Önálló laboratórium beszámoló

Szoftverfejlesztés .NET platformon

Készítette: Horváth Szabolcs
Konzulens: Simon Balázs

A Blazor framework

A modern, adat vezérelt webalkalmazások fejlesztése jellemzően két részre bomlik: szerveroldali fejlesztés és kliensoldali fejlesztés. Szerveroldali fejlesztés jellemző nyelvei a C#, Java, PHP, stb…, míg kliensoldalon szinte kizárólag JavaScript-et és a nyelvet kiegészítő keretrendszereket használjuk. A két fejlesztési oldal elválik egymástól programozási nyelv mentén. Egy full-stack fejlesztőnek értenie kell, mind a backend nyelveihez, mind a frontend rengeteg JavaScript frameworkjéhez.

A Blazor az utóbb említett nehézséget igyekezik kiküszöbölni azzal, hogy lehetőséget ad kihasználni a C# fejlett szerveroldali lehetőségeit, mellette pedig akár a kód egy részének újra felhasználásával, megírni a kliensoldali webes interface-t, ugyanazon a nyelven.

A Blazor egy open-source keretrendszer, ami WebAssembly segítségével lehetővé teszi összetett webes felületek létrehozását .NET-ben. Így kihasználhatjuk a .NET keretrendszer hasznos funkcióit és könyvtárait, mint például a LINQ.

Blazor projektünk létrehozásakor két hosting modelből választhatunk:

- *Blazor WebAssembly Hosting Model*
  Ennek a modelnek a használatakor, a C# kódunk kliensoldalon fut, a böngészőben. A szükséges könyvtárak letöltődnek a böngészőbe, amikor az oldalra navigálunk. Ennek előnye, hogy nincs szükség állandó kapcsolatra egy szerverrel, az egész kliensoldal a böngészőben fut.

  A Blazor WebAssembly alkalmazásunk lehet teljesen kliensoldali. Ilyenkor csak statikus fájlokból dolgozik az alkalmazás, így akár teljesen offline is tud működni, miután betöltöttük azt.

  Másik lehetőségünk, hogy egy ASP.NET projekttel együtt hozzuk létre. Ilyenkor az ASP.NET-es projekt lesz a backendje az alkalmazásnak, és a Blazor WebAssembly-s projekt ezen keresztül tud dinamikusan adatot lekérni a szervertől. Nagy előnye így együtt létrehozni a két projektet, hogy könnyű kódot újra felhasználni a két projekt között (pl.: model osztályok, validáció).

- *Blazor Server-Side Hosting Model*
  Ennek a modelnek a használatakor az alkalmazás a szerveren fut, a böngésző pedig egy SignalR kapcsolaton keresztül kommunikál a szerverrel. A böngészőnek nem kell letöltenie a .NET könyvtárakat, a szerver elküldi neki a megjelenítendő tartalmat.

  Az eventeket a kapcsolaton keresztül kezeli az alkalmazás, és ha az oldal tartalma változik, a szerver kiszámolja a különbséget a böngészőben lévő és a újonnan generált HTML között, és a böngészőnek csak azt a tartalmat küldi el, ami változott. Ennek köszönhetően az alkalmazás gyorsnak, reszponzívnak érződik.

  A model nagy hátránya, hogy folyamatos kapcsolatra van szüksége a szerverrel. Ha a szerver leáll, az alkalmazás használhatatlan. Mivel a szerver végzi a munka nagy részét, az ezt a modelt használó alkalmazásaink jóval rosszabbul skálázódnak, mint a WebAssembly-s model esetében. További probléma lehet a magas késleltetés ami egy leterhelt szerver, vagy egy rossz internet kapcsolat esetén jelentkezhet.

A MudBlazor komponens könyvtár

Blazor-höz sokféle komponens könyvtár érhető el, melyek több funkcionalitást biztosító, illetve natív Blazor-ben nem létező komponenseket biztosítanak a fejlesztők számára. Ezek egyike a MudBlazor, ami egy ingyenes, open-source könytár sokféle komponenssel és extrával. A könyvtárat többek között a JetBrains is támogatja, illetve a készítő szeretne képzéseket, és több előre elkészített stílust és template-et kínálni, így talán elmondható, hogy a könyvtárat hosszabb távon is támogatni és fejleszteni fogják.

A könyvtár Material Design-ra alapoz, és igyekszik olyan komponenseket biztosítani, amelyek mellett nem lesz szükségünk egyáltalán CSS vagy JavaScript kód írására. A könyvtár .NET 6.0-ot javasol, de kompatibilis .NET 5.0-val is. A könyvtárt telepíteni kifejezetten egyszerű a dokumentáció „Getting Started” oldala alapján, de akár használhatjuk az egyik Template-jüket is, amiben minden elő van készítve ahhoz, hogy a könyvtárt használjuk. Alapvető layoutokat is tudunk másolni a dokumentáció „Wireframes” oldaláról.

A könyvtár készítői egy playground oldalt is biztosítanak a könyvtárhoz TryMudBlazor néven, ahol ki lehet próbálni a könyvtárat kisebb elemeit online.

A könyvtárhoz le lehet még tölteni egy extra package-et is Theme Manager néven, ami meg tudja könnyíteni az oldal stílusának tesztelését azzal, hogy futás közben lehet vele változtatni az alkalmazás témájának tulajdonságait.

A könyvtár biztosít lehetőséget színek, betűtípusok és árnyékok testreszabására, illetve rendelkezik ikonoknak egy nagy, kereshető gyűjteményével, így nem kell azokat külső forrásból beszerezni.

A könyvtár a Bootstrapes osztályokat is jól kezeli, így az abban megszokott osztályokkal tudunk margint, paddinget és sok mást állítani. A Bootstrapes classokon kívül definiál saját CSS osztályokat is, mint például a .mud-elevation-x osztály.

MudBlazor komponens könyvtár vizsgálata

Layout

- **Össze lehet-e rakni a Blazor alap elrendezését a könyvtár segítségével? (Fent fejléc, Bal oldalon menü, Menü összecsukható)**
  A könyvtár sokféle lehetőséget biztosít az alap layout kialakítására:
  - <MudAppBar>: Fejléc, melyre ikonokat, szöveget, linkeket tudunk elhelyezni többek között.
  - <MudDrawer>: Navigation Drawer, melynek fejlécét a <MudDrawerHeader> komponenssel tudjuk személyre szabni. A <MudDrawerContainer> komponenssel a Drawer tartalmához tudunk extra formázást adni. A @bind-Open="…" használatával a Drawer automatikusan bezáródik navigáció hatására. A MudDrawer több variációval rendelkezik:
    - Temporary: Az oldal tartalma fölé gördül be. Elem kiválasztására, vagy egyéb akcióra magától bezáródik.
    - Persistent: Arrébb tolja az oldal tartalmát mikor kinyílik. Nem csukódik be magától, csak explicit Open="false" parameter beállításra.
    - Responsive: Széles képernyőn Persistent-ként viselkedik, kis képernyőn pedig Temporaryként. Ha az ablak mérete túl kicsi, magától bezáródik. A képernyő méret breakpointját testre lehet szabni.
    - Mini: Ha MudNavLink komponenseket használunk a MudDrawerben, akkor ez a típus Open="false" állapotban nem tűnik el, hanem összezsugorodik akkorára, hogy a navigációs linkek ikonjai még láthatóak maradjanak. (A screenshoton ez látható)

  Be lehet állítani, hogy a Drawer az AppBart eltolja vagy fölé gördüljön, illetve hogy a teteje az AppBarral egy szinten, vagy az AppBar alatt legyen.

  - <MudContainer>: Egyszerű komponens az oldal tartalmának középre igazításához. Fixed="true" beállítás mellett a legközelebbi breakpoint mérethez igazodik. Fixed="false", azaz Fluid állapotban automatikusan méreteződik, de meg lehet adni neki MaxWidth és egyéb paramétereket.
  - <MudGrid>: A Bootstraphez hasonló 12 oszlopos grid, melyen meg lehet adni az egyes elemeknek, adott képernyőméreten hány oszlopot foglaljanak el. További Class-ok hozzáadásával az egyes elemek közötti távolságokat, azok igazítását, margint és paddinget lehet állítani többek között.
  - <MudPaper>: Ebbe csomagolva más komponenseket, megkönnyíti azok formázását.
  - <MudToolBar>: AppBarhoz hasonló megjelenésű toolbar leszámítva, hogy akárhova elhelyezhető.

Táblázat

Movies

Search by title

Title	IMDb Rating	Runtime	Year	Director(s)	Release Date
Druk	7,7	117	2020	Thomas Vinterberg	2020-09-12
The Silence of the Lambs	8,6	118	1991	Jonathan Demme	1991-01-30
Reservoir Dogs	8,3	99	1992	Quentin Tarantino	1992-01-21
Schindler's List	9	195	1993	Steven Spielberg	1993-11-30
The Chestnut Man	7,7	50	2021		2021-09-29
Spider-Man: No Way Home	8,7	148	2021	Jon Watts	2021-12-13

Rows per page: 10 1-10 of 128

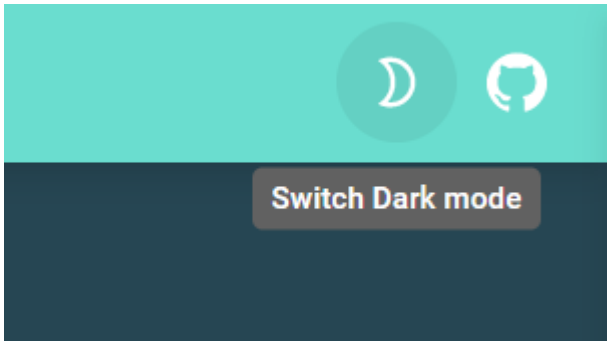
1. ábra Táblázat egy keresőmezővel a Toolbar-jában

- **Van-e táblázat komponens?**
  Van, <MudTable> a megnevezése. A legalapabb elrendezésében tartalmaz egy <HeaderContent> komponenst, amiben az oszlopcímeket tudjuk beállítani, illetve egy <RowTemplate> komponenst, amiben az egyes oszlopokban lévő adatokat tudjuk definiálni. A <RowTemplate> elemen belül a @context attribútumon keresztül tudjuk elérni magát az adatot. A cella tartalmát <MudTd> elemekben tudjuk testreszabni. A táblázat elemeit a <MudTable> komponens Items paraméterével tudjuk beállítani.
- **Ha sok sor van, lehet-e lapozni? Lehet-e lejjebb görgetéssel újabb sorokat betölteni?**
  A <MudTable> komponenshez hozzá lehet adni a <MudTablePager> komponenst, ami a lapozást teszi lehetővé. A pager-nek lehet adni PageSizeOptions[] paramétert, amiben lévő int értékek reprezentálják a választási lehetőségeinket, hogy egy oldalon hány elemt szeretnénk megjeleníteni.
- **Lehet-e fixálni sorokat és oszlopokat, hogy görgetéskor mindig látszódjanak?**
  A <MudTable> komponens FixedHeader és/vagy FixedFooter paraméterét true-ra állítva, az oszlopcímeket tartalmazó sor fixen látható lesz görgetés közben.
- **Lehet-e rendezni a sorokat adott oszlop tartalma szerint?**
  A sorokat rendezhetővé úgy tudjuk tenni, ha az oszlopcímeket tartalmazó <MudTh> tag-en belül <MudTableSortLabel> címkéket használunk, melyeknek SortBy paraméterében egy delegate-t megadva definiálhatjuk hogyan hasonlítsa össze az elemeket. A <MudSortLabel> címkékre kattintva az oszlop szerint rendeződik a táblázat. Újabb kattintásra fordítva rendez, majd még egyszer kattintva megszünteti a rendezést. A rendezést le lehet tiltani az Enabled paraméterrel.
- **Lehet-e egyszerre több oszlop tartalma rendezni a sorokat?**
  Nem, egyszerre csak egy oszlop szerint lehet rendezni.
- **Lehet-e szűrni a sorokat adott oszlop tartalma szerint?**

Lehet, ha a <MudTable> komponensen beállítjuk a Filter paramétert egy delegate-re, ami egy bool-t visszaadva megmondja, hogy egy adott sor illeszkedik-e a filterre. A keresés paraméterét megadhatjuk mondjuk egy keresőmezővel, aminek értékét bind-oljuk egy változóhoz, majd a delegate-nek átadott függvényben a változó alapján eldöntjük mikor illeszkedik egy sor és mikor nem.

- **Lehet-e egyéni sablon alapján tartalmat adni a celláknak?**
  A <RowTemplate> tagen belül lévő <MudTd> elembe akármit belerakhatunk. Így olyan layoutot és olyan elemeket tudunk egy cellán belül létrehozni, amilyet csak akarunk.
- **Lehet-e egyéni stílust adni oszlopoknak, soroknak vagy celláknak?**
  <MudTable> komponensbe elhelyezve egy <ColGroup> elemet, egyesével tudunk stílust beállítani az oszlopoknak. (pl.: `<col style="width: 10%" />`)
- **El lehet-e rejteni oszlopokat?**
  Nincs szép, beépített megoldás erre a MudBlazorben, viszont az oszlop stílusát a fentebb említett módon tudjuk állítani, így például egy `style="display: none"` -t beállíthatunk, ezzel elrejtve az oszlopot.
- **Át lehet-e rendezni oszlopokat?**
  Nincs rá beépített lehetőség.
- **Lehet-e dinamikusan generálni az oszlopokat? Ha pl. nagyon sok oszlop van, egy Selectből/ComboBox-ból kiválasztott érték alapján más-más oszlopok látszódjanak.**
  Oszlopok elrejtésére és megjelenítésére nincs könnyenhasználható beépített megoldás, azonban olyat tudunk csinálni, hogy egy-egy sorhoz a <ChildRowContent> elembe elhelyezhetünk olyan adatokat, amik az adott sorhoz tartoznak, de mondjuk csak egy gomb megnyomására jelenjenek meg (pl.: egy Person típust megjelenítő sorhoz, az Address típust megjelenítő táblázat mint ChildRowContent). A <ChildRowContent>-be akármit elhelyezhetünk, és itt is rendelkezésünkre áll a @context változó, ami reprezentálja a sorhoz tartozó adatot.

Tooltip/Popover

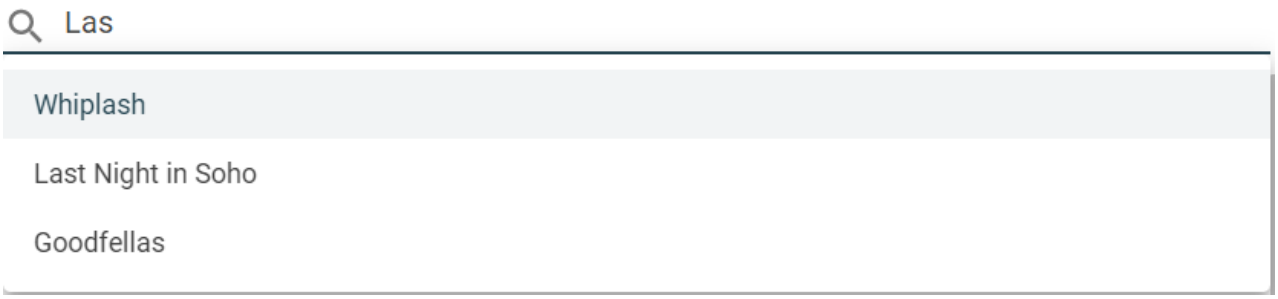


2. ábra Tooltip megjelenik, amikor az ikonra visszük az egerünket

- **Van-e felugró ablak komponens, amely akkor jelenik meg, ha az egérrel egy adott elem fölé megyünk?**
  Van, <MudTooltip> a neve. Egyszerűen körbe kell venni egy komponens a <MudTooltip> komponenssel, és megadni a Text tulajdonságot. A tooltipet lehet színezni, kis nyilat adni hozzá, késleltetve megjeleníteni, és beállítani melyik oldalon jelenjen meg.
- **Lehet-e HTML tartalommal feltölteni a felugró ablak tartalmát?**
  Igen, a tooltip tartalmát a <MudTolltip> tagen belül egy <TooltipContent> tagen belül tudjuk definiálni. Ilyenkor a komponenst, amihez akarjuk rendelni ezt a tooltipet, egy <ChildContent> taggel kell körbevenni.
- **Lehet-e körbe forgó töltés ikont megjeleníteni a felugró ablakban, amíg a tartalom dinamikusan töltődik a háttérben?**
  Igen, egy @if feltételében megvizsgálhatjuk a tartalom meglétét, és amíg az nem töltődött be, addig például egy <MudProgressCircular>-t jelenítünk meg.

Select/ComboBox/Autocomplete

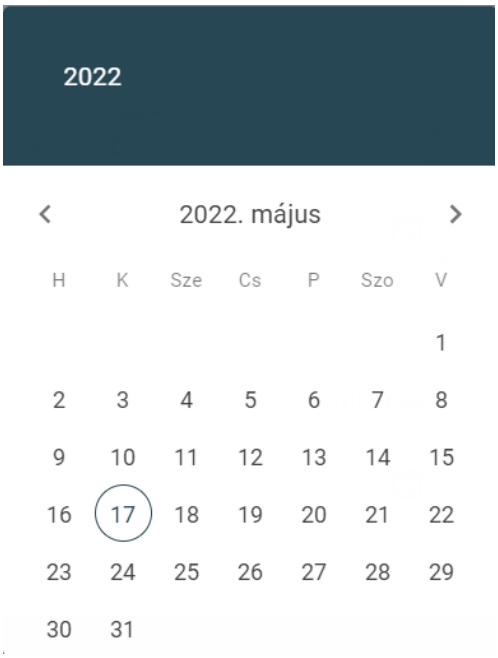
3. ábra Lenyitott Select komponens



4. ábra Autocomplete mező amivel filmekben lehet keresni cím alapján

- **Van-e olyan komponens, amely fel tud sorolni kiválasztható alternatívákat (mint a Select/ComboBox) és lehetőséget ad arra is, hogy egyéni értéket írjunk bele (mint az InputText)?**
  Van elemeket felsoroló komponens, <MudSelect> a neve. Egyéni értéket nem tudunk közvetlenül a Select komponensnek megadni. Ha komponensre kattintunk és leütünk egy karaktert, kitölti az első illeszkedő választási lehetőséggel, de ennél mélyebb autocomplete-et nem implementál.
- **Ha nincs, szimulálható-e ez a viselkedés Autocomplete segítségével?**
  Lézetik külön <MudAutoComplete> komponens is, ami amellett, hogy tud hasonlóan viselkedni mint a <MudSelect> komponens, képes egyéni bemenetet is kezelni, illetve meg lehet neki adni egy search function-t, aminek segítségével autocomplete viselkedést valósít meg.
- **Az Autocomplete beállítható-e úgy, hogy mindig kinyíljon, ha a fókusz a komponensre kerül?**
  Ez az alapvető viselkedése a komponensnek. A különböző beállításaitól függ, hogy rögtön megjeleníti-e a lista elemeit, vagy csak miután elkezdtünk gépelni (pl CoerceText attribútum).
- **Lehet-e lenyitó ikont rakni az Autocomplete-re, hogy ha erre az ikonra kattintunk, akkor kinyíljon?**
  A komponensen alapból van lenyitó ikon, amit az Adornment tulajdonsággal lehet testreszabni.

5. ábra Dátumválasztó komponens



6. ábra Az előző ábra jobb oldali ikonjára kattintva ez jelenik meg

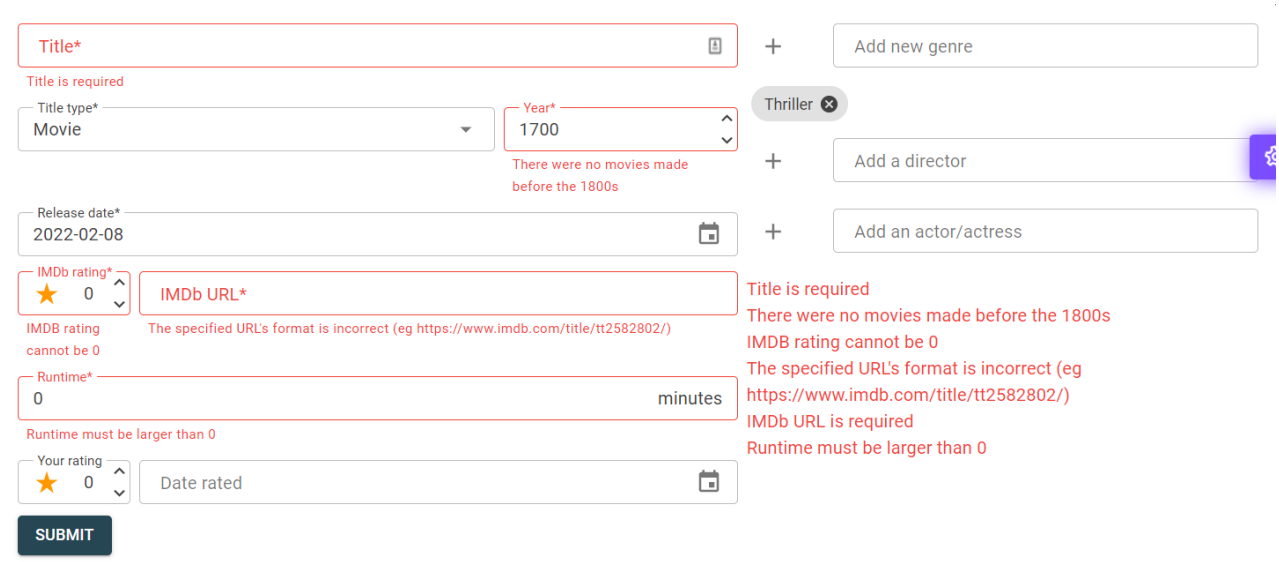
- **Van-e dátumválasztó komponens?**
  Van, <MudDatePicker> a komponens neve.
- **Lehet-e konfigurálni az időzónát?**
  Időzónát közvetlenül nem kezeli a komponens. Ha a @bind-Date paraméter segítségével összekötjük egy DateTime változóval, akkor van objektumunk a helyi időre, amit aztán a TimeZoneInfo.ConvertTimeToUtc metódussal UTC-re konvertálhatunk majd tetszőleges eltolást alkalmazhatunk rajta.
- **Lehet-e konfigurálni a dátum megjelenési formátumát?**
  A DateFormat paraméterben megadhatjuk a dátum megjelenési formáját (pl: „yyyy-MM-DD”). Megadhatunk egy DateMask paramétert is, ami kézzel való begépelés során segít betartani a megadott dátum formátumot (pl: kirakja helyettünk a kötőjeleket). A dátumválasztást természetesen egy felugró dialógusablakkal is véghez lehet vinni, ami hasonlít az alap HTML type=date input element dialógusára. A dátumválasztó-dialógus rengeteg testreszabási lehetőséggel rendelkezik, és természetesen a kiválasztott dátum illeni fog a DateFormat-ra. (Fontos, hogy DateMask használat esetén az értéke megegyezzen a DateFormat értékével)

ProgressBar

7. ábra Egy indeterminate progress bar

- **Van-e lehetőség kijelezni, hogy hol tart egy háttérfolyamat?**
  Van, a <MudProgressCircular> és <MudProgressLinear> komponensekkel. A komponens lehet Determinate, ha meg lehet becsülni hol tart egy adott folyamat, illetve Indeterminate ha csak azt akarjuk mutatni, hogy valamilyen háttérfolyamatra, de nem tudjuk az hol tart. Testre lehet szabni a színt, méretet, legyen-e százalék kiírás, stb…

Form



8. ábra Egy összetett form, többféle validációs hibával

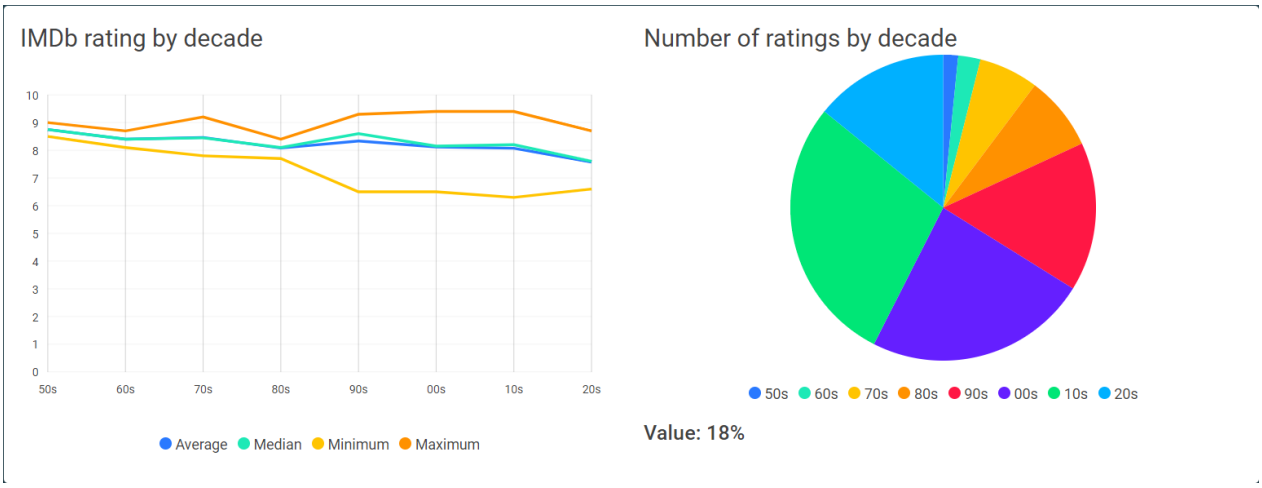
- **Formok validációja hogyan történik?**
  Többféle lehetőség van validáció megvalósítására.
  A <MudForm>-ot legegyszerűbben úgy lehet validálni, hogy az egyes bemeneti mezőkhöz megadjuk a Required illetve Validation attribútumokat. A <MudForm> rendelkezik egy bind-IsValid attribútummal, amivel a validáció sikerességét tudjuk változóba menteni, illetve egy bind-Errors attribútuma, amivel a hiba stringeket tudjuk összegyűjteni. A validációt a Mudform.Validate függvény indítja.
  Másik lehetőség a Blazor-ös <EditForm> használata, aminek megadunk egy model osztályt, ami a form beviteli mezőit fogja össze, és ebben az osztályban a tulajdonságokhoz annotációval tudunk felvenni validációt.
  A <MudForm> lehetőséget nyújt arra is, hogy a FluentValidation könyvtárat használjuk, amivel jól olvasható és könnyen implementálható validációt tudunk megvalósítani. Ráadásul a validációs osztályokat újra fel tudjuk használni a szerveroldali validációhoz is ASP.NET-ben, azonban pillanatnyilag a FluentValidation csak Third-Party könyvtárakon keresztül használható Blazorrel problémamentesen. Legalábbis .NET 6 környezetben le se lehet fordítani a solutiont, ha FluentValidation függőségünk van.

- **Látszik-e pirossal a hibásan kitöltött mező, zölddel helyesen kitöltött/javított mező?**
  A validációs hibák megjelennek a beviteli mezők alatt, illetve a mező körvonala is pirosra vált validációs hiba esetén. Ha az input megfelel a validációnak, azt nekünk kell külön C# kódból jelezni az input mező stílusának változtatásával, vagy más módon.
- **Lehet-e a hibaüzeneteket a mezők mellé/alá írni?**
  A hibaüzeneteket alapból megjelennek a beviteli mezők alatt validációs hiba esetén.
- **Ki lehet-e külön gyűjteni a hibaüzeneteket?**
  A hibaüzeneteket form-szinten is össze lehet gyűjteni egy listába, a bind-Error attribútummal, és kiíratni azokat.
- **Lehet-e saját ellenőrzési szabályt felvenni?**
  Az egyes beviteli mezők Validate tulajdonságában tetszőleges stringgel vagy IEnumerable<string>-el visszatérő delegate-et meg tudunk adni, ami megvalósíthatja a custom validációt.

Naptár

- **Van-e naptár komponens (Google Calendar-hoz hasonló)?**
  A könyvtár nem rendelkezik Naptár komponenssel. Ha mindenképp ilyen komponensre lenne szükségünk, a Radzen komponenskönyvtár Scheduler nevű komponense egy jó megoldás lehet.
- **Lehet-e eseményeket felvenni, törölni, módosítani?**
  A Radzen Scheduler komponense rendelkezik ezekkel a funkciókkal.

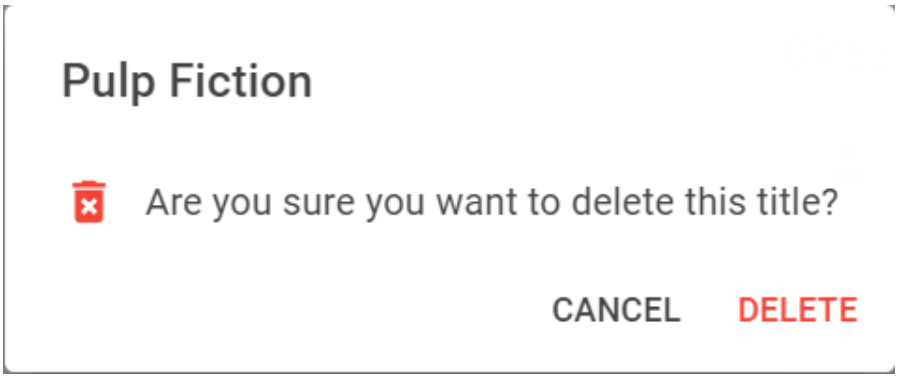
Grafikon



9. ábra Egy Line- és egy PieChart

- **Lehet-e grafikonokat rajzolni?**
  Lehet, a <MudChart> nevű komponens segítségével 4 fajta grafikont lehet rajzolni. A ChartType property segítségével választható ki a grafikon típusa. A grafikonnak lehet adni egy ChartOptions objektumot, amiben pár megjelenést érintő tulajdonságot lehet beállítani. A grafikonok testreszabási lehetőségei viszonylag szerények: segédvonalakat, beosztást, vonalvastagsot lehet állítani, de ezeken kívül mást nem nagyon. A grafikonokra lehet saját SVG contentet rakni, a <CustomGraphics> tag-et beágyazva.
- **Milyen grafikonok vannak? (kördiagram, oszlopdiagram, stb.)**
  - Bar: Oszlopdiagram, ChartSeries nevű objektumokban kapja az adatait, az oszlop label-öket pedig az XAxisLabels propertyben.
  - Line: Vonaldiagram, hasonló a használata az oszlopdiagramhoz.
  - Pie: Kördiagram, ChartSeries-ek helyett egy double[] tömböt kap a Data propertyjébe.
  - Donut: Fánkdiagram, azonos viselkedés a kördiagrammal.
- **Lehet-e real-time grafikont rajzolni? (pl. tőzsdei árfolyam)**
  A grafikonok adatait lehet real-time változtatni a kódból, de gyakorlatban nehézkes lehet komplexebb viselkedést megvalósítani úgy, hogy az jól is nézzen ki, a szűk testreszabási lehetőségek miatt.

Dialógus ablak



10. ábra Dialógus ablak custom contenttel (ikon + szöveg)

- **Van-e lehetőség saját dialógus ablak megjelenítésére?**
  Van. Az oldal valamelyik központi komponensében fel kell venni egy <MudDialogProvider> komponenst. Ezt más komponensekből úgy érjük el, hogy @inject-el injektálunk egy IDialogService-t.
  A dialógusablak layoutját érdemes felvenni egy külön komponensenek. Ebben a komponensben egy <MudDialog> komponens definiálja a dialog layoutját. A <MudDialog>-on belül a <DialogContent> határozza meg a fő tartalmat, a <DialogActions>-ben pedig a dialógus alján lévő akciókat definiálhatjuk.
  A dialógust megjeleníteni a DialogService.Show<> metódusával tudjuk megjeleníteni. A dialógus típusát a Show függvény template paraméterében tudjuk definiálni. A Show függvényben opciókat is meg lehet adni (pl. pozíció, méret, Esc billentyű bezárja-e, stb…).
  A dialógus kaphat paramétert, hasonlóan bármely másik komponenshez, [Parameter] annotációval ellátott propertykben.
- **Van-e lehetőség beépített dialógus ablak megjelenítésére (pl. szöveg + csak OK gomb, csak OK Cancel gomb tartalommal)**
  Nincs alap dialógus, mindenképp meg kell adni a Show függvénynek egy típust, ami egy <MudDialog>-gal rendelkező komponens. Azonban egy csak OK Cancel gombokkal rendelkező dialógust össze lehet rakni 4 sorból.
- **Egy dialógus ablakból lehet-e újabb, de másfajta dialógus ablakot nyitni?**
  Lehet, a dialógus komponensében egy [Inject] annotációval ellátott propertyben kaphat DialogService-t, amit akadály nélkül használhat.
- **Egy dialógus ablakból lehet-e újabb, de ugyanolyan dialógus ablakot nyitni?**
  Nincs semmi akadálya.

Üzenetek

5. ábra Megjelenő Snackbar személy sikeres hozzáadásakor

- **Lehetőség van-e üzenetek megjelenítésére? (pl. történt valami, vagy bejelentkezéskor egy értesítés)**
  Többféle komponens is van feedback megjelenítésre:
  - <MudAlert>: Egyszerű kis komponens, ami egy ikont és kis szöveget tud megjeleníteni. A bordert és a színeket testre lehet szabni. Alapvetően statikus, de a sarkában lehet megjeleníteni kis X-et amivel bezárhatjuk, azonban ezt pár sor kóddal kell megoldanunk.
  - <MudSnackbarProvider>: Hasonlóan a MudDialogProviderhez, ezt is érdemes globálisan egyszer felvenni, majd injektálni ahol kell. A Snackbart, az Alert-tel ellentétben, kódból dinamikusan tudjuk csak megjeleníteni a SnackbarProvider Add metódusával. A tartalma lehet ikon, szöveg, action button, de akár még custom HTML markup is.
  - <MudBadge>: Más komponensek sarkába tudunk rakni egy kis színes pöttyöt, amin jelezhetjük például a bejövő üzenetek számát, de akár ikont is rakhatunk bele.
- **Lehet-e saját HTML tartalma az üzenetnek?**
  A Snackbar a tartalmát HTML stringként kezeli, ezért akármilyen markup-ot adhatunk neki. Arra a dokumentáció is felhívja a figyelmet, hogy user content-et nem szabad Snackbarban így megjeleníteni, mivel az biztonsági kockázatot jelent.

Fájlok feltöltése

- **Van-e lehetőség egy vagy több fájl feltöltésére?**
  Van, a natív <InputFile> komponenst felhasználva. Ha bármilyen MudButton for attribútumát beállítjuk az input id attribútumának értékére. Ennek a gombnak aztán olyan stílust adhatunk amilyet csak szeretnénk, de akár másféle komponenst is használhatunk, ami képes Click event kezelésére. Akár Drag-and-Drop fájlfeltöltést is meg tudunk valósítani, ha olyan komponenst használunk az input label-jének, ami tudja kezelni a drag eventeket.
  Magát a feltöltés kezdeményezését az input OnChanged eventjével tudjuk kezelni. Az InputFileChangeEventArgs paraméteren keresztül tudjuk bekérni a fájlokat.
- **Van-e lehetőség jelezni, hol tart a feltöltés?**
  Nincs a könyvtárnak beépített megoldása erre. Saját kézzel úgy oldhatjuk meg ezt, hogy chunkonként olvassuk be a fájlt, és valamilyen Timer segítségével frissítjük a progress mérőt.