SZAKDOLGOZAT/  
diplomadolgozat

Kazal Soma

2022

Pannon Egyetem

Mechatronikai és Méréstechnikai Kutatócsoport

Mechatronikai mérnöki alapszak

SZAKDOLGOZAT/  
diplomadolgozat

Hajó-szárazföld kommunikáció, telemetria, és megjelenítő rendszer tervezése és megvalósítása elektromos versenyhajóhoz.

Kazal Soma

Témavezető: Dr. Medvegy Tibor

Konzulens: Dr. Medvegy Tibor

2022

A hallgató feladata egy elektromos versenyhajó fedélzeti központi egysége és egy szárazföldi szerver közötti adatkapcsolat megvalósítása, valamint a hajó által küldött telemetriás adatok megjelenítése a szárazföldi számítógépeken és okostelefonokon.

Nyilatkozat

Alulírott Kazal Soma diplomázó hallgató, kijelentem, hogy a szakdolgozatot a Pannon Egyetem Mechatronikai és Méréstechnikai Kutatócsoportnál készítettem a Mechatronikai mérnöki alapszak (B. Sc. in Mechatronics) diploma megszerzése érdekében.

Kijelentem, hogy a szakdolgozatban/diplomadolgozatban foglaltak saját munkám eredményei, és csak a megadott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy a szakdolgozatban/diplomadolgozatban foglalt eredményeket a Pannon Egyetem, valamint a feladatot kiíró szervezeti egység saját céljaira szabadon felhasználhatja.

Veszprém, 2022. május 14.

hallgató aláírása

Alulírott Dr. Medvegy Tibor témavezető kijelentem, hogy a szakdolgozatot/diplomadolgozatot Kazal Soma a Pannon Egyetem Mechatronikai és Méréstechnikai Kutatócsoportnál készítette a Mechatronikai mérnöki alapszak (B. Sc. in Mechatronics) diploma megszerzése érdekében.

Kijelentem, hogy a szakdolgozat/diplomadolgozat védésre bocsátását engedélyezem.

Veszprém, 2022. május 14.

témavezető aláírása

Tartalmi összefoglaló

A feladatom a Pannon Rája Solar Boat csapat versenyhajójában lévő fedélzeti központi egység és egy szárazföldi szerver közötti adatkapcsolat megvalósítása, valamint a hajó által küldött telemetriás adatok megjelenítése a szárazföldi számítógépeken és okostelefonokon. Ennek megvalósítására a szárazföldi egységen létrehozok egy web szervert, mely egy erre a célra készült weboldalt fog futtatni. A szárazföldi egység és a hajó belső rendszere Wifi kapcsolattal fog kommunikálni. Az elkészült weboldalon különálló menüpontokban lehet a hajó telemetriáját csoportokra bontva megtekinteni. Például: Energetika menüpont alatt lehet megtekinteni az akkumulátor töltöttségét, a napelemek állapotát stb.

**Kulcsszavak:** adatkapcsolat, telemetria, kommunikáció, weboldal, szerver

Abstract

My task is to design the dataflow between the on-board unit inside the Pannon Ray Solar Boat team’s racing boat and the costal unit, as well as to display the telemetric data sent by the boat’s unit on smartphones and PCs. To achieve this, I will create a web server on the costal unit, which will run a website designed for this porpuse. The boat’s on-board system and the costal unit will communicate through Wifi connection. On the finished webside, the recieved data can be viewed broken down into different categories in the menu. For example: Under the „Energetics” option one can view the charge of the batteries and the condition of the solar panels etc.

**Kulcsszavak:** dataflow, telemetrics, communication, website, server

Tartalomjegyzék

[1 Bevezetés 8](#_Toc121824667)

[Technológia megvalósítás 9](#_Toc121824668)

[1.1 Szenzorok 9](#_Toc121824669)

[1.1.1 Szenzorok típusai 9](#_Toc121824670)

[1.1.2 Gyűjtött adatok 9](#_Toc121824671)

[1.1.3 Kommunikáció a szenzorokkal 10](#_Toc121824672)

[1.2 Adatbázis 10](#_Toc121824673)

[1.2.1 Használt technológiák 11](#_Toc121824674)

[1.2.2 Tárolt adatok 11](#_Toc121824675)

[1.3 Backend 12](#_Toc121824676)

[1.3.1 Szerepe a projektben 12](#_Toc121824677)

[1.3.2 C# 12](#_Toc121824678)

[1.3.3 Entity Framework 13](#_Toc121824679)

[1.3.4 .NET Core 14](#_Toc121824680)

[1.3.5 .NET Framework 14](#_Toc121824681)

[1.3.6 Controllerek 14](#_Toc121824682)

[1.3.7 Modellek 17](#_Toc121824683)

[1.3.8 Service-ek 17](#_Toc121824684)

[1.4 Frontend 18](#_Toc121824685)

[1.4.1 Szerepe a projektben 19](#_Toc121824686)

[1.4.2 HTML 19](#_Toc121824687)

[1.4.3 Javascript 19](#_Toc121824688)

[1.4.4 CSS 19](#_Toc121824689)

[1.4.5 React 19](#_Toc121824690)

[1.4.6 Axios 19](#_Toc121824691)

[2 A hajó és a szárazföldi egység kommunikációja 20](#_Toc121824692)

[2.1 Wifi 20](#_Toc121824693)

[2.2 Kommunikáció 20](#_Toc121824694)

[3 Kliensprogram megvalósítása 20](#_Toc121824695)

[3.1 Valami 20](#_Toc121824696)

[3.2 Valami2 20](#_Toc121824697)

[Irodalomjegyzék 21](#_Toc121824698)

[Mellékletek 21](#_Toc121824699)

[Köszönetnyilvánítás 21](#_Toc121824700)

# Bevezetés

A mai világban egyre nagyobb problémává válik a klímaváltozás és a fosszilis energiahordozókból történő energiakinyerés által keletkezett széndioxid, és az ennek következtében kialakuló klímaváltozás. Ezen probléma miatt manapság egyre többen a szénhidrogénekkel működő járművek helyett az elektromossággal működő járművek felé fordulnak. Viszont a legtöbben ezt a faja közlekedést is megkérdőjelezik, mivel az igaz, hogy így közvetlenül a járművünk nem állít elő üvegház hatású gázokat, viszont az áramot, amivel járművünket feltöltjük használat előtt, nagy valószínűséggel még mindig olyan erőműben állították elő, melynek széndioxid kibocsátása rendkívül magas.

Sokan - köztük én is - azt látják a legfentarthatóbb megoldásnak, ha nem csupán a járműveink működnek elektromossággal, hanem az ezeket működtető elektromosságot is környezetbarát módon állítjuk elő. Ilyen alternatív megoldások lehetnek például: szélerőmű, napelem, vízturbina, vagy akár a geotermikus erőművek.

Talán az előbb leírtak miatt tetszett meg ennyire a Solar Boat Challenge névre hallgató nemzetközi versenysorozat, melyhez kötődik szakdolgozatom témája is. Sőt a fentiekkel ellentétben ebben a versenyben, nem, hogy egy környezetbarát módon működő erőmű látja el árammal az általunk épített elektromos hajót, hanem ez a verseny olyan hajók építéséről szól, melyek teteje napelemekkel van beborítva, így a közlekedéshez szükséges energiát maga termeli meg, így megspórolhatjuk az energia szállítása során keletkező veszteséget is.

A projekten belül az én feladatom, és egyben a szakdolgozatom témája is, a hajó és a szárazföldi központként funkcionáló számítógép közötti kommunikáció megvalósítása. A hajó és a szárazföldi modul Wifi kapcsolaton keresztül fognak kommunikálni. Ez a gyakorlatban azt jelenti, hogy fel kell dolgoznom a hajó belső számítógépe és a benne található szenzorok által küldött adatokat, majd létre kell hoznom a szárazföldi számítógépen egy web szervert, mely ezeket az adatokat egy adatbázisban tárolja, és egy weboldalon a kívánt formában megjeleníti.

# Technológia megvalósítás

Ebben a fejezeben külön kitérek a feladat megvalósításához felhasznált technológiákra.

.

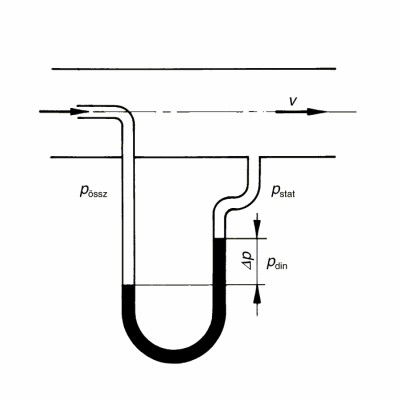
## Szenzorok

A szenzorok definíciójuk szerint olyan eszközök, melyek egy fizikai mennyiséget - mint a hőmérséklet, a távolság, vagy a nyomás - a vezérlés- és szabályozástechnikában felhasználható jellé alakítják át. Ilyen lehet egy elektromos vagy pneumatikus jel. Ebben a fejezetben a hajóban használt szenzorokkal fogok foglalkozni.

### Szenzorok típusai

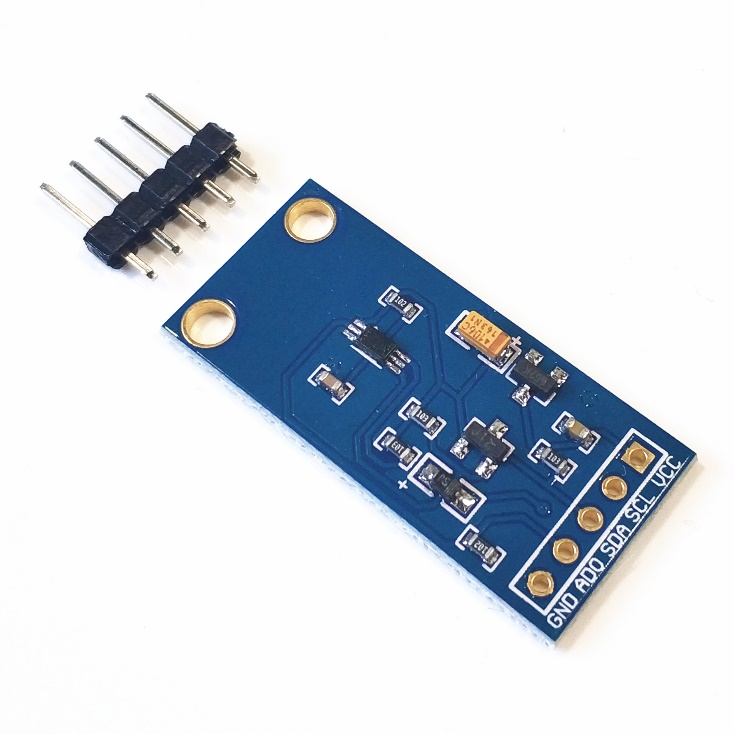
A szenzorokat többféle módon csoportosíthatjuk. Először is csoportosíthatjuk őket a mért változó típusa szerint: Eszerint egy szenzor lehet bináris (pl.: nyomáskapcsoló, vagy kapcsoló-hőmérő), és lehet analóg (pl.: erő- és nyomatékmérő, vagy hőmérsékletmérő szenzor).

Az első szenzorunk egy Pitot-cső. A Pitot-cső egy áramvonalas homlokfalú cső melyet az áramlással szembe fordítva helyezünk el. Ennek a csőnek a belső furatában torlónyomás keletkezik az áramlás hatására a, miközben a furat másik végén egy nyomásváltozás mérésére alkalmas eszköz található. Röviden összefoglalva ez egy műszer, ami áramlások sebességének mérésére alkalmazható. Jelen esetben a hajó vízhez viszonyított sebességének a meghatározására használjuk.



1.1. ábra: Pitot-cső

Második szenzorunk egy BH1750FVI egy digitális környezeti fényérzékelő IC busz interfészekhez. Ez az IC alkalmasa környezeti fény adatok megszerzésére, és általában mobiltelefonok és laptopok LCD és billentyűzet háttérvilágításának beállításához használják. Nagy felbontás mellett széles fénytartomány észlelhető vele. A hajó számára a napelemek működésének megfigyelése miatt fontos.



1.2. ábra: GY-30-BH1750FVI Fényérzékelő

A következő szenzor az SCT013 nem invazív áramváltók érzékelő, amely a vezetőt keresztező áram intenzitását méri anélkül, hogy a vezetéket meg kellene vágni vagy módosítani. Ezt az érzékelőt egy Arduino-val a terhelés intenzitásának vagy teljesítményének mérésére használjuk.

A mérés elektromágneses indukcióval történik. Ezek az érzékelők egy bilincshez hasonlóan osztott maggal rendelkeznek, amely lehetővé teszi a felhasználó számára, hogy bekapcsolja az elektromos berendezéseket anélkül, hogy a vezetéket el kellene vágnia.



1.3. ábra: YHDC SCT-013-100-AX Árammérő

Az SCT013 sorozatban vannak olyan szenzorok, amelyek áram- vagy feszültségkimenetként biztosítják a mérést. Mi ezt a szenzort árammérésre használjuk, azon belül is a motoráram mérésére.

Valamint használunk még a hajótestben egy P31-FLOATSW típusú úszókapcsolót is, a hajótestbe kerülő nedvesség detektálására. Az úszókapcsolók egyfajta szintérzékelő eszközök, amelyek egy tartályban lévő folyadék szintjének érzékelésére szolgálnak. Könnyen használható szintszabályozó eszköz; ráadásul nem is bonyolult áramkör és nem is ad áramot. A szenzor egy mechanikus kapcsolóval rendelkezik, mint az átlagos kis méretű kapcsolók.



1.4. ábra: P31-FLOATSW Úszókapcsoló

Amint ez a kapcsoló jelet ad, bekapcsol a hajótestben a szivattyú.

### Gyűjtött adatok

A Pitot-cső a hajótest vízhez viszonyított sebességét méri. Ez a hajótest vízből történő kiemelkedése miatt fontos. Az ettől a szenzortól kapott értékeket sebesség értékekként tároljuk.

A GY-30-BH1750FVI Fényérzékelő fényáramot mér. Az ettől a szenzortól kapott értékeket fényáramként lumen-ben tároljuk.

YHDC SCT-013-100-AX Árammérőkkel több helyen is mérünk áramerősséget a hajótesten belül. A motoráram az egyetlen mennyiség, amit ilyen típusú szenzorral mérünk és átadjuk a szárazföldi egységnek. Itt egy áramerősség értéket tárolunk.

Valamint átadjuk még a szárazföldi egységnek az akkumulátor áram és feszültség értékét is.

### Kommunikáció a szenzorokkal

A hajón belül egy CAN-protokoll névre hallgató, járművek belsejében alkalmazott kommunikációs protokollt használunk. Az ezzel továbbított adatokat egy rádió adó-vevő készülékkel továbbítjuk a szárazföldre, ahol szintén egy rádió adó-vevő készülékkel kapjuk meg ezeket. Fontos, hogy a hajó és a szárazföldi egység közötti kommunikáció egy duplex kommunikáció, mivel nem csak a hajó küld adatokat a szárazföldi egységnek, hanem fordítva is, azaz a szárazföldi egység is küld adatokat a hajónak, egy utasítást, mely az akkumulátor vagy a motor hűtését hivatott átkapcsolni, azaz ha ki van kapcsolva, bekapcsolni, ha be van kapcsolva, akkor kikapcsolni.

## Adatbázis

Adatbázis alatt köznapi értelemben rendezett, és meghatározott szisztéma szerint tárolt adatok összességét értjük, melyek nem feltétlenül számítógépen kerülnek tárolásra. Természetesen ebben az esetben számítógépes adatbázist fogok használni, mely definíció szerint azonos minőségű és többnyire strukturált adatok összessége, amelyet egy azok tárolására, lekérdezésére és szerkesztésére alkalmas szoftvereszköz kezel. Ebben a fejezetben azt fogom taglalni, hogy a hajóból érkezett adatokat milyen módon tárolom el, hogy később alkalmasak legyenek elemzésre.

Diagram

Description automatically generated

1.5. ábra: A rendszer adatfolyamának ábrázolása.

### Használt technológiák

Az adatbázis létrehozásához és üzemeltetéséhez szükséges technológiákról fogok beszélni. Ilyen az adatbázis szerver és a szkriptnyelv és elemei.

### Tárolt adatok

A szenzorok által továbbított adatokat adatbázisba és táblákba mentjük, és a mérés időpontja szerint rendezzük, így az adatok később is elemezhetővé és értékelhetővé válik.

Jelen projectben MSSQL adatbázist használtam.

Az adatbázis létrehozása a következő scripttel történt:

IF DB\_ID('SolarBoat') is null

Begin

Create database SolarBoat;

End

Go

Az adatbázis egyetlen táblát (Data tábla) tartalmaz, mely időrendben tartalmazza a hajótól (illetve a tesztkörnyezettől) érkező valamennyi adatot.

A tábla leírása:

CREATE TABLE [dbo].[ApiJob](

[JobId] [int] IDENTITY(1,1) NOT NULL,

[SurveyId] [int],

[ClientId] [int],

[LocationId] [int],

[ShopperId] [int],

[JobDate] [DATE],

[DueDate] [DATE],

[ReportDate] [DATE],

[SubmitDate] [DATE],

[JobStatus] [int],

[WaveId] [int],

[ProcessedScore] [int],

[SchedulerManagerId] [int],

[ShopperGrade] [int],

[Changed] [DATETIME],

[ShopViewUrl] [Varchar](500),

[NumberOfResponsesInApollo] int null

CONSTRAINT [PK\_ApiJob] PRIMARY KEY CLUSTERED

(

[JobId] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, FILLFACTOR = 85) ON [PRIMARY]

) ON [PRIMARY]

## Backend

A backend a programoknak, weboldalaknak a hátsó, a felhasználó elől rejtett, a tényleges számításokat végző része. Feladata a projektben a frontend, az adatbázis és a hajó felől érkező adatok feldolgozása, és az eredményeknek a frontend illetve az adatbázis felé történő visszajuttatása, valamint utasítások küldése a hajó felé.

A mi esetünkben a backend egy önálló console alkalmazás, mely nem igényel a futtatáshoz külső eszközt (IIS – Internet Information Services Manager), így könnyebben konfigurálható, ráadásul a program működése is egyszerűen nyomon követhető a console ablakban megjelenített üzeneteken keresztül.

### Szerepe a projektben

Az elkészült projektben a backend fogja biztosítani a kommunikációt a frontend (tehát a felhasználói felület), az adatbázis és a hajó között. Az általam használt backend „Database First” technológiával készült, mely egy alternatívát kínál a „Code First” megközelítés helyett. Míg a “Code First” megközelítés esetében a backend hozza létre saját maga számára az adatbázist induláskor, a „Database First” technológiával működő backend készítése idején egy úgynevezett “scaffolding” folyamat során térképezzük fel a már létező adatbázist. Ez a folyamat elkészíti a backend és az adatbázis közötti kommunikációhoz szükséges osztályokat és kódrészleteket.

### C#

A C# egy modern objektumorientált nyelv, kényelmes és gyors lehetőséget biztosítva ahhoz, hogy .NET keretrendszer alá alkalmazásokat készítsünk, legyen az akár számolás, akár kommunikációs alkalmazás.

### Entity Framework

Az Entity Framework (magyarul Entitás-keretrendszer) egy objektum-relációs leképező keretrendszer a .NET keretrendszerhez. Gyakorlatilag az adatbázis leképezését és kezelését biztosítja a backendben.

Az adatbázissal való kommunikáció a Microsoft.EntityFarmeworkCore.DbContext osztályon alapul. A mi projectünk esetében a SolarBoatContext osztály valósítja meg a működést. Ebben az osztályban írjuk le az adatbázist:

public partial class SolarBoatContext : DbContext

{

public SolarBoatContext()

{

}

public SolarBoatContext( DbContextOptions<SolarBoatContext> options) : base(options)

{

}

public virtual DbSet<Data> Data { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

if (!optionsBuilder.IsConfigured)

{

string constr = ConfigurationManager.AppSettings["connectionstring"];

optionsBuilder.UseSqlServer(constr);

}

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Data>(entity =>

{

entity.Property(e => e.Idopont).HasColumnType("datetime");

});

OnModelCreatingPartial(modelBuilder);

}

partial void OnModelCreatingPartial(ModelBuilder modelBuilder);

}

### .NET Core

A .NET Core a .NET egy implementációja. A .NET Core egyik jelentős újítása a Frameworkhöz képes a platformfüggetlensége. A .NET Framework, illetve a vele készült alkalmazások korábban csak Windows alatt voltak elérhetők, ezzel szemben a .NET Core-os alkalmazások az eddig megszokott Windows-os környezet mellett elérhetők lehetnek macOS-en és különböző Linux-disztribúción is.

A projectben a hajót szimuláló tesztkörnyezet készült .Net Core technológiával.

### .NET Framework

Ide is kell valami. A backend .Net Framework 4.7.2 alkalmazásával készült.

### Controllerek

A controllerek a backend azon részei melyek közvetlenül reagálnak az alkalmazásban található HTTP lekérdezésekre.

A controllerekben action metódusok találhatók. Minden egyes action metódus egy végpontot képez az alkalmazásban, azaz ezek fogadják az egyedi url-lel és típussal rendelkező webrequesteket, azaz külső klienstől - jelen esetben a frontendtől vagy a hajótól - érkező kéréseket.

A mi backendünkben egy controller található, a ValuesController. Az egyes végpontok a REST API követelményeinek megfelelően lettek kialakítva, azaz az url minden esetben azonos (http://localhost:8080/api/values), a metódusok csak a hívás típusában (GET, POST, PUT, DELETE) térnek el, azaz a backend a hívás típusától függően választja ki, melyik actionnek adja át a vezérlést.

public class ValuesController : ApiController

{

DataService dataService;

private static HttpClient client = new HttpClient()

{

BaseAddress = new Uri("http://localhost:8081/"),

};

public ValuesController()

{

dataService= new DataService();

}

// GET api/values

public async Task<HttpResponseMessage> Get()

{

Console.WriteLine(DateTime.Now.ToString() + " - Incoming GET request");

List<Data> result = new List<Data>();

try

{

result = await dataService.GetAllAsync();

}

catch (Exception)

{

}

return Request.CreateResponse(HttpStatusCode.OK, result, Configuration.Formatters.JsonFormatter);

}

// GET api/values/5

public async Task<HttpResponseMessage> Get(int id)

{

Console.WriteLine(DateTime.Now.ToString() + " - Incoming GET request with id");

Data result = await dataService.GetByIdAsync(id);

return Request.CreateResponse(HttpStatusCode.OK, result, Configuration.Formatters.JsonFormatter);

}

// POST api/values

public async Task<HttpResponseMessage> Post([FromBody] Data value)

{

Console.WriteLine(DateTime.Now.ToString()+" - Incoming POST request");

if (value!=null)

{

value.Idopont = DateTime.Now;

int id = await dataService.AddNewAsync(value);

return Request.CreateResponse(HttpStatusCode.OK, new { Id = id }, Configuration.Formatters.JsonFormatter);

}

return Request.CreateResponse(HttpStatusCode.NotAcceptable, new { Error="Missing data" }, Configuration.Formatters.JsonFormatter);

}

// PUT api/values/5

public async Task<HttpResponseMessage> Put([FromBody] CommunicationType data)

{

string json = System.Text.Json.JsonSerializer.Serialize(new {tipus=data.Tipus });

HttpContent content = new StringContent(json, System.Text.Encoding.UTF8, "application/json");

await client.PostAsync(client.BaseAddress, content);

return Request.CreateResponse(HttpStatusCode.OK);

}

// DELETE api/values/5

public async Task<HttpResponseMessage> Delete()

{

await dataService.DeleteAllAsync();

return Request.CreateResponse(HttpStatusCode.OK);

}

}

### Modellek

Ahogy már írtam, a backend tartalmazza az adatbázis leképezését. A mi esetünkben egyetlen tábla - a Data tábla létezik - ezért a backend szintén egyetlen modell osztályt - Data.cs - tartalmaz. A modell osztály felépítése megfelel az adatbázis Data táblájának felépítésének:

public partial class Data

{

public int Id { get; set; }

public DateTime? Idopont { get; set; }

public int? AkkuFeszultseg { get; set; }

public int? AkkuToltoaram { get; set; }

public int? AkkuHomerseklet { get; set; }

public int? MotorHomerseklet { get; set; }

public int? MotorAram { get; set; }

public bool? AkkuHutes { get; set; }

public bool? MotorHutes { get; set; }

}

### Service-ek

A service-eken belül oldjuk meg a backend adatbázissal való kommunikációját.

A backendben egy service került kialakításra, a DataService. Ezen belül olyan, az adatbázissal kapcsolatos alapvető feladatokat valósítottam meg, mint az adatok listázása, egy konkrét adat kikeresése id alapján, új adat beszúrása, vagy az adatok törlése. A service metódusai a controller action-metódusai által kerülnek meghívásra.

internal class DataService

{

private readonly SolarBoatContext context;

public DataService()

{

context=new SolarBoatContext();

}

internal async Task<int> AddNewAsync(Data value)

{

await context.Data.AddAsync(value);

await context.SaveChangesAsync();

return value.Id;

}

internal async Task DeleteAllAsync()

{

await context.Database.ExecuteSqlRawAsync("TRUNCATE TABLE [Data]");

}

internal async Task<List<Data>> GetAllAsync()

{

int limit = 60;

int c=await context.Data.CountAsync();

int n = c-limit;

n= Math.Max(n, 0);

return await context.Data.Skip(n).ToListAsync();

}

internal async Task<Data> GetByIdAsync(int id)

{

return await context.Data.FirstOrDefaultAsync(d => d.Id == id);

}

}

## Frontend

A frontend a programoknak, weboldalaknak az a része, amelyik a felhasználóval közvetlenül kapcsolatban van. Feladata az adatok megjelenése, befogadása a felhasználó felől.

Weboldal esetén a frontend fejlesztő HTML, CSS vagy JavaScript kódokkal foglalkozik.

A frontendet HTML, Javascript, CSS és a React keretrendszer alkamazásával készítettem el.

A frontend gyakorlatilag egy egyetlen (de nagyon összetett) HTML oldalból álló weblap. React esetében a weblap komponensekből épül fel. A komponensek webrequesteket indítanak a backend felé, és feldolgozzák valamint megjelenítik a backendtől kapott adatokat.

### Szerepe a projektben

Az elkészült projektben a frontend határozni a felhasználói felületet, mellyel a felhasználók közvetlenül interakcióba fognak lépni.

### HTML

A HTML (magyarul: hiperszöveges jelölőnyelv) egy leíró nyelv, mely weboldalak készítéséhez lett kifejlesztve, és mára már internetes szabvánnyá vált. Naprakész változata a HTML5.

### Javascript

A JavaScript egy objektumorientált programozási nyelv, prototípus-alapú szkriptnyelv, melynek weboldalakon elterjedt a használata. Bár papíron ez a nyelv szabványosított, különböző böngészők részben mégis különbözően implementálják. Ebből kifolyólag a projekt szempontjából fontos kérdés, hogy az elkészült weblapot milyen böngészőből futtatjuk majd.

### CSS

A CSS egy stílusleíró nyelv, mely a HTML dokumentumok megjelenését írja le, azaz CSS-el formázhatjuk meg a weboldalunk megjelenését. Például állíthatjuk be a szöveg színét, vagy a képek elhelyezkedését stb.

### React

A React egy ingyenes nyílt-forráskódú Javascript könyvtár felhasználói felületek fejlesztéséhez.

### Axios

Az Axios segítségével küldünk http lekérdezéseket a backednek, illetve ezzel fogadjuk a backend felől érkező válaszokat is.

# A hajó és a szárazföldi egység kommunikációja

A hajó és a szárazföldi egység közti kommunikáció rádió kapcsolat segítségével lesz megoldva.

## Wifi

A Wifi egy vezeték nélküli kommunikációs technológia, amely lehetővé teszi elektronikus berendezések, például számítógépek, táblagépek, okostelefonok vagy mobiltelefonok stb. Csatlakoztatását az internethez rádiófrekvenciák használatával vagy infravörös kapcsolaton keresztül az információk továbbításához.

## Kommunikáció

A hajó duplex kommunikációt használva cserél adatokat a backenddel. Webrequestet indítva JSON formátumban küldi az adatokat a backendnek. A backend felől érkező utasításokat a HttpListener osztály használatával fogadja. A backend részéről szintén JSOn formátumban érkeznek az utasítások.

Mivel dolgozatok leadásnak határidejéig a hajó és a szárazföldi szerver közötti kommunikációt lehetővé tevő rádió nem érkezett meg, ezért a továbbiakban a tesztkörnyezet működéséről fogok beszélni.

Ide írd le a tesztkörnyezet működését.

# Kliensprogram megvalósítása

Ez valójában a backend része lett, úgyhogy szerintem erre nincs szükség.

## Valami

Ezt a részt ki kell majd dolgozni a hajó technikai kialakításától függően.

## Valami2

Ezt a részt ki kell majd dolgozni a hajó technikai kialakításától függően.

# Irodalomjegyzék

[1] Krámli György, Festo*,* Szenzorika tanfolyami jegyzet, 4.oldal

[2] Krámli György, Festo*,* Szenzorika tanfolyami jegyzet, 6.oldal

[7] Rónyai Lajos, Algoritmusok, 2004

[4] Douglas Bell, Programtervezés, 2003

https://www.mouser.com/datasheet/2/348/bh1750fvi-e-186247.pdf

https://pdf1.alldatasheet.com/datasheet-pdf/view/1160229/YHDC/SCT013-005.html

# Mellékletek

Ide a kész szakdolgozatból fognak kódrészletek kerülni.

# Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani témavezetőmnek, Dr. Medvegy Tibornak, aki szakértelmével, hasznos magyarázataival hatalmas segítséget nyújtott szakdolgozatom elkészüléséhez.

Hálával tartozom továbbá szüleimnek és testvéremnek, akik nélkül ez a szakdolgozat nem jöhetett volna létre. Köszönöm nekik, hogy tanulmányaim során türelemmel és megértéssel támogattak, és minden helyzetben mellettem álltak.

Külön köszönöm barátaimnak is, hogy egyetemi éveim során sok bíztató szóval támogatták céljaim elérését, és nélkülözhetetlen tanácsaikkal valamint segítségükkel hozzájárultak e dolgozat megírásához.

Köszönöm mindenkinek!