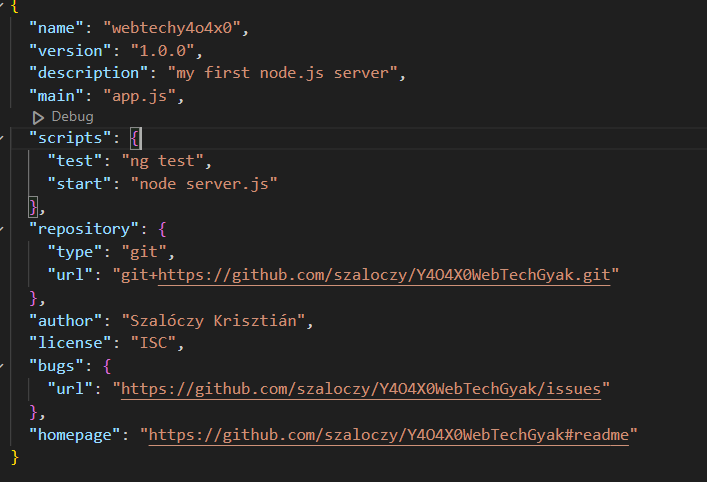
A weboldalakon szereplő óra márkák modelljeinek adatait egy model.json fájlban tárolom. Amit a feladatnak megfelelően egy Ajax kéréssel töltök be az oldalra. Az url amelyre a kérés megy ebben az esetben a model.json fájlra mutat. A dataType a várt adat formátumot jelöli. A success egy függvény amely akkor fut le ha az AJAX kérés sikeresen végrehajtódik. Az itt megadott **data** paraméter tartalmazza a válaszként kapott adatokat. Ebben az esetben az adatokat egy **models** változóba mentem el.

**error**: Egy függvény, amely akkor fut le, ha az AJAX kérés hibával tér vissza. Ez az eset lehetőséget ad a hibakezelésre, és a felhasználó értesítésére a problémáról. Ez a függvény megkapja a hibát leíró paramétereket, mint például a HTTP státuszkódot és az üzenetet.



## 3.2 Szorgalmi: node.js inicializálása, package.json, server.js, package-lock.json

Az **’npm** **init’** parancsot kiadtam terminálban, amivel egy új **Node**.**js** projektet inicializálok. Ezután a **Node** **Packege** **Manager** számos kérdést feltett a konfigurációval kapcsolatban. Ezek közé tartoznak például a projekt neve, verziószáma, leírása, fő fájl neve és egyéb metainformációk. Ennek eredménye ként létre jött a **package**.**json** ami a konfigurációt és a függőségeket fogja tartalmazni.



Az **npm** **install** parancsot kiadva létre jön a **package-lock.json**, ez a fájl pontosan rögzíti, hogy mely verziókban települtek le a függőségek, valamint azok függőségei és azok verziói.

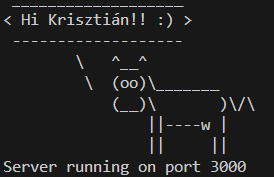


Létrehoztam a **server**.**js** fájlt ami tartalmaz gy egyszerű Node.js szerveralkalmazást definiál, amely a beépített **http**, **fs** és **path** modulokat használja a szerver működtetéséhez és a statikus fájlok szolgáltatásához. Ezek a modulok részei a Node.js alaptelepítésnek, és nem szükségesek külön telepíteni őket az **npm**-en keresztül.



A **node**\_**modules** mappa létrehozásához be kell húznom egy függőséget. Egy szimpla modult választottam a <https://www.npmjs.com/package/cowsay> oldaláról, aminek az a neve hogy **cowsay.** Az **npm** **install** **cowsay** paranccsal lehúztam ezt a modult a **node**.**js** projektembe.

Ha futtatom a **node** **server**.**js** paranccsal akkor kifogja rajzolni a konzolra a **cowsay**.**say** függvény által visszaadott értéket.

****