Miskolci Egyetem
Gépészmérnöki és Informatika Kar
Általános Informatikai Intézeti Tanszék

Szakmai Gyakorlat Beszámoló

Készítette:

Baumel Márton Benedek

Neptun kód: *O09CTQ*

Mérnökinformatikus BSc hallgató

Korszerű Web technológiák szakirány

1, Tartalomjegyzék:

2, Cég bemutatása

Az Evosoft Hungary Kft. több mint 25 évvel ezelőtt kezdte el a működésest egy pár fős cégként, de ma már Magyarország egyik legmeghatározóbb informatikai vállalatává vállt. Az országban Budapesten, Szegeden és Miskolcon is megtalálható a vállalatnak egy irodája és több mint 1400 alkalmazottal rendelkezik jelenleg.

A cég profiljában bele tartozik az ipari automatizálás-technika, a hajtástechnika, az elektromos autók, orvostechnika és az energetikai automatizálás. Ez a széles profil annak köszönhető, hogy a cég a Siemens világvállalat része, amivel lehetőség nyílik a technológia valamennyi szakterületén jelen lennie és folyamatosan az aktuális legfejletebb technológiával dolgozni.

Emellett a cég folyamatosan keresi a kapcsolatot egyetemi hallgatókkal olyan formában, hogy rengeteg rendezvényt biztosítanak, ami szabadon látogatható, sok szakmai programot rendeznek, emellett pedig az egyetemen is vendégelőadóként is találkozhatunk a cég munkatársaival.

2016-óta pedig tehetségtámogató programot is biztosítanak hallgatóknak evoCampus néven, ahol egy félév alatt kell egy szabadon választott projektet megvalósítani egy csapattal olyan formában, hogy a félév végére az prezentálható állapotba kerüljön az elkezdett projekt. Ehhez olyan formában biztosítanak szakmai segítséget, hogy a csapatokhoz rendelnek mentorokat, akik a cég dolgozói és velük együtt halad a fejlesztés, úgy, mint egy éles szakmai környezetben.

3, Feladatok Bemutatása

Még a munka megkezdése előtt küldenem kellett egy önéletrajzot, ahol is szerepelnie kellett annak, hogy milyen technológiák felé szeretnék orientálódni és ezek alapján történt meg a csapatokba való besorolás. Az első kettő napban történet meg az aktuális projektek átbeszélése és kiválasztása, a csapat menedzserével egyeztetve.

Én előzetesen jeleztem, hogy szeretnék webes technológiákkal foglalkozni, ezen belül is a React frontend keretrendszerrel és .NET backend technológiával szerettem volna dolgozni és végül olyan projektbe is osztottak be, ahol ezzel tudtam foglalkozni a szakmai gyakorlat alatt, de külön is lehetőséget biztosítottak arra, hogy dönthessünk arról, hogy specifikusan milyen projektbe is szeretnénk bekerülni a szakmai gyakorlat alatt.

Ezt azért volt fontos átbeszélni a legelején mert, úgy próbáltak minket szakmai gyakorlatosokat beosztani, hogy figyelembe vették a saját személyes céljainkat és olyan projektre osztottak be minket, hogy a legtöbbet tudjuk a 8 hét alatt fejlődni az adott technológiával.

A fő feladat csoportok időrendi sorrendbe:

1. A számítógép és a fejlesztő környezet összeállítása
2. Céges eszközök megismerése
3. Kötelező oktatások elvégzése
4. Szakami segédanyagok elvégzése és a specifikus ismeretek megismerése
5. Projekt megismerése
6. Projekt munkálataiban részt venni

4, Munkanapló

4.1. Első Hét

Kitűzött célok:

- *Cég megismerése*
- *Munkaállomás beüzemelése*
- *Projekthez tartozó technológiák megismerése*

Az első nap az összes szakmai gyakorlaton részt vevőknek megtörtént a céggel való szerződés kötése és a kötelező szerződések aláírása. Utána megkaptuk a belépő kártyánkat azután pedig bemutatták az irodát és elmagyarázták, hogy hol mit is találunk pontosabban. Az eligazítás után a csapat vezetőm vett át és bevitt abba az irodába, ahol is dolgozni fogok és meg is kaptam az első feladatomat, ami az volt, hogy a munkaállomásomat kellett beüzemelnem. A munkámhoz kaptam egy új laptopot, 2 monitort és egyéb perifériákat. A kicsomagolás utána pedig elkezdtem a cég által előírt programokat, frissítéseket és biztonsági eszközöket felépíteni és miután befejeztem ezeknek az eszközök beszerzésest elkezdtem a fejlesztő eszközeim letöltését és telepítésest.

Teljesen ezzel a feladattal 2-3 nap alatt végeztem, mert rengeteg engedélyt kellett megszereznem a géphez és a frissítések is sok időbe teltek.

A héten még a projekthez nem tudtam hozzáférni, ezért a fennmaradó időben elkezdtem tanulni a projekthez tartozó technológiákat annak érdekében, hogy minél gyorsabban betudjak csatlakozni majd a projektbe.

Projekt technológiák, amiket megkellett ismernem:

- TypeScript
- React
- C#/.NET 6.0
- ASP.NET
- Entity Framework

A héten az utolsó feladatomat a projekten dolgozó kolléga adta, ami egy algoritmizáló feladat volt TypeScript-be.

Feladat: Van egy tömböm, amiben JSON elemek vannak (az elemek rendelkeznek egy név és egy szám tulajdonsággal), egy olyan függvényt kell írnom, ami megkapja ezt a tömböt és a paraméter listájában szerepelnie kell egy olyan paraméternek, aminek legalább egy tulajdonsága megegyezik a tömb egyik tulajdonságával és a paraméter alapján a rendezést végre lehet hajtani és a végén kiíratjuk a régi és az új tömb elemeit.

Megoldásom:

objectTypeInterface.ts:

```
export default interface ObjectType{
    name: string,
    nr: number
}
```

objectQueryType.ts:

```
export default interface ObjectQueryType {
    name: boolean,
    nr: boolean
}
```

index.ts:

Az algoritmust leíró logika:

```
  objectList.sort((a, b) =>
    `${query.name && a.name} ${query.nr && a.nr}` <
    `${query.name && b.name} ${query.nr && b.nr}`
      ? -1
      : 1
  );
```

A feladatot befejezése után átnéztük és átbeszéltük, hogy hol lehetne még a programban javítani és még milyen alternatív megoldások vannak.

Ez a feladat azért volt érdekes mert a kiindulási feladat egy rendes projektből származott és az algoritmizáláshoz tartozó témaköröket tudtam jobban megismerni a megoldásával.

4.2. Második Hét

Kitűzött célok:

- *A projekthez tartozó technológiák további tanulása*
- *Projekt megismerése*

A hét első felében a React-os és a TypeScript-es ismereteimet bővítettem a típus kezelés mélyebb megismerésével és a Redux-ot kezdtem el használni a useContext hook helyett.

Sikerült a projektem témáját és a feladat körömet legyeztetnem a projekt architect-el, és végül a projektem témája, ahova beosztottak egy nyilván tartó rendszer lesz a Siemens számára.

Az én feladatköröm a frontend-hez fog főképpen tartozni, de lesznek a backend-hez tartozó feladatok is és ezért meg kell ismernem a programhoz tartozó különböző strukturális logikáknak a felépítésével ezért a projekt architect kiadott előzetesen egy kisebb feladatott annak érdekében, hogy megismerhessem ezeknek a programoknak a felépítési mintáját. A kettő fő tervezési minta, amit a projektnél használnak a CQRS *(Command and Query Responsibility Segregation)* és a Mediator.

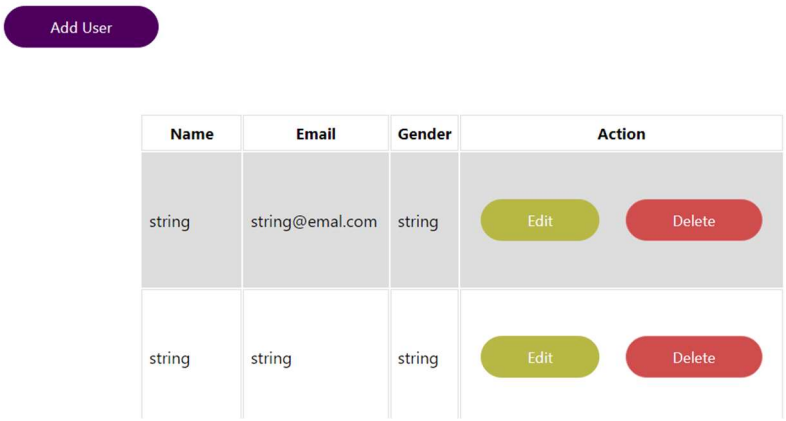
Feladat: Az eddigi ismereteim alapján létre kell hoznom egy olyan webalkalmazást, ami megvalósítja a CRUD *(Create, Read, Update, Delete)* műveleteket, React és C# *(ASP.NET/.NET 6.0)* környezetben. Ez lesz a kiindulás feladat, amit a későbbiekben átfogok fejleszteni a tervezési minták alapján.

Ez a feladat azt a célt szolgálja, hogy minél átláthatóbbá tudja tenni számomra a minták megértését és használatát emellett a projektnél bevezetett kódolási elveket is megtudom ismerni és a rendes projektnél már nem kell ezeket újra megtanulnom, hanem egyből rész tudok már benne venni, mert rendelkezni fogok a kellő ismeretekkel.

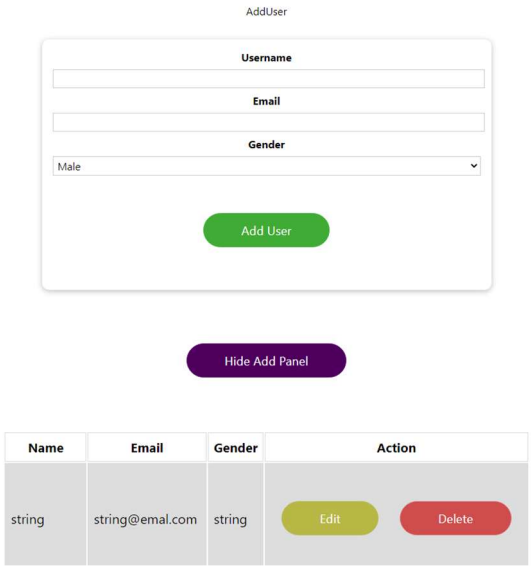
Az alkalmazás témája egy egyszerű nyilvántartó rendszer, amivel különböző emberek nevét, email címét és nemét lehet tárolni. Az oldal megmutatja az adatbázisban szereplő összes elemet egy táblázatban. Lehetőség van az oldalon újabb embert hozzá adni a táblázathoz emellett a meglévőket is lehet szerkeszteni és törölni.

Első lépésben megterveztem az alkalmazásom felületét és beégetett adatokkal leprogramoztam a React frontend oldalt.

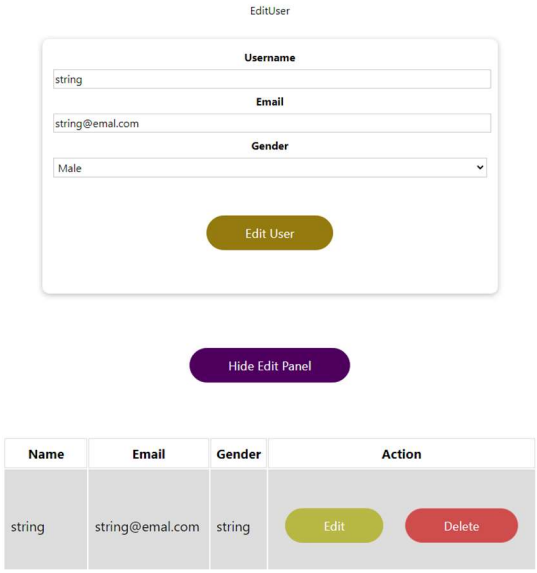
Főoldal:



Új felhasználó felvételére szükséges komponens:



Meglévő felhasználó szerkesztésére szolgáló komponens:



A következő lépésben elkezdtem az alkalmazásom backend-jét megtervezni és leprogramozni.

Létrehoztam a felhasználó modelljét, amiből az Entity Framework ORM segítségével letudtam generálni az adatbázisomat.

UserEntity.cs:

```
public class UserEntity
{
    1 reference | Baumel Márton, 4 days ago | 1 author, 1 change
    public int Id { get; set; }
    6 references | Baumel Márton, 4 days ago | 1 author, 1 change
    public string Username { get; set; } = string.Empty;
    4 references | Baumel Márton, 4 days ago | 1 author, 1 change
    public string Email { get; set; } = string.Empty;
    2 references | Baumel Márton, 4 days ago | 1 author, 1 change
    public string Gender { get; set; } = string.Empty;
}
```

Miután létrehoztam az adatbázist elkezdtem megírni az alkalmazáshoz tartozó controller osztályt, amiben szerepelni fognak a végpontok a frontend számára és itt lesznek megvalósítva a CRUD műveletek.

A következő kódrészletek mind a UserController.cs-ből származnak:

Mivel EntityFramwork-őt használtam ezért a controller osztályban használnom kell a DataContext-et amivel eltudom érni az adatbázisomat:

```
[Route("api/[controller]")]
[ApiController]
1 reference | Baumel Márton, 4 days ago | 1 author, 2 changes
public class UserController : ControllerBase
{
    private readonly DataContext _context;

    0 references | Baumel Márton, 4 days ago | 1 author, 1 change
    public UserController(DataContext context)
    {
        _context = context;
    }
```

A CRUD műveletek megvalósítását a Create paranccsal kezdtem. Ezzel a művelet csoporttal új elemet hozhatok létre:

```
[HttpPost]
0 references | Baumel Márton, 4 days ago | 1 author, 2 changes
public async Task<ActionResult<List<UserEntity>>> AddEntity(UserEntity user)
{
    var regex = new Regex(@"^([\w\.\-]+)@([\w\-]+)((\.(\w){2,3})+)$");
    var emailMatch = regex.Match(user.Email);

    if ((user.Username.Trim().Length > 3 && user.Username.Trim().Length < 15) && emailMatch.Success)
    {
        _context.Users.Add(user);
        await _context.SaveChangesAsync();

        return Ok(await _context.Users.ToListAsync());
    }
    else
    {
        return BadRequest("Bad input data");
    }
}
```

A következő parancs, amit implementáltam az a Read volt. Ezzel a művelet csoporttal módosítás mentes lekérdezéseket lehet létrehozni.

Az összes elem lekérése:

```
[HttpGet]
0 references | Baumel Márton, 4 days ago | 1 author, 1 change
public async Task<ActionResult<List<UserEntity>>> Get()
{
    return Ok(await _context.Users.ToListAsync());
}
```

Csak egy adott Id-val rendelkező elem lekérdezése:

```
[HttpGet("{id}")]
0 references | Baumel Márton, 4 days ago | 1 author, 2 changes
public async Task<ActionResult<UserEntity>> GetEntityById(int id)
{
    var user = await _context.Users.FindAsync(id);

    if (user == null)
    {
        return BadRequest("User not founded");
    }
    else
    {
        return Ok(user);
    }
}
```

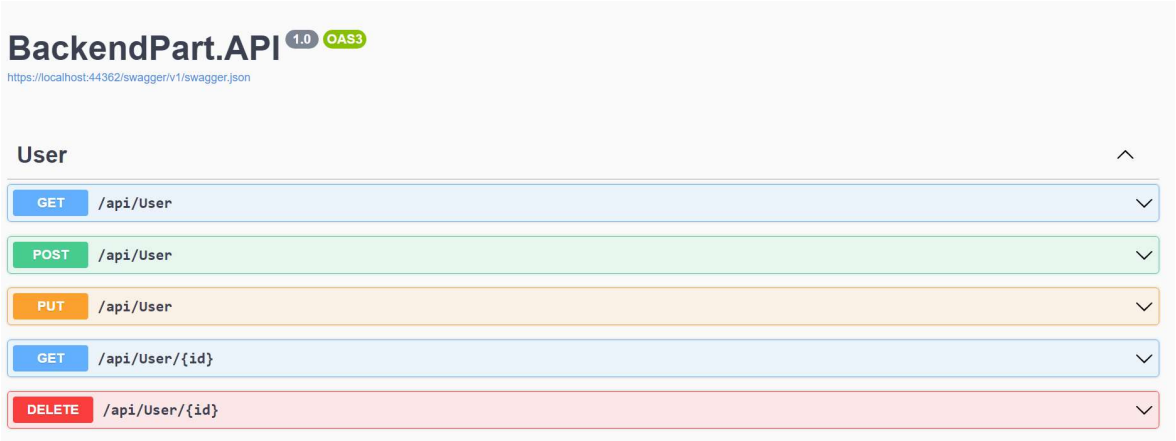
A következő parancs, amit implementáltam az a Delete volt. Ezzel a művelet csoporttal törlést tudunk végre hajtani

```
[HttpDelete("{id}")]
0 references | Baumel Márton, 4 days ago | 1 author, 2 changes
public async Task<ActionResult<List<UserEntity>>> DeleteEntityById(int id)
{
    var user = await _context.Users.FindAsync(id);

    if (user == null)
    {
        return BadRequest("User not founded");
    }
    else
    {
        _context.Users.Remove(user);
        await _context.SaveChangesAsync();

        return Ok(await _context.Users.ToListAsync());
    }
}
```

Az utolsó parancs, amit implementáltam az Update volt. Ezzel a művelet csoporttal a meglévő elemeket tudjuk módosítani.

```
[HttpPut]
0 references | Baumel Márton, 4 days ago | 1 author, 2 changes
public async Task<ActionResult<List<UserEntity>>> UpdateEntity([FromBody] UserEntity userRequest)
{
    var regex = new Regex(@"^([\w\.\-]+)@([\w\-]+)((\.(\w){2,3})+)$");
    var emailMatch = regex.Match(userRequest.Email);

    if ((userRequest.Username.Trim().Length > 3 && userRequest.Username.Trim().Length < 15) && emailMatch.Success)
    {
        var user = await _context.Users.FindAsync(userRequest.Id);

        if (user == null)
        {
            return BadRequest("User not founded");
        }

        user.Username = userRequest.Username;
        user.Email = userRequest.Email;
        user.Gender = userRequest.Gender;
        await _context.SaveChangesAsync();

        return Ok(await _context.Users.ToListAsync());
    }
    else
    {
        return BadRequest("Bad input data");
    }
}
```

Az alkalmazásomat a Swagger nevezetű fejlesztői eszközzel leteszteltem és a végső állapota ilyen lett:



Ezek után annyi feladatom maradt, hogy a frontend-en szereplő beégetett adatokat kicseréljem egy API hívásra és az alkalmazásom elkészült

Felhasználó adatok feltöltése:

```
useEffect(() => {
    fetch("https://localhost:44362/api/User")
      .then((response) => response.json())
      .then((json) => setUserData(json));
  }, [userData]);
```

Felhasználó hozzáadása:

```
const addUserHandler = (user: IUserDataModel) => {
    fetch("https://localhost:44362/api/User", {
      mode: "cors",
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(user),
    })
      .then((res) => {
        if (res.status === 200) {
        } else {
          alert("Something went wrong");
        }
      })
      .catch(function (error) {
        console.log(error);
      });
  };
```

Felhasználó törlése:

```
const deleteUserHandler = (id: number) => {
   fetch(`https://localhost:44362/api/User/${id}`, {
     mode: "cors",
     method: "DELETE",
     headers: { "Content-Type": "application/json" },
   })
     .then((res) => {
       if (res.status === 200) {
       } else {
         alert("Something went wrong");
       }
     })
     .catch(function (error) {
       console.log(error);
     });
 };
```

Felhasználó szerkesztése:

```
const onEditUserHandler = (user: IUserDataModel) => {
   fetch("https://localhost:44362/api/User", {
     mode: "cors",
     method: "PUT",
     headers: { "Content-Type": "application/json" },
     body: JSON.stringify(user),
   })
     .then((res) => {
       if (res.status === 200) {
       } else {
         alert("Something went wrong");
       }
     })
     .catch(function (error) {
       console.log(error);
     });
 };
```

Ezzel a lépéssel pedig az alkalmazásom első változata elkészült. Ezután a projekt architect-el átbeszéltük a megoldásomat és kielemezte azokat a részeket, amiket javítanom kellett az első változatban.

A hibáim az algoritmizáláshoz és a kód minőséghez volt köthető, de ezeket a hibákat gyorsan tudtam javítani és ezek után engedélyt kaptam arra, hogy elkezdjem a második változatott lefejleszteni, amiben megvalósítom a CQRS és a Mediator tervezési mintákat.

A héten még részt vettem egy kolléga által tartott előadáson, ami az agilis fejlesztés alapelveiről és a Scrum keretrendszerban való fejlesztésről szólt. Ez azért volt hasznos mivel a projekten, amin dolgozni fogok ott is Scrum keretrendszer alapján történik a fejlesztés ezért jó volt megismerkedni az alapvető mintákkal és a szabályokkal.

1. ábra

4.3. Harmadik Hét

Kitűzött célok:

- Kötelező betanulási feladtok elvégzése

Mielőtt még a projekt feladatokba bele tudnék csatlakozni azelőtt a cég által előírt kötelező oktatásokon kell részt vennem. Ezek az oktatások online történnek ezért a saját tempómba tudok rajtuk végig haladni, de mivel sok kurzust kellett teljesítenem ezért a hetemet ez kitöltötte.

4.4. Negyedik Hét

Kitűzött célok:

- Kötelező betanulási feladatok befejezése
- CQRS felbontás

A héten sikerült a kötelező oktatásokat befejeznem ezért elkezdtem a becsatlakozás folyamatát projektbe.

A betanulási feladatomba sikerült átalakítania Backend részét, hogy kövesse a CQRS tervezési mintát. A Command and Query Responsibility Segregation minta azt a célt szolgálja, hogy eltudjuk különíteni az olvasó és a változtató parancsokat. Ez azért hasznos mert az alkalmazás teljesítménye így szabadon alakítható, jobban meglehet oldani a kód biztonságot és jobban átlátható kódot tudunk írni így.

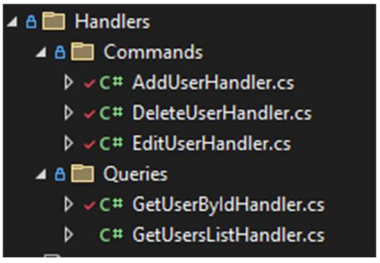
Kezdésképpen létrehoztam kettő új projektet:

- BackendPartUpdated.API: ez a rész fogja biztosítani az alkalmazásom végpontjait

- BackendPartUpdated.DataManagment: ez a rész fogja biztosítani a Backend oldal kommunikációját a kérések és az adatbázis között

Az előző projektben szereplő adatbáziskezelő műveleteket átmásoltam a DataManagment részbe és létrehoztam ugyan azt a User entitást, amit a régebbi változatban használtam.
A CQRS mintát követve az előző változatban lévő logikai műveleteket átszerveztem az új projektbe.

Jól látszik, hogy a lekérdező parancsok a Handlers mappán belül a Queries mappába kerültek és a jobb átláthatóság kedvéért nem egy osztályba raktam bele az összes műveletet, ami a felhasználók adatával térne vissza, hanem a feladatuk alapján szeparáltam őket.



Míg a változtató parancsok a Commands mappába kerültek ugyan úgy elkülönítve egymástól.

4.5. Ötödik Hét

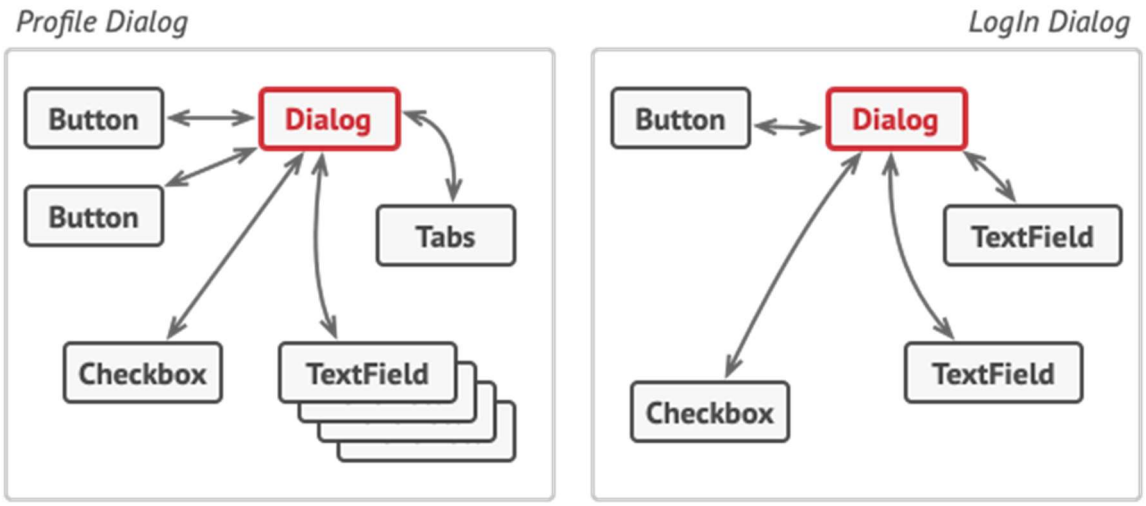
Kitűzött célok:

- Startnap-on való részvétel
- Mediator alkalmazása
- Feladat értékelése

Annak érdekében, hogy a cég szakami ágazatait jobban áttudják látni az újonnan érkezőket ezért a cég időközönként szervez egy Startnap-ot, ahol a cég újoncainak az ágazati vezetők bemutathatják, hogy az ő részlegük mivel is foglalkozik pontosabban ezzel pedig a céges beilleszkedés folyamata is gyorsabb lehet, emellett a cég munka kultúrájába is bevezetés kaptunk. Ez a rendezvény Budapesti irodában volt, ahol személyesen kellett részt venni.

A programozási feladatom utolsó lépéseként pedig a korábbi munkámból alkalmazott üzleti logikát átültettem a Meditor tervezési mintára.

A Mediator tervezési mintának az a célja, hogy a különböző hívásokat egy *"elosztó"* osztály kezelje le annak érdekében, hogy a kód objektumai között átláthatóbbak és tisztábbak legyenek a folyamatok. Másrészről ez direkt kommunikációt biztosít a komponensek között.



2. ábra

Alkalmazása:

A kérések 2 részből fognak állni:

1. A kérés meghívására szolgáló rész

```
3 references | 0 changes | 0 authors, 0 changes
public class GetUsersListQuery : IRequest<Result<List<UserEntityDto>>>
{
}
```

2. A kérés lekezelése

```
1 reference | 0 changes | 0 authors, 0 changes
public class GetUsersListHandler : IRequestHandler<GetUsersListQuery, Result<List<UserEntityDto>>>
{
    private readonly IDataRepository _dataRepository;

    0 references | 0 changes | 0 authors, 0 changes
    public GetUsersListHandler(IDataRepository dataRepository)
    {
        _dataRepository = dataRepository;
    }

    0 references | 0 changes | 0 authors, 0 changes
    public async Task<Result<List<UserEntityDto>>> Handle(GetUsersListQuery request, CancellationToken cancellationToken)
    {
        var userList = await Task.FromResult(_dataRepository.GetUsers());

        //LINQ
        var result = new Result<List<UserEntityDto>>(userList.Select(x => new UserEntityDto(x.Id, x.Username, x.Email, x.Gender)).ToList());

        return result;
    }
}
```

Viszont, ha specifikus adattal szeretnénk ellátni a kérést akkor pedig a record-ként is tudjuk a kérés meghívására szolgáló részt definiálni

```
4 references | 0 changes | 0 authors, 0 changes
public record AddUserCommand(string Username, string Email, string Gender) : IRequest<Result<UserEntityDto>>;
```

A legvégső feladat pedig a FluentValdiation alkalmazása volt. Amikor valamilyen adatot adok át a kérésnek, ezzel a külső csomaggal az objektum létre jövettele előtt megtudjuk vizsgálni, hogy a kért paraméterek helyesek a saját feltételeinknek alapján és csak akkor fogja létrehozni az objektumot, ha a paraméterek helyesen voltak megadva ezzel pedig a validációhoz szükséges kód sokkal rövidebb lehet. Ezek alapján az összes logikai műveletet átalakítottam erre a formára.

```
2 references | Baumel Márton, 15 days ago | 1 author, 1 change
public class AddUserValidator : AbstractValidator<AddUserCommand>
{
    1 reference | Baumel Márton, 15 days ago | 1 author, 1 change
    public AddUserValidator()
    {
        RuleFor(t => t.Username).NotEmpty().WithMessage("Username is empty");
        RuleFor(t => t.Username.Trim()).MinimumLength(3).WithMessage("Username is too short");
        RuleFor(t => t.Username.Trim()).MaximumLength(15).WithMessage("Username is too long");
        RuleFor(t => t.Email.Trim()).EmailAddress().Matches(@"^([\w\.\-]+)@([\w\-]+)((\.(\w){2,3})+)$").WithMessage("Not good email format");
    }
}
```

Az applikációm frontend résszén nem kellett alakítanom mivel ugyan azokat a műveleteket használtam itt is mint az előző backend verzióban. Ebből is látszik, hogy az applikációm frontend és backend oldala képes egymástól függetlenül is működni, azaz, ha valamilyen nagyobb frissítést szeretnék a későbbiekben végre hajtani, mint az egyik nyelv teljes lecserélését a frontend vagy a backend oldalon akkor nem kell az applikációm mindkét részét átírnom.

Végezettül projekt architect-el átbeszéltük utoljára ezt a feladatot és elfogadta a megoldásomat ezután pedig elkezdhettem a becsatlakozási folyamatot a tényleges projektbe.

4.6. Hatodik Hét

Kitűzött célok:

- Azure DevOps engedélyek megszerzése
- TFS megismerése
- Projekt feladatok megismerése

A hét első felében a projekt verzió követő rendszeréhez (ez a Azure DevOps nevű alkalmazás) kellett az engedélyeket megszereznem és a felület kezelését megkellett tanulnom. Ezután elkezdtem megismerkedni a projekthez tartozó munkafolyamattal, ami a Git-es ismeretimhez képest újdonság volt emellett a projekthez írt coding guideline-ot tanulmányoztam.

Ezek utána a projektet futtathatóvá tettem a munkaállomásomon és elkezdtem megismerni a működését, emellett pedig a projekthez tartozó feladatokat kezdtem el jobban felfedezni.

4.7. Hetedik Hét

Kitűzött célok:

- Frontend feladat elvégzése
- Csapattal való megismerkedés
- Scrum eseményeken való részvétel

A hét elején megkaptam a projektben az első feladatomat, ami a frontenddel kapcsolatos volt.

Feladat: az oldalon szerepelnek olyan felhasználó interakcióra váró elemek, amikhez lehet újabb sorokat létrehozni és itt kell a felhasználó felületet úgy átalakítani, hogy ezeket a sorokat jobban eltudjam választani egy vastagított fekete vonallal.

A megvalósítás alatt sikerült a projektet jobban megismernem és sikerült a SCCS (Syntactically Awesome Style Sheet) segítségivel és a React-on belüli inline style beállítással megoldanom a feladatot.

Emellett a héten elkezdtem részt venni a napi Stand Up-on. Ilyenkor a fejlesztők össze ülnek és a Scrum szabályai szerint elmondják, hogy:

1, A tegnapi nap mivel foglalkoztak

2, A mai nap mivel fognak foglalkozni

3, Van-e valamilyen olyan tényező, ami akadályozza ezt

Ez a megbeszélés a nap elején történik meg és a célja az, hogy a csapaton belül mindenki láthassa azt, hogy ki mivel foglalkozni az adott nap vagy ha valamilyen probléma lépet volna fel akkor azt itt meg lehet beszélni, hogy minél hamarabb ki lehessen azt javítani.

Ezen a héten még részt vettem egy Refinement megbeszélésen, aminek a célja az volt, hogy a PO-val (Product Owner) egyeztetve átbeszéljük a feladat listára újonnan felkerülő feladatokat. Ilyenkor is, ha valami nem teljesen világos a megoldandó feladattal kapcsolattal akkor a PO feladata az, hogy utánajárjon, hogy a megrendelő, hogyan is gondolta pontosan. Ezzel is a fejlesztés alatt fellépő hibákat lehet csökkenteni.

4.8. Nyolcadik Hét

Kitűzött célok:

- Visszajelzések alkalmazása
- Review-n való részvétel

A kész kódomat értékelték a munka társaim és az észrevételeik alapján javítottam és végzetül a kódom bele került a tényleges kódbázisba.

Ezen a héten részt vettem még egy Review eseményen, aminek az a célja, hogy a projekten lévő munkatársak betudják mutatni, hogy az aktuális fejlesztési ciklusban milyen fejlesztési munkákat hajtottak végre. Itt én is bemutattam milyen feladatokat hajtottam végre, a feladatokat, hogyan valósítottam meg és a megvalósítás folyamán milyen nehézségekbe ütköztem. A saját feladataim bemutatása után pedig meghallgattam, hogy a többiek milyen fejlesztéseket hajtottak végre.

Miután mindenki elmondta, hogy mivel haladt utána pedig a következő hetek eseményeit beszéltük át és elindult a következő fejlesztési ciklus.

5. A szakmai gyakorlat alatt szerzett ismeretek bemutatása

Az elmúlt 8 hét alatt a rengetget sikerült fejlödnőm szakmailag és sikerült a szakmához tartozó nem programozói képességeket is jobban megismernem.

A programozási ismereteim a frontend és backend témakörben is bővültek olyan formában, hogy látthattam, hogy egy valódi projektben, hogyan is működik a C# ASP.NET, mire kell oda figyelnem a kód tisztasága szempontjából és hogyan lehet jól átgondolt kódot írni úgy, hogy a jövőben is használható lehessen mindenki számára.

A React ismeretim pedig olyan formában javultak, hogy sokkal jobban megértem azt, hogy miért is fontos egy jól át gondolt felhasználói felület és ezt, hogyan is lehet logikusan komponensekre bontani, emellett pedig, hogy lehet CSS helyett az SCSS-t használni.

Emellett megismertem azt, hogy hogyan is működik egy nagy multinacionális informatikai cég. Megtanultam azokat a szokásokat, amiket követni kell annak érdekében, hogy mindenki számára könnyedén és gyorsan mehessen a munkája. Ezeken felül pedig sikerült új kapcsolatokat építenem, amik a karrierem szempontjából felbecsülhetetlen lehet.

Viszont a leghasznosabb új ismeretnek azt tartom, hogy sikerült bele látnom egy programozó csapat működésében, megértettem a különböző folyamatok miértjét, hasznosságát és összeségében végre láttom azt a folyamatot, hogy hogyan lesz egy vevői megrendelősből egy működő képes alkalmazás, amit akár több százan is használhatnak függetlenül attól, hogy a világ melyik pontján is élnek.

6, Források megjelölése

1.ábra:

https://www.schaffrath.de/fileadmin/user_upload/magazin/artikel/Scrum_Cheat_Sheet.jpg

2.ábra:

https://refactoring.guru/design-patterns/mediator