Razvoj spletnih aplikacij za razvijalce zalednih sistemov

Mitja Krajnc, Jani Kajzovar, Andraž Leitgeb

Databox, Ptuj, Slovenija
mitja.krajnc@databox.com, jani.kajzovar@databox.com, andraz.leitgeb@databox.com

Pojav frontend razvojnih ogrodij je še povečal vrzel med frontend in backend razvojem. V podjetju smo želeli razširiti funkcionalnosti interne spletne aplikacije in s tem zagotoviti večjo učinkovitost ekip, vendar nismo imeli prostih frontend razvijalcev. Zato smo se odločili raziskati alternativne pristope, ki bi omogočil razvoj frontend in backend aplikacij v enakem jeziku in okolju. Primerno rešitev ponuja razvoj aplikacij z .NET 8 Blazor in MudBlazor, kar bomo predstavili v članku. Z .NET 8 Blazor lahko razvijemo robustne in interaktivne spletne aplikacije v C#, brez potrebe po učenju dodatnega jezika. To poenostavi in pohitri razvoj, saj se razvijalci lahko osredotočijo na logiko in funkcionalnost aplikacije. Še večjo poenostavitev predstavlja uporaba knjižnice MudBlazor, ki temelji na Material UI in je zasnovana za Blazor. Ponuja izdelane in prilagodljive komponente, kar omogoča hitro izdelavo uporabniških vmesnikov. Tako smo izkoristili prednosti .NET razvoja in omogočili backend razvijalcem enostaven pristop k razvoju spletnih rešitev. V članku bomo predstavili Blazor in MudBlazor, razvojno strukturo, izzive in rezultate. Rešitev predstavlja spletna aplikacija, ki po strukturi spominja na backend projekte in omogoča hiter razvoj internih aplikacij, in s tem opolnomoči backend razvijalce. Dodali smo tudi peskovnik za preizkušanje konceptov uporabniške izkušnje brez dodatnega dela za frontend razvijalce.

Ključne besede:
Blazor
.NET
MudBlazor
spletne aplikacije
zaledni sistemi

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.7

1 Uvod

V svetu razvoja programske opreme je hitra prilagodljivost ključna. Z nenehnim razvojem novih tehnologij je pomembno, da ekipe čim prej osvojijo in izkoristijo nove priložnosti. Ena izmed takšnih tehnologij je .NET 8 Blazor [1], ki prinaša sveže inovacije in izboljšave za razvoj spletnih aplikacij. Z njim lahko backend razvijalci, ki so že seznanjeni z .NET, hitro in učinkovito ustvarjajo privlačne uporabniške vmesnike brez potrebe po dodatnem znanju frontend knjižnic, kot so React, Vue ali Angular.

Zadnja različica Blazor ponuja vrsto novih funkcionalnosti, ki olajšajo delo razvijalcem. Med najpomembnejšimi so izboljšave zmogljivosti, razširitev podpore za WebAssembly, izboljšana orodja za razvoj in razhroščevanje ter številne nove komponente, ki poenostavijo gradnjo kompleksnih aplikacij.

Za potrebe našega projekta smo se odločili uporabiti zbirko komponent MudBlazor [2], ki je komponentna knjižnica za Blazor. Omogoča enostavno in hitro ustvarjanje lepih in odzivnih uporabniških vmesnikov. Z uporabo MudBlazor lahko naši backend razvijalci hitro integrirajo napredne funkcionalnosti in izboljšajo uporabniško izkušnjo, brez potrebe po dodatnem učenju in raziskovanju frontend tehnologij.

Zastava celotnega projekta je temeljila na čim večji domačnosti backend razvijalcev, tako da smo se v strukturi projekta zgledovali predvsem po obstoječih projektih, ki se uporabljajo za različne backend storitve in ne toliko po strukturi projektov, ki jih prinašajo tipični frontend projekti. Osnova projekta je bila tako imenovana čista arhitektura (clean architecture) [3], pri čemer smo v eno rešitev združili vse od storitvenega nivoja do uporabniškega vmesnika.

S tem smo povečali produktivnost ekipe in zmanjšali odvisnost od zunanjih virov. To nam omogoča hitrejši razvoj in lažjo prilagodljivost. S strateško odločitvijo uporabe Blazor in MudBlazor smo izkoristili moč .NET ekosistema in opolnomočili backend razvijalce, kar nam omogoča bolj učinkovito delo in hitrejše doseganje ciljev. S tem pristopom smo backend razvijalcem omogočili, da ustvarjajo visokokakovostne interne aplikacije, ki so tako funkcionalne kot estetsko dovršene, kar je ključno za uspešno delovanje naše ekipe in podjetja.

2 Blazor

Blazor je tehnologija za razvoj spletnih aplikacij, ki omogoča ustvarjanje interaktivnih uporabniških vmesnikov z uporabo C# in .NET, namesto tradicionalnih JavaScript ogrodij. Z izdajo .NET 8 Blazor je Microsoft še dodatno nadgradil in izboljšal to orodje, kar prinaša številne prednosti in novosti za razvijalce. O samem Blazorju je bilo na preteklih OTS konferencah že povedanega marsikaj, zato bomo predstavili samo nekatere izmed novosti, ki so prišle v zadnji verziji.

2.1. Novosti v .NET 8 Blazor

Zadnja verzija .NET, ki je v uradnem izidu, je stara sicer nekaj več kot pol leta, a prinaša s seboj kar nekaj izboljšav, ki jih bomo predstavili v spodnjih odstavkih [4];

- **Boljša zmogljivost in hitrost:** Ena izmed glavnih prednosti zadnje inačice Blazor je izboljšana zmogljivost. Optimizacije pri nalaganju in izvajanju aplikacij omogočajo hitrejše odzivne čase in manjšo porabo virov. To pomeni, da lahko razvijalci ustvarijo odzivnejše aplikacije. Le-te se hitreje nalagajo in delujejo gladko tudi na napravah z nižjimi specifikacijami.
- **Razširjena podpora za WebAssembly:** WebAssembly omogoča, da v brskalniku teče visoko zmogljiva koda, kar izboljšuje zmogljivost in odzivnost aplikacij. Z zadnjo verzijo se razširja podpora za WebAssembly, kar pa omogoča še boljšo integracijo z obstoječimi aplikacijami in orodji, ter prinaša boljše zmogljivosti in večjo prilagodljivost pri razvoju.

- **Izboljšana orodja za razvoj in razhroščevanje:** Novi razhroščevalniki, boljša diagnostika napak in napredna orodja za profiliranje aplikacij omogočajo hitrejše odkrivanje in odpravljanje težav ter izboljšujejo kakovost končnih izdelkov.
- **Nove komponente in razširitve:** Osma verzija tudi prinaša vrsto novih komponent in razširitev, ki poenostavljajo razvoj kompleksnih aplikacij. Te komponente vključujejo napredne uporabniške vmesnike, izboljšane možnosti za oblikovanje in prilagoditve, ter boljšo integracijo z drugimi .NET orodji in knjižnicami.
- **Večja podpora za mobilne naprave:** Več pozornosti je bilo namenjene tudi podpori za mobilne naprave. Pri tem imamo v mislih predvsem prilagoditev za različne zaslone, optimizacijo dotika in izboljšana zmogljivost na mobilnih napravah. S tem je zagotovljena boljša uporabniška izkušnja za uporabnike.
- **Optimizacija nalaganja:** Inačica vključuje tudi napredne tehnike za optimizacijo nalaganja, kot so lazy loading, zmanjšanje velikosti prenosa in boljše upravljanje z odvisnostmi. Te izboljšave zmanjšujejo čas nalaganja aplikacij in izboljšujejo uporabniško izkušnjo.
- **Izboljšave pri upravljanju stanja in navigaciji:** Upravljanje stanja in navigacije je učinkovitejše in poenostavljeno. Novi mehanizmi za upravljanje stanja, izboljšana podpora za večstranske aplikacije in boljša integracija z usmerjanjem omogočajo lažje upravljanje kompleksnih aplikacij.

2.2. Načini upodabljanja

Zadnja različica prinaša več novosti in izboljšav pri načinih upodabljanja. Ti načini omogočajo večjo fleksibilnost in dodatno zmogljivost pri ustvarjanju spletnih aplikacij. V tem poglavju bomo predstavili, kakšni so ti načini, kako delujejo in katere prednosti prinašajo [5].

- **Interactive Server:** Je ena izmed ključnih funkcionalnosti Blazor. Omogoča izvajanje aplikacije na strežniku, medtem ko se uporabniški vmesnik prikazuje v brskalniku. V tej verziji so bile narejene pomembne izboljšave glede zmogljivosti in zanesljivosti, kar omogoča boljše delovanje aplikacij pri večjem številu uporabnikov.
- **Interactive WebAssembly:** Omogoča izvajanje C# kode neposredno v brskalniku, brez potrebe po strežniški infrastrukturi. Kot že prej omenjeno je bila razširjena podpora za WebAssembly, kar prinaša hitrejše nalaganje aplikacij, boljšo integracijo z obstoječimi JavaScript knjižnicami in večjo zmogljivost. Omogočeno je enostavno klicanje JavaScript funkcionalnosti direktno iz C# kode.
- **Interactive Auto:** Nov hibridni način upodabljanja, kjer se del aplikacije izvaja na strežniku in je na voljo uporabniku takoj. Del aplikacije pa se izvaja neposredno v brskalniku s pomočjo WebAssembly.
- **Static Server:** Omogoča pred-upodabljanje vsebin na strežniku in njihovo pošiljanje kot statične HTML strani. To bistveno pospeši nalaganje strani in izboljša SEO, saj so vsebine dostopne iskalnikom že ob prvem nalaganju.

Za izdelavo všečnega uporabniškega vmesnika, je ne glede na način upodabljanja na voljo več različnih komponentnih knjižnic, ki se razlikujejo tako po izgledu, funkcionalnostih, enostavnosti uporabe in ceni. Pregledali smo kar nekaj popularnih knjižnic, denimo: Blazorise, Telerik UI for Blazor, MudBlazor, Ant Design Blazor, Radzen. Po tehtnem premisleku in glede na naš primer uporabe, smo se odločili za uporabo knjižnice MudBlazor.

3 MudBlazor

MudBlazor je knjižnica komponent za Blazor, odprtokodni okvir, ki omogoča izdelavo interaktivnih spletnih aplikacij z uporabo C# in HTML. MudBlazor je zasnovan na principih Material Designa, kar pomeni, da nudi moderni, vizualno privlačen uporabniški vmesnik, ki je hkrati enostaven za uporabo in prilagajanje. Z MudBlazor lahko tudi backend razvijalci hitro in enostavno ustvarijo odzivne in estetsko dovršene spletne aplikacije Pri tem potrebujejo minimalno CSS kode, dodatno pa se izognejo poglobljenemu znanju frontend tehnologij, kot so React, Vue ali Angular [2].

3.1. Prednosti MudBlazor

- Enostavna uporaba in integracija: MudBlazor je zasnovan za hitro integracijo v obstoječe Blazor projekte. Knjižnica je dobro dokumentirana in ponuja obsežne primere uporabe, kar omogoča razvijalcem, da hitro začnejo z delom. Integracija v projekt je preprosta in vključuje namestitev paketa in konfiguracijo nekaj nastavitev.
- Material Design: MudBlazor temelji na Material Design smernicah, ki jih je razvilo podjetje Google. To zagotavlja sodoben in enoten videz komponent, kar prispeva k boljši uporabniški izkušnji. Knjižnica ponuja širok spekter predlog in komponent, ki jih je mogoče enostavno prilagoditi potrebam projekta.
- Odzivnost in prilagodljivost: Komponente MudBlazor so zasnovane tako, da so odzivne in se prilagajajo različnim velikostim zaslonov, kar omogoča uporabo na različnih napravah, od mobilnih telefonov do namiznih računalnikov. To omogoča razvijalcem, da ustvarijo aplikacije, ki so dostopne širokemu spektru uporabnikov.
- Bogata zbirka komponent: MudBlazor vključuje številne vnaprej pripravljene komponente, kot so gumbi, obrazci, tabele, navigacijske vrstice, dialogi in še več. To omogoča razvijalcem, da se osredotočijo na poslovno logiko aplikacije, namesto da bi izgubljali čas z oblikovanjem osnovnih elementov uporabniškega vmesnika.
- Aktivna skupnost in podpora: MudBlazor ima aktivno skupnost razvijalcev, ki redno prispeva k razvoju in izboljšavam knjižnice. To zagotavlja, da je knjižnica vedno posodobljena in da so morebitne težave hitro odpravljene.
- Z uporabo MudBlazorja lahko naši backend razvijalci hitro in učinkovito ustvarijo privlačne in funkcionalne spletne aplikacije, ne da bi morali pridobiti dodatno znanje iz frontend tehnologij. To povečuje produktivnost, zmanjšuje stroške razvoja in omogoča hitrejše doseganje poslovnih ciljev.

Odprtokodnost knjižnice in aktivna skupnost ima kot dodaten bonus tudi razširitve osnovnih knjižnic, tako da se nabor komponent, ki jih lahko razvijalec uporabi še dodatno poveča z uporabo katere izmed nadgradenj, kot sta denimo MudExtensions [6] in Mudex [7].

3.2. Kako začeti z MudBlazor

Namestitev

Razvoj z knjižnico MudBlazor je v osnovi možen na dva različna načina:

- uporaba predloge in izdelava novega projekta s pomočjo te predloge,
- ročna namestitev knjižnice v že obstoječi projekt.

Zaradi načine strukturiranja projekta za backend razvijalce, pri čemer smo kot že omenjeno uporabili našo strukturo rešitve, smo se zato določili za drugo možnost in sicer za ročno namestitev. Ta poteka po enakem postopku kot pri .NET knjižnicah z uporabo package managerja. V obstoječi Blazor projekt smo dodali paket Mudblazor.

```
dotnet add package MudBlazor
```

Naslednji korak je bil dajanje referenc v datoteko _Imports.razor, da je knjižnica dostopna na vseh straneh in Blazor komponentah, ki jih bomo ustvarili.

```
@using MudBlazor
```

Popolnoma identičen postopek se uporabi pri uporabi katerekoli nadgradnje, v našem primeru je bila to knjižnica MudExtensions.

Glede na način upodabljanja je potrebno v "index.html" ali "App.razor" dodati tudi osnovne CSS stile in pa JavaScript datoteke.

```
<linkhref="_content/MudBlazor/MudBlazor.min.css" rel="stylesheet" />
<script src="_content/MudBlazor/MudBlazor.min.js"></script>
```

Konfiguracija

Znotraj datoteke Program.cs dodamo referenco na MudBlazor storitve in jih registriramo.

```
using MudBlazor.Services;
builder.Services.AddMudServices();
```

Po registraciji MudBlazor storitev je na vrsti še vključitev osnovnih ponudnikov, za delovanje ogrodja. Pri tem je potrebno poudariti, da je samo ThemeProvider obvezen, ostale pa lahko poljubno dodamo v kolikor jih potrebujemo.

```
<MudThemeProvider/>
<MudPopoverProvider/>
<MudDialogProvider/>
<MudSnackbarProvider/>
```

Po tej konfiguraciji smo pripravljeni za izdelavo spletnih strani s pomočjo MudBlazor komponent.

Uporaba komponent

Po dokončanju konfiguracije, smo pripravljeni za uporabo MudBlazor komponent. Na spodnjem primeru kode smo definirali stran, ki uporabi MudBlazor storitev za delo z dialogi in preda tej storitvi načrt kako naj se dialog izriše in s katerimi nastavitvami. V drugem delu pa smo definirali MudBlazor dialog, ki uporablja MudBlazor dialog API za prikaz preprostega sporočila, ki ga preko parametrov pridobi od glavnega okna. Načrt dialoga vključuje samo vsebino, medtem, ko se nastavitve oblike in obnašanja kot že omenjeno nastavijo prej.

Spodnji izseku kode je del strani, ki vsebuje gumb, ki odpre definirano metodo. V sami metodi se nato poveča nek števec in odpre dialog.

```
@inject IDialogService DialogService

<MudButton @onclick="OpenDialogAsync" Color="Color.Primary">

  Open Counter

</MudButton>

@code {

  private int _counter = 1;

  private Task OpenDialogAsync()

  {

    var parameters = new DialogParameters();

    parameters.Add("Text", "Hello, MudBlazor!");

    parameters.Add("Counter", _counter.ToString());

    _counter++;

    var options = new DialogOptions() {

      CloseButton = true,

      MaxWidth = MaxWidth.Large

    };

    return DialogService.ShowAsync<DialogBlurryExample_Dialog>("Simple Dialog", parameters,
options);

    }

}
```

Naslednji izsek kode pa prikazuje ločeno komponento za izris dialoga v katerega preko parametrov pridobimo tekst in števec ter vse to prikažemo znotraj dialoga.

```
<MudDialog>
  <DialogContent>
    @Text
  </DialogContent>
  <DialogContent>
    @Counter
  </DialogContent>
  <DialogActions>
    <MudButton OnClick="Cancel">Cancel</MudButton>
    <MudButton Color="Color.Primary" OnClick="Submit">Ok</MudButton>
  </DialogActions>
</MudDialog>

@code {
  [CascadingParameter]
  private MudDialogInstance MudDialog { get; set; }
  [Parameter]
  public string Text { get; set; }
  [Parameter]
  public string Counter { get; set; }
  private void Submit() => MudDialog.Close(DialogResult.Ok(true));
  private void Cancel() => MudDialog.Cancel();
}
```

4 Zastavljena struktura rešitve

Zasnova celotne rešitve, je bila načrtovana z namenom, da našim razvijalcem omgočimo kar se da lažji razvoj. Pri tem smo preučili različne projektne strukture iz različnih ogrodij in jih uskladili ter prilagodili s strukturo, ki jo uporabljamo za razvoj naših storitev in mikrostoritev. V osnovi smo tako uporabili princip čiste arhitekture, ki smo ga malo prilagodili z uporabo vzorca Backend for Frontend.

4.1. WebAssembly kot hrbtenica spletne rešitve

WebAssembly je platforma, ki omogoča izvajanje programov, napisanih v različnih programskih jezikih, neposredno v brskalniku. Namesto da bi bil omejen le na JavaScript, lahko WebAssembly izvede kode, napisane v C, C++, C#, Rust in drugih jezikih, po predhodni pretvorbi v Wasm format. Ta format je strojno neodvisen, kar pomeni, da lahko deluje na različnih napravah in operacijskih sistemih brez sprememb kode.

- Visoka zmogljivost: WebAssembly je zasnovan za nameonom, da se izvaja s hitrostjo blizu standardnih aplikacijam. Zaradi binarnega formata in učinkovite uporabe procesorskih zmogljivosti je v marsičem hitrejši od interpretiranega JavaScript-a.

- Kompaktna velikost: Wasm datoteke so kompaktne, kar omogoča hitrejše nalaganje in manjšo porabo pasovne širine. To je še posebej pomembno za aplikacije z omejenimi viri in počasnimi internetnimi povezavami.

- Interoperabilnost: WebAssembly lahko sodeluje z obstoječim JavaScript-om in spletnimi API-ji. To omogoča postopno uvajanje WebAssembly komponent v obstoječe spletne aplikacije, ne da bi bilo potrebno popolno prepisovanje.

- Večjezična podpora: Z WebAssembly lahko razvijalci uporabljajo različne programske jezike, ki so jim bolj domači ali primerni za določene naloge. To omogoča ponovno uporabo obstoječih knjižnic in kodnih baz, kar zmanjšuje čas razvoja in stroške.

- Varnost: WebAssembly deluje v izoliranem peskovniku v brskalniku, kar zagotavlja visoko raven varnosti. Vgrajeni varnostni mehanizmi preprečujejo nedovoljen dostop do sistema in občutljivih podatkov.

Zaradi zgoraj naštetih prednosti je bil Blazor z uporabo Web Assembly izbran za izdelavo naših internih rešitev. S tem smo pridobili vse prednosti dinamične spletne strani, ki se izvaja na odjemalcu (podobno kot Single Page Application) po vrhu vsega pa je večina implementacije izvedla v C# jeziku.

4.2. Uporaba vzorca Backend for Frontend

Backend za Frontend (BFF) je arhitekturni vzorec, ki vključuje ustvarjanje ločenega backend sloja za vsako frontend aplikacijo ali tip uporabnika [8]. Glavna ideja BFF je, da se vsakemu uporabniškemu vmesniku ali napravi omogoči lasten backend, ki je prilagojen specifičnim potrebam in zahtevam tega vmesnika. To omogoča boljšo optimizacijo in prilagodljivost ter zmanjšuje kompleksnost komunikacije med frontend in backend sistemi.

Prednosti BFF

- **Prilagojena API izpostavljenost:** BFF omogoča prilagoditev API-jev specifičnim potrebam posamezne frontend aplikacije, kar zmanjšuje količino prenesenih podatkov in povečuje učinkovitost komunikacije .

- **Izboljšana varnost:** BFF dodaja dodatni sloj varnosti med frontend in glavnimi backend sistemi, kar omogoča boljšo zaščito pred zlonamernimi napadi in nepooblaščenim dostopom .

- **Ločevanje odgovornosti:** Uporaba BFF vzorca omogoča boljšo organizacijo kode, lažje vzdrževanje in boljšo skalabilnost sistema .

- **Optimizacija zmogljivosti:** BFF omogoča optimizacijo zmogljivosti za različne vrste naprav in uporabniških vmesnikov, kar zagotavlja optimalno uporabniško izkušnjo .

- **Hitrejši razvoj in uvedba funkcionalnosti**: Z ločenimi backendi za vsako frontend aplikacijo je mogoče hitreje uvajati nove funkcionalnosti in posodobitve, kar omogoča agilnejši razvoj in manjše tveganje za napake.

Z uporabo vzorca Backend for Frontend smo pridobili prilagojeno in učinkovito komunikacijo med frontend in backend sistemom.

Temelj komunikacije med uporabniškim vmesnikom in med kompleksnimi zalednimi sistemi je tanka plast API vmesnikov, ki jih backend razvijalci z enostavnostjo iztpostavijo iz obstoječim mikrostoritev ali novo napisanih mikrostoritev.

4.3. Struktura Blazor projekta

V ogrodju .NET je rešitev sestavljena iz večjega števila projektov. Pri tem smo posamezne projekte razdelili na vmesnike do zunanjih storitev, na poslovno logiko, podporne storitve in na predstavitveno plast. Kot dodaten del napram strukturi naših storitev je prav ta predstavitvena plast tista, ki vsebuje Blazor in posledično je za backend razvijalce celotna struktura povsem domača. Za čim hitrejši razvoj izpostavljajo vse naše mikrostoritve svoje SDK-je. Ti so potem vključeni v projekt, ki predstavlja Backend for Frontend za uproabniški vmesnik. Backend for Frontend pa izpostavi SDK, ki ga potem Blazor projekt preko vmesnika tudi uporablja. Tudi sama struktura projekta, ki vsebuje ogrodje Blazor je čimbolj približana backend razvijalcem in je sestavljena iz sledečih delov:

- **Vmesniki do ostalih projektov v rešitvi:** Predstavljajo sloj, preko katerega uporabniški vmesnik komunicira z API klici namenjeni uporabniku in so tesno povezani z uporabo vzorca Backend for Frontend. Tukaj je uporabljen SDK od Backend for Frontend projekta, ki izpostavlja vse API klice, ki jih uporablja spletna rešitev.
- **Notranje storitve uporabniškega vmesnika:** Abstrakcija storitev, ki se uporabljajo na uporabniškem vmesniku, da se izpostavijo storitve za večkratno uporabo. V to spadajo storitve, kot je delo z brskalnikom, delo s lokalno shrambo in piškotki, delo z navigacijo itd. Tukaj smo pridobili predvsem skrivanje določenih kompleksnosti, ki se jih razvijalci potem ne rabijo vedno zavedati.
- **Modeli za pogled:** Podatki, ki jih pridobimo iz API klicev niso nujno vedno povsem prilagojeni za prikaz na uporabniškem vmesniku, zato se takšne podakte preoblikuje v modele za pogled, pri čemer se doda določene specifike samo za uporabniške poglede. V tem koraku se zgodijo še formatiranja datumov, pretvorbe iz internih šifrantov v uporabniku prijazne podatke in dodatno izpopolnjevanje podatkov, ki je potrebno samo na uporabniškem vmesniku.
- **Strani:** Osnova Blazor aplikacije so posamezne strani, ki vključujejo osnovno pot, kjer so dostopne. Strani smo razdelili na čisti uporabniški dizajn in na podporno kodo. V Blazorju je to možno s pristopom tako imenovanega "code behind" načina, kjer posamezno stran razdelimo na dve datoteki. Prva datoteka je sestavljena iz same strani in komponent, medtem ko je druga datoteka delni razred, v katerem je večina C# kode, ki upravlja s stranjo in komponentami.
- **Komponente:** V primeru kompleksnejših ali ponavljajočih se kompoment, smo za voljo izboljšane berljivosti le-te prestavili v poseben del projekta. Take komponente so potem na voljo za večkratno uporabo. Primer tega so denimo potrditveni dialogi, ki so povsod identični.

5 Zaključek

Z uporabo .Net Blazor in ogrodja MudBlazor smo tudi backend razvijalcem, ki v večini primerov niso vešči vseh sodobnih pristopov glede gradnje spletnih strani omogočili izdelavo internih spletnih rešitev. Takšen način dela sicer ne bi priporočali za gradnjo spletnih rešitev, kjer želimo čimbolj dovršen spletni vmesnik, saj uporaba obeh tehnologij skupaj vseeno ni na nivoju ogrodij kot so React, Angular, Vue ali podobni.

Prav tako ima uporaba WebAssembly svoje specifike, ki je povsem drugačno kot pa pri standardnih rešitvah. Naprimer ločen peskovnik med posameznimi zavihki in začetno nalaganje. Potrebno je sicer poudariti, da so bili interni uporabniki nad delovanjem celotne rešitve več kot zadovoljni, še posebej pa jim je všeč hitrost posodobitev, saj je za interne aplikacije vedno veljalo, da je kovačeva kobila vedno bosa in je bilo težko najti čas za posodobitve.

Tudi iz vidika samega izgleda je bila uporaba MudBlazor komponent sprejeta odlično, saj je bil sodelavcem Material Design poznan. Vendar pa smo tudi tukaj zaradi načina razvoja upoštevali omejitve. Odločili smo se, da kompleksnih lastnih komponent ne bomo uporabljali ampak samo tiste ki že obstajo ali pa jih je mogoče

sestaviti iz več osnovnih. Na ta način so se tudi interni uporabniki zavedali, da smo omejeni z oblikovnega vidika in da nebo ni meja, kot je to ponavadi pri našem glavnem produktu, ki je namenjen končnemu uporabniku.

V celoti gledano je tak način razvoja prinesel obilo prednosti za naše interne uporabnike in razbremenitev za frontend razvijalce. Tudi backend razvijalci pa so sprejeli izziv izdelave spletnih rešitev, saj so lahko ostali na tehnologijah, ki so jih že dobro poznali.

Literatura

[1] docs.microsoft.com/en-us/aspnet/core/blazor/, Microsoft Blazor Documentation, Obiskano 24. 7. 2024

[2] mudblazor.com/, MudBlazor - Blazor Component Library, Obiskano 22. 7. 2024

[3] blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html, The Clean Architecture, Obiskano 19. 7. 2024

[4] learn.microsoft.com/en-us/aspnet/core/release-notes/aspnetcore-8.0, What's new in ASP.NET Core 8.0, Obiskano 23. 7. 2024

[5] learn.microsoft.com/en-us/aspnet/core/blazor/components/render-modes?view=aspnetcore-8.0, ASP.NET Core Blazor render modes, Obiskano 22. 7. 2024

[6] codebeam-mudextensions.pages.dev/, MudExstensions Page, Obiskano 22. 7. 2024

[7] www.mudex.org/, Mudex Mudblazor Extensions, Obiskano 22. 7. 2024

[8] learn.microsoft.com/en-us/azure/architecture/patterns/backends-for-frontends, Backends for Frontends pattern, Obiskano 23.7.2024