Szegedi Tudományegyetem

Informatikai Intézet

A képen szöveg, embléma, címerpajzs, jelvény látható

Automatikusan generált leírás

Szakdolgozat

Akvárium automatizálás és távfelügyelet megvalósítása

Implementation of aquarium automation and remote supervision

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Készítette: |  | Témavezető: |  |
|  | Takács Ábel |  | Dr. Mingesz Róbert Zoltán |  |
|  | mérnökinformatika szakos hallgató |  | egyetemi adjunktus |  |

Szeged

2024

**A feladat kiírása**

Ahhoz, hogy egy akvárium mikroklímája megfelelő legyen, számos berendezés kell összehangoltan működjön. Ilyenek pl. a vízforgató és szűrő berendezések, fényforrások, fűtőszálak, ventilátorok és CO2 adagoló rendszerek. Ezek egy jó részét egyszerű időzítőkkel szokták vezérelni, működésüknek a monitorozása pedig általában nincs megoldva. Számos műveletet (pl. halak etetése, vegyszerek adagolása) pedig csak manuálisan lehet elvégezni. A hallgató feladata egy olyan rendszer megvalósítása, mely egyrészt vezérelni tudja a különböző eszközöket, másrészről pedig monitorozza az akvárium állapotát, a mért paramétereket pedig távolról is le lehet kérdezni. A megvalósítás során figyelni kell az üzembiztonságra, vagyis a rendszer semmilyen körülmény között nem fagyhat le.

A következő funkciók jöhetnek szóba:

* Hőmérséklet és fénymennyiség monitorozása
* Világítás ki/be kapcsolása, több világítótest
* Ventillátorok ki/be kapcsolása
* Folyadékszínt monitorozása
* Folyadékszínt utántöltése
* Vegyszerek adagolása
* Haleledel adagolása
* Zajszintmérés, hibajelzés
* Szivárgásdetektálás
* Kémiai tulajdonságok monitorozása.

**Tartalmi összefoglaló**

**A téma megnevezése:**

Akvárium automatizálás- és távfelügyeletre használható rendszer tervezése és megvalósítása

**A feladat megfogalmazása:** Egy olyan rendszer megtervezése és letrehozása, mely képes egy adott akvárium jellemzőit (Hőmérséklet, Ph, Vízszint, Fénymennyiség) monitorozni, valamint a fenntartó berendezések (Levegőztető(k), Lámpatest(ek), Etetés) időzítését ellátni. Emellett megbízható, üzembiztos és sokáig képes működni. A felhasználó telefonjáról WiFi-n keresztül bárhonnan személyre szabhatja az időzítéseket, a monitorozott adatokat megtekintheti, valamint a preferált jellemzőket beállíthatja, értesítéseken keresztül tájékozódhat a beállított takarítási/karbantartási periódusokról. Figyelmeztetést kap, ha esetleg valamely jellemző elhagyja a preferált tartományt vagy az akvárium szivárog.

**A megoldás módszere:** A rendszer funkcionalitási elvárásainak felmérése, kutatás a felhasználandó eszközök után. A rendszer megtervezése, tesztelése, majd sajátcélú otthoni használat, így is feltárva az esetleges tesztek által fel nem fedett hibákat.

**A megoldáshoz használt módszerek, eszközök:**

* ESP8266 12E NodeMCU mikrovezérlő/WiFi modul
* Visual Studio Code, PlatformIO
* KiCAD áramkörtervező szoftver
* React Native keretrendszer, Expo kliens
* Hőmérséklet, Ph, Vízszint szenzorok, fotorezisztor, analóg multiplexer, shift regiszter
* Relé modul, LCD kijelző

**Elért eredmények:** A kész rendszer üzembe helyezése és sikeres működtetése, sikeres távirányítás és monitorozás az applikáción keresztül. A rendszer lefagyás nélküli üzemeltetése több héten át. A tervezett nyomtatott áramkör és az ESP-re tervezett szoftver is bővíthető további egy szenzorral és egy időzíthető konnektorral.

**Kulcsszavak:** *Akvárium, IoT, Mobil applikáció, Mikrovezérlő*

**Tartalomjegyzék**

Tartalmi összefoglaló 3

Bevezetés 5

[Irodalmi és technológiai áttekintés 6](#irodalmi_attekinto)

1. Hardver 6
   1. Mikrovezérlő 6
   2. Tápegység és tápfeszültség 7
   3. Szenzorok 7
   4. Aktuátorok 10
   5. Multiplexer 11
   6. Shift regiszter 12
   7. Kijelző 13
2. Szoftver
   1. Visual Studio Code és PlatformIO
   2. Az ESP programozásához használt függvénykönyvtárak
   3. Az adatbázihoz és szerverhez használt technológiák
   4. React Native, Typescipt és Expo
   5. React Native *thrid-party* komponensek

**Bevezetés**

Az intelligens eszközök egyre nagyobb részben vannak jelen a mindennapi életben, hiszen nemcsak kényelmi funkciókat látnak el, de rengeteg idő is megspórolnak nekünk. Ilyen például a napjainkban már szinte mindenhol előforduló termosztátos fűtési renszer, mely lehetővé teszi, hogy a fűtés automatikusan tartsa a beállított hőfokot a lakásban. De már sok helyen előfordulnak okos hűtők is, melyeken bevásárló listát lehet írni, vagy akár még rendelni is tud nekünk. Ezek a rendszerek sokban megkönnyíthetik a mindennapi életünket. Ez utóbbi beletartozik az ’IoT’ világába, melynek lényege, hogy a berendezéseinket interneten keresztül elérhetjük, irányíthatjuk.

Az akváriumok nagyon szép díszei lehetnek bármely lakásnak, feltéve hogy rendszeresen karban vannak tartva és az érzékeny mikroklímájuk megfelelően van beállítva és szabályozva. Napi szinten személyesen tapasztalom, mennyire szép és megnyugtató látványt tud nyújtani egy jól karbantartott akvárium, hiszen az íróasztalom, ahol napjaim nagy részét töltöm, helyet ad egy 54 literes akváriumnak. Annak érdekében, hogy egy ilyen rendszer megfelelően szolgálja lakóit (és persze minket), a berendezéseknek folyamatosan összhangban kell működniük, a vílágítást érdemes időzíteni, hogy mindig ugyanakkor kapcsoljon ki és be, valamint a halakat etetni kell, az akváriumot pedig tisztán tartani. Emellett pedig a halaknak megfelelő környezet megteremtése is kulcsfontosságú, hiszen ezek az állatok érzékenyek a hőmérséklet- és Ph-változásokra.

Ennek a dolgozatnak a célja, hogy egy olyan rendszert valósítak meg, mellyel ezek a feladatok leegyszerűsödnek, néhány pedig teljesen automatává válik. Önmagában az időzítési feldatok ellátása nem egy bonyolult dolog, viszont ezeknek a személyre szabhatósága és a beállítás valós idejű frissítése már okozhat fejtörést. A monitorozás a szenzorok kezelésén és a megfelelő mérési technika kiválasztásán alapul. A leginkább elgondolkodtató rész pedig az automata etetési funkció, melynek mechanikai háttere, valamint a különféle eledelekkel való kompatibilitás is gondos tervezést igényel.

Az elején ismertetni fogom a felhasznált eszközöket, ezek működését és használatát, majd pedig bemutatom hogyan is lehet ezen eszközök felhasználásával egy az imént említett rendszert létrehozni. A fejezetek tartalmazzák a hardveres és szoftveres megoldásokat, ideértve a használt áramkörök tervezetét, a szoftverek (Mikrovezérlő és telefonos applikáció) tervét és megvalósítási módját. Valamint a felmerült problémákat és az ezekre nyújtott megoldásokat.

**Irodalmi és technológiai áttekintés**

1. **Hardver**

**1.1 Mikrovezérlő**

A mikrovezérlő kiválasztásánál fontos szempont volt, hogy beépített WiFi antennával rendelkezzen, valamint az Arduino családba tartozzon, így lehetővé téve a rengeteg előre elkészített függvénykönyvtár használatát. A választásom egy **ESP8266 12E NodeMCU** [1] mikrovezérlő modulra esett. Ez a modul kifejezetten IoT területen lévő felhasználásra van kifejlesztve, valamint a lap amire integrálva van lehetővé teszi a breadboardon való kísérletezést és az egyszerű USB-n keresztül történő programozást, minimális felhasználói felülettel (reset, flash gombok).

A circuit board with many wires

Description automatically generated

* 1. ábra ESP8266 12E NodeMCU és lábkiosztása [1]

Ez egy 32 bites mikrovezérlő modul, melyben 64KB utasítás memória és 96KB adatmemória áll rendelkezésre, emellett pedig 4MB flash memória használható adatok tárolására, ez esetemben bőven elegendő a WiFi hálózat és a beállítások adatainak tárolására. Mivel ez a modul 3.3V-on üzemel viszont pár szenzor 5V-os tápfeszültésget igényel, így szükségem volt egy erre a célra megfelelő tápegységre is melynek 5V-os kimenetét lehet ennek a mikrovezérlőnek a meghajtására is használni Vin-en keresztül.

**1.2 Tápegység és tápfeszültség**

A tápegység kiválasztásánál az egyik legfontosabb tényező a méret volt, hiszen egy viszonylag kis dobozban kell majd helyet foglalnia, ugyanakkor képesnek kell lennie elegendő áramot biztosítani a mikrovezérlő, a szenzorok, a kijelző és a relék számára. Így a választásom a Mean Well   
RS-25-5 típusú tápegységre [1] esett. Ennek fontosabb adatai az alábbi táblázatban láthatóak [2].

A close-up of a power supply

Description automatically generatedA hand holding a power outlet

Description automatically generated

1.2.1 ábra Mean Well RS-25-5 tápegység [1] és az SPST csatlakozó [3]

Az RS-25-5 tápegység fontosabb jellemzői [2]

|  |  |
| --- | --- |
| DC feszültésg | 5V |
| Max. áram | 5A |
| Kimeneti áram tartomány | 0A ~ 5A |
| Bemeneti feszültésg (VAC) | 88 ~ 264V |
| Hatékonyság | 78.5 % |
| Túlfeszültség védelem | 5.75 ~ 6.75V |
| Méretek | 78 \* 51 \* 28mm |

Ezt a tápegységet felhasználva sikeresen meg tudom táplálni az összes 5V-ot igénylő eszközt, valamint a mikrovezérlőt is. A hálózati feszültség csatlakoztatására egy SPST típusú csatlakozóval ellátott aljzatot [3] választottam, mely beépített kapcsolóval és biztosítékfoglalattal rendelkezik.

**1.3 Szenzorok**

A hőmérséklet figyeléséhez egy **DS18B20** típusú vízálló digitális hőmérőt [1]. Ennek 3 kivezetése van, VCC és GND valamint az adat átvitelére szolgáló DAT, ezt egy tetszőleges digitális bemenetre kössük. Utóbbi vezetéket egy 4,7kΩ-os felhúzó ellenállással együtt kell bekötni, ezt a kivitelező egy a szenzorhoz mellékelt adapterrel meg is oldotta, így csak az adaptert kell megfelelően bekötni.

A close-up of a circuit board

Description automatically generated

1.3.1 ábra DS18B20 digitális hőmérő adapterrel [1]

A Ph mérésére egy Peukerty gyártótól származó **analóg** **Ph szenzort** [2] választottam, mivel ezt volt a legegyszerűbb beszerezni amazonon keresztül. Ez egy szenzor csövet és egy adapter modult tartalmaz, mellyel viszonylag egyszerűen lehet a mikrovezérlőhöz kapcsolni. Az adapter [3] 7 portjából csak 3-at fogok használni, mivel ennyi elegendő a Ph megállapításához. Fontos megjegyezni, hogy ezt a szenzort kalibrálni kell a pontos használathoz.

A soldering iron and a circuit board

Description automatically generatedA diagram of a device

Description automatically generated

* + 1. ábra Ph szenzor adapterrel [2] és az adapter adatai [3]

A vízszint mérésére egy **kapacitív vízszintmérő szenzort** [4] választottam. Ennek 3 kivezetése van, VCC, GND és egy S névre hallgató, ami az adatot közvetíti. A szenzor elve, hogy a benne elhelyezett áramkörnek köszönhetően folyadék kontaktus nélkül képes az üvegfal (vagy bármilyen nem fémes anyag) túloldalán lévő folyadékszint meghatározására, mivel a felépítésének köszönhetően képes érzékelni a folyadék jelenlétével létrejövő kapacitás változásokat. Természetesen egyszerre csak egy pontot képes mérni, így például szivárgásdetektálásra sajnálatos módon önmagában nem alkalmas, viszont cserébe hatékony és hosszútávú megoldást nyújt egy adott vízszint mérésére. Ez esetemben elegendő is arra, hogy a felhasználó manuálisan beállítsa a kívánt vízszintet és a szenzor ennek a szintnek a hiányát érzékelhesse.

A black wire with a white square with a green sticker on it

Description automatically generatedA diagram of a circuit

Description automatically generated

1.3.3 ábra, Kapacitív vízszint szenzor és működési elve [4]

Végezetül pedig a fényméréshez választott szenzorom egy vízálló **fotorezisztor** [5]. Ennek a szenzornak a működési elve, hogy fény hatására csökken az ellenállása, így kiszámítható milyen erősségű fény éri. Azért ezt a szenzort választottam mivel ez a legkézenfekvőbb és költséghatékonyabb megoldás, elegendő pontosságot nyújt ahoz, hogy 5 nagyobb kategóriára osszuk a mért fénymennyiséget, és ezt közvetítsük a felhasználó felé. Valamint tökéletes arra a célra, hogy esetleges kiégett lápmatest, vagy más hiba közben fellépő fényhiányt érzékelje.

// TODO kép

A close-up of a resistor

Description automatically generated

1.3.4 ábra, Fotorezisztor [5]

**1.4 Aktuátorok**

Az automatikus etetés funkciójához egy **MG90S** [1] típusú fém mechanikájú szervómotort választottam mint beavatkozó szerv. Ez egy kis motor, melyet egy adatvezetéken keresztül PWM alapon lehet irányítani, 3 vezetékkel rendelkezik [2] és 5V-on üzemel. A választott motor 180 fokot képes fordulni, így elegendő mechanikai behatást tud nyújtani egy rotációs alapon működő etetőhöz. Valamint elegendően nagy erőt tud kifejteni (1,8kgf \* cm) ahoz, hogy az etetőt elforgassa.

A small electronic device with wires

Description automatically generatedA close up of text

Description automatically generated

1.4.1 ábra, MG90S szervómotor [1] és a vezetékek értelmezése [2]

Az időzítésekhez egy 4 csatornás **relé modult** [3] választottam. A relék működési elve, hogy amikor az irányításra szolgáló bemenetén logikai magas értéket kap akkor a relében található elektromágnes bekapcsol, kapcsolva a benne lévő mágneses kapcsolót. Így lehetővé téve hálózati feszültség (~220V) kapcsolását a mikrovezérlő feszültségével (3.3V). Ehez a modulhoz egy 3 kivezetésű konnektorállást választottam, így az akvárium 3 berendendezését lehet rácsatlakoztatni (3 csatorna elegendőaz itt használt berendezéseknek, ugyanakkor bővíthető).

A close-up of a circuit board

Description automatically generated

1.4.2 ábra, 4 csatornás relé modul [3]

**1.5 Multiplexer**

A választott mikrovezérlőnek csupán 1 ADC-vel ellátott analóg portja van, viszont nekem 3 analóg szenzorom, így szükségem volt egy analóg multiplexerre. Erre a célra a CD4052 2 \* 4 csatornás analóg multiplexer/demultiplexer-t [1] választottam. A multiplexer lényege, hogy egy bemenetét vezérlővonalak írányítása alatt képes több kimenete egyikére kapcsolni. Esetemben 2 vezérvonal  
(A, B) és 2 \* 1 bemenet (x, y) valamint 2 \* 4 kimenet (0-3x, 0-3y) van [2]. Ezen keresztül tudom a szenzorok jeleit az analóg portra kapcsolni.

A black and silver electronic chip

Description automatically generatedA diagram of a circuit board

Description automatically generated

1.5.1 ábra, CD4052 multiplexer [1] és lábkiosztása [2]

**1.6 Shift regiszter**

Mivel az ESP-n korlátozott számú GPIO láb található, így (leginkább a relék miatt), így szükségem volt egy kis eszközre aminek segítségével ezen portok számát bővíthettem, a választásom egy **74HC595** [1] típusú shift regiszterre esett, melynek egyszerű használata és szélekörű működési paraméterei kedvezőek voltak számomra. A shift regiszter működése abból áll, hogy három vezetéken (adat, órajel, kapu/latch) keresztül lehet a nyolc kivezetésére logikai értékeket megjeleníteni, az órajel felfutó élére a regiszter beolvassa az adatvonalon lévő értéket, és a benne található D-tárolók segítségével minden egyes olvasás után az adatsorozat egyel elcsúszik (shift). Ebben a regiszterben pedig még egy, a kimeneteke állapotváltásának szinkronizálására szolgáló, kapu vezérvonal is szerepel, melynek egy órajelciklusa alatt az adat a belső D-tárolókból a kimenetekre kapcsolt D-tárolókra mozognak, így biztosítva, hogy a kimenetek egyszerre váltsanak értéket, miután mind a 8 bit bekerült a regiszterbe.

|  |  |  |
| --- | --- | --- |
|  | Min | Max |
| Vcc | 2V | 6V |
| Vin | 0V | Vcc |
| Vout | 0V | Vcc |
| Iout | -20mA | 20mA |
| Vo (H and L) | -0,3V | Vcc +0,5V |

A white rectangular object with black text

Description automatically generated

1.6.1 ábra, 74HC595 shift regiszter és fontosabb paraméterei [1]

**1.7 Kijelző**

A rendszernek szüksége van valamiféle adatmegjelenítő felületre is, már csak önmagában a WiFi hálózat konfigurálásához szükséges adatok megjelenítésére, hiszen a mikrovezérlő önmagában nem képes adatot megjeleníteni. Erre a célra egy 20 \* 4 karakteres LCD kijelzőt [1] választottam.

A close-up of a circuit board

Description automatically generated

1.7.1 ábra, LCD kijelző I2C interfésszel [1]

Az LCD kijelzőt a gyártó már felszereli egy I2C interfésszel, így a kommunikáció és bekötés is sokkal egyszerűbb. A kijelzőnk ezzel az interfésszel összesen 4 portot kap, természetesen itt is van VCC és GND emellett pedig egy SCA (adat) és egy SCL (órajel) pineket találunk. Utóbbi két vezetéken keresztül van megvalósítva a szinkron soros adatátvitel.

**1.8 Tranzisztorok**

Mivel egy kevert jelű (3.3V és 5V) rendszeren dolgozok, elkerülhetetlenek a tranzisztorok használatai, valamint a relék miatt kifejezetten ajánlott. Erről bővebben a hardveres megoldások (3) fejezetben beszélek. A projekthez egy logikai jelszintű MOSFET (IRLZ34N) típusú térvezérlésű tranzisztort választottam. A tranzisztor egy félvezető áramköri alkatrész, három kivezetése van (térvezérlésű esetén ezek Gate, Source és Drain), ezek közül a Gate szabályozza a Source és Drain közt folyó áram mennyiségét. Így nagyszerűen használható arra, hogy 5V-os jeleket kapcsoljak az ESP 3.3V-os jeleivel.

**2 Szoftver**

A dolgozat megvalósításához több különféle fejlesztői eszközt és kész függvénykönyvtárat használtam, ebben a fejezetben ezeket mutatom be, többek között az ESP mikrovezérlő programozásához használt környezetet és könyvtárakat, valamint a mobil alkalmazáshoz felhasznált technológiákat.

**2.1 Visual Studio Code és PlatformIO**

Az ESP mikrovezérlő programozása többféle környezet felhasználásával is történhet, egyik legelterjedtebb talán az Arduino IDE, azonban ez a környezet számomra alkalmatlannak bizonyult, mivel viszonylag nagyobb forráskódot kell létrehoznom, a könnyebb átláthatóság érdekében inkább a VSCode mellett döntöttem. Az IDE PlatformIO bővítménye [1] kifejezetten mikrovezérlők programozásához lett kifejlesztve. Itt kiválasztható a programozni kívánt modul (ez esetemben egy ESP8266 12E NodeMCU), majd a kiválasztás után a bővítmény automatikusan telepíti a szükséges függőségeket, mely esetemben egy *esptool* névre hallgató python program, mely lényegében a fordítási és telepítési lépések automatizálására használható.

A screenshot of a computer

Description automatically generated

2.1.1 ábra, VSCode PlatformIO bővítménnyel (PIO kezdőlap és .ini fájl) [1]

**2.2 Az ESP programozásához használt C/C++ függvénykönyvtárak**

Az ESP alapú mikrovezérlők előnye, hogy kompatibilisek az összes Arduino alapú függvénykönyvtárral, így ezen könyvtárak széles választéka közül szabadon tudtam válogatni. Ebben a fejezetben felsorolásszerűen minimálisan részletezve mutatom be a használt könyvtárakat.

* **Arduino –** Az Arduinon használható alapfüggvények széles tárháza.
* **ESP8266WiFi** – WiFi kapcsolat létesítésében játszik fontos szerepet.
* **ESP8266 HTTPClient –** HTTP kérések küldésében játszik szerepet.
* **ArduinoJson –** JSON fájlok kezelésére használható (szérializáció, deszérializáció).
* **NTPClient –** A pontos idő hálózaton keresztül történő szinkronizálására szolgál.
* **WiFiUdp –** Az NTPClient használatához szükséges UDP protokollt kezeli.
* **TimeLib –** A mikrovezérlő belső RT órájához való egyszerű hozzáférést biztosítja.
* **EEPROM** – A mikrovezérlő memóriájához (flashez is) biztosít egyszerű hozzáférést.
* **LiquidCrystal\_I2C** – I2C interfésszel felszerelt LCD kijelzők használatához biztosít függvényeket.
* **OneWire** – Egyvezetékes szenzorok jeleinek olvasásához tartalmaz függvényeket.
* **DallasTemperature** – A DS18B20-as egyvezetékes hőmérséklet szenzor egyszerű kezeléséhez biztosít függvényeket (a OneWire ennek a függősége).

**2.3 Az adatbázihoz és szerverhez használt technológiák**

A használt adatbázis egy MariaDB alapú MySQL adatbázis, melynek tárhelye egy saját szerveren található, melyen php nyelven íródott API fogadja a kéréseket.

Ez a nyelv széleskörűen elterjedt első sorban webes alkalmazások területén, hiszen beágyazható HTML kód, valamint dinamikusan is képes HTML kódot generálni. Emelett pedig rengeteg beépített nyelvi funkcióval és függvénykönyvtárral rendelkezik különféle problémákra. Emelett jól moduralizálható, támogatja az OOP megközelítést is. Az adatbázis hozzáféréshez és kezeléshez a *mysqli* könyvtárat választottam, használatával viszonylag egyszerűen tudunk kommunikálni az adatbázissal. // TODO írni még 2 sort

**2.4 React Native, Typescipt és Expo**

A mobilalkalmazás fejlesztéséhez olyan nyelvet és rendszert próbáltam találni aminek használatában van tapasztalatom és léteznek felhasználható könyvtárak/csomagok amiknek segítségével egyszerűsödhet az app létrehozása, így a választásom a React Native-ra esett melyet az Expo keretrendszerrel használva egyszerűen, az újrafelhasználhatóságot és karbantarthatóságot szem előtt tartva lehet multiplatform alkalmazásokat fejleszteni akár web, IOS vagy Android rendszerre.

A React Native a Facebook által fejlesztett nyílt forráskódú keretrendszer, mely lehetővé teszi a multiplatform alkalmazásfejlesztést egy kódbázis felhasználásával. Magát a keretrendszert lehet Javascript vagy Typescript nyelven is használni, melyek közül egyértelműen a Typescriptet választottam.

Számos fejlesztést elősegítő keretrendszer létezik, én az Expo-t választottam, mellyel lehetőségem nyílik az úgynevezett *hot-reload* technika (az alkalmazás követi a forráskód változtatásai újrafordítás nélkül) használatára, valamint a buildelési folyamatokat is kezelni tudja.

A screenshot of a computer

Description automatically generated

2.4.1 ábra, Expo-cli [1]

**2.5 React Native *thrid-party* komponensek**

Mint korábban említettem azért választottam a React Native keretrendszert, mivel számos már előre elkészített nyílt forráskódú komponens található hozzá, amivel a fejlesztési időt jelentősen le lehet csökkenteni. Ebben a fejezetben a legfontosabbakat fogom röviden ismertetni.

* **React-native-async-storage** – Az eszköz tárhelyéhez való hozzáférést és azzal való műveletek (írás, olvasás) elvégzését biztosítja aszinkron függvényekkel.
* **React-native-community/datetimepicker –** Dátumés idő kiválasztásához alkalmas komponenst biztosít, mely az adott eszköz operációs rendszerének megfelelő választófelületet jelenít meg.
* **React-navigation/bottom-tabs –** A navigációhoz nyújt segítséget, egy az ablak alján elhelyezhető navigációs vezérlő segítségével.
* **React-navigation/stack –** Navigációs komponens, alapvető navigációs funkciókat lát. Ennek segítségével lehet navigálni pl. gomb megnyomására.
* **Expo-notifications –** Expo keretrendszer segítségével küldhető értesítések használatára.
* **Expo-secure-store –** Egy titkosítással ellátott tárhelyet biztosít az eszközön, érzékeny adatok tárolására.
* **Expo-status-bar –** Egy a töltés folyamatát jelző animációs komponenst biztosít.
* **React-native-animatable –** Komponensek animálását teszi lehetővé.
* **React-native-bouncy-checkbox –** Egy animált checkbox komponenst biztosít.
* **React-native-chart-kit –** Különféle diagramok egyszerű létrehozásához szükséges komponenseketr tartalmaz.
* **React-native-dropdown-select-list –** Egy legördülő lista komponens.
* **React-native-keyboard-aware-scroll-view –** Görgethető megjelenítőfelületet, melyet a billentyűzet jelenléte nem befolyásol.
* **React-native-vector-icons –** Különféle ikonokat tartalmaz különféle családokból.

**3 Szoftveres megoldások**

**3.1 Az ESP mikrovezérlőhöz használt szoftveres megoldások**

A választott mikrovezérlőm programozása viszonylag egyszerűnek bizonyult a PlatformIO használatával és mivel az ESP programozását lehet C++ nyelven végezni, így egyszerűen tudtam osztályokra bontott megoldást nyújtani a feladat megvalósítására. Az ESP-n futó forráskód egyszerűsített osztálydiagramja a 3.1-es mellékletben található.

Az osztályokat a különféle feladatok elvégzése szerint hoztam létre, valamint az OOP ideológiája szerint azokat a struktúrált elemeket melyekből viszonylag sok egyed előfordulhat is osztályoztam. Így készült egy osztály a különféle szenzorok adatainak struktúrált tárolására és egy osztály a szenzorok kezelésére. Ezen elképzelés mentén készítettem egy osztályt a beállítások/preferenciák struktúrálására és egy másikat ezek kezelésére. Készült egy-egy osztály a memória, az aktuátorok, a felhasználói felület, a szerver-kapcsolat és persze a WiFi kapcsolat kezelésére is. A vezérlésért felelős osztályokat lehetőség szerint *singleton* módon implementáltam, valamint globálisan kerültek példányosításra, így elkerülvén a nem kívánatos ütközéseket és hibákat (A memória-EEPROM kezelésért felelős osztály szigorúan *singleton*). Az osztályokon felül készült még négy *enum* is, melyek a szenzor adatok és beállítások működéséhez szükséges konstansokat tartalmaznak.

A teljes osztálykomplexumot a mikrovezérlőn futó *main* fájlban található setup és loop függvény bírja együttműködésre (A setup indításkor egyszer fut le, majd a loop fog egy végtelen ciklusban futni). Ezeken felül pedig néhány segédfüggvény szerepel még, melyek a komplexebb és/vagy többször használt műveletek kiszervezésére szolgálnak. A forráskód részleteire nem térnék ki, csupán néhány fontosabb elemet, függvényt emelnék ki.

Az első és talán legfontosabb működési szempontból a WiFi és a szerver kezelése, hiszen ezek nélkül nem létezhetne a megtervezett rendszer. Az első osztály a *ServerConnector* [3.1.1], ez felel a WiFi kapcsolat megfelelő felállítására, valamint a szerver-kapcsolatért. Ez az osztály intelligensen inicializálja magát (a konstruktor felépítése a 3.1.1-es ábrán látható), ami alatt azt értem, hogy már a konstruktorban ellátja a létfontosságú műveleteket, például a *WiFiConfig* addattagja inicializálását, mely ekkor megnézi léteik-e elmentett hálózat a memóriában (ehez a *MemoryHandler* osztályt használja, pontosabban egy referenciát a *MemoryHandler* belső példányára).

A screen shot of a computer program

Description automatically generated3.1.1 ábra, A *ServerConnector* osztály konstruktora

Ha talált elmentett adatokat, akkor sikeres a konstruktor futása, ellenkező esetben a friss rendszer inditásával azonos módon készít egy szervert az ESP modulból. Így a felhasználó a telefonjáról rácsatlakozhat a mikrovezérlő által generált hálózatra, ahol adott IP címen keresztül eléri a WiFi konficurációs oldalt. Itt megadhatja az otthoni hálózata adatait, majd mentés után az ESP újraindul, és a frissen mentett adatok miatt már biztosan sikeres lesz a *WiFiConfig* inicializálása. Természetesen a folyamat állapotáról folyamatos visszajelzést kapunk a beépített LCD kijelzőn. Ha a mentett adatokat sikerült betölteni, akkor, következhet az internethez való csatlakozás, melynek sikertelensége (adott időkeret utáni sikertelen csatlakozás) implikálja, hogy a *WiFiConfig* által elmentett adatok hibásak. Ebben az esetben a rendszer elfelejti a mentett adatokat és újraindítja a konfigurációhoz használatos szervert. Sikeres csatlakozás esetén, ha a rendszer még nincs regisztrálva (nincs a memóriájában mentett azonosító) akkor a szerverhez csatlakozva regisztráljuk és kiosztunk neki egy azonosítót (ha ez a folyamat sikertelen akkor egyértelműen nem jó az internet kapcsolat, tehát újra kell konfigurálni). Ezután minden esetben frissítésre kerül az NTP szerver pontos ideje, melyhez az belső órát szinkronizáljuk. A konstruktor ezen felül inicializálja az értesítések elküldöttségét jelző flag-eket, ezekre azért van szükség, hogy egy adott értesítést ne küldjünk el minden percben, hanem maximum napi egyszer.

A rendszer egy másik fontos pillére a beállítások és preferenciák struktúrálását és kezelését végző *ConfigData* és *ConfigHandler* osztályok.

4 ESP8266 (Szoftveres megoldások)

**5 Adatbázis és API**

Az adatbázis egy fontos eleme a projektnek, hiszen itt kerülnek eltárolásra az adatok és a beállítások is, valamint innen tudjuk a telefonos applikációban megjeleníteni a szenzorok adatit is. Az itt eltárolásra kerülő adatok a felhasználó adatai (email cím, név, jelszó), az akvárium (egy felhasználónak lehet több is) adatai (azonosító, név, méretek), a konfigurációs adatok az adott akváriumhoz valamint az adott akvárium szenzorai által mért adatok. Utóbbiakat a mikrovezérlő adott időközzel mintavételezi (ez külön konfigurálható, max 15 perc), és feltölti azokat az adatbázisba. Ezen felül az értesítések a mikrovezérlőről érkeznek a PhP alapú API felé, ahonnan tovább lesznek küldve a telefonos applikációba úgynevezett ’push notification’ formájában.

**5.1 Az adatbázis**

Az adatbázist belső kapcsolatait leíró egyed-kapcsolat diagramot a // TODO melléklet tartalmazza. Ennek a digramnak a relációsémákká való leképzése itt látható (az Outlet helyett az OL rövidítést használjuk):

User(email, firstName, lastName, deviceToken, authToken);

Aquarium(Id, name, length, depth, height, fishCount);

HaveAquarium(*email*, *Id*);

Config(*Id*, minTemp, maxTemp, minPh, maxPh, OL1On, OL1Off, OL2On, OL2Off, OL3On, OL3Off, waterLvlAlert, prefLight, feedingTime, foodPortions, filterClean, waterChange, sampleTime, lastModifiedDate);

SensorSample(*Id*, When, temperature, Ph, waterLvl, lightAmount);

Ezek alapján már egyértelműen felvehetjük az adatbázisunk tábláit, melyeknek SQL leírása megtalálható a // TODO számú mellékletben. Értelemszerűn összekapcsoljuk a táblákat, hogy esetleges felhasználó törlés esetén az ehez tartozó adatok is törlődjenek, valamint ha mondjuk a felhasználó kitörli egyik akváriumát akkor az ahoz tartozó konfigurációk és minták is törlődjenek.

Az akvárium azonosítója automatikusan generálódik majd, amikor a rendszer először csatlakozik az adatbázishoz, ez a WiFi hálózat megadása után meg is történik (a rendszer létrehoz egy akváriumot az adatbázisban alapértelmezett paraméterekkel, melyek később módosíthatók), ekkor a felhasználó a rendszer kijelzőjén láthatja majd a kiosztott azonsítót, ezt a telefonos applikációba való regisztrációkor meg kell adni, vagy ha már regisztrált (ez implikálja, hogy már van egy rendszere), ilyenkor az új akvárium felvételéhez kell megadni az azonosítót. Ennek megadása nélkül nem köthető össze a felhasználó és az akvárium így ez kötelező. Ezután a felhasználó már tetszőlegesen állíthatja az akvárium további adatait.

Minden konfigurációnak van egy azonosítója, amely az akvárium azonosítójával egyezik meg, külső kulcsot alkotva, így minden akváriumhoz csak egy konfiguráció tartozhat és fordítva, melyet tetszőlegesen lehet állítani az applikációban bejelentkezés és az adott akvárium kiválasztása után.

**5.2 Php API**

A screen shot of a computer

Description automatically generatedAz API (Access Point Interface) minden alkalmazás esetén létfontosságú, hiszen ezen keresztül történnek az adatok kezelése és a kérések kiszolgálása is, ez adja a kapcsolatot az alkalmazás és az adatbázis közt. Esetünkben ez az API Php alapú, és kezeli mind a mikrovezérlő, mind a telefonos applikáció által küldött és fogadott adatokat. Implementálásra kerülnek benne a használt adatszerkezetek modelljei, az ezek alapján az adatbázissal interaktáló DAO (Data Access Object) valamint a különféle kérések kiszolgálására létrehozott vezérlések (minden kérés egy-egy különböző fájlba fut be, mely kezeli azt). Az API mappaszerkezetének felépítése a 5.2.1-es ábrán látható. Természetesen a CONTROLS mappa tartalma jelentősen több, hiszen ide kerül minden funkció megvalósítását ellátó fájl is, de szemléletesség céljából itt ennyi is elegendő. 5.2.1 ábra

**Mellékletek**

A diagram of a diagram

Description automatically generated

1. Melléklet - A koncepció blokkvázlata