Széchenyi István Egyetem

Gépészmérnöki, Informatikai és Villamosmérnöki Kar

Informatika Tanszék

**SZAKDOLGOZAT**

**Borbély Bálint**

**Mérnök informatikus BSc szak**

2024



**SZAKDOLGOZ****AT**

**Appointy időpontkezelő webkomponens**

**Borbély Bálint**

**Mérnök informatikus BSc szak**

**2024**

**A képen szöveg, képernyőkép, Betűtípus, levél látható

Automatikusan generált leírás**

# Nyilatkozat

Alulírott Borbély Bálint (VVO4KH), Mérnökinformatikus Bsc. szakos hallgató kijelentem, hogy az Appointy időpontkezelő webkomponens című szakdolgozat feladat kidolgozása a saját munkám, abban csak a megjelölt forrásokat, és a megjelölt mértékben használtam fel, az idézés szabályainak megfelelően, a hivatkozások pontos megjelölésével.

Eredményeim saját munkán, számításokon, kutatáson, valós méréseken alapulnak, és a legjobb tudásom szerint hitelesek.

Győr, 2024. 12. 09.

hallgató

# Kivonat

Appointy időpontkezelő webkomponens

A dolgozat egy népszerű online naptárakat összekötő, és azok funkcióit a fejlesztők és felhasználók számára összhangba helyező webkomponens elkészítésének menetét, funkcionalitását és a felhasznált technológiáit fogja bemutatni. A komponens fejlesztésével a cél az, hogy a webfejlesztők egy olyan eszközt kapjanak, amelynek segítségével a cég, amelynek a weboldalt fejlesztik a felhasználóinak egységes felületen tudja megmutatni a csarnokaik foglaltságát, a fogorvos szabad időpontjait, vagy az egyetemi rendezvényeket a naptáraikból, amelyek akár több naptár között is szét lehetnek szórva. A komponens konfigurálható a felhasználási cél elérése érdekében, például, ha van ilyen megadva, akkor a felhasználók ezen keresztül csak a kritériumoknak megfelelő időpontfoglalásra tehetnek kérelmet, a megfogalmazott kritériumoknak megfelelve, amit az erre illetékes személy később jóváhagyhat, vagy elutasíthat.

# Abstract

Appointy appointment management web component

This paper will describe the process, the features and the technologies used to create a web component that connects online web calendars and harmonizes their functionality for business use, for developers and users. By developing the component, the goal is to provide web developers with a tool that allows the company whose website is being developed to show their users in a single interface the occupancy of their halls, the dentist's free appointments, or university events from their calendars, which may be scattered among several calendars. The component may be configured to achieve intended use, for example, if provided, users can request an appointment booking through the component itself, according to the criteria defined, which can be later approved or rejected by the person responsible.

# Tartalomjegyzék

[Nyilatkozat 4](#_Toc184647243)

[Kivonat 5](#_Toc184647244)

[Abstract 6](#_Toc184647245)

[Tartalomjegyzék 7](#_Toc184647246)

[1. Bevezetés 1](#_Toc184647247)

[1.1. A naptár 1](#_Toc184647248)

[1.2. Online naptár 1](#_Toc184647249)

[1.3. Vállalatok haszna 2](#_Toc184647250)

[2. Online naptár szolgáltatás 3](#_Toc184647251)

[2.1. Google Calendar 3](#_Toc184647252)

[2.1.1. A Google Calendar múltja 4](#_Toc184647253)

[3. A projekt oka és célja 6](#_Toc184647254)

[4. Technológiák 8](#_Toc184647255)

[4.1. Az oldal felépítésére és stílusára vonatkozó technológiák 8](#_Toc184647256)

[4.1.1. HTML 8](#_Toc184647257)

[4.1.2. CSS 10](#_Toc184647258)

[4.1.3. SCSS 11](#_Toc184647259)

[4.2. Az oldal viselkedését befolyásoló technológiák 14](#_Toc184647260)

[4.2.1. JavaScript 14](#_Toc184647261)

[4.2.2. TypeScript 16](#_Toc184647262)

[4.3. A felhasznált keretrendszer 17](#_Toc184647263)

[4.4. A használt csomagkezelő 18](#_Toc184647264)

[4.5. A használt fejlesztői környezet 19](#_Toc184647265)

[5. Fejlesztés 21](#_Toc184647266)

[5.1. Naptár elrendezése 21](#_Toc184647267)

[5.1.1. Alapelrendezés 21](#_Toc184647268)

[5.1.2. Túlcsodrulás 22](#_Toc184647269)

[5.1.3. Naptártest felépítése 24](#_Toc184647270)

[5.1.4. Görgetés 25](#_Toc184647271)

[5.2. A megjelenítés logikai elemei 27](#_Toc184647272)

[5.2.1. Hét napjai 27](#_Toc184647273)

[5.2.1.1. Weekdays 29](#_Toc184647274)

[5.2.1.2. Dates 30](#_Toc184647275)

[5.2.2. Megjelenített hónap 33](#_Toc184647276)

[5.2.3. Lapozás 35](#_Toc184647277)

[5.3. Események 36](#_Toc184647278)

[5.3.1. Lekérés 36](#_Toc184647279)

[5.3.2. Megjelenítés 38](#_Toc184647280)

[5.3.3. Esemény létrehozás űrlap 44](#_Toc184647281)

[5.3.4. Validátorok 45](#_Toc184647282)

[5.3.5. E-mail 53](#_Toc184647283)

[5.3.6. Megegyezés 55](#_Toc184647284)

[5.4. Testreszabás 58](#_Toc184647285)

[6. Összegzés 63](#_Toc184647286)

[Irodalomjegyzék 1](#_Toc184647287)

[Ábrajegyzék 4](#_Toc184647288)

[Mellékletek 7](#_Toc184647289)

# Bevezetés

## A naptár

Az emberiség már évezredek óta használ naptárakat[1], hogy rendszerezett módon tartsák nyilván az idő múlását és az eseményeket. A naptárak segítenek nekünk az életünk szervezésében, a teendőink hatékony tervezésében és megfelelő időbeosztás kialakításában. Az idő az egyik legértékesebb erőforrásunk, amivel nem mindegy mit, hogyan és miért kezdünk, és a naptárak segítenek abban, hogy hatékonyabban kihasználhassuk a nekünk jutott időt.

## Online naptár

Az online naptárak használata különösen előnyös lehet számos szempontból. Először is, az online naptárak könnyen hozzáférhetők bárhol és bármikor, amíg internetkapcsolat rendelkezésre áll. Ez lehetővé teszi, hogy bárhonnan elérjük a naptárunkat, legyen szó akár otthonunkról, munkahelyünkről vagy egy vonatról út közben.

Másodszor, az online naptárak szinkronizálhatók más eszközökkel, például okostelefonokkal vagy számítógépekkel. Ez azt jelenti, hogy ha egy eseményt vagy találkozót adunk a naptárunkhoz egy eszközön, az azonnal megjelenik és frissül az összes többi eszközön is, amelyen be van jelentkezve ugyanaz a naptár. Ez segít abban, hogy mindig naprakészek legyünk, és ne felejtsünk el fontos eseményeket vagy találkozókat.

Harmadszor, az online naptárak gyakran rendelkeznek automatikus emlékeztető funkcióval. Ez azt jelenti, hogy beállíthatjuk, hogy a naptár automatikusan értesítést küldjön nekünk egy esemény vagy találkozó előtt, így biztosítva, hogy ne felejtsük el időben felkészülni rá.

Az online naptárak emellett könnyen megoszthatók másokkal. Például egy család vagy egy csoport munkatársai között lehetőség van arra, hogy egy közös naptárat használjanak, így mindig tisztában lehetnek egymás teendőivel és időbeosztásával. Ez segíthet abban, hogy jobban együttműködjenek és hatékonyabban szervezzék meg az eseményeiket és találkozóikat.

Az online naptárak használata tehát számos előnnyel jár a fizikai, papír alapú naptárakkal szemben. Rugalmasságot, hozzáférhetőséget és szinkronizációt biztosítanak, amelyek segítenek abban, hogy hatékonyan szervezzük meg az életünket és időnket. Természetesen hátránya is van, nevezetesen a már megemlített szükséges internetkapcsolat, és ide sorolható még a számítógép, okostelefon, vagy táblagép, amellyel rendelkeznünk kell, ha ezeket az online naptárakat igénybe szeretnénk venni. Véleményem szerint azonban az előnyök súlya messze meghaladja a hátrányokét, így én is online naptárakat használok.

## Vállalatok haszna

Az online naptárak használata vállalatok számára is kiváló lehetőségeket kínál, ezek bevezetése és használata számos előnnyel járhat egy vállalat számára.

Először is, az online naptárak lehetővé teszik a hatékonyabb időbeosztást és tervezést a vállalati tevékenységek számára épp úgy, ahogy magánszemélyek számára is. Az alkalmazottak könnyen megoszthatják a naptárakat egymással, lehetővé téve számukra, hogy lássák egymás elfoglaltságait és időbeosztásait. Ez segíthet abban, hogy elkerüljék az időpontbéli ütközéseket, és hatékonyabban szervezzék meg a találkozóikat, megbeszéléseiket és egyéb eseményeket.

Másodszor, az online naptárak lehetővé teszik a rugalmasabb munkavégzést. Az alkalmazottak könnyebben hozzáférhetnek a naptárakhoz bárhol és bármikor, ami lehetővé teszi számukra, hogy rugalmasabban alakítsák ki a munkaidejüket és az ütemtervüket. Ez különösen fontos az olyan vállalatok számára, amelyek rugalmas munkarendet biztosítanak a munkavállalóiknak, vagy amelyeknek globális szinten kell együttműködniük különböző időzónákban dolgozó munkatársaikkal.

Végül, az online naptárak segíthetnek abban, hogy javuljon a hatékonyság és a produktivitás a vállalaton belül. Az alkalmazottak könnyebben tervezhetik meg a munkaidőt és az ütemtervet a naptárak segítségével, és kevesebb időt tölthetnek az időpontok összehangolásával és az ütközések kezelésével. Ez lehetővé teszi számukra, hogy több időt fordítsanak az érdemi munkára és a vállalat céljainak elérésére, ami hozzájárulhat a vállalati teljesítmény és hatékonyság növekedéséhez.

# Online naptár szolgáltatás

A felsorolt érvek alapján könnyen elhihető tény, hogy rengeteg, elsősorban informatikai vállalat döntött úgy, hogy készít egy ilyen online naptárat, amelynek használatát elérhetővé teszi a világ számára is. Ezek mindegyike természetesen nem puszta szívjóságból, hanem profitszerzés céljából teszi ezt, viszont nehéz lehet választani közülük, hiszen annyi elérhető opció van.

## Google Calendar

Az egyik lehetőség, amely az egyik legnépszerűbb is egyben a Google által kínált naptár, a Google Calendar. Ez az egyik legelterjedtebb online naptárkezelő szolgáltatás, amit az egész világon használnak. A képen egy interneten talált példa látható a Google Calendar felületére.

[2]



1. ábra: A Google Calendar felülete [2]

### A Google Calendar múltja

Ahogy már írtam, a Google Calendar a Google által kifejlesztett időmenedzsment és időpontkezelési naptárszolgáltatás, amelynek múltjáról remek áttekintést nyújt ennek az összefoglaló a forrása [3]. Mike Samuel (2. ábra)[4] hozta létre a Google-nál végzett 20%-os projektjének részeként (a 20%-os projekt felelős a Google számos szolgáltatásának fejlesztéséért. Az alapítók, Sergey Brin és Larry Page azt tanácsolták, hogy a dolgozók „idejük 20%-át azzal töltsék, amiről úgy gondolják, hogy a Google számára a leghasznosabb”. [5]). 2006. április 13-án bétaverzióban, majd 2009 júliusában általános kiadásban vált elérhetővé a weben és mobilalkalmazásként Android és iOS platformokra.

2. ábra: Mike Samuel, a Google Calendar atyja [3]

Kezdetben csak weben keresztül, és Android operációs rendszeren volt elérhető, 2015. március 10-én azonban megjelent az iOS-alkalmazás is.

2010 decemberében a Google lehetővé tette a felhasználók számára, hogy kiválasszák a létrehozott események időzónáját, ami korábban hiányzott, és hiányát a média is kritizálta.

2015 augusztusában a Google hozzáadta az „Events from Gmail” (Események a Gmail-ból) funkciót, amelynek segítségével a felhasználók Gmail üzeneteiből származó eseményinformációk automatikusan bekerülnek a Google Naptárba. Az alapértelmezés szerint engedélyezett funkció az újonnan beérkezett e-mail üzenetek, például járatkésések figyelembevételével frissíti is az eseményeket új információkkal.

2015 decemberében a Google hozzáadta a „Reminders” (Emlékeztetők) funkciót, amely lehetővé teszi a felhasználók számára, hogy a teendőiket emlékeztetőként adják hozzá a naptárhoz, és ezek a tevékenységek az eddig megszokott események mellett megjelennek. A Reminders úgynevezett szolgáltatásközi funkcióként szolgál, ami azt jelenti, hogy az emlékeztetők a Gmail beérkezett üzenetei közt, a Google Now-ban és a Google Keep-ben is megjelennek.

2016 januárjában a Google a mobilalkalmazásokban a Google Naptárat kiegészítette „Intelligens javaslatokkal” (Smart Suggestions). Az intelligens javaslatok az események címeit, valamint hozzájuk tartozó helyszíneket és kapcsolatokat ajánlanak fel. Az intelligens javaslatokkal egy időben a Google 54 új ország ünnepnaptárát is hozzáadta, így összesen 143 országspecifikus ünnepnaptárral rendelkeznek.

2016 áprilisában a „Goals” (Célok) funkció következett. A célok olyan tevékenységek, amelyeket a felhasználó szeretne elvégezni. Néhány rövid kérdés megválaszolása után (például a „Milyen gyakran?” és a „Legjobb időpont?”) a naptár „megtalálja a legjobb időablakokat a cél eléréséhez”, és idővel alkalmazkodik is a felhasználó időbeosztásához, például átütemezi a célt szolgáló tevékenységet, ha olyan esemény kerül felvételre, amely közvetlenül ütközik a cél időpontjával. A funkció 2017 januárjában bővült a Google Fit és az Apple Health támogatással.

2017 márciusában az iOS-alkalmazás frissült, hozzáadva az iPad támogatást, majd júliusban ismét frissült, hogy az iOS „Today” paneljéhez widgetet adjon hozzá.

2017 júniusában, miután májusban bejelentette, hogy a Google több szolgáltatásában is bevezette a „Family Groups” funkciót, a Google elkezdte a családi naptárak bevezetését a felhasználók számára.

2022. szeptember 20-án a Google bejelentette, hogy a Google Calendar és a Google Assistant emlékeztetői egyesülnek a Google Tasks-al. Az átállás 2023 márciusában kezdődött néhány felhasználó kiválasztásával, majd 2023 második negyedévében opcionális lehetőségként a többi felhasználóra is kiterjesztették, és 2023. május 22-es befejezéssel a vállalati szektor számára.

# A projekt oka és célja

A meghatározóbb online naptárak bemutatását és felsorolását követően magáról a projektről, ezen belül is a projekt létrehozásának okáról és céljáról lesz szó.

A projekt célja alapvetően egy olyan webkomponens létrehozása, ami vállalatok és fejlesztők számára több online naptárat köthet össze. Ez azt is jelenti, hogy ezáltal lehetővé teszi a felhasználók számára, hogy egyszerűen és hatékonyan kezeljék az adott céggel kapcsolatos időpontokat, így számos előnnyel jár mind a felhasználók, mind a fejlesztők számára. A modern digitális környezetben az időgazdálkodás és az események szervezése kulcsfontosságú szerepet játszik mind az egyének, mind a vállalkozások számára. Ezért egy olyan platform létrehozása, amely egyszerre összekapcsolja és egységesíti a különböző online naptárszolgáltatásokat, és az azokban található időpontokat, jelentős előrelépést jelenthet a vállalat és a vállalat ügyfelei számára időgazdálkodás és az eseménytervezés terén.

Egy felhasználási eset, hogy a weboldalakba beágyazott naptárak legtöbb esetben (a Google és a Microsoft esetében is) csak a már létrehozott események megtekintésére, esetleg a naptár saját naptárhoz való hozzáadására alkalmasak, a felhasználó nem tud velük interakcióba lépni, sem új időpontfoglalást kezdeményezni.

Még egy előnye ennek a webkomponensnek, hogy a fejlesztőknek nem kell sem külső oldalra navigálni a felhasználókat a naptárak megtekintéséhez, és több naptár használata esetén nem kell több naptárat sem beágyazni a weboldalba, hiszen ezzel a komponenssel egy felületen jeleníthetik meg őket. Ezáltal minimalizálódik az időveszteség és a folyamatok bonyolultsága, ami hozzájárul az üzleti hatékonyság növekedéséhez.

Egy olyan platform létrehozása, amely lehetővé teszi több naptár közötti naptáresemények egységes összehangolását, segít megelőzni a különböző időpontok ütközését és a kényelmetlenségeket, amelyek gyakran felmerülnek az ütemezés során. Ezáltal javul a szervezettség és az időgazdálkodás hatékonysága mind az egyének, mind a vállalkozások számára.

A webkomponens segítségével egyszerűen lehetőség nyílik az időpontfoglalási kérelmek kezelésére és az időpontok egyszerű elbírálására, ami jelentős idő- és energia megtakarítást jelent mind a felhasználók, mind a szolgáltatók számára.

A felhasználók számára az időpontok elutasításáról, elfogadásáról, mozgatásáról, szerkesztéséről küldött értesítés lehetővé teszi, hogy mindig naprakészek legyenek az aktuális eseményekkel kapcsolatban, és könnyedén reagálhassanak a változásokra.

A sokszínű paraméterezhetőség lehetővé teszi a fejlesztők számára, hogy testre szabják a komponenst az egyes szervezetek és felhasználók igényei szerint, így maximalizálva a funkcionalitást és az alkalmazkodóképességet. Megadhatnak több használandó naptárat, paraméterezhetik az időpontok ütközésének engedélyezését, az időpontok maximális és minimális hosszát, és még rengeteg mást, ami könnyíti az időpontkérések validálását mielőtt az megérkezne az időpontok kezelőjéhez.

Terv szerint a komponens a kinézetének személyre szabására is ad majd lehetőséget, hogy minden fejlesztő az adott cég színpalettáját használhassa a komponens színeiként, ezzel is jobban integrálva azt az oldalba, és az ismerősség érzését nyújtva a felhasználóknak.

Összességében, a webkomponens létrehozása, ami összeköti az online naptárakat és lehetővé teszi az időpontok kezelését és foglalását, számos előnnyel jár mind a felhasználók, mind a fejlesztők számára. Ez jelentős hozzáadott értéket képvisel a modern digitális környezetben, hiszen a gördülékeny, akadálymentes ügyintézés az egyik legfontosabb szempont, amikor egy felhasználó egy vállalattal kapcsolatba lép. Ezen a gördülékenység hiánya egy olyan elem, amely akár el is rémisztheti a potenciális ügyfeleket a vállalattal való kapcsolatba lépéstől, és végső soron az üzlet megköttetésétől is.

# Technológiák

Most, hogy áttekintettük a projekt céljait és előnyeit, mélyebben belemerülünk a technikai részletekbe. A projekt során fontos szerepet játszik a megfelelő technológiai alapok kiválasztása, amelyek lehetővé teszik a webkomponens hatékony és kifinomult működését. Számos különböző technológia áll rendelkezésre, amelyek közül választhatunk, attól függően, hogy milyen szintű funkcionalitást és skálázhatóságot szeretnénk elérni. A felhasználói felület kialakításához és az ott végbemenő interakciók kezeléséhez különféle frontend keretrendszereket (például a React vagy Vue.js) lehet alkalmazni. Fontos az is, hogy a projektet felhőalapú szolgáltatások segítségével is kiegészítsük, például az e-mail értesítések küldéséhez. Összességében a technológiai stack kiválasztása és az ezek közötti megfelelő integráció kulcsfontosságú a projekt sikeréhez és a felhasználói élmény maximalizálásához.

## Az oldal felépítésére és stílusára vonatkozó technológiák

### HTML

A HTML (3. ábra, 4. ábra) a HyperText Markup Language rövidítése, és ez a webböngészőben történő megjelenítésre tervezett dokumentumok szabványos jelölőnyelve. A HTML dokumentumok olyan struktúrát határoznak meg, amelyek meghatározzák egy weboldal szerkezetét, és lehetővé teszik a tartalom hierarchiájának leírását. A HTML elemeket tag-ekkel alakítjuk ki, amelyek különböző tulajdonságokkal rendelkeznek és egymásba ágyazhatók. Az elemek egyedi azonosítókkal (id) és osztályokkal (class) rendelkezhetnek, amelyek segítségével stílusokat és formázásokat lehet alkalmazni rájuk az inline stílusadás mellett. Gyakran olyan technológiák segítik, mint a CSS (Cascading Style Sheets) és az olyan szkriptnyelvek, mint a JavaScript. [6]

3. ábra: HTML5 logó [6]

A webböngészők a HTML-dokumentumokat webkiszolgálóról vagy helyi tárolóból fogadják, és a dokumentumokat multimédiás weboldalakká renderelik.

A HTML-elemek a HTML-oldalak építőkövei. A HTML-elemek segítségével képek és más objektumok, például interaktív űrlapok ágyazhatók be a megjelenített oldalakba. A HTML-elemeket a „kisebb, mint” és a „nagyobb, mint” relációs jelek, avagy „kacsacsőrök” segítségével írt úgynevezett „tag”-ek, vagy magyarul címkék jelölik ki. Az olyan tagek, mint például a <img> és a <input> közvetlenül „önmaguktól” vezetik be a tartalmat az oldalba, más címkék, mint például a <p> és ennek a befejező párja a </p> körülveszik a dokumentum bizonyos részeit, és azzal kapcsolatos információkat határoznak meg. Ezesetben például a <p> a benne lévő szövegről azt árulja el, hogy új bekezdésként (paragrafusként) kell értelmezni, és sort kell emelni előtte. Az ilyen körülvevő tagek tartalmazhatnak altag-eket is, amik olyan tagek, amelyek körül vannak véve a szülő tag-gel, például: <div><p></p></div> esetében a <div> a szülő tag, és a <p> a benne található altag. A böngészők nem jelenítik meg ezeket a HTML-tageket, hanem az oldal tartalmának értelmezéséhez és formázásához használják fel őket. [7]

A következő linken rengeteg információt lehet találni a HTML szabványról, amely a dolgozatba már nem fér bele: [8].

<!DOCTYPE html>

<html lang="hu">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <h1>Hello World!</h1>

</body>

</html>

4. ábra: Példa HTML kódra (saját)

### CSS

A CSS (5. ábra, 6. ábra), avagy Cascading Style Sheets egy olyan stíluslap-nyelv (style sheet language) amelyet egy jelölőnyelvvel, például HTML-lel vagy XML-lel (beleértve az olyan XML-fajtákat, mint az SVG, MathML vagy az XHTML) írt dokumentum megjelenítésének és stílusának meghatározására használnak. A CSS segítségével megadhatók a különböző elemek méretei, színei, betűtípusai, elrendezései és egyéb vizuális tulajdonságai. A CSS stílusok külön fájlokban vagy közvetlenül a HTML dokumentumokban is elhelyezhetők. A CSS a HTML és a JavaScript mellett a World Wide Web egyik alaptechnológiája.

5. ábra: CSS3 logó [9]

A CSS-t úgy tervezték, hogy lehetővé tegye a tartalom és a megjelenítés szétválasztását, beleértve az elrendezést, a színeket és a betűtípusokat. Ez a szétválasztás javíthatja a tartalom hozzáférhetőségét, nagyobb rugalmasságot és kontrollt biztosít a megjelenítési jellemzők meghatározásában, lehetővé teszi, hogy több weblap megossza a formázására vonatkozó CSS-t .css fájlban történő megadásával, ami csökkenti a HTML összetettségét és ismétlődését, és lehetővé teszi a gyorsítótárazást, ezzel javítva az oldal betöltési sebességét.

A formázás és a tartalom szétválasztása azt is lehetővé teszi, hogy ugyanazt az oldalt különböző stílusban jelenítsük meg különböző megjelenési módok esetén, például nagy vagy kis képernyőn, nyomtatásban, hanggal (beszédalapú böngésző vagy képernyőolvasó segítségével) és Braille-alapú tapintású tapintással használható eszközökön.

A World Wide Web Consortium (W3C), a HTML korábbi, és a CSS szabványok jelenlegi karbantartója, 1997 óta a CSS használatát szorgalmazza a kifejezett prezentációs HTML helyett. [9]

A HTML-hez hasonló módon itt is rengeteg mindenről lehetne írni a box model-től kezdődően, a flexboxon át egészen az animációkig, amikről bővebben itt lehet olvasni: [10].

h1 {

    color: red;

    text-align: center;

}

.small {

    font-size: small;

}

6. ábra: Példa CSS kódra (saját)

### CSS szintaxisok

A képen Grafika, Betűtípus, embléma, Grafikus tervezés látható

Automatikusan generált leírásVannak bizonyos funkciók, amelyeket a CSS nem kínál, mint például a szelektor öröklődés, aminek segítségével a stílusokat örökölhetik egymástól a szelektorok [11] (8. ábra), vagy a mixin, amivel egy többször felhasznált stílusrészletet tudunk csoportba rendezni, amit aztán később felhasználhatunk, hasonló koncepció alapján, mint ahogy a függvények működnek [12] (9. ábra) és még sok más. Ahhoz, hogy a fejlesztők hozzáférjenek ezekhez a funkciókhoz, CSS preprocesszorokat használnak. A CSS preprocesszor egy olyan alkalmazás, amely lehetővé teszi a CSS generálását saját egyedi szintaxisából. A CSS preprocesszorok megkönnyítik és hatékonyabbá teszik a CSS-sel való munkát. [13] [14]

7. ábra: SASS logó [14]

A képen szöveg, Betűtípus, képernyőkép, nyugta látható

Automatikusan generált leírásA képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

8. ábra: SCSS szelektor öröklődés [11]

9. ábra: A mixin és az include [12]

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírásAz felső képen egy példa látható az SCSS és a CSS közti alapvető szintaktikai különbségre, az alsón pedig, hogy ugyanez hogy néz ki alap SASS szintaxist követve. [15]

. ábra: Példa az SCSS és a CSS szintaxis közötti különbségre [15]

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

11. ábra: A 12. ábrán látható példa alap SASS szintaxissal [15]

Jelenleg rengeteg CSS preprocesszor áll rendelkezésre. A SASS és az SCSS a két legnépszerűbb CSS-előfeldolgozó. Bár mindkettő hasonlóan működik, van néhány fontos különbség a kettő között, ami megosztja a fejlesztői közösséget abban, hogy az egyiket válasszák a másik helyett. A webfejlesztés világában a SASS a Syntactically Awesome Style Sheets rövidítése, amely egy preprocesszor szkriptnyelv, amelyet vagy egy CSS fájlba értelmez (interpret), vagy fordít (compile). A gyakran Sassy CSS-nek nevezett SCSS a SASS fő szintaxisa, amely a már létező CSS-szintaxisra épül. A SASS és az SCSS egyaránt képes importálni egymást. A SASS matematikai és változótámogatása valójában a CSS erejét növeli. A CSS és SASS változók közti egyik fő különbség a CSS változók deklaratív, és a SASS változók imperatív mivolta, tehát ha valahol használunk egy SASS változót, majd ez megváltozik később, ez a korábbi felhasználást nem fogja érinteni. A CSS változók változásai ezzel szemben minden azokat felhasználó elemre hatással vannak. Erről a témáról tovább lehet olvasni itt: [16]. A dolgozat során mindkét változó fajta használatra került.

## Az oldal viselkedését befolyásoló technológiák

### JavaScript

A képen Betűtípus, Grafika, embléma, tervezés látható

Automatikusan generált leírásMint korábban már említésre került, a HTML az oldal szerkezetét írja le, ebbe viszont beágyazhatunk olyan scriptnyelvvel írt programokat, mint például a JavaScript, amelyek befolyásolják a weboldalak viselkedését és akár a tartalmát is.

A JavaScript, vagy annak valamely változata, ma már alapvető fontosságú a weboldalak fejlesztésében. Képes arra, hogy interaktívvá és élénkké tegye az online felületeket azáltal, hogy felugró ablakokat, interaktív űrlapokat és egyéb dinamikus elemeket adunk vele az oldalakhoz.

12. ábra: JavaScript logó [18]

Ezen túlmenően, az aszinkron kommunikáció révén a JavaScript lehetővé teszi az adatok háttérben történő letöltését anélkül, hogy ez (mint egyszálas futás esetén) megakasztaná az oldal scriptjének futását vagy, hogy az egész oldal frissülne, ami gyorsabb és simább felhasználói élményt eredményez. Ettől függetlenül a JavaScript ebben az esetben továbbra is egy szálon futtatja az aszinkron függvényeket, viszont a Web Worker-ek 2009-es bevezetésével képesek lettünk több szál használatára is. [17] [18]

A JavaScript (JS) értelmezett (interpretált), vagy „just-in-time” fordított (just-in-time compiled) programozási nyelv first-class függvényekkel, ami annyit jelent, hogy a függvényeket az adott nyelven úgy kezelik, mint bármely más változót. Egy ilyen nyelvben például egy függvényt át lehet adni paraméterként más függvényeknek, egy függvény visszaadhatja azt, és értékként hozzárendelhető egy változóhoz is [19]. Bár leginkább a weboldalak szkriptnyelveként ismert, számos olyan környezetben is használják, amely nem böngésző, például a Node.js alatt.[20]

A JavaScript egy prototípus-alapú *(a prototípus-alapú programozás az objektumorientált programozás olyan formája, amelyben az osztályokat nem explicit módon definiálják, hanem úgy származtatják, hogy tulajdonságokat és metódusokat adnak hozzá egy másik osztály példányához*[21]*)*, többparadigmás *(több programozási paradigmát támogat, mint például az objektumorientált és a funkcionális* [22]*)*, egyszálú, dinamikus nyelv *(dinamikus típusokat használ, tehát futásidőben a változók típusa változhat),* amely támogatja az objektumorientált, imperatív és deklaratív (pl. funkcionális programozási) stílusokat. [23]

### TypeScript

A TypeScript (TS) egy kibővített verziója a JavaScript programozási nyelvnek, amelyet gyakran használnak a modern webfejlesztés során. [24]

Az egyik legfőbb különbsége a vanilla *(olyan szoftver, hardver vagy algoritmus, amelyet nem alakítottak át vagy módosítottak eredeti formájáról)* JavaScripthez képest, hogy statikus típusokat és szigorúbb típusellenőrzést biztosít, így segítve a fejlesztőket a kód hibáinak korai felfedezésében és megelőzésében.

13. ábra: TypeScript logó [24]

Ez azt jelenti, hogy a TypeScript lehetővé teszi a kódon belül átadott adatok típusainak fejlesztő általi meghatározását és korlátozását, tehát egy ilyen módon korlátozott változó a típusát nem, csak az értékét változtathatja meg. Emellett, és ennek köszönhetően a TypeScript képes hibajelzést küldeni, ha a típusok nem egyeznek meg. Erre azért képes, mert a TypeScript fordítási idejű típusellenőrzést használ. Ez azt jelenti, hogy a kód futtatása *előtt* ellenőrzi, hogy a megadott típusok egyeznek-e, nem pedig a kód futtatása közben [25].

A TypeScript például hibát jelez, ha egy stringet adunk át egy olyan függvénynek, amely mondjuk számot várna valamilyen számításhoz. A JavaScript ugyanilyen esetben nem szól, hanem futásidőben megpróbálja értelmezni a számokkal való műveleteknek szánt műveleteket a stringen. Ha ez nem sikerül, akkor futásidőben hibát dob, ha mégis sikerül, akkor pedig végrehajtja azt (például a „+” jel jelenti a számok összeadását, de a stringek összefűzését is). Sok esetben ez csak még több problémát eredményez, hiszen az így kiszámolt adat kikerülhet egy felületre, ahol így a kontextusban nem értelmezhető, esetleg érzékeny információ kerül megjelenítésre. Megtörténhet az is, hogy továbbadjuk egy másik függvénynek, az pedig a következőnek mindaddig, amíg az egyik függvény valamely művelete hibára nem fut, és ekkor el kell kezdeni visszakeresni, hogy melyik függvényben történik a hiba, amely sok függvény esetén sok időt vehet igénybe.

A fejlesztők által írt TypeScript kód fordítás után egyszerű JavaScriptté fordul, így bármely böngésző tudja azt futtatni, amely megfelelően friss verziószámú JavaScript értelmezésére alkalmas. A típusok kikényszerítése így csak arra szolgál, hogy a fejlesztő még felhasználók általi használat előtt kiszűrhesse a típusokkal való műveletekre vonatkozó hibákat, így egy stabilabb és kevesebb hibával rendelkező kódot tudjon szállítani a megrendelőnek.

Ez a nyelv nemcsak dinamikus weboldalak létrehozására, hanem teljes értékű webalkalmazások fejlesztésére is alkalmas, támogatva számos népszerű frontend keretrendszert, például az Angular-t.

A TypeScript folyamatos fejlődése és széleskörű támogatottsága miatt továbbra is elengedhetetlen eszköz a webfejlesztők számára.

## A felhasznált keretrendszer

A képen Betűtípus, embléma, szöveg, Grafika látható

Automatikusan generált leírásAz Angular egy TypeScriptre épülő, nyílt forráskódú fejlesztési platform és keretrendszer, amelyet elsősorban webes alkalmazások építéséhez terveztek.[26] Komponensalapú felépítése lehetővé teszi, hogy a fejlesztők az alkalmazásukat moduláris, újrahasznosítható egységekre bontsák, ezzel átláthatóbbá téve a kódot és egyszerűsítve a karbantartást. Az Angular-t a Google fejleszti és karbantartja, valamint folyamatosan frissítik. Integrált könyvtárainak gyűjteménye számos funkciót lefednek, beleértve például az útválasztást(routing) és az űrlapok kezelését(forms) is. [27]

14. ábra: Angular logó [27]

Ez a keretrendszer olyan programozási technikákat is támogat, amelyek jelentősen megkönnyíthetik a fejlesztési folyamatot, például a dependency injection, amely lehetővé teszi, hogy a különböző komponensek és objektumok, vagy függvények ne közvetlenül hozzanak létre és kezeljenek más objektumokat, hanem azok külsőleg legyenek biztosítva számukra. Ez a technika nem csak a kód modularitását növeli, hanem megkönnyíti a tesztelést és a karbantartást is, mivel csökkenti a különböző részek közötti szoros kapcsolódást, és a kód duplikációt, így az rugalmasabbá válik. Ilyen fontos funkció a modern weboldalak tekintetében még például a lazy loading (késleltetett betöltés), amelyet az Angular is támogat, és amelynek lényege, hogy az alkalmazás bizonyos részei csak akkor kerülnek betöltésre, amikor a felhasználónak valóban szüksége van rájuk. Ez a technika optimalizálja a teljesítményt, mivel csökkenti az alkalmazás kezdeti betöltési idejét és az erre felhasznált erőforrásokat, különösen nagyobb alkalmazások esetében.

Utolsóként pedig megemlíteném a fejlesztők munkáját megkönnyítő Angular CLI-t, amely egy parancssoros eszköztár. Ez olyan feladatok automatizálásában nyújt segítséget, mint a nagyrészt változatlan, vagy kevés változtatást igénylő sablonkódok (boilerplate kód) generálása, de emellett létrehozhatók projektek, valamint elvégezhetők a különböző build, tesztelési és telepítési műveletek is. [28]

## A használt csomagkezelő

Az npm (Node Package Manager) a világ legnagyobb szoftver-nyilvántartása. Az egész világon rengeteg fejlesztő használja az npm-et szoftvercsomagok megosztására, és számos szervezet használja a magánfejlesztéseinek kezelésére és egyszerűsítésére is. [29]

15. ábra: npm logó [26]

Nevéből adódóan, az npm kezeli a projekt függőségeit, tehát telepíteni, frissíteni és törölni is lehet vele csomagokat, például meg lehet neki adni, hogy egy adott csomagból egy adott verziót telepítsen, de azt is, hogy mindig a legújabbat használja. Alapvetően JavaScript-hez készült, de TypeScript-tel is használható és ez a Node.js futtatási környezet alapértelmezett csomagkezelője is. Egy parancssori kliensből, (az npm-ből), egy weboldalból (ahol például fiókot lehet létrehozni a saját csomagok létrehozásához és feltöltéséhez), és egy online adatbázisból, az npm registry-ből áll, amely a nyilvános ingyenes, és a privát fizetős csomagokat is tartalmazza. A regiszterhez a klienssel lehet hozzáférni, az elérhető csomagok pedig az npm weboldalon keresztül böngészhetők és kereshetők is. [30]

## A használt fejlesztői környezet

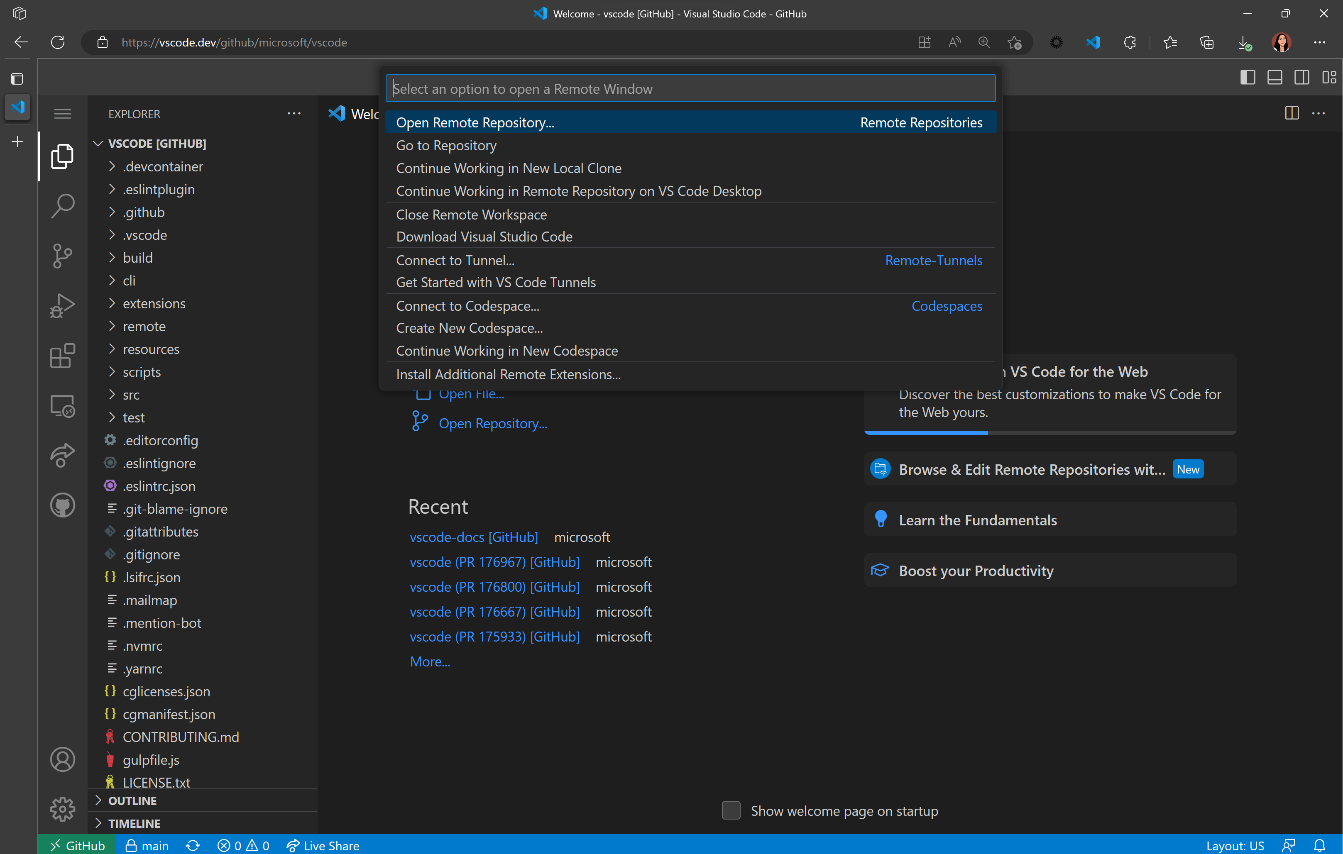
A WebStorm a VSCode-hoz nagyban hasonlító IDE (Integrated Development Environment), az ezt kiadó JetBrains pedig a VSCode legnagyobb piaci versenytársának mondható. A UI tekintetében is kifejezetten hasonlít a két környezet, mintha csak egy másik skint húznánk a VSCode-ra (17. ábra, 18. ábra), de nem véletlen kedvelik ennyire az emberek ezeket az IDE-ket, kényelmes őket használni. [31]

16. ábra: WebStorm logó [31]

Amiben a leginkább eltér a kettő az egyrészt az áruk, hiszen a VSCode ingyenesen használható, a WebStorm-ért pedig licenszdíjat kell fizetni, másrészt pedig nem más, mint hogy a Visual Studio Code valójában egy egész egyszerű szövegszerkesztő, amelynek ereje a bővítménykönyvtárából telepíthető szoftvercsomagokban rejlik, amelyek segítségével szinte bármilyen programozási nyelvre és keretrendszerre fel lehet készíteni a szerkesztőt. Ezek a bővítmények származhatnak magától a Visual Studio Code tulajdonosától, a Microsofttól, de nagyon sok esetben a felhasználók írnak hozzá csomagokat, majd teszik mindenki számára elérhetővé, ezzel növelve a funkcionalitását.

A JetBrains ezzel szemben a legtöbb nyelvre külön IDE-t ad ki, amelyek rendelkeznek az adott nyelvre és környezetre jellemzően szükséges eszközökkel, és amelyek kinézete nagyon hasonlít egymásra, de más funkcionalitással bírnak. Külön található szerkesztő Python fejlesztőknek, webfejlesztőknek, PHP fejlesztőknek, C és C++ fejlesztőknek, és ezt a sort még lehetne folytatni. VSCode esetében ezt a specializációt bővítményekkel lehet megoldani. Természetesen a JetBrains IDE-iben, tehát a WebStorm-ban is van lehetőség pluginok telepítésére, de ezek általában kisebb mértékben befolyásolják a szerkesztő működését. Ennek a hozzáállásnak előnye is van, mivel a legtöbb felhasználói visszajelzés alapján a WebStorm hatékonyabban képes nagy kódbázisokat kezelni, kevésbé „lassul le” a vetélytársával szemben, amelyet a sok bővítmény telepítése visszahúzhat ebben a versenyben.

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Automatikusan generált leírás

17. ábra: WebStorm UI [32]

18. ábra: VSCode UI [33]

[32] [33]

# Fejlesztés

## Naptár elrendezése

### Alapelrendezés

A képen szöveg, képernyőkép, sor, Betűtípus látható

Automatikusan generált leírásA fejlesztés során a funkcionális, vagy logikailag összetartozó egységeket komponensekbe választottam szét. Az első lépés a naptár alapfelületének megtervezése, és elkészítése volt. Ennek alapjául a már évtizedek óda elterjedt, más online- és asztali naptárak esetén is használt elrendezést vettem, miszerint egy adott oldalon 7 oszlop található hétfőtől vasárnapig. Ezen oszlopok fejlécében a dátum és a nap neve olvasható. Az oszlopok alsó, fejléc alatti test-részben pedig minden oszlopot téglalapokra osztottam, ahol egy téglalap egy órányi időtartamot ölel fel, hogy az időbeosztás vizuálisan is jól követhető legyen. Ebből értelemszerűen minden oszlopba 24 darab kell, ezzel lefedve egy nap 24 óráját.

. ábra: Az Appointy egy nap-oszlopának kezdete (saját)

A képen szöveg, képernyőkép, sor, szám látható

Automatikusan generált leírásEzzel az időpontok megjelenítési felülete megtervezésre került, viszont még nem látszott, hogy egy napon belül melyik téglalap melyik órához tartozik. Ezért készítettem, és a naptár bal oldalához toldottam egy időskálát is, amely mutatja, hogy milyen időszeletek hol találhatóak meg a naptárban.

. ábra: Az Appoiny időskálával bővített felülete (saját)

### Túlcsordulás

Mivel a téglalapok megfelelően nagy méretűek kell, hogy legyenek, hogy az oda kerülő időpontok olvashatóak és egyáltalán jól láthatóak maradjanak, a 24 órához tartozó téglalapok a legtöbb monitoron vertikális, avagy y tengely menti irányban, de némelyen horizontálisan is túlcsordulnak a monitoron megjelenített viewport-on (a viewport egy weboldal felhasználó által látható területe; egyszerűbben fogalmazva: „Ami egyszerre ráfér a kijelzőre”). A böngészők ezt általában görgetés bevezetésével oldják meg, viszont ez a viselkedés böngészőnként eltérhet, és idővel változhat, nem beszélve arról, hogy ezesetben a komponens felhasználójának kell megoldani a túlcsordulás kezelését. Ezt el szerettem volna kerülni, hogy minél egyszerűbb legyen a komponens használata, ezért a komponensen belül van megadva az overflow a megfelelő komponensre (amelynél értelme van), így attól függően, hogy a fejlesztő mennyi helyet ad a komponensnek, az igyekszik azt kitölteni, és megfelelően görgethetővé válni (21. ábra, 22. ábra).

A képen szöveg, sor, szám, Párhuzamos látható

Automatikusan generált leírás

21. ábra: Az Appointy felülete teljes képernyőn (saját)

A képen szöveg, szám, képernyőkép, Betűtípus látható

Automatikusan generált leírás

22. ábra: Az Appointy felülete egy 500\*800 pixeles helyre szorítva (saját)

Az ábrákon láthatóan a komponens nem lóg ki kis helyre szorított helyzetben sem, vertikálisan és horitonzálisan is görgethető túlcsordulás esetén. A naptártest magasságának kiszámítására egy matematikat képletet kellett írnom, hiszen nem 100%-ra kell azt skálázni, mivel az aránytaalanul összenyomná, deformálná az órákat jelentő téglalapokat, valamint a test túllógna a megadott magasságon, így vertikálisan kétszeresen görgethető lenne a komponens, ami UX (User eXperience) szempontból nem optimális.

height: calc(100% - $date-container-height - $column-top-padding - var(--hour-cell-height));

23. ábra: A naptártest magasságának számítására alkalmazott képlet

A képletben a $date-container-height a komponens tetején található év-hónap kijelzés, és lapozó gombok magasságára utal, a $column-top-padding a naptár oszlopainak felső bélése, a --hour-cell-height pedig az órákat felölelő téglalapok magassága, ami megegyezik a fejléc magasságával. A --hour-cell-height a képletben a többi változóval ellentétben azért nem SCSS, hanem CSS változóként van definiálva, mert ezt lehet kódból lekérni és használni.

### Naptártest felépítése

Az előző pontban megfogalmazottak alapján a naptár hét fő oszlopból áll, amelyeknek van fejléce és teste. A fejléc tartalma már tárgyalásra került, a testben a már említett órákat megtestesítő téglalapok találhatóak. Az oszlopok saját komponens alapján jönnek létre, amelyeket az Angular által biztosított @for vezérlővel állítottam elő, valamint az oszlopokon belüli téglalapok is ezt a módszert követik. Ennek a generálásnak funkcionális oka van, amely később tárgyalásra kerül.

@for (date of dates; track $index) {  
 <div class="d-flex flex-column grow">  
 <Appointy-calendar-column [date]="date" class="day\_column" [id]="'calendarColumn' + date.getDate()" (loaded)="handleColumnLoad($event, $index)"/>  
 </div>  
}

24. ábra: A naptár oszlopainak generálása (saját)

A dates tömbben az adott hét dátumai vannak felsorolva Date objektumként (ez 7 elemet jelent), amelyek alapján generálódnak és töltődnek fel az oszlopok fejlécei, valamint ez alapján kapja az id-ját az oszlop maga is, amelynek szintén funkcionális oka van, és jelentőségére ez a dokumentum később tér ki. A végső elem, amit láthatunk ezen a komponensen a loaded Event visszaadása a szülőnek, ami időzítésbéli problémákat old meg.

<div>  
 @for (hour of [].constructor(24); track $index) {  
 <div class="hour-cell" [id]="getID($index)" (click)="openNewEventModal($index)"></div>  
 }  
</div>

25. ábra: A naptár oszlopain belüli téglalapok generálása (saját)

A téglalapok generálása az oszlopokéhoz nagyban hasonlít, gyakorlatilag egy 24 elemű üres tömböt hoztam létre a [].constructor(24) segítségével, amelyen végigiterálva hozom létre a cellákat. Ezek szintén azonosítót kapnak a jelenlegi $index alapján, amely ezesetben pont az az óra, amely az adott cella kezdő időpontja (például a 0. indexet az első cella kapja meg, ami a 00:00 - 01:00 időintervallumot öleli fel).

Az üres elemeket tartalmazó tömb létrehozására sajnos szükség volt, mivel a @for vezérlő nem képes azt az alapvető for ciklus funkcionalitást végezni, miszerint elindul egy értéktől, és addig tesz valamit (például növeli eggyel), amíg el nem éri, hogy a neki megadott feltétel igaz legyen (26. ábra). Ezt a @for vezérlőt azért találták ki, hogy listákon lehessen iterálni vele, és ebből kifolyóleg nem alkalmas az előbb említett funkcionalitásra, azonban ahogy ez látható a mellékelt képen (25. ábra) szerencsére meg lehet kreatívan oldani, hogy így is működhessen.

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

//output: 0 1 2 3 4 5 6 7 8 9

26. ábra: Általános for ciklus funkcionalitás (saját)

### Görgetés

A vertikális görgetés, mint korábban említésre került, nem az egész komponensre vonatkozik, csak a „táblázat” testére, tehát a fejléc, egy helyben marad. Ennek a megoldására az első, és leginkább intuitív gondolat a position: sticky használata, ami számomra is az első próbálkozás volt, azonban ez a property nem szokott rendesen működni, ha bármelyik szülőjén az overflow property hidden, auto, vagy scroll értékre van állítva [34]. Ezért döntöttem amellett, hogy egy másik, kicsit bonyolultabbnak bizonyult megoldást választok, miszerint a naptártestet és a mellette lévő időskálát lehessen csak görgetni.

A képen szöveg, képernyőkép, sor, szám látható

Automatikusan generált leírásEz rögtön láthatóvá tett egy problémát, miszerint így a fejléc, és a test egymáshoz képest mért elrendezése elcsúszott az időskála miatt. Egyértelműen ez okozta a problémát, hiszen a napok oszlopai végig egyenlő szélességűek, ez volt az egyetlen különbség a két sor között. Ezt a problémát egy fejlécbe helyezett filler elem használatával tudtam korrigálni, ami ugyanazt a szélességet kapta, mint az időskála oszlopa.

27. ábra: Időskála oszlopának stílusa (saját)

.hours\_column {  
 min-width: $hours-column-width;  
 min-height: 2rem;  
 padding-top: 0;  
 margin-top: -5px;  
 margin-left: 15px;  
}

28. ábra: Elrendezést segítő filler elem (saját)

A második probléma, ami sokáig fel sem tűnt pusztán azért, mert Firefox-ot használtam a fejlesztéshez, maga a görgetősáv volt. Alapértelmezés szerint Firefox Developer Edition alatt a görgetősáv DOM-ban elfoglalt szélessége 0, úgy jelenik meg, mint abszolút pozícióval az oldal jobb szélére tapasztott elem. Ez a legtöbb böngészőben nem így van, például a világ jelenlegi legnépszerűbb böngészőjében, a Google Chrome-ban sem, tehát ezek a görgetősávok az összes velük egy szinten lévő elemet a saját szélességükkel összenyomják. Ez is egy elrendezési inkonzisztenciát eredményezett, miszerint a fejléc és a test egymáshoz képest elcsúszott állapotban jelentek meg. Szerencsére egyszerű megoldást lehet eszközölni erre, mégpedig a scrollbar-width (29. ábra) property segítségével meg lehet adni pontosan milyen széles legyen az adott elemben feltűnő görgetősáv. Ezzel ez a probléma is megoldódott.

29. ábra: A görgetősáv szélesség megoldása (saját)

.scrollable {  
 overflow-y: auto;  
 …  
 scrollbar-width: none;  
}

A komponensbe egy kényelmi funkció is bekerült, ami oldalbetöltéskor reggel 7:00-hoz görgeti a naptártestet (ahogy ez a korábbi képeken is látható volt), hiszen az emberek általában nem hajnalban, hanem nap közben aktívak, így ez a kis QoL (Quality of Life) feature gördülékenyebbé teszi a használatot mindneki számára. Ez az időskála komponens betöltésekor rögtön megtörténik, a komponens TypeScript file-jában.

const scrollTo = document.getElementById("7");  
const scrollable = document.getElementById('scrollable');  
  
if (scrollTo && scrollable) {  
  
 const { top } = scrollTo.getBoundingClientRect(); *// scrollTo helye* const { top: scrollableTop } = scrollable.getBoundingClientRect(); *// scrollable helye* const scrollY = top - scrollableTop; *// mennyit kell görgetni* scrollable.scrollBy({  
 top: scrollY,  
 behavior: 'smooth'  
 });  
}

30. ábra: A kezdőidőponthoz görgetés (saját)

## A megjelenítés logikai elemei

### Hét napjai

A naptár fejlécében fontos, hogy látni lehessen az adott hét napjait és a dátumokat, avagy, hogy melyik nap hányadika van. Ez a fejléc is külön komponensekre van bontva, amelyek megfelelően megjelenítik a nekik szolgáltatott információt.

A fejléc elemei az ábrán látható módon kerülnek generálásra. A weekdays egy 7 elemű tömb, amelyben a hét napjai vannak string-ként tárolva, a dates tömbben pedig az adott héthez tartozó napok dátumai vannak. Ezek feltöltéséről rögtön szó lesz. Mint látható a komponens megkapja a hozzá tartozó napot és dátumot @Input adatként, amit csak meg kell jelenítenie az alább látható módon.

@for (day of weekdays; track $index) {  
 <Appointy-weekday [day]="day" [date]="dates[$index].getDate()" class="weekdays"/>  
}

31. ábra: Fejléc generálása (saját)

<div class="header-cell d-flex flex-column">  
 <small class="text-center">{{ day }}</small>  
 <h3 class="text-center">{{ date }}</h3>  
</div>

32. ábra: Nap és dátum megjelenítése (saját)

Ezzel már látható a naptár tetején a nap és a dátum is, ahogy azt korábbi ábrákon (pl.: 22. ábra) már látni lehetett.

### Weekdays

Az előző pontban említett módon a weekdays tömb 7 elemű, és a hét napjai vannak string-ként tárolva benne. Ennek a feltöltéséért a fillWeekDays függvény felel.

Ez a függvény egy locale változót vár bemenetként, amely valójában nem más, mint a nyelv, amelyen a napokat le szeretnénk kérni.

fillWeekDays(locale: string) {  
 let baseDate: Date = *startOfWeek*(new Date(), {weekStartsOn: 1});for (let i: number = 0; i < 7; i++) {  
 let day: string = baseDate.toLocaleDateString(locale, {weekday: 'long'});  
 day = day[0].toUpperCase() + day.slice(1);  
 this.weekdays.push(day);  
 baseDate.setDate(baseDate.getDate() + 1);  
 }  
}

33. ábra. A hét napjainak feltöltése (saját)

Első lépésként létrehozok egy baseDate változót, amelyet egy hétfői dátumra állítok be a date-fns könyvtárból származó startOfWeek függvény segítségével. Ennek egy kötelező paramétere van: a dátum, amely dátum hetének az elejét szeretnénk megkapni, de nekem egy másik opcionális, de a helyes működés szempontjából fontos paramétert kellett még megadnom, ez pedig a weekStartsOn, amely a korábban megismertekhez hasonlóan egy számot vár 0 és 6 között, ahol a 0 a vasárnap, 1 a hétfő és 6 a szombat. Ezzel adható meg, hogy melyik nappal kezdődjön a hét, tehát melyik napot adja vissza a függvény.

Mikor sikeresen beállítottunk egy hétfői dátumot, a cikluson belül lekérjük az adott dátumra vonatkozó nap nevét az általunk beállított nyelven, és weekday: long konfigurációval. Enélkül egy YYYY. MM. DD. formátumú string-et kapnánk vissza, így viszont a nap teljes magyar nevét (hétfő, kedd, stb.).

Ezután a napot nagy kezdőbetűssé teszem, hozzátoldom a weekdays tömb végéhez, majd a baseDate változómat egy nappal tovább léptetem. Ekkor kezdődik az előbb leírt ciklus elölről, amíg nincs meg mind a 7 nap.

### Dates

A dátumok feltöltése bonyolultabbnak bizonyult a hét napjainál, hiszen itt figyelembe kell venni azt, ha a hét visszanyúlik az előző hónapba, vagy átnyúlik a következőbe, ezért ez kisebb darabokban kerül bemutatásra.

this.dates = [];  
let prevMonthUsed: boolean = false;  
let nextMonthUsed: boolean = false;  
  
const currentDate: number = this.currentDate.getDate();  
const currentDay: number = this.currentDate.getDay();  
this.startDate = currentDate - (currentDay != 0 ? currentDay : 7) + 1;

34. ábra: Kezdődátum kezdeti meghatározása (saját)

Első lépésként a feltöltendő tömböt kiürítem, hogy véletlen se maradjon benn olyan elem, amire már nincs szükségem, és elronthatja a feltöltést vagy a megjelenítést. Ezután létrehozok két boolean változót prevMonthUsed és nextMonthUsed néven, amelyek azt jelölik majd, hogy a héten átnyúltunk-e másik hónapba, és ha igen melyikbe.

A következő szekcióban a kezdődátumot keresem meg, ahol is a jelenlegi dátumból kivonok annyi napot, ahányadik nap van a héten, így optimális esetben megtudom, hogy milyen dátumra esik a hét első napja. Itt jön számításba az, hogy ha a hét eleje még az előző hónaphoz tartozna, akkor erre a startDate változó úgy reagálna, hogy az értéke lecsökken 1 alá.

A megfogalmazott feltétel pont ezt vizsgálja, amely, ha igaz, akkor tudjuk, hogy az előző hónap is megjelenik a héten, tehát a megfelelő flaget igazra állítjuk, a kezdődátumot pedig valós értékre változtatjuk. Ez utóbbit az előző hónap hosszának lekérésével, majd ahhoz a startDate hozzáadásával tehetjük meg, hisz ebben az esetben a startDate vagy 0, vagy negatív szám. Egy példa kedvért, ha a startDate -2 értékre jönne ki első körben, akkor az előző hónap végéhez ezt hozzáadva (és feltéve, hogy az előző hónap 30 napos volt) tudjuk, hogy a hét kezdő dátuma 28.-a lesz.

35. ábra: Kezdődátum korrigálása (saját)

const lastDayOfLastMonth: number = new Date(this.currentDate.getFullYear(), this.currentDate.getMonth(), 0).getDate();  
if (this.startDate < 1) { *// hét eleje még az előző hónap* prevMonthUsed = true;  
 this.startDate = lastDayOfLastMonth + this.startDate;  
}  
  
let date: number = this.startDate - 1;  
const lastDayOfThisMonth: number = new Date(this.currentDate.getFullYear(), this.currentDate.getMonth() + 1, 0).getDate();

Az előző hónap hosszának lekérésében észrevehetünk egy érdekességet, miszerint a this.currentDate.getMonth() metódussal lekérem, hogy jelenleg milyen hónap van, majd ezt változtatás nélkül visszaadom a Date konstruktorának, pedig az előző hónap végét keresem, nem a jelenlegiét. Ez azért így van elkészítve, mert a getMonth a getDay-hez hasonlóan 0-tól kezdődik, és a hónap esetében 11-ig tart, ahol 0 a január, és 11 a december. A Date konstruktora viszont 1-től 12-ig tartó számot vár a hónap beállításához, tehát az egész el van csúszva eggyel, így nem kell változtatni a lekért adaton.

Ezután létrehozok egy date változót, ami kezdetileg a startDate értékénél 1-gyel kisebb értéket kap, majd lekérem a jelenlegi hónap hosszát az előző bekezdésben megismert sajátosságot figyelembe véve.

Ez után kerülünk bele a ciklusba, ahol elkezdjük feltölteni a tömböt. Az elején láthatóan azok a feltételek foglalnak helyet, amik azt ellenőrzik, hogy ebben az iterációban fog-e történni két hónap közti váltás, vagy sem, és ez alapján állítják be a date (jelenlegi dátum) értékét.

for (let i = 0; i < this.weekdays.length; i++) {  
 if (prevMonthUsed && lastDayOfLastMonth < date + 1) { *// átlépünk az előző hónap végéről elsejére* date = 1;  
 } else if (lastDayOfThisMonth < date + 1) { *// átlépünk az akt. hónap végéről a köv. hónap elsejére* nextMonthUsed = true;  
 date = 1;  
 } else {  
 date++;  
 }  
  
 let copiedDate: Date = new Date(this.currentDate);  
 copiedDate.setDate(date);  
  
 if (nextMonthUsed) { *//következő hónapba lépünk* copiedDate.setMonth(copiedDate.getMonth() + 1);  
 this.dates.push(copiedDate);  
 nextMonthUsed = false;  
 } else if (prevMonthUsed) { *//előző hónapba lépünk* copiedDate.setMonth(copiedDate.getMonth() - 1);  
 this.dates.push(copiedDate);  
 prevMonthUsed = false;  
 } else {  
 this.dates.push(copiedDate);  
 }  
}

36. ábra: A hét dátumainak feltöltése (saját)

Amikor ez megtörtént, lemásolom a jelenlegi dátumot egy másik copiedDate nevű változóba (mert a komponens szintű jelenlegi dátumot más is használja, ezért nem azt szeretném változtatni), majd ezen beállítom a kezdődátumot.

Az ezek utáni feltételek már csak azt vizsgálják, hogy melyik hónapban vagyunk. Az első feltétel azt vizsgálja, hogy át kell-e lépnünk a következő hónapba, amelyet a nextMonthUsed változó vizsgálatával megtehetünk, de ha átléptünk a következőbe, akkor biztosak lehetünk benne, hogy még egyszer nem kell ezt megtennünk ugyanazon a héten, ezért ezt a flaget false-ra állíthatjuk. Ugyanezen logika mentén van ellenőrizve az előző hónapba lépés is, valamint, ha semerre nem kell lépni a hónapok között, akkor egyszerűen hozzáfűzzük a dátumot a tömbhöz.

### Megjelenített hónap

A hónap kijelzésénél felmerül a kérdés, hogy mi alapján döntsük el, hogy milyen hónap van az adott héten. A legtöbb esetben ez nem probléma, hiszen szeptember 16 és 22 között egyértelműen szeptember van. A kérdés akkor lesz fontos, ha a hétbe belecsúszik az előző hónap vége, vagy a következő hónap eleje. Erre azt a megoldást találtam ki, hogy azt a hónapot jelenítjük meg, amelyből több nap található a héten. Tehát ha például a hét a következőképpen néz ki: szeptember 29, 30, október 1, 2, 3, 4, 5; akkor a megjelenítendő hónap az október.

. ábra: Év- és hónap kijelző, lapozó gombok (saját)

Szerencsére nem szükséges megszámolni ezeket a napokat, elegendő, ha megvizsgáljuk, hogy a hét középső napja, azaz a csütörtök melyik hónapban van, ezzel meg is tudtuk, hogy ennek a hónapnak legalább 4 napja található az adott héten, míg a másiknak maximum 3, tehát már egyértelmű melyik a megjelenítendő hónap.

38. ábra: Hét középső napjának meghatározása (saját)

this.currentDate = new Date();  
this.middleDate = new Date();

this.middleDate.setDate(this.currentDate.getDate()

+ (4 - (this.currentDate.getDay() == 0 ? 7 : this.currentDate.getDay())));

Ezt a középső napot hivatott jelenteni a middleDate, amelyet a feljebb látható módon állítok be.

A logika a következő: mindenekelőtt lekérem a jelenlegi dátumot a Date objektum alap konstruktorával (new Date()), és ezt bemásolom mind a currentDate, mind a middleDate változómba. Ezután lekérem a jelenlegi dátumból, hogy hányadika van a this.currentDate.getDate() metódussal, amely értelemszerűen egy számot ad vissza 1-től az adott hónap utolsó napjáig.

Ehhez a dátumhoz hozzáadom a (4 – (this.currentDate.getDay() == 0 ? 7 : this.currentDate.getDay())) kifejezés eredményét, amelyet egyszerűbb darabokra bontva elmagyarázni. A this.currentDate.getDay() a hét napját adja vissza számként, egy 0 és 6 között terjedő skálán. Ezt az eredményt 4-ből kivonom, mivel a 4 (csütörtök) a hét középső napját jelölő szám, így tehát ha jelenleg csütörtök van, akkor 4-4-et, tehát 0-t ad hozzá a jelenlegi dátumhoz, de ha kedd van, akkor 4-2-t, tehát 2-t ad hozzá, így ismét a csütörtököt kapjuk.

Természetesen ez sem lehet ilyen egyszerű, mivel ez a 0-6 skála nem hétfőtől vasárnapig terjed, hanem vasárnaptól szombatig, tehát amerikai szokás szerint a vasárnap a hét kezdőnapja, és annak az értéke a 0. A háromoperandusú operátor csak a vasárnap edge case (olyan eset, ahol az általánosan megfogalmazott működés hibás eredményre vezet) jellegét hivatott kiküszöbölni, miszerint, ha az visszakapott érték 0, akkor 7-et vonjunk ki a 4-ből, de ha nem az, akkor minden mehet tovább, mint eddig. Így vasárnappal is működik a képlet: 4-7-et, tehát 3-at von ki a dátumból, ha vasárnaptól 3 napot visszalépünk, akkor pedig pont a csütörtököt kapjuk, ahogy kell.



39. ábra: Megjelenített hónap egy kevert hónapú héten (saját)

### Lapozás

Igazán fontos feladata egy naptárnak, hogy ne csak az aktuális, vagy egy adott hetet lássunk, és kérhessünk le rajta, hanem amelyiket szeretnénk. Ezt többféleképp is meg lehet oldani, például egy kicsinyített nézetben az adott hónap minden hetét megjelenítve, és egy hétre kattintva azt nagyítva, vagy esetleg, amit én választottam, egyszerre egy hetet mutatva, és lapozással a hetek között lépkedve. Ez a hozzáállás szerintem ennek a komponensnek jobban áll, hiszen arra hivatott, hogy az oldalra látogatók láthassák a lefoglalt időpontokat, és maguk is időpontot foglalhassanak. Szerintem feltételezhető, hogy ezt az időpontfoglalást legtöbb esetben nem három hónap múlvára igyekeznek megtenni, hanem maximum néhány héttel későbbre.

Korábban látott módon a lapozógombok a fejlécben kaptak helyet (pl.: 39. ábra), és ezekre kattintva lehet egy lépéssel vissza és tovább lapozni a naptárban.

stepWeek(amount: number) {  
 amount \*= 7;  
 this.currentDate.setDate(this.currentDate.getDate() + amount);  
 this.middleDate.setDate(this.middleDate.getDate() + amount);  
  
 this.fillDatesOfWeek();

}

40. ábra: A hét léptetése (saját)

A lapozás tulajdonképpen úgy történik, hogy a stepWeek metódust meghívva egy paramétert adok át neki, ami gyakorlatilag az irány, hogy merre léptessük a naptárat. Ez az érték hátra léptetés esetén -1, előre léptetésnél +1. Ez után ez az érték rögtön felszorzódik héttel (mivel egy héten hét nap van), majd ezzel a +7, vagy -7 nappal megnövelt, vagy épp csökkentett dátumot állítom be mind a currentDate, mind a middleDate változó értékének. Ez után az eddig tárgyalt funkciók közül már csak az van hátra, hogy a hét dátumait ismét feltöltsük, amelynek működéséről korábban már volt szó.

## Események

Most, hogy a naptár felülete valójában kész, a következő lépés az események lekérése, és az a naptár azokkal való feltöltése. A kiválasztott naptár a Google Calendar lett, mivel egyrészt jelenleg az egyik, ha nem a legnépszerűbb online naptárszolgáltató, másrészt az Outlook Calendar API hozzáféréséhez Azure Developer fiók kell, amely alapvetően nem ingyenes, csak egy 12 hónapos ingyen próbaidőt biztosít. A Calendly általam fontosnak talált funkciónak használatához pedig autentikáció szükséges, tehát az események lekéréséhez is, szóval nem lehet publikussá tenni naptárat, hogy a megjelenítést bejelentkezés nélkül lehessen végezni.

### Lekérés

Az események témakörében a fejlesztés során első lépésként a már meglévő események lekérését és megjelenítését valósítottam meg, amelyet a naptár korábban tárgyalt struktúrájának betöltése után kezdem meg az initCalenar metódus segítségével.

initCalendar() {  
 this.subs.push(this.calendarService.getCalendarEvents()  
 .subscribe((res) => {  
 for (let i = 1; i < res.length; i++) {  
 res[0].items = [...res[0].items, ...res[i].items];  
 }  
 this.displayEvents(res[0].items);  
 }))  
}

41. ábra: Az események lekérése az initCalendar-ban (saját)

Itt az RxJs[35] Observable[36] kollekcióját használom, melyet a getCalendarEvents()

ad vissza. Ez egy olyan kollekció, amely a Promise-hez hasonlítható, annyi különbséggel, hogy több adatot is képes emittálni, míg a Promise csak egyet (szigorúan véve a Promise is tud több adatot visszaadni, ha azok objektumba, vagy tömbbe vannak foglalva, de az Observable több tömböt és objektumot is vissza tud adni). Leggyakrabban aszinkron folyamatok esetén hasznos, amikor nem tudjuk, hogy mikor fogunk kapni adatot, és esetleg azt sem, hogy mennyit. A subscribe() metódussal egy Subscription objektum jön létre, amelyről ha nem iratkozunk le tovább foglalja a memóriát úgy is, ha nincs használatban többé az a feliratkozás. Emiatt van belerakva egy subs tömbbe, amely a komponens életciklusának végén törlődik.

A subscribe-on belül a visszakapott eredmény egybe fűzését láthatjuk, mivel több naptár használata esetén ezek elemei külön kerülnek visszaadásra. Ezek itt egységes formára vannak hozva. Ezután a lapítás után történik a megjelenítés a displayEvents függvénnyel.

getCalendarEvents(date: Date = new Date()): Observable<any[]> {  
 const startOfWeek = this.getStartOfWeek(date);  
 startOfWeek.setHours(0, 0, 0, 0);  
 const offset = startOfWeek.getTimezoneOffset(); *//percben adja vissza* startOfWeek.setMinutes(startOfWeek.getMinutes() - offset);  
  
 const endOfWeek = this.getEndOfWeek(date);  
 endOfWeek.setHours(23, 59, 59, 999);  
 endOfWeek.setMinutes(endOfWeek.getMinutes() - offset);  
  
 const startDateTime = startOfWeek.toISOString();  
 const endDateTime = endOfWeek.toISOString();  
  
 const requests: Observable<any>[] = [];  
  
 for (let i = 0; i < this.constService.CALENDAR\_IDS.length; i++) {  
 const url = `https://www.googleapis.com/calendar/v3/calendars/${this.constService.CALENDAR\_IDS[i]}/events?timeMin=${startDateTime}&timeMax=${endDateTime}&key=${this.constService.API\_KEY}`;  
 requests.push(this.http.get(url));  
 }  
  
 return *forkJoin*(requests).pipe(  
 *map*(res => res.flat())  
 );  
}

42. ábra: Egy hét eseményeinek lekérése (saját)

A 42. ábrán az események egy hétre való lekérése látható. A for ciklus előtti részben csak az időintervallum beállítása kap helyet, a cikluson belül a naptárakhoz tartozó URL-ek vannak összeállítva. A CALENDAR\_IDS tömbben a naptárak ID-i találhatóak, amellyel egyértelműen azonosítható egy-egy naptár. Ezt átadva URL paraméterként egyértelműen megadhatjuk,hogy melyik naptár érdekel minket, ekkor már csak a lekérendő idő intervallumot kell megadni, és az API kulcsot. Ez az összeállított URL bekerült a requests tömbbe, amelyben található URL-ekre a függvény végén a forkJoin használatával egyszerre kérést küldünk, és összevárjuk az eredményeket, majd ezt visszaadjuk a függvényt hívó kódnak.

### Megjelenítés

Az események lekérésének áttekintésével rátérhetünk a megjelenítésre. Ezt az előző pontban látható displayEvents() függvény oldja meg,

displayEvents(results: CalendarEvent[]) {  
 this.calendarEvents = [];for (const res of results) {  
 this.calendarEvents.push(new CalendarEvent(res.summary, res.start, res.end, res.description, res.location));  
 }  
  
 let startElement: HTMLElement | null;  
  
 for (let i = 0; i < this.calendarEvents.length; i++) {startElement = document.getElementById(this.calendarEvents[i].startDate.getFullYear() + "." + this.calendarEvents[i].startDate.getMonth() + "." + this.calendarEvents[i].startDate.getDate() + "." + this.calendarEvents[i].startDate.getHours());

…

}

…  
}

43. ábra: Események rendszerezése (saját)

Bemenetként megkapja a lekérésből visszakapott eseményeket, amiket rendszerez a calendarEvents tömbbe, így egységesen elérhetőek lesznek a fontos információk. Következő lépésként meg kell keresni, hogy hol kell megjeleníteni az eseményt.

Itt jön a képbe a téglalapok id-val való ellátása, amiről korábban szó volt Ez alapján az azonosító alapján minden jelenleg legenerált celláról megmondható, hogy milyen év milyen hónapjának melyik napját, és hányadik óráját reprezentálja.

displayEvents(results: CalendarEvent[]) {  
 …  
 for (let i = 0; i < this.calendarEvents.length; i++) {  
 …  
 if (startElement) {const eventContainer = document.createElement('div');  
 eventContainer.classList.add("event\_container");  
 startElement.appendChild(eventContainer);  
this.eventRefs[i] = this.appRef.bootstrap(EventComponent, eventContainer);this.eventRefs[i].instance.calendarEvent = this.calendarEvents[i];  
 this.eventRefs[i].instance.id = `card${i}`;  
this.eventRefs[i].instance.updateVariables();  
 }  
 }  
  
 this.now = new Date();  
 this.setNowMarker();  
}

44. ábra: Esemény elhelyezése a felületen (saját)

Ha ezt az elemet megtaláltuk, belehelyezzük az eseményt reprezentáló EventComponent-et, amelynek bementi adatként megadjuk a hozzá tartozó esemény adatait, majd az updateVariables() függvénnyel küldünk neki egy jelet, hogy frissítse a bemeneti adatait, ezzel észre véve a változást (tehát, hogy adatot adtunk neki). Ez megtörténik minden lekért esemény esetében, majd ez után csak a now marker kerül elhelyezésre, amely a jelenlegi időt jelzi, hogy könnyebben lehessen látni hol tartunk az eseményeinkkel.

A képen szöveg, képernyőkép, szám, Betűtípus látható

Automatikusan generált leírás

45. ábra: Megjelenített események (saját)

A képen szöveg, képernyőkép, Betűtípus, sor látható

Automatikusan generált leírás

46. ábra: Now marker (saját)

A megjelenített események az 45. és 46. ábrán láthatóak, előbbi a már múltban megtörtént (past event), a másodikon pedig a még meg nem történt, vagy aktuális (future / current event) látható, színekkel is jelölve ezt a érthetőbb megjelenítésért. Az 46. ábrán látható piros vonal maga a now marker, amelynek pozíciója fél percenként frissül, így aktuális marad.

Mindenek előtt a 44. ábrán meghívott updateVariables() metódus elnevezéséhez híven frissíti a komponens esemény méretezéséhez fontos változóit, majd az Angularba beépített ChangeDetectorRef[37] osztály segítségével manuálisan frissítem a view-ban megjelenített adatokat. Ezután következik az updatePosition(), ami a kártyák pozicionálását és méretezését végzi.

47. ábra: Esemény kártya változóinak frissítése (saját)

48. ábra: Esemény kártya pozíciójának beállítása (saját)

updatePosition() {  
 const card = document.getElementById(this.id)!;  
 const cellHeight: number = *parseFloat*(*getComputedStyle*(document.documentElement).getPropertyValue('--hour-cell-height'));  
 this.setMarginTop(card, cellHeight);  
 this.setHeight(card, cellHeight);  
 this.setColor(card);  
}

updateVariables() {  
 this.startHour = this.calendarEvent.startDate.getHours();  
 this.startMinute = this.calendarEvent.startDate.getMinutes();  
 this.endHour = this.calendarEvent.endDate.getHours();  
 this.endMinute = this.calendarEvent.endDate.getMinutes();  
  
 this.cdr.detectChanges();  
  
 this.updatePosition();  
}

Az események helyes pozícióban való megjelenítéséhez nem elég a megfelelő cella megtalálása, hiszen nem biztos, hogy az adott időpont például pontban 11 órakor kezdődik, és 12 órakor ér véget. Ezért pontosabban meg kell határozni a kártyák elhelyezkedését, amelyhez mindenek előtt lekérjük az adott kártyára vonatkozó referenciát a DOM-ból. Ezután a már korábban látott –hour-cell-height css változó lekérése következik, ebből megtudjuk mekkora helyet kell elfoglalnia 1 órának. Ha ez megtörtént a vezérlés átkerül a setMarginTop, a setHeight és a setColor metódusokhoz.

setColor(card: HTMLElement) {  
 const now: Date = new Date();  
  
 if (this.calendarEvent.endDate < now) {  
 card.classList.add("past\_event");  
 } else {  
 card.classList.add("future\_event");  
 }  
}

51. ábra: Kártya színének beállítása (saját)

50. ábra: Kártya magasságának beállítása (saját)

49. ábra: Kártya felső margójának beállítása (saját)

setHeight(card: HTMLElement, cellHeight: number) {  
 if (card) {  
 const height: number = cellHeight \* (this.endHour - this.startHour) + cellHeight \* ((this.endMinute - this.startMinute) / 60);  
 card.style.height = `${height}rem`;  
 }  
}

setMarginTop(card: HTMLElement, cellHeight: number) {  
 if (card) {  
 const marginTop: number = cellHeight \* (this.startMinute / 60);  
 card.style.marginTop = `${marginTop}rem`;  
 }  
}

Ezek az eljárások könnyen érthető logika mentén számítják ki az adott kártyának megfelelő felső margót, magasságot és dönti el a színét az esemény kezdő- és végidőpontjait használva.

Egy eseményre kattintva megtekinthetjük részleteit jobban kibontva. Ekkor felugrik egy párbeszédablak, amely bemeneti adatként megkapja az adott esemény részleteit, amely ezeket megjeleníti az 53. ábrán látottak alapján.

<Appointy-event-details-modal *\*ngIf*="eventDetailsVisible" [eventDetails]="eventDetails" [events]="calendarEvents"/>

52. ábra: Esemény részletei modal (saját)

Az első sorban az esemény címe (summary-ja) található, alatta a leírás, amelyben részletezhető az esemény, ezért sok esetben hosszú is lehet, tehát szükség van egy ilyen bővített nézetre is. Ezek után a helyszín következik (ha van ilyen beállítva), majd a dátum, és az időintervallum, amikor az esemény történik. A meghívottak listája adatvédelmi okok miatt nincs itt felsorolva, hiszent ezt a modalt bárki láthatja aki használja a weboldalt, és rákattint erre az eseményre.

A képen szöveg, képernyőkép, képernyő, szoftver látható

Automatikusan generált leírás

53. ábra: Egy esemény részletei (saját)

### Esemény létrehozás űrlap

A képen szöveg, elektronika, képernyőkép, képernyő látható

Automatikusan generált leírásEsemény létrehozásához a naptár bármely szabad felületére kell kattintani, aminek hatására megnyílik egy modal, amelyben megadhatjuk a létrehozandó eseményünk adatait. Megnyíláskor a komponens előre kitölti a dátum és a kezdő- és végidőpontot az alapján, hogy hova kattintunk a naptáron. Például, ha 2024 novemberének végét vizsgálja a felhasználó, majd a 26.-i napon a 11-12 órás cellára kattint, akkor ezzel lesz kitöltve az időintervallum és a dátum is.

. ábra: Esemény létrehozása modal (saját)

Vannak kötelezően kitöltendő, és opcionálisan kitöltendő mezők az adatok közt. Kötelezően kitöltendő a cím, a kezdő- és végidőpontok, a helyszín, valamint az E-mail cím, amely a felhasználó saját E-mail címe. Ez utóbbi azért kötelező, mert ez alapján lehet hozzárendelni őt az eseményhez, és ez az alapja az esemény elfogadásáról, törléséről és szerkesztéséről való értesítéseknek és műveleteknek is. Az opcionális mezők a leírás és a további résztvevők E-mail címei.

A képen szöveg, képernyőkép, képernyő, szoftver látható

Automatikusan generált leírás

55. ábra: Példa kitöltött eseményűrlapra (saját)

### Validátorok

Az űrlap mezőinek validálásához az Angular által is támogatott Validátorokat [38] használtam, amelyekből van néhány eleve a keretrendszerbe épített, például a Validators.required és a Validators.email.

…

location: new FormControl('', [Validators.*required*]),  
email: new FormControl('', [Validators.*required*, Validators.*email*]),

…

56. ábra: Beépített validátorok ( saját)

Ha specifikusabb módon szeretnénk űrlap bemenetet validálni, akkor magunk is írhatunk validátorokat, amelyeket használhatunk az űrlapajainkon. Ezeket Reactive Formok [39] esetén a FormGroup konstruktorában második paraméterként kell megadni egy objektumba, ezen belül pedig egy tömbbe foglalva.

this.eventForm = new FormGroup({  
 summary: new FormControl('', [Validators.*required*]),

…  
 attendees: new FormControl(''),  
}, { validators:  
 [  
 *startBeforeEndValidator*(),  
 *disallowedTimeValidator*(

this.constService.DISALLOWED\_DATES, this.eventDetails.calendarEvent!.startDate!, this.eventDetails.calendarEvent!.endDate!),  
 *overlapValidator*(

this.constService.ALLOW\_OVERLAPS,

this.constService.LOCATIONS,

this.events,

this.eventDetails.calendarEvent!.startDate!,

this.eventDetails.calendarEvent!.endDate!)  
 ]  
});

57. ábra: Custom validátorok alkalmazása (saját)

Ugyanitt adhatók meg a bemeneti paramétereik függvényhíváshoz hasonlóan, ha az általunk írt validátornak szüksége van ilyenekre.

A validátorok felépítése az 58. ábrán látható. Az első sorban a függvényt deklaráljuk, és megadjuk a visszatérési értékét, ami ValidatorFn lesz. Ez után a visszaadandó függvény megírása következik, amely bemenetként megkapja az űrlap mezőit, majd a függvény testében ezeket vizsgálva vagy null-t ad vissza, amely esetben a validáció sikeres volt, vagy pedig visszaadhatunk valamilyen flaget, hogy ezzel azonosítsuk a hiba okát. Az első példán annyi került vizsgálatra, hogy a kezdő dátum a végdátum előtt található-e, és ha nem, akkor visszadobja a startBeforeEnd flaggel a hibát.

58. ábra: startBeforeEndValidator (saját)

export function *startBeforeEndValidator*(): ValidatorFn {  
 return (control: AbstractControl): ValidationErrors | null => {  
 const start = control.get('start')?.value;  
 const end = control.get('end')?.value;  
  
 if (start && end && start >= end) {  
 return {startBeforeEnd: true};  
 }  
  
 return null;  
 };  
}

A másik két általam készített validátor az overlapValidator és a disallowedTimeValidator. Ezek közül az előbbi az esemény más eseményekkel való átfedését vizsgálja a beállított konfigurációtól függően, a második pedig az időpontfoglalásra nem engedélyezett intervallumokkal való metszetet validálja.

59. ábra: overlapValidator tömb szekciója (saját)

export function *overlapValidator*(overlaps: boolean | boolean[], locations: string[], events: CalendarEvent[], startDate?: Date, endDate?: Date): ValidatorFn {

…

if (Array.isArray(overlaps)) {  
 let r = null;  
 for (let i = 0; i < overlaps.length; i++) {if (!overlaps[i] && control.get('location')?.value == locations[i]) {for (const event of events) {  
 if (event.location == locations[i]) { *//locationok egyeznek* r = decideOverlap(start, end, event);  
 if (r != null) return r;  
 }  
 }  
 }  
 }  
 return r;  
}  
 …

}

Az overlapValidator működése a fejlesztő beállításaitól függ. Az overlaps változó lehet boolean tömb, vagy egy boolean a konfigurációtól függően, a validátor első sora pedig erre tesztel rá. Ha ez tömb, akkor az elemei az adott helyszínekre vonatkoznak, tehát ha azt szeretnénk, hogy az első beállított helyszínen ne lehessen átfedésben foglalni, de a másodikon igen, akkor [false, true] értékre kell állítanunk ezt a tömböt.

Ezen a tömbön addig iterál a validátor, míg meg nem találja a kiválasztott helyszínhez kapcsolódó értéket, majd, ha nem engedélyeztük az átfedést, végignézi a decideOverlap() függvény segítségével az adott napra és helyszínre vonatkozó eseményeket is, hogy található-e köztük átfedés.

60. ábra: decideOverlap függvény (saját)

function decideOverlap(start: Date, end:Date, event:CalendarEvent){  
 if (start < event.endDate && start > event.startDate) { *//start overlapel* return {startInOverlap: true};  
 }  
 if (end < event.endDate && end > event.startDate) { *//end overlapel* return {endInOverlap: true};  
 }  
 if (start <= event.startDate && end >= event.endDate) { *// start és end közrefog egy másik eventet* return {timesEnvelopOverlap: true};  
 }  
 return null;  
}

Az első feltétel azt vizsgálja, hogy az általunk foglalni kívánt időpont kezdete beleesik-e egy másik, már létező esemény intervallumába. Ha igen, akkor visszaküldi a startInOverlap flaget, amelyből így tudjuk, hogy az időpont kezdete miatt nem jó a bemenetünk. A második feltétel ugyanezt vizsgálja, csak az időpontunk végével, és az endInOverlap flaget adja vissza, ha nem megfelelő a bemenet, a harmadik pedig azt vizsgálja, hogy az időpontunk körbe ölel-e más, a validátor szempontjából releváns időpontot (tehát az eleje a másik időpont előtt, a vége a másik időpont után van).

61. ábra: Az overlapValidator változó szekciója (saját)

…

else if(!overlaps) {  
 for (const event of events) {  
 r = decideOverlap(start, end, event);  
 }

return r;  
 }  
 return null;  
 };  
}

Ha az overlaps változó nem tömb, hanem egy egyszerű boolean, akkor az értéke minden eseményre és minden helyszínre vonatkozik. Ekkor, ha ez a változó hamis, akkor belépünk az elágazás másik ágába, amely ugyanazt vizsgálja, amit a tömb esetén is vizsgáltunk (start, vagy end átfedésben van-e, vagy az egész közre fog-e egy másik eseményt), csak helyszíntől függetlenül.

Végül, ha nincs megadva overlaps változónk, vagy igaz értékre van állítva, akkor null-t adunk vissza, ami azt jelenti, hogy minden rendben van. Ebben az esetben persze nem futott érdemi input validáció, hiszen engedélyeztük az átfedéseket.

A harmadik általam készített validátor a disallowedTimeValidator, ami ugyanarra a logikára épül, mint az overlapValidator, csak erre a use case-re faragva, ezért nem fejtem ki a működését külön.

export function *disallowedTimeValidator*(disallowedDates: (string | string[])[], startDate?: Date, endDate?: Date): ValidatorFn {  
…  
 for (const d\_date of disallowedDates) {  
 if (Array.isArray(d\_date)) {  
 const d\_start: Date = new Date(d\_date[0]);  
 const d\_end: Date = new Date(d\_date[1]);  
 if (start < d\_end && start > d\_start) {return {startInDisallowed: true};  
 }  
 if (end < d\_end && end > d\_start) {return {endInDisallowed: true};  
 }  
 if (start <= d\_start && end >= d\_end) {return {timesEnvelopDisallowed: true};  
 }  
  
 } else if (d\_date) {  
 if (*isSameDay*(d\_date, start)) {  
 return {appointmentOnDisallowedDay: true};  
 }  
 }  
 }  
 return null;  
 };  
}

62. ábra: disallowedTimeValidator (saját)

Az időpontkérésre nem engedélyezett intervallum vizuálisan is reprezentálva van a naptárban, ezzel növelve a felhasználói élményt, és egyértelműbbé téve idpontfoglalásra alkalmas intervallumot.

A képen szöveg, képernyőkép, diagram, Diagram látható

Automatikusan generált leírás

63. ábra: Időpontfoglalásra alkalmatlan intervallumok reprezentációja (saját)

Ezt az alkalmatlam területet félig áttetsző overlay reprezentálja, amelyen világos- és sötétszürke egyenesek váltják egymást 45 fokban döntve. Ezen a területen a korábban említett kattintás (amelyre előkerül az időpontfoglaló űrlap) is le van tiltva, hiszen ezen intervallumban nem lenne értelme ezt engedélyezni.

64. ábra: Alkalmatlan intervallum megjelenése kódban (saját)

background: repeating-linear-gradient(  
 -45deg,  
 hsl(360 0% 90% / 0.75),  
 hsl(360 0% 90% / 0.75) 10px,  
 hsl(360 0% 85% / 0.75) 10px,  
 hsl(360 0% 85% / 0.75) 20px  
);

### E-mail

Az új esemény rögzítésére irányuló űrlap kitöltése után az OK gombra kattintáskor a komponens e-mailt küld a fejlesztő által felkonfigurált e-mail címre (ennek a megadásáról később szó lesz). Az e-mail küldés logikájának egy részlete a 66. ábrán látható. Az egész logikát nem illesztem be a dokumentációba, mivel nagyon hosszú lenne, viszont véleményem szerint sok pluszt nem adna a megértés elősegítéséhez.

Az e-mail küldéséhez az emailJS [40] online levélküldő szolgáltatást használtam fel.

Lényegében a függvény megkapja bemenetként a létrehozandó esemény adatait, a címet amire a levelet küldjük, a címet ahova a választ várjuk, és az esemény egyeztetésének státuszát. Az adatokat rendszerezzük a levél sablon [41] által várt formátumra, amelyet betöltünk a request\_data változóba, ezután a finishedState alapján eldöntüjk milyen levelet küldünk, majd a függvény végén megtörténik a levél küldése.

…

if (this.constService.EMAIL\_BACKEND\_URL && this.constService.EMAIL\_BACKEND\_URL != "") {  
 return *lastValueFrom*(this.http.post(this.constService.EMAIL\_BACKEND\_URL, request\_data));  
} else {  
 return emailjs.*send*(this.constService.SERVICE\_ID, template\_id, request\_data, this.constService.EMAILJS\_PUBLIC\_KEY);  
}

…

65. ábra: E-mail szerver opciók (saját)

Lehetőség van saját e-mail szerver használatára is, amelyet a fejlesztő beállíthat. Ha ez megtörténik, a kérés erre a beállított url-re fog érkezni, tehát saját szerver is használható hozzá.

66. ábra: E-mail küldő kód részlet (saját)

sendMail(formValue: any, to\_email?: string, from\_email?: string, finishState: string = "inProgress") {

…  
 let request\_data: any = {  
 to\_email: to\_email ?? this.constService.COMPANY\_EMAIL,  
 reply\_to: from\_email ?? formValue.email,  
 appointment\_date: date,  
 start\_time: start,  
 end\_time: end,  
 summary: formValue.summary,  
 description: formValue.description,  
 location: formValue.location,  
 attendees: formValue.attendees  
 };  
 let template\_id;  
  
 if (finishState == "deleted") {request\_data.mail\_subject = deleted\_request\_data.mail\_subject;  
 request\_data.mail\_title = deleted\_request\_data.mail\_title;

request\_data.mail\_text = deleted\_request\_data.mail\_text;  
 request\_data.mail\_details = deleted\_request\_data.mail\_details;  
 request\_data.view\_link = this.constService.BASE\_URL;  
  
 template\_id = this.constService.FINISHED\_TEMPLATE\_ID;  
  
 } else if (finishState == "accepted") {

…

}

…

return emailjs.*send*(this.constService.SERVICE\_ID, template\_id, request\_data, this.constService.USER\_ID);

…

}

### Megegyezés

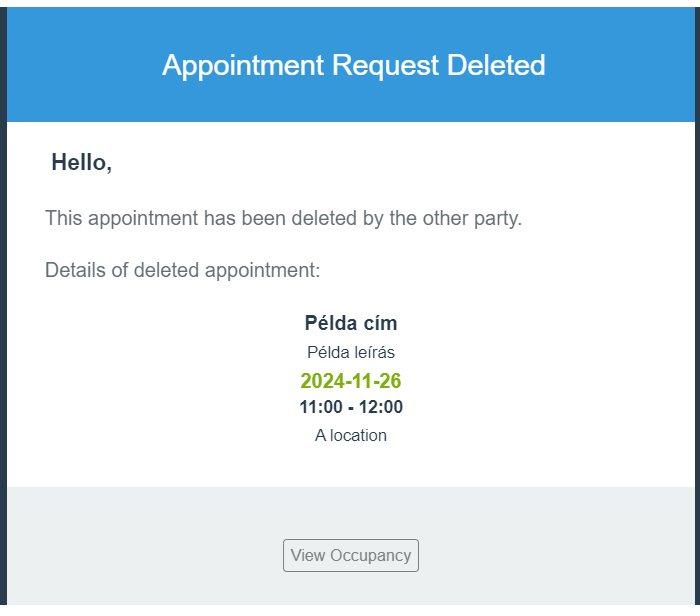
A képen szöveg, képernyőkép, Weblap, szoftver látható

Automatikusan generált leírásÚj időpontkérelem esetén a beállított illetékes egy 67. ábrához hasonló levelet fog kapni a kérésről. Ebben szerepel az időpontfoglaló e-mail címe, a foglalandó időpont címe, leírása, dátuma, intervalluma, valamint a kiválasztott helyszín.

67. ábra: Új időpontkérés beérkezése (saját)

Erre háromféleképp reagálhat a címzett.

Elutasíthatja az időpontkérelmet, amely a Delete proposal gombra nyomással történik. Ekkor a címzett az 68. ábrán látható oldalra kerül irányításra, és az időpontfoglaló értesítőt kap, hogy az időpontját törölték (69. ábra). A View Occupancy gomb itt a bekonfigurált oldalra visz, amelyen láthatóak a foglalt időpontok.

 A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

68. ábra: Időpont törlés oldal (saját)

69. ábra: Időpont törlés értesítő (saját)

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírásMásodik opcióként szerkesztheti az időpontot, amely esetben a 70. ábrán látható oldalra kerül, amely gyakorlatilag azokat a mezőket tartalmazza, mint a főoldalon felugró modal az e-mail címeken kívül. Itt az OK gombra kattintva a 67. ábrához nagyon hasonló értesítést kap a másik fél annyi különbséggel, hogy nem új időpontról, hanem szerkesztett iőpontról van benne szó.

70. ábra: Időpont szerkesztése (saját)

A harmadik opció az időpont elfogadása, amely kétféle képpen nézhet ki. Ha az illetékes adminisztrátor fogadja el az időpontot (tehát az utolsó szerkesztő az időpontfoglaló volt), akkor a 71. ábrán látható felületet kapja, ahol bejelentkezés után az időpont rögzítésre kerül a naptárban, és a másik fél értesül arról, hogy az időpontja elfogadásra került.

A képen szöveg, képernyőkép, Betűtípus, sor látható

Automatikusan generált leírás

71. ábra: Mindkét fél által elfogadott időpont (saját)

Ha az időpontot az időpont foglalója fogadja el, akkor a 72. ábrán látható felületre jut, és az adminisztrátor kap egy értesítést az időpont elfogadásáról, viszont itt az adminnak még lehetősége van szerkeszteni rajta, elutasítani és elfogadni is (a 67. ábrához hasonlóan). Ha a törlést, vagy a szerkesztést választja, a korábban tárgyalt procedúrák következnek be, ha elfogadja az időpontot pedig ismét a 71. ábrán látható oldalra kerül.

A képen szöveg, Betűtípus, képernyőkép, fehér látható

Automatikusan generált leírás

72. ábra: Foglaló által elfogadott időpont (saját)

Minden előbb felsorolt felületen a sikeres műveletről egy snackbar feltűnése biztosítja az aktuális felhasználót, amelyet a 68. ábrán láthatunk, majd 8 másodperccel ezután átirányításra kerülnek a beállított oldalra, jelen esetben a főoldalra, ahol látható a már sokat mutatott Appointy naptár felület.

## Testreszabás

A projekt egy fontos tényezője a testreszabhatóság, mind funkcionalitásban, mind vizuálisan, ezért felkonfigurálható egy appointy.json és egy appointy.scss fájllal.

{  
 "redirectURL": "http://localhost:4200",  
 "locale": "hu-HU",  
 "apiKey": "AIzaSyBGYpsRXbu27SlAYE93OLs4BXz4ADI3FXc",  
 "clientID": "703772084263-ngg5a6tfdd920qh60gf694ouodr718gc.apps.googleusercontent.com",  
 "calendarIDs": [  
"0bad952e0331a7207fc33d2a2289cc7567000bceaf1c509ca255f9a984814738@group.calendar.google.com",  
"76f7dd5c8f8118dbb084270794f6d24d6998ff95dafb423138b36f2c094d69e2@group.calendar.google.com"  
 ],  
 "discoveryDocs": "https://www.googleapis.com/discovery/v1/apis/calendar/v3/rest",  
 "scope": "https://www.googleapis.com/auth/calendar",

…

}

73. ábra: appointy.json 1/4 (saját)

A JSON [42] fájlban a funkcionalitás szempontjából fontos paramétereket lehet konfigurálni, amelyek a következők:

* *redirectURL*: Az az URL cím, amelyre az időpont egyeztetési feladatállomások végén átirányítjuk a felhasználót
* *locale*: A naptárfelületen a hónap és napok nevének nyelvéért felelős. Alapértelmezés szerint ezek angolul jelennek meg.
* *Google:*
  + *apiKey*: Az időpontok létrehozásakor szükséges Google API kulcs, enélkül nem lehet időpontot felvenni.
  + *clientID*: A Google API-kban használt egyedi azonosító, amely az alkalmazást azonosítja. Google API használatához kötelező megadni.
  + *calendarIDs*: A kezelésre szánt naptárak azonosítói. Minden naptárat egy ilyen ID egyértelműen azonosít.
  + *discoveryDocs*: A Google API-hoz tartozó, kódból olvasható JSON file. Tartalmazza az API specifikációját, ezzel az alkalmazás megismeri az API szerkezetét.
  + *scope*: A Google által használt OAuth 2.0 hitelesítési folyamat egyik eleme. Meghatározza, hogy az alkalmazás milyen mértékű hozzáférést kér a felhasználó adataihoz. Esetünkben a naptár ezek a naptárakkal történő műveletek.

{

…

"emailjsPublicKey": "\_KlycIcm2HzL4TL3v",  
 "serviceID": "service\_vsa8qqp",  
 "inProgressTemplateID": "template\_h0h44bn",  
 "finishedTemplateID": "template\_99ge21g",  
 "companyEmail": "bbalint0404@gmail.com",

…

}

* *Levelezés:*

74. ábra: appointy.json 2/4 (saját)

* + *emailjsPublicKey*: Az EmailJS levélküldő szolgáltatás által fiókokhoz rendelt egyedi publikus kulcs.
  + *serviceID*: A bekonfigurált e-mail szolgáltató egyedi azonosítója. Ez határozza meg, hogy az EmailJS melyik e-mail szolgáltatón keresztül küldi el a leveleket (pl.: Gmail, Yahoo, stb.)
  + *inProgressTemplateID*: A folyamatban lévő egyeztetéshez használt e-mail template azonosítója.
  + *finishedTemplateID*: A befejezett egyeztetésekről küldött értesítől e-mail template azonosítója.
  + *companyEmail*: Az időpontokat kiadó cég, vállalat, vállalkozó, vagy szervezet e-mail címe, ahova a kérelmeket várják.

75. ábra: appointy.json 3/4 (saját)

{

…

"baseURL": "http://localhost:4200",  
 "editRoute": "edit-event",  
 "acceptRoute": "accept-event",  
 "deleteRoute": "delete-event",  
 "createEventRoute": "create-calender-event",

…

}

* *Routing:*
  + *baseURL*: A route-ok alap URL címe, a lejjebb beállított route-ok ehhez vannak hozzáfűzve.
  + *editRoute*: Az időpontkérelem szerkesztése gombra kattintáskor megnyíló oldal route-ja. Az EventEditComponent-et erre a route-ra kell helyezni.
  + *acceptRoute*: Az időpontkérelem elfogadásakor megnyíló oldal route-ja. Az EventAcceptComponent-et erre a route-ra kell helyezni.
  + *deleteRoute*: Az időpontkérelem elutasításakor megnyíló oldal route-ja. Az EventDeleteComponent-et erre a route-ra kell helyezni.
  + *createEventRoute*: Az egyeztetett időpont felvételénél használt route. A CreateCalendarEventComponent-et erre a route-ra kell helyezni.

76. ábra: appointy.json 4/4 (saját)

{

…

"locations": ["A location", "B location"],  
 "maxAttendees": [5, 2],  
 "disallowedDates": [  
 "2024-11-07",  
 [  
 "2024-11-09T10:22:00Z",  
 "2024-11-09T13:40:00Z"  
 ],  
 [  
 "2024-11-04T14:08:00Z",  
 "2024-11-06T08:40:00Z"  
 ]  
 ],  
 "allowOverlaps": [false, true]

}

* *locations*: Az időpont kérelmező űrlapon megjelenő helyszínek, amelyek közül a felhasználó választhat. Ha több naptár és helyszín van megadva, a sorrendnek megfelelően veszi fel az eseményeket a naptárba, tehát az első helyszínen lévők az első naptárba kerülnek, a másodikon lévők a másodikba, és így tovább.
* *maxAttendees*: Az időpont kérelmező űrlapon megadható extra megadható résztvevők száma. Egy számérték esetén minden helyszínre az a korlátozás vonatkozik, tömbbe foglalt számok esetén a sorrendnek megfelelő helyszínre vonatkozik a korlátozás (az első szám az első helyszínre, és így tovább).
* *disallowedDates*: A foglalásra nem engedélyezett intervallumokat lehet itt megadni. lehet egész napot, egy napon belüli intervallumot, vagy több napon átívelő intervallumot megadni.
* *allowOverlaps*: Egy boolean érték esetén globálisan beállítható, hogy engedélyezünk-e időpont ütközést, tömbben megadva viszont a sorrendnek megfelelő helyszínre vonatkozik (az első érték az első helyszínre, és így tovább).

Az appointy.scss fájlban a komponens színeit tudjuk testre szabni, hogy a lehető legmegfelelőbben illeszkedjen egy adott cég brandjéhez, vagy egy tulajdonos ízléséhez.

:root {  
 --background-color: hsl(0 100% 100%);  
 --primary-color: hsl(215, 75%, 40%);  
 --secondary-color: hsl(215, 90%, 60%);  
  
 --secondary-text-color: white;  
}

77. ábra: appointy.scss (saját)

A megadható színek a következők:

* *--background-color*: A teljes komponens háttérszínét állítja
* *--primary-color*: Gombok háttérszínét, és a jövőbeli, valamint aktív események naptáron megjelenő hátterét határozza meg
* *--secondary-color*: A múltbeli (már megtörtént) időpontok naptáron megjelenő hátterét határozza meg
* *--secondary-text-color*: Másodlagos szín, gombok feliratának a színét, és a jövőbeli, valamint aktív események naptáron megjelenő szövegének színét határozza meg

# Összegzés

Összességében a diplomamunka elkészítése alatt sok tapasztalatot szereztem a scope menedzselésben, az időgazdálkodás területén, és felmerülő vizuális, vagy funkcionális hibák javításában.

A dolgozatom projektje a felgyorsult világ egyik legfontosabb és legdrágább erőforrásával foglalkozik, az idővel, hiszen egy időpontkezelő webkomponenst fejlesztettem Angular keretrendszerben, amely segítséget biztosít mind a vállalatok, mind a felhasználók számára az időpontok foglalásához. A komponens célja, hogy testre szabható és integrálható legyen különböző weboldalakba, miközben intuitív, felhasználóbarát felületet kínál a felhasználók részére. Az alkalmazás a Google Calendar API-n keresztül valósítja meg az események kezelését, biztosítva a felhasználók számára időpontjaik egyszerűbb kezelését.

A fejlesztés során külön figyelmet fordítottam a felhasználói élmény javítására, és az Angular komponens-alapú struktúrájával könnyen bővíthető és karbantartható megoldást biztosítottam, amelyet végeláthatatlanul lehetne bővíteni újabb és újabb funkciókkal.

Az eredmények alapján a komponens sikeresen kommunikál a Google Calendar API-val, és biztosítja a reszponzív felhasználói felületet. Az Angular környezet és a Google Calendar API integrációja során több technikai kihívással szembesültem, például az OAuth 2.0 alapú hitelesítés megoldásával, amelyet sikeresen kezeltem.

A jövőbeli fejlesztési irányok között szerepel például a konfigurálhatóság új szintekre juttatása és a Google Calendar API újabb funkcióinak integrálása. Összességében úgy vélem, hogy a dolgozatban bemutatott időpontkezelő webkomponens jelentős mértékben hozzájárulhat a naptár-alapú alkalmazások fejlesztéséhez, és elősegíti az egyszerűbb és hatékonyabb időpontkezelést a webes platformokon.

# Irodalomjegyzék

[1] S. M. W. Design és flo.rooke, „What is the history behind the calendar?”, Portals to the Past. Elérés: 2024. december 9. [Online]. Elérhető: https://www.portalstothepast.co.uk/history-behind-calendar/

[2] „9 Google Calendar features you should start using now | Zapier”. Elérés: 2024. április 18. [Online]. Elérhető: https://zapier.com/blog/google-calendar-schedule/

[3] „Google Calendar”, *Wikipedia*. 2024. január 29. Elérés: 2024. április 12. [Online]. Elérhető: https://en.wikipedia.org/w/index.php?title=Google\_Calendar&oldid=1200623137

[4] „Mike Samuel - Stealth | LinkedIn”. Elérés: 2024. április 18. [Online]. Elérhető: https://www.linkedin.com/in/mikevsamuel

[5] D. C. Contributor, „Google’s »20% rule« shows exactly how much time you should spend learning new skills—and why it works”, CNBC. Elérés: 2024. április 18. [Online]. Elérhető: https://www.cnbc.com/2021/12/16/google-20-percent-rule-shows-exactly-how-much-time-you-should-spend-learning-new-skills.html

[6] „HTML5”, *Wikipedia*. 2024. április 18. Elérés: 2024. április 18. [Online]. Elérhető: https://en.wikipedia.org/w/index.php?title=HTML5&oldid=1219524906

[7] „HTML”, *Wikipedia*. 2024. április 16. Elérés: 2024. április 18. [Online]. Elérhető: https://en.wikipedia.org/w/index.php?title=HTML&oldid=1219285383

[8] „HTML Standard”. Elérés: 2024. május 6. [Online]. Elérhető: https://html.spec.whatwg.org/multipage/

[9] „CSS”, *Wikipedia*. 2024. április 9. Elérés: 2024. április 18. [Online]. Elérhető: https://en.wikipedia.org/w/index.php?title=CSS&oldid=1217996434

[10] „Cascading Style Sheets”. Elérés: 2024. május 6. [Online]. Elérhető: https://www.w3.org/Style/CSS/Overview.en.html

[11] „How to use Selector Inheritance and Multiple Inheritance ?”, GeeksforGeeks. Elérés: 2024. december 9. [Online]. Elérhető: https://www.geeksforgeeks.org/how-to-use-selector-inheritance-and-multiple-inheritance/

[12] „Sass: @mixin and @include”. Elérés: 2024. május 6. [Online]. Elérhető: https://sass-lang.com/documentation/at-rules/mixin/

[13] „SASS Vs SCSS: What’s The Difference?”, InterviewBit. Elérés: 2024. április 18. [Online]. Elérhető: https://www.interviewbit.com/blog/sass-vs-scss/

[14] „File:Sass Logo Color.svg - Wikipedia”. Elérés: 2024. május 6. [Online]. Elérhető: https://commons.wikimedia.org/wiki/File:Sass\_Logo\_Color.svg

[15] „Sass: Sass Basics”. Elérés: 2024. április 18. [Online]. Elérhető: https://sass-lang.com/guide/

[16] „Sass: Variables”. Elérés: 2024. május 6. [Online]. Elérhető: https://sass-lang.com/documentation/variables/

[17] M. Peng, „Multithreading Javascript”, Hackmamba. Elérés: 2024. április 24. [Online]. Elérhető: https://medium.com/techtrument/multithreading-javascript-46156179cf9a

[18] R. via C. Williams, *English: Unofficial JavaScript logo by Chris Williams, from GitHub logo.js, under very permissive licensing (WTFPL).* 2011. Elérés: 2024. április 24. [Online]. Elérhető: https://commons.wikimedia.org/wiki/File:JavaScript-logo.png

[19] „First-class Function - MDN Web Docs Glossary: Definitions of Web-related terms | MDN”. Elérés: 2024. április 24. [Online]. Elérhető: https://developer.mozilla.org/en-US/docs/Glossary/First-class\_Function

[20] „14 Things You Can Do with JavaScript”. Elérés: 2024. december 9. [Online]. Elérhető: https://www.orientsoftware.com/blog/what-can-you-do-with-javascript/

[21] „Prototype-based programming - MDN Web Docs Glossary: Definitions of Web-related terms | MDN”. Elérés: 2024. december 9. [Online]. Elérhető: https://developer.mozilla.org/en-US/docs/Glossary/Prototype-based\_programming

[22] „JavaScript Programming Paradigms”. Elérés: 2024. május 6. [Online]. Elérhető: https://byby.dev/js-paradigms

[23] „JavaScript | MDN”. Elérés: 2024. április 24. [Online]. Elérhető: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[24] „TypeScript”, *Wikipedie*. 2023. október 30. Elérés: 2024. április 24. [Online]. Elérhető: https://cs.wikipedia.org/w/index.php?title=TypeScript&oldid=23328659

[25] „Documentation - The Basics”. Elérés: 2024. május 6. [Online]. Elérhető: https://www.typescriptlang.org/docs/handbook/2/basic-types.html

[26] „Angular - What is Angular?” Elérés: 2024. december 9. [Online]. Elérhető: https://v17.angular.io/guide/what-is-angular

[27] W. A. Solutions, „The Complete Guide to Angular — Past, Present & Future”, Medium. Elérés: 2024. szeptember 30. [Online]. Elérhető: https://medium.com/@whitenappsolutions/the-complete-guide-to-angular-past-present-future-af6ec4c99a55

[28] „Angular”. Elérés: 2024. szeptember 30. [Online]. Elérhető: https://angular.dev/

[29] „npm”, *Wikipedia*. 2024. szeptember 12. Elérés: 2024. szeptember 30. [Online]. Elérhető: https://en.wikipedia.org/w/index.php?title=Npm&oldid=1245308296

[30] „About npm | npm Docs”. Elérés: 2024. szeptember 30. [Online]. Elérhető: https://docs.npmjs.com/about-npm

[31] A. Janes, „Why WebStorm is the most productive IDE for JavaScript development”, Medium. Elérés: 2024. szeptember 30. [Online]. Elérhető: https://aaron-janes.medium.com/why-webstorm-is-the-most-productive-ide-for-javascript-development-3223918b197a

[32] „WebStorm: The JavaScript and TypeScript IDE, by JetBrains”, JetBrains. Elérés: 2024. szeptember 30. [Online]. Elérhető: https://www.jetbrains.com/webstorm/

[33] *microsoft/vscode*. (2024. szeptember 30.). TypeScript. Microsoft. Elérés: 2024. szeptember 30. [Online]. Elérhető: https://github.com/microsoft/vscode

[34] „How to Fix Issues With CSS Position Sticky Not Working”, DEV Community. Elérés: 2024. november 12. [Online]. Elérhető: https://dev.to/robmarshall/how-to-fix-issues-with-css-position-sticky-not-working-4a18

[35] „RxJS”. Elérés: 2024. november 25. [Online]. Elérhető: https://rxjs.dev/

[36] „RxJS - Observable”. Elérés: 2024. november 25. [Online]. Elérhető: https://rxjs.dev/guide/observable

[37] „ChangeDetectorRef”. Elérés: 2024. november 26. [Online]. Elérhető: https://angular.dev/api/core/ChangeDetectorRef

[38] „Validators”. Elérés: 2024. november 26. [Online]. Elérhető: https://angular.dev/api/forms/Validators

[39] „Reactive Forms”. Elérés: 2024. november 26. [Online]. Elérhető: https://angular.dev/guide/forms/reactive-forms

[40] „Send email directly from your code”, EmailJS. Elérés: 2024. november 25. [Online]. Elérhető: https://www.emailjs.com/

[41] U. Inc, „Appointment Reminder email Template | Unlayer”. Elérés: 2024. november 25. [Online]. Elérhető: https://unlayer.com/templates/appointment-reminder

[42] „JSON”. Elérés: 2024. november 27. [Online]. Elérhető: https://www.json.org/json-en.html

# Ábrajegyzék

[1. ábra: A Google Calendar felülete [2] 3](#_Toc184651339)

[2. ábra: Mike Samuel, a Google Calendar atyja [3] 4](#_Toc184651340)

[3. ábra: HTML5 logó [6] 8](#_Toc184651341)

[4. ábra: Példa HTML kódra (saját) 9](#_Toc184651342)

[5. ábra: CSS3 logó [9] 10](#_Toc184651343)

[6. ábra: Példa CSS kódra (saját) 11](#_Toc184651344)

[7. ábra: SASS logó [14] 11](#_Toc184651345)

[8. ábra: SCSS szelektor öröklődés [11] 12](#_Toc184651346)

[9. ábra: A mixin és az include [12] 12](#_Toc184651347)

[10. ábra: Példa az SCSS és a CSS szintaxis közötti különbségre [15] 13](#_Toc184651348)

[11. ábra: A 12. ábrán látható példa alap SASS szintaxissal [15] 13](#_Toc184651349)

[12. ábra: JavaScript logó [18] 14](#_Toc184651350)

[13. ábra: TypeScript logó [24] 16](#_Toc184651351)

[14. ábra: Angular logó [27] 17](#_Toc184651352)

[15. ábra: npm logó [26] 18](#_Toc184651353)

[16. ábra: WebStorm logó [31] 19](#_Toc184651354)

[17. ábra: WebStorm UI [32] 20](#_Toc184651355)

[18. ábra: VSCode UI [33] 20](#_Toc184651356)

[19. ábra: Az Appointy egy nap-oszlopának kezdete (saját) 21](#_Toc184651357)

[20. ábra: Az Appoiny időskálával bővített felülete (saját) 21](#_Toc184651358)

[21. ábra: Az Appointy felülete teljes képernyőn (saját) 22](#_Toc184651359)

[22. ábra: Az Appointy felülete egy 500\*800 pixeles helyre szorítva (saját) 23](#_Toc184651360)

[23. ábra: A naptártest magasságának számítására alkalmazott képlet 23](#_Toc184651361)

[24. ábra: A naptár oszlopainak generálása (saját) 24](#_Toc184651362)

[25. ábra: A naptár oszlopain belüli téglalapok generálása (saját) 24](#_Toc184651363)

[26. ábra: Általános for ciklus funkcionalitás (saját) 25](#_Toc184651364)

[27. ábra: Időskála oszlopának stílusa (saját) 26](#_Toc184651365)

[28. ábra: Elrendezést segítő filler elem (saját) 26](#_Toc184651366)

[29. ábra: A görgetősáv szélesség megoldása (saját) 26](#_Toc184651367)

[30. ábra: A kezdőidőponthoz görgetés (saját) 27](#_Toc184651368)

[31. ábra: Fejléc generálása (saját) 28](#_Toc184651369)

[32. ábra: Nap és dátum megjelenítése (saját) 28](#_Toc184651370)

[33. ábra. A hét napjainak feltöltése (saját) 29](#_Toc184651371)

[34. ábra: Kezdődátum kezdeti meghatározása (saját) 30](#_Toc184651372)

[35. ábra: Kezdődátum korrigálása (saját) 31](#_Toc184651373)

[36. ábra: A hét dátumainak feltöltése (saját) 32](#_Toc184651374)

[37. ábra: Év- és hónap kijelző, lapozó gombok (saját) 33](#_Toc184651375)

[38. ábra: Hét középső napjának meghatározása (saját) 33](#_Toc184651376)

[39. ábra: Megjelenített hónap egy kevert hónapú héten (saját) 34](#_Toc184651377)

[40. ábra: A hét léptetése (saját) 35](#_Toc184651378)

[41. ábra: Az események lekérése az initCalendar-ban (saját) 36](#_Toc184651379)

[42. ábra: Egy hét eseményeinek lekérése (saját) 37](#_Toc184651380)

[43. ábra: Események rendszerezése (saját) 38](#_Toc184651381)

[44. ábra: Esemény elhelyezése a felületen (saját) 39](#_Toc184651382)

[45. ábra: Megjelenített események (saját) 40](#_Toc184651383)

[46. ábra: Now marker (saját) 40](#_Toc184651384)

[47. ábra: Esemény kártya változóinak frissítése (saját) 41](#_Toc184651385)

[48. ábra: Esemény kártya pozíciójának beállítása (saját) 41](#_Toc184651386)

[51. ábra: Kártya színének beállítása (saját) 42](#_Toc184651387)

[50. ábra: Kártya magasságának beállítása (saját) 42](#_Toc184651388)

[49. ábra: Kártya felső margójának beállítása (saját) 42](#_Toc184651389)

[52. ábra: Esemény részletei modal (saját) 43](#_Toc184651390)

[53. ábra: Egy esemény részletei (saját) 43](#_Toc184651391)

[54. ábra: Esemény létrehozása modal (saját) 44](#_Toc184651392)

[55. ábra: Példa kitöltött eseményűrlapra (saját) 45](#_Toc184651393)

[56. ábra: Beépített validátorok ( saját) 45](#_Toc184651394)

[57. ábra: Custom validátorok alkalmazása (saját) 46](#_Toc184651395)

[58. ábra: startBeforeEndValidator (saját) 47](#_Toc184651396)

[59. ábra: overlapValidator tömb szekciója (saját) 48](#_Toc184651397)

[60. ábra: decideOverlap függvény (saját) 49](#_Toc184651398)

[61. ábra: Az overlapValidator változó szekciója (saját) 50](#_Toc184651399)

[62. ábra: disallowedTimeValidator (saját) 51](#_Toc184651400)

[63. ábra: Időpontfoglalásra alkalmatlan intervallumok reprezentációja (saját) 52](#_Toc184651401)

[64. ábra: Alkalmatlan intervallum megjelenése kódban (saját) 52](#_Toc184651402)

[65. ábra: E-mail szerver opciók (saját) 53](#_Toc184651403)

[66. ábra: E-mail küldő kód részlet (saját) 54](#_Toc184651404)

[67. ábra: Új időpontkérés beérkezése (saját) 55](#_Toc184651405)

[68. ábra: Időpont törlés oldal (saját) 56](#_Toc184651406)

[69. ábra: Időpont törlés értesítő (saját) 56](#_Toc184651407)

[70. ábra: Időpont szerkesztése (saját) 56](#_Toc184651408)

[71. ábra: Mindkét fél által elfogadott időpont (saját) 57](#_Toc184651409)

[72. ábra: Foglaló által elfogadott időpont (saját) 57](#_Toc184651410)

[73. ábra: appointy.json 1/4 (saját) 58](#_Toc184651411)

[74. ábra: appointy.json 2/4 (saját) 59](#_Toc184651412)

[75. ábra: appointy.json 3/4 (saját) 60](#_Toc184651413)

[76. ábra: appointy.json 4/4 (saját) 61](#_Toc184651414)

[77. ábra: appointy.scss (saját) 62](#_Toc184651415)

# Mellékletek

A projekt GitHub linkje:

https://github.com/ThisIsJustAGuy/szakdoga