Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Návrh nízkonákladového řešení regulační a ovládací jednotky pro akvária a terária

Bakalářská práce

Michal Papaj

Školitel: PhDr. Milan Novák, Ph.D.

České Budějovice 2021

Bibliografické údaje

Papaj, M., 2021: Návrh nízkonákladového řešení regulační a ovládací jednotky pro akvária a terária. [Design of a low-cost solution for regulation and control unit for aquariums and terrariums. Bc. Thesis, in Czech] – 129 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Práce se zabývá návrhem nízkonákladového řešení regulační a ovládací jednotky pro akvária a terária. Zkoumá nároky živočichů na prostředí a možnosti vhodných technologií pro akvária a terária. V rámci analýzy je zvolena vhodná nízkonákladová technologie pro vlastní regulační jednotku. Praktická část se zabývá návrhem a konstrukcí systému regulační jednotky. Jednotka je schopna monitorovat či regulovat mimo jiné teplotu, pH, výšku vodní hladiny nebo osvětlení. Je taktéž navržena a implementována mobilní a webová aplikace pro řízení regulační jednotky. Výsledný systém umožňuje kontrolovat stav prostředí a vzdáleně jej regulovat.

Annotation

Bachelor's thesis is focused on a design of a low-cost regulation and control unit for aquariums and terrariums. Initial part examines demands of animals on the environment and possibilities of suitable technologies for aquariums and terrariums. A suitable low-cost technology for the regulation unit is selected. Practical part aims on the design and construction of the control unit system. It is capable of monitoring or regulating, among other things, temperature, pH, water level or lighting. Mobile and web applications are designed and implemented. The system allows a user to oversee the state of the environment and regulate it remotely.

Prohlášení

Prohlašuji, že jsem autorem této kvalifikační práce a že jsem ji vypracoval pouze s použitím pramenů a literatury uvedených v seznamu použitých zdrojů.

V Českých Budějovicích dne 8. 12. 2021

.....

Michal Papaj

Poděkování

Rád bych poděkoval své rodině a přátelům za trpělivost a podporu.

Obsah

1 Úvod 1
1.1 Cíle práce 1
2 Analýza 3
2.1 Terária 3
2.1.1 Nároky živočichů v teráriích 3
2.1.2 Současná řešení ovládání terárií 6
2.2 Akvária 8
2.2.1 Nároky živočichů v akváriích 8
2.2.2 Současná řešení ovládání akvárií 9
2.3 Specifikace požadavků 11
2.3.1 Funkční požadavky 11
2.3.2 Nefunkční požadavky 12
2.4 Scénáře 13
2.4.1 Přihlášení uživatele do aplikace 13
2.4.2 Přidání vivária 13
2.4.3 Nastavení vivária online 14
2.4.4 Nastavení vivária při Bluetooth spojení 15
2.4.5 Upozornění na stav akvária 15
2.4.6 Zobrazení historie 16
2.5 Zvažované technologie 17
2.5.1 Arduino 17
2.5.2 Particle 19
2.6 Zvolené technologie 21
2.6.1 ESP32 21
2.6.2 Flutter 22
2.6.3 Firebase 24
3 Design 30
3.1 Architektura systému 30
3.1.1 Architektura s centrální jednotkou 30
3.1.2 Architektura bez centrální jednotky 31
3.2 Regulační a ovládací jednotka vivária 32
3.2.1 Interní moduly 32

3.2.2 Externí moduly ... 35
3.2.3 Návrh akvária ... 42
3.3 Kamera ... 47
3.4 Flutter aplikace ... 48
3.4.1 Obrazovky aplikace ... 48
4 Implementace ... 58
4.1 Firebase ... 58
4.1.1 Realtime Database ... 58
4.1.2 Storage ... 60
4.1.3 Testy bezpečnostních pravidel ... 61
4.2 Flutter aplikace ... 62
4.2.1 Nastavení mobilní aplikace ... 62
4.2.2 Nastavení webové stránky ... 63
4.2.3 Named Routes ... 63
4.2.4 Provider ... 66
4.2.5 Použité knihovny ... 67
4.2.6 Struktura kódu ... 71
4.2.7 Models ... 71
4.2.8 Bluetooth ... 74
4.2.9 Správa uživatelů ... 78
4.2.10 Správa vivárií ... 79
4.2.11 Testování ... 81
4.3 Kamera ... 82
4.3.1 Běh programu ... 83
4.3.2 Kód ... 83
4.4 Vivárium ... 87
4.4.1 Bluetooth ... 88
4.4.2 Wi-Fi ... 90
4.4.3 Firebase ... 90
4.4.4 Paralelní vlákno ... 92
4.4.5 Rozhraní modulů ... 93
4.4.6 Ovládací panel ... 97
4.4.7 Aktualizace firmwaru ... 98
4.4.8 Třída Vivarium ... 100
4.4.9 Vybrané moduly ... 102
4.4.10 Testování ... 108

5 Testovací provoz ... 110
5.1 Nastavení a ověření běhu programu ... 110
5.2 Nalezené problémy a jejich řešení ... 112
5.2.1 Únik paměti ... 112
5.2.2 SSL Certifikáty ... 112
5.2.3 Zamrzávání hlavního vlákna ... 112
5.3 Testování vybraných modulů v praxi ... 113
5.3.1 Výška vodní hladiny ... 113
5.3.2 PID regulace teploty ... 114
5.3.3 Použití dvoustavového přepínače ... 116
5.3.4 Chlazení ... 118
6 Závěr ... 119
7 Bibliografie ... 121
8 Seznam použitých symbolů a zkratek ... 124
Seznam tabulek ... 125
Seznam schémat ... 125
Seznam grafů ... 125
Seznam kódu ... 125
Seznam obrázků ... 127
Seznam příloh ... 129

1 Úvod

V dnešní době zvyšující se popularity zařízení připojených do sítě IoT je jen málo odvětví, které by touto inovací neprošly. Jednou z nejvýznamnějších oblastí je rozšiřování produktů zapojených do chytré domácnosti. V rámci chytré domácnosti jsou jednotlivá zařízení připojená do jedné sítě tak, aby uživateli umožňovala pohodlné ovládaní jednotlivých prvků domácnosti. V současnosti může majitel takového systému ovládat některé domácí spotřebiče, mnohem známější je však využití v oblasti zabezpečení a úspory energie. Domov může být monitorován a dveřní senzory mohou reagovat na vstup osob. Úspora energie může mimo jiné spočívat v regulaci teploty v závislosti na přítomnosti osob v bytové jednotce. Veškeré inovace v oblasti IoT slouží ke zvýšení komfortu uživatele. Zařízení IoT většinou disponují připojením k internetové síti, což umožňuje snadné ovládání prostřednictvím webové stránky či aplikace.

Jedním z odvětví, kterému dosud nebyla věnována téměř žádná pozornost, je využití IoT v případě akvárií a terárií. V současné době lze nalézt ukázku několika soukromých projektů pracující s myšlenkou inteligentních akvárií či terárií. Jedním z úspěšných projektů v tomto odvětví je startupový projekt arménské firmy Bluenero. Projekt této firmy je však komerční a výhradně zaměřený na akvária, přičemž nabízí již hotový produkt včetně akvária samotného.

1.1 Cíle práce

Konečným výstupem této práce je modulární regulační a ovládací jednotka pro akvária a terária. K jednotce lze připojit vstupně-výstupní periferie pro zvýšení komfortu uživatele tak, aby bylo jeho vlastní činnosti vyžadováno co nejméně. Za základní moduly této jednotky lze považovat automatický dávkovač krmení s volitelným intervalem podávání, PID regulátor teploty jak pro vodní, tak pro suchozemské prostředí. Mezi podpůrné module řadíme čidla pro zlepšení přehledu uživatele nad zařízením. Jedná se především o detektory hladiny, tepelná čidla, stmívače či ovládání světel.

Pro splnění hlavního cíle této práce je třeba splnit následující úkoly:

- Analyzovat možnosti technologií pro řízení akvárií a terárií
- Analyzovat současná řešení v oblasti řízení a regulace akvárií a terárií
- Specifikovat funkční a nefunkční požadavky
- Vytvořit uživatelské scénáře použití
- Navrhnout architekturu systému regulační a ovládací jednotky
- Zkonstruovat regulační a ovládací jednotku
- Implementovat dílčí části zkonstruované regulační a ovládací jednotky
- Otestovat funkčnost systému regulační a ovládací jednotky a jejích částí

2 Analýza

Aby bylo možné vytvořit plně funkční regulační a ovládací jednotku, je nezbytné určit požadavky živočichů, které lze pomocí jednotky regulovat a řídit. Největší změny nastanou při rozhodování mezi chovem suchozemských a vodních živočichů. V obou případech je třeba kromě prostorových nároků dbát i na další biologické potřeby jednotlivých živočichů tak, aby byly vytvořeny podmínky k zabezpečení jejich pohody. [1]

Zároveň je nutné analyzovat současná řešení. Bude potřeba určit klady a zápory jednotlivých řešení a na základě této analýzy poté provést návrh vlastního řešení.

2.1 Terária

Pro chov plazů, obojživelníků a v některých případech i malých savců se využívá terárií. [2]

2.1.1 Nároky živočichů v teráriích

Ústřední komise pro ochranu zvířat vydala doporučení, které stanovuje podmínky pro chov suchozemských živočichů v teráriích. Patří k nim především vytvoření předpokladů pro uspokojení biologických potřeb a zabránění vzniku stresových situací, výbavou chovných prostor a výběrem vhodných materiálu použitých pro v daném prostředí. [1]

Denní světlo

Světelné podmínky v terárium se musí co nejvíce přiblížit podmínkám ve volné přírodě. Především plazi potřebují velmi světlé prostředí a nedostatek světla může zapříčinit jejich postupné chřadnutí. I při dobrém umístění terária je potřeba dodat interní osvětlení. Pro tento účel se hodí především zářivky a kompaktní žárovky se světelným spektrem vhodným právě pro plazy.

Mezi chovanými druhy dochází k velké diverzitě vzorců chování, především z důvodu jiné lokality původu. Je tedy třeba regulovat délku doby, intenzitu a spektrum světla. Pralesní druhy požadují zastínění v části chovného prostoru. [1]

UV záření

Plazi a obojživelníci potřebují pro svůj kvalitní život zdroj UV záření, konkrétně typ UVB a UVA. Každý z těchto typů hraje jinou roli pro zdraví chovaných zvířat. UVA záření má u plazů i obojživelníků velký vliv na jejich psychiku. Motivuje je k aktivitě, podporuje chuť k jídlu a reprodukční činnost. V neposlední řadě též velmi ovlivňuje jejich zrak. UVB záření

pomáhá se syntézou provitaminu D na vitamín D3, který plazi potřebují pro správné zpracování vápníku.

UV záření neprojde sklem ani plastem, je tedy třeba umístit jeho zdroj s přímou viditelností dovnitř terária. [3]

UVA záření

Pro plazy je UVA záření velmi důležité. Lidské oko obsahuje tři funkční typy čípků, kdy každý z nich je specializován na detekování jiné barvy. Nicméně většina plazů má čtyři funkční typy čípků, přičemž ten čtvrtý je schopen detekovat právě UV záření. Tito plazi tedy vidí větší rozsah barev a absence UV záření může razantně snížit jejich schopnosti rozpoznat jejich druhy nebo dokonce odlišit potravu od zbytku okolí. [4] [5]

UVB záření

Plazi potřebují UV záření pro tvorbu vitaminu D3, který posléze používají na zpracování vápníku. Z tohoto důvodu tráví většinu dne na přímém slunci.

Při chovu plazů a obojživelníků je tedy třeba poskytnout náhradní zdroje UV záření, aby se mohli zdravě vyvíjet a růst. V případě chybějícího zdroje UVB záření dochází k deformacím kostry, problémům s trávením či nedostatečné aktivitě živočicha. Následkem těchto prvotních problémů dochází k dalším nemocem a k celkovému oslabení imunity.

Je potřeba zajistit, aby byli plazi vystaveni hodnotám UVB záření odpovídající jejich přirozenému prostředí. Živočich by měl mít možnost opustit prostor výskytu takového záření. Jakékoliv UVB záření nad rámec požadavku pro tvorbu vitamínu D3 nepřináší žádný užitek, neboť další ultrafialové světlo už pouze rozkládá vytvořený přebytek na biologicky neaktivní látky [6]

Teplo

Většina plazů je velice citlivá na změny teploty, je tedy třeba dbát o správnou teplotu se zvýšenou péčí. Plazi a obojživelníci spadají do skupiny ektotermních živočichů. Na rozdíl od endotermních živočichů produkují ektotermové metabolické teplo velmi pomalu. Mnohdy mají špatnou teplotní izolaci, což jim pomáhá s příjímáním tepla z okolního prostředí. Svou teplotu potom regulují přesouváním mezi oblastmi s různou teplotou. [7]

V přírodě si tak plazi vybírají slunné prostředí pro zahřátí a naopak, pokud potřebují svou teplotu snížit, vyhledávají stín nebo se schovají z povrchu pod zem. Ektotermním živočichům tedy nestačí konstantní teplota po celém objemu terária. Je třeba vytvořit prostředí, kde bude

teplota rozložena tak od nejnižší hodnoty na jednom konci po nejvyšší na konci druhém. Jedině tak může být pro živočichy zajištěna možnost volby ideální teploty dle jejich aktuálních potřeb. [8]

Pro správně distribuovanou teplotu je třeba veškeré zdroje tepla umístit na jednu stranu terária. V případě potřeby světelných zdrojů UVA a UVB záření je třeba jejich umístění do stejného prostoru spolu se zdroji tepla. Toto opatření vede k simulaci přirozeného života živočichů. Většina plazů také vyžaduje střídání denního a nočního režimu, přičemž v nočním režimu vyžadují podstatné snížení teploty. Je tedy třeba omezit vytápění terária v tomto cyklu. [8]

Voda a vlhkost

Distribuci vody a vlhkost v teráriu je třeba nastavit s ohledem na chovaný druh živočichů. Voda u průměrného obojživelníka či plaza tvoří přibližně dvě třetiny jeho hmotnosti a je potřebná nejen pro jeho tělesné funkce, ale ovlivňuje též i chování. Pro většinu plazů a obojživelníků je dostatek vody v jejich okolí důležitý pro proces rozmnožování a inkubaci vajíček. Mimo stojící vody je v životě těchto živočichů potřebný i déšť, který zajišťuje potravu novorozeným potomkům. [9]

Zdroje vody v teráriu lze rozdělit do 4 kategorií:

- **Terarijní misky** – základním zdrojem stojaté vody v teráriu je terarijní miska. Čerstvá a čistá voda by v teráriu až na několik výjimek měla být dostupná veškerý čas. Zde je důležitá velikost misky, neboť některé druhy vyžadují dostatečný objem vody pro ponoření celého těla. Voda v misce může být jedním z faktorů vlhkosti v teráriu. Miska může být umístěna do blízkosti tepelného zdroje pro urychlení jejího odpařování. [9]
- **Kapátka** – některé druhy živočichů, převážně ty žijící v oblastech s dostatkem zeleně, mohou vyžadovat vodu kapající na určité místo v teráriu. [9]
- **Mlhovač/odpařovač vody** – mlha může být zdrojem pití pro některé malé živočichy. Zařízení odpařující vodu má největší podíl na celkové vlhkosti terária. [9]
- **Rosící zařízení** – rosící zařízení je pro mnohé chované jedince nepostradatelné. Existují druhy plazů, kteří ignorují stojící vodu a pijí pouze v případě deště například z listů rostlin. Déšť také pomáhá ke zvýšení aktivity plazů a obojživelníků. [9]

2.1.2 Současná řešení ovládání terárií

Regulace teploty za pomoci ventilátoru

Jakub Fürbach ve své práci *Regulátor prostředí terária* přichází s řešením regulace teploty za pomoci regulace otáček dvou ventilátorů. Předpokládá zde vytápění jedinou žárovkou s UVA zářením, která je schopná vytopit testované terárium. Tepelná čidla poté kontrolují změnu teploty a dle odchylky od teploty požadované jsou poté upraveny otáčky ventilátorů. Pokud požadovaná teplota odpovídá aktuální, jsou otáčky ventilátorů nastaveny na 50 %. Správným nastavením bylo dosáhnuto regulace teploty s přesností na 1,5 °C oběma směry. [10]

Pro osvětlení je využita žárovka s UVA zářením pro vytápění a žárovka s UVB zářením pro přidání denního spektra. V teráriu existují dva režimy osvětlení (denní a noční), jejichž změna je řízena spínáním relé. [10] Vypnutím UVA žárovky je dosaženo snížení teploty v průběhu noci, nicméně v práci není zajištěna kontrola této noční teploty a jako cíl zůstává teplota z denního režimu. Teplota tak může v závislosti na vnějším nestabilní. Řešením vysoké teploty by mohlo být přidání druhé proměnné pro ovládání ventilátorů v nočním režimu. Stále by však chyběla možnost pro navýšení teploty v případě jejího přílišného poklesu.

V práci není zmíněna kontrola gradientu teploty napříč teráriem, není tedy jisté, do jaké míry by bylo možné rozložit a regulovat teplotu dle požadavků chovaných živočichů.

Také je v dané práci opomíjen vliv ventilátorů na celkovou vlhkost vnitřního prostředí. Lze předpokládat, že bude spolu s teplem docházet i ke ztrátě vlhkosti. Ventilátor tedy záporně ovlivňuje dvě veličiny v teráriu – teplotu a vlhkosti. Při využití vhodných nástrojů, které zajistí nezávislost nárůstu těchto hodnot, lze regulace obou proměnných řešit v rámci vlastních systémů. Lze například použít ultrazvukový mlhovač, který případné ztráty vlhkosti vyrovná.

Tato práce tedy pouze reguluje při denním režimu horní hranici teploty v rámci celého terária a v nočním režimu je patrná absence tepelného zdroje, a tedy i možnost regulace teploty. Pro využití v regulační a řídíce jednotce akvárií a terárií lze pravděpodobně využít pouze poznatků o regulaci horní teplotní hranice pomocí ventilátorů.

Regulace teploty a vlhkosti

Christopher G. Stanton ve své práci *Vivarium Temperature and Humidity Control with Arduino* zveřejněné na webu hackster.io popisuje s vytvořenou řídící jednotku se základem podobným předchozí analyzované práci. Regulace prostředí probíhá ve vertikálně postaveném teráriu, gradient teploty tedy směřuje vzhůru. [11]

Základem správně regulovaného terária je správné umístění čidel. V teráriu se proto nachází dvě teplotní čidla pro měření hladiny nejnižší teploty na dně a nejvyšší teploty v horní oblasti terária. O vytápění se stará výhřevná podložka a tepelná žárovka směřující na sklo terária. Vytápějící prvky jsou napojeny na termostat, který se stará o jejich spínání dle potřeby. Obě dříve zmíněná tepelná čidla nejsou součástí soustavy s termostatem, jsou samostatně připojena a slouží ke kontrole aktuální teploty pro případ dysfunkce termostatu. [11]

Jak sám Stanton zmiňuje, samostatný termostat nevyhovuje všem podmínkám, neboť v případě zvýšení teploty nedokáže tento systém žádným způsobem teplotu znovu snížit. Základem pro snížení teploty je využití několika ventilátorů odvětrávací teplý vzduch mimo prostředí terária. Řízení těchto ventilátorů je závislé na hodnotách monitorovaných teplot prostřednictvím teplotního čidla DS18B20. Toto čidlo umožňuje číst teplotu s rozlišením na 0,5 °C, zdá se tak být dostatečně vyhovujícím pro potřeby terárií, kde se teploty pohybují v rozmezí nejvýše desítek stupňů Celsia a doporučené teploty pro chov plazů a obojživelníků se udávají v jednotkách.

Nainstalované ventilátory je možné ovládat pomocí pulzně šířkové regulace (PWM), což umožňuje regulovat otáčky ventilátorů i pro velmi nízké hodnoty oproti lineární regulaci. Původní Stantonova myšlenka spočívala ve škálování otáček ventilátoru dle naměřených hodnot, což by znamenalo plné využití PWM. Tuto část však kvůli nedostatečnému výkonu zvoleného mikrokontroleru a potřebě refaktorovat již napsaný kód nebyl schopen implementovat. Základem je tedy pouze ověřování horního limitu teploty a ventilátor se po překročení tohoto limitu spustí na plný výkon. Toto řešení se nakonec ukázalo jako dostačující pro základní potřeby terária, nicméně je terárium připraveno o zdroj stálé cirkulace vzduchu. Stanton zároveň upozorňuje, že kombinace termostatu a ventilátorů umožňuje regulaci teploty pouze v případě, že je okolní prostředí chladnější. Tuto podmínku však nelze vždy dodržet. Je tedy třeba myslet i na alternativy pro ochlazení terárií. Zde by se dle Stantona mohlo uplatnit využití Peltierova článku, který by mohl například ochlazovat vodu využívanou pro systém zvlhčovače. Jiným způsobem by mohlo být vytvoření chladící jednotky pomocí Peltierova článku a ventilátoru.

Stanton zde myslí i na změnu vlhkosti, a proto ve své práci popisuje řešení použité pro regulaci vlhkosti. Jako čidlo pro tuto soustavu využil vlhkoměru DHT11 monitorující úroveň vlhkosti a upozorňující na její nízkou úroveň. Pokud vlhkost klesne pod kritickou úroveň, je sytém pro tvorbu vlhkosti spuštěn tak, aby opět nastal optimální stav. Stanton zároveň nedoporučuje pro

regulaci vlhkosti používat časovač, neboť nelze reagovat na vnější vlivy a v nejhorším případě může dojít i k vytopení terária.

Mezi výhody Stantonovy práce lze zařadit možnost regulovat teplotu oběma směry za podmínky nižší okolní teploty. Systém na kontrolu vlhkosti a jejího zvýšení se zdá být dostačující. Vlhkost v teráriu přirozeně klesá vlivem vytápění a působení ventilátorů, pokud i přesto přeroste kritický stav, je možné ji snížit déletrvajícím během ventilátorů.

Nevýhodou je absence denního a nočního režimu, teplota je stabilní během celého dne. Tento nedostatek lze vyřešit například výměnou termostatu za RTC hodiny a přenesením logiky termostatu do vlastního kódu v termostatu s rozlišením denní doby.

2.2 Akvária

2.2.1 Nároky živočichů v akváriích

Filtrace

Filtrace pomáhá udržovat kvalitu vody v akváriích. Existují tři typy filtrace – mechanická, biologická a chemická. Mechanická filtrace pomáhá odstraňovat pevné částice z vody. Biologická filtrace je postavena na kultuře bakterií, které přeměňují amoniak na dusitany a ty poté na dusičnany. Chemická filtrace se stará o odstranění rozpuštěných látek ve vodě. Velmi často je k tomuto účelu využito aktivní uhlí umístěné do akvarijního filtru. [12]

Osvětlení

Podle průvodce *Basic aquarium guide* firmy Hagen je potřeba při nastavení osvětlení dbát na rostliny v akváriu, které vyžadují přibližně 12 hodin světla denně. Ryby lépe reagují na pravidelné změny světla, je tedy doporučeno spouštět osvětlení v pravidelných intervalech pomocí časovače. Náhlé změny světla ryby stresují, ideální je tedy postupné přidávání jasu. Krmení by mělo probíhat nejdřív 30 minut po zapnutí světel a nejpozději 30 minut před jejich vypnutím. [13]

Pro osvětlení akvárií je možné využít mnoho druhů světelných zdrojů od klasických zářivek přes metal-halogenidové výbojky a aktinické žárovky až po LED diody. V závislosti na fauně a flóře akvária může být vhodné využít více zdrojů osvětlení pro simulaci změny světla v průběhu dne. Elektronická kniha *Freshwater Aquarium* radí použít aktinickou žárovku pro simulaci svítání a setmění, případně simulovat svit měsíce pomocí LED diod. Ty mají univerzální použití a v závislosti na typu je lze použít i pro celodenní osvětlení. Oproti

klasickým zářivkám generují podstatně méně tepla, mají nižší energetickou spotřebu a delší životnost. Mohou být navíc nastaveny přesně podle požadavků na změnu osvětlení v průběhu dne. [12]

Teplota

Ryby mají specifické požadavky na teplotu vody v akváriu. Pro její stabilizaci je vhodné použít topného tělesa. Doporučuje se využít tyčové topítko se zabudovaným termostatem. Topné těleso pak automaticky udržuje požadovanou teplotu. Další možností vytápění mohou být tělesa, která se zabudují do dna akvária. Jejich teplo by mělo podpořit růst rostlin. [14]

V závislosti na umístění akvária je nutné zvážit potřebu chlazení vody v akváriu. Teplotu je možné snížit pomocí ventilátorů namířených na vodní hladinu, neboť pohyb vzduchu urychluje tepelnou výměnu. Pomocí ventilátorů je možné snížit teplotu vody v akváriu o několik stupňů Celsia. Další možností je využití termoelektrického chlazení, které funguje na základě Peltierova jevu. Voda protéká skrze zařízení a ochlazená se vrací zpět do akvária. Tento způsob je finančně náročnější jak na pořízení, tak na spotřebu energie. [15]

pH vody

Pro určení kyselosti vody se využívá pH stupnice. Hodnota pH je ovlivňována mnoha různými faktory, mimo jiné tvrdostí vody, rozpuštěnými minerály a obsahem kyslíku ve vodě. Hodnota pH není v přírodě úplně stabilní, mění se s ročním počasím i mírně v průběhu dne. V akváriu je možné udržovat pH v přibližné hodnotě vyhovující chovaným druhům ryb. Změny pH v akváriích mohou způsobovat také rostliny. Při fotosyntéze v průběhu dne zpracovávají oxid uhličitý na kyslík, čímž zvyšují hodnotu pH. V noci pak oxid uhličitý zase vylučují a pH se snižuje. Snižovat pH lze umělým dodáváním CO_2, případně použít chemické přípravky, které mohou i pH zvyšovat. [12]

2.2.2 Současná řešení ovládání akvárií

Sledování stavu akvária za využití AWS a mobilního připojení

Guillermo Perez Guillen popsal ve své práci *Aquarium Monitoring with AWS-Seeed-Soracom* popisuje vytvoření systému pro kontrolu stavu svého akvária pro chov želv. Cílem projektu bylo měřit a informovat o stavu teploty a vlhkosti prostředí, výšky vodní hladiny a vibrace způsobené čerpadlem či samotnými želvami. [16]

Základ projektu tvoří mikrokontroler firmy Seeed, který ovládá periferie a zároveň umožňuje skrze operátora mobilní virtuální sítě nahrávat data do AWS, k nimž lze přistupovat z mobilní

aplikace. Pro měření vlhkosti a teploty vzduchu je použito čidlo DHT11, pro určení výšky vodní hladiny bylo využito ultrazvukového čidla SRF04 a pro detekci vibrací byl zvolen akcelerometr ADXL345. [16]

Využití virtuálního operátora zajišťuje lepší dostupnost dat z akvária nezávislou na Wi-Fi připojení. Toto řešení však zároveň zvyšuje cenu provozu projektu, která se může zvyšovat spolu s objemem posílaných dat. Nevýhodou tohoto řešení je absence obrazového výstupu přímo na místě, data lze získat pouze skrze webovou aplikaci. Uživatel taktéž nemůže naměřené hodnoty nijak ovlivnit.

Felix Smart

Felix je produkt nedávného startupu kanadské společnosti Felix Smart. Základním prvkem je ovládací zařízení o rozměrech 137x285mm, které disponuje 6 USB konektory a 8 zásuvkami pro připojení dalších prvků. Zařízení je ovládáno pomocí speciální mobilní aplikace, lze jej taktéž ovládat skrze hlasového asistenta. Aplikace podporuje systém IFTTT pro nastavení jednoduchých pravidel pro spínání jednotlivých zásuvek. K zařízení lze dokoupit sondy pro měření pH, NH_3, teploty a vodní hladiny a 360° kameru. Ceny se pohybují v rozmezí od 429 dolarů za základní ovladač až po 1179 dolarů za kompletní řešení pro monitorování stavu akvária.

Výhodou tohoto řešení je vysoká univerzálnost. Uživatel může zapojit ovládací prvky do ovladatelných zásuvek a USB konektorů, v zařízení pak nastaví, kdy se má který prvek spouštět. Toto řešení zároveň jako jediné z nalezených využívá podvodní kameru. Ovládací prvky tato firma nenabízí, uživatel je musí dokoupit zvlášť.

IMKS

Česká firma IMK system nabízí modulární systémy na kontrolu pH a Led osvětlení. Systém na řízení pH umožňuje zapojení pH sondy, teploměru a CO_2 ventilu. Druhý systém umožňuje zapojení led osvětlení a jeho konfiguraci. Firma nabízí i možnost zakoupení modulární verzi systému LED osvětlení, ke kterému lze pomocí datového vodiče připojit ovládání pH. Zároveň vyvíjí unifikované webové rozhraní pro ovládání obou systémů. Jsou plánovány další moduly rozšiřující vlastnosti chytrého akvária.

2.3 Specifikace požadavků

Na základě předchozích zjištění lze stanovit vstupní a výstupní funkce regulační a ovládací jednotky a taktéž podmínky pro úspěšné dosažení cíle projektu. Jsou specifikovány požadavky na chování jednotky, regulované veličiny a možnosti, jakými může uživatel jednotku ovládat.

S ohledem na množství měřených a regulovaných veličin není praktické vytvořit jednu univerzální jednotku schopnou dle požadavků uživatele ovládat akvárium nebo terárium. Úspornější variantou se zdá řešení v podobě menších jednotek vytvořených tak, aby byly schopné plně ovládat pouze jeden druh vivária. Tyto jednotky budou v rámci jednoho místa řízeny tzv. centrální jednotkou, která jediná bude mít přístup do vnější sítě.

2.3.1 Funkční požadavky

V rámci řízení a regulace terária je požadováno, aby jednotka dovolovala

- Měřit a regulovat teplotu v teráriu
- Měřit a regulovat vlhkost v teráriu
- Ovládání osvětlení včetně časování intervalu spouštění
- Doplňovat vodu do terária
- Vizuální kontrolu
- Možnost zapojit pouze část periferií
- Zobrazení údajů na zařízení

V rámci řízení a regulace akvária musí jednotka umožnit:

- Měřit a regulovat teplotu vody v akváriu
- Detekovat výšku vodní hladiny a v případě nutnosti ji dorovnat
- Ovládat osvětlení včetně časování intervalu spouštění
- Manuální spuštění PH sondy
- Automatické krmení včetně volby časování
- Vizuální kontrolu
- Zobrazení údajů na zařízení

- Možnost zapojit jen část periferií pro vymezení činností
- Lokální uložení nastavení uživatele

Uživatel je schopen:

- Přihlásit se do aplikace
- Přidat a odebrat svá vivária

  Získat informaci o stavu přidaných vivárií
- Získat informaci o kritickém stavu vivária v reálném čase
- Získat informace o vývoji stavu vivária za určitý časový úsek
- Nastavit hodnoty regulovaných veličin

2.3.2 Nefunkční požadavky

Aby bylo možné zajistit správnou funkcionalitu všech prvků výsledného systému, je třeba definovat požadavky těchto prvků:

Aplikace

- Použití frameworku Flutter
- Vytvoření příkladu použití
- Zabezpečená komunikace s ovládací regulační jednotkou
- Čitelný a okomentovaný zdrojový kód

Regulační a ovládací jednotka

- Použití jazyka Wiring
- Čitelný a okomentovaný zdrojový kód
- Kód musí umožnit modularitu periferií
- Zabezpečené odesílání dat vivária uživateli

2.4 Scénáře

Realizace a následná implementace kódu funkčních prvků se odvíjí od základních uživatelských scénářů. V rámci každého scénáře je popsán algoritmus, funkční požadavky a předpoklady, které musí být splněny.

2.4.1 Přihlášení uživatele do aplikace

Popis

Po spuštění aplikace bude uživatel vyznán k přihlášení pomocí svého Google účtu, případně pomocí e-mailu a hesla. Pokud se jedná o nové přihlášení uživatele, je vytvořen nový záznam v databázi.

Funkční požadavky

1. Implementace přihlašovací obrazovky
2. Implementace metod pro registraci a přihlášení
3. Metoda pro vytvoření nového uživatele v databázi

Předpoklady

- Knihovna *google_sign_in* pro přihlášení prostřednictvím Google a knihovna Firebase_auth pro zaslání přihlašovacích údajů Flutter.

2.4.2 Přidání vivária

Popis

Nové vivárium má v defaultním nastavení zapnutý Bluetooth modul a je tedy připraveno pro prvotní přihlášení uživatele. Uživatel si zobrazí seznam dostupných zařízení a klikne na plovoucí tlačítko pro přidání dalšího zařízení.

Aplikace zkontroluje stav Bluetooth a přidělená oprávnění pro lokaci. Pokud není některá z podmínek navázání Bluetooth komunikace splněna, je uživatel vyzván k udělení oprávnění či zapnutí Bluetooth na zařízení.

Následně je uživateli zobrazen seznam dostupných Bluetooth zařízení. Nové vivárium má jasně rozpoznatelný název „Vivarium service“. Uživatel si v seznamu vybere požadované vivárium a posléze je přesunut na stránku pro jeho přidání. Zároveň je na pozadí aplikace spuštěno ověřování vybraného Bluetooth zařízení.

Nejprve se aplikace pokusí provést párování se zařízením, které pro tuto činnost zobrazí nově vygenerovaný párovací pin na své obrazovce. Po dokončení párování se aplikace zeptá, zda již nebylo zařízení přiřazeno jakémukoliv uživateli. Pokud je již zařízení přiřazeno, je proces přidání zrušen.

Pokud zařízení dosud nebylo přiřazeno, může uživatel zadat nový název zařízení a přihlašovací údaje k Wi-Fi, ke které se poté bude zařízení vivárium přihlašovat. Po potvrzení dojde na pozadí aplikace k uložení nových dat do Bluetooth zařízení a zároveň nahrání nového zařízení do databáze. Pokud kdykoliv v průběhu procesu přiřazení nastane chyba, jsou veškeré kroky vráceny zpět a uživateli zobrazena příslušná hláška.

Po úspěšném přidání je Bluetooth zařízení odpojeno a uživatel přesunut na stránku přehledu zařízení.

Funkční požadavky

1. Implementace rozhraní pro vybrání Bluetooth zařízení
2. Implementace rozhraní pro vytvoření nové regulační a ovládací jednotky
3. Uložení nových údajů do databáze
4. Implementace komunikace mezi mobilní aplikací a zařízením pro potřeby nastavení Bluetooth modulu

Předpoklady

- Uživatel je přihlášen
- Knihovna pro práci s Bluetooth
- Vivárium je zapojeno do elektrické sítě
- Uživatel se v aplikaci nachází na přehledu vlastních zařízení

2.4.3 Nastavení vivária online

Popis

Uživatel má možnost měnit nastavení svého vivária. Jedná se například o čas svícení nebo krmení, požadovanou teplotu a vlhkost. Uživatel přejde na záložku vivária, které si přeje nastavit. Aplikace automaticky ukládá změny aktualizace do databáze.

Funkční požadavky

1. Implementace rozhraní pro nastavení
2. Metoda pro aktualizaci databáze

Předpoklady

- Uživatel přidal zařízení
- Uživatel je přihlášen

2.4.4 Nastavení vivária při Bluetooth spojení

Popis

Pokud se vivárium nachází v oblasti bez Wi-Fi připojení, je možné jej nastavit napřímo pomocí Bluetooth spojení. V panelu zařízení klikne možnost připojení k zařízení. V takovém případě nejsou data do zařízení streamována z databáze, ale skrze Bluetooth. Jakékoliv změny nastavení v aplikaci se tak kromě databáze nahrávají přímo do zařízení. Změny jsou prováděny okamžitě, projeví se po přerušení Bluetooth spojení.

Funkční požadavky

1. Metoda pro získání dat z databáze
2. Metoda pro navázání komunikace s regulační a ovládací jednotkou
3. Metody pro komunikaci s regulační a ovládací jednotkou
4. Implementace rozhraní pro nastavení jednotek prostřednictví Bluetooth

Předpoklady

- Uživatel přidal zařízení
- Uživatel je přihlášen

2.4.5 Upozornění na stav akvária

Popis

Ve viváriu mohou nastat kritické situace vyžadující interakci uživatele. Může například dojít k rychlému snížení vodní hladiny, zvýšení teploty nebo změně pH vody. V takových případech regulační a ovládací jednotka upozorní uživatele. Pro upozornění je potřeba mít vivárium připojené k Wi-Fi síti.

Každý modul pravidelně kontroluje svůj stav a porovnává jej s nastavením prahových hodnot. Jakmile dojde k jejich překročení, inicializuje modul odeslání notifikace na zařízení uživatele. Zařízení notifikaci zachytí a v případě, že je aplikace na pozadí (background) či vypnutá (terminated), zobrazí se notifikace ve stavovém panelu zařízení. Pokud má uživatel aplikaci spouštěnou na popředí, dojde k zobrazení informace v zařízení.

Funkční požadavky

1. Detekce kritických stavů
2. Implementace procesu odeslání notifikace
3. Implementace zachycení a zobrazení notifikace v mobilním zařízení

Předpoklady

- Uživatel povolil zobrazení notifikací
- Aplikace běží na pozadí
- Nastane kritický stav
- Vivárium je připojené k Wi-Fi síti.

2.4.6 Zobrazení historie

Popis

Pro zjištění vývoje stavu regulační a ovládací jednotky je potřeba ukládat předchozí hodnoty do databáze. Vivárium se v pravidelných intervalech dotazuje stavu jednotlivých modulů a následně odesílá získaná data do databáze. Aplikace poté streamuje obsah posledních několik stovek záznamů, které zobrazuje v podobě grafů pro snadné informování uživatele.

Funkční požadavky

1. Implementovat nahrávání dat do databáze na straně kontroléru
2. Implementovat zpracování dat a následné zobrazení uživateli
3. Vytvořit vhodnou strukturu v databázi

Předpoklady

- Vivárium je připojeno k internetu
- V aplikaci je zobrazen přehled vivária

2.5 Zvažované technologie

S ohledem k rozsahu práce a nárokům na vyšší výpočetní výkon mikrokontroleru je nutné zvolit odpovídající hardware. Pokud je přihlédnuto i k pořizovací ceně, nabízejí se tři možnosti: Arduino, Particle a ESP. Každá z těchto technologií byla analyzována a zároveň došlo k vytvoření prototypu regulační a ovládací jednotky pro ověření proveditelnosti implementace finálního kódu.

2.5.1 Arduino

Arduino je open-source elektronická platforma založena na lehce použitelném hardwaru a softwaru. Každá deska obsahuje mikrokontroler, který lze naprogramovat nahráním sady instrukcí. Arduino je pomocí pinů schopné přijímat vstupní signály a podle nahraného programu převést vstup na odpovídající výstup. Pro programování je využit stejnojmenný programovací jazyk Arduino založený na jazyku Wiring. [17]

Arduino Mega2560

Arduino nabízí širokou škálu desek s různým typem mikrokontrolerů. Mezi základní desky patří Arduino UNO s mikrokontrolerem ATmega328P a 14 programovatelnými piny nebo Arduino NANO s mikrokontrolerem ATmega328 a 22 programovatelnými piny. [18]

Deska regulační a ovládací jednotky bude ovládat velké spektrum vstupních a výstupních periferií a základní typy Arduino desek nemají dostatečný počet pinů. Z toho důvodu bylo zvažováno k použití desky MEGA s 54 piny. Oproti desce UNO má MEGA též větší kapacitu flash paměti pro vlastní aplikaci. S ohledem na množství použitých knihoven by bylo zapotřebí složité optimalizace knihoven pro snížení velikosti kódu pod 31.5 kB.

V rámci snížení nákladů dle zadání této práce by byl použit klon firmy RobotDyn. Využívá stejného mikrokontroleru ATmega2560 jako originální verze Arduina. Rozdíl je v převodníku z USB na sériovou komunikaci. Zatímco originál využívá programovatelný ATmega16U2, klon využívá převodník CH340G bez možnosti jej dále programovat. V této práci není programování druhého mikrokontroleru na desce potřeba. Ve většině běžných projektů ani programování ATmega16U2 není zapotřebí. Dalším rozdílem je využitý typ konektoru. Originály Arduino využívají USB B konektor, klon lze napájet a programovat prostřednictvím micro USB konektoru.

V tabulce jsou zobrazeny technické parametry klonu i originálu MEGA a pro porovnání též i základní verze UNO. Údaje byly převzaty ze stránky výrobce v případě Arduina a ze stránky

oficiálního distributora pro Českou republiku v případě klonu. Rozdíl v povoleném proudu na pinu lze vysvětlit tím, že Arduino uvádí doporučený maximální proud, zatímco distributor uvádí limitní maximální proud trvalé zátěže.

Deska	**Arduino UNO**	**Arduino MEGA**	**RobotDyn MEGA**
Mikrokontroler	ATmega328P	ATmega2560	ATmega2560
Frekvence krystalu (MHz)	16	16	16
Flash paměť (kB)	32 (0,5 kB pro bootloader)	256(8 kB pro bootloader)	256
SRAM (kB)	2	8	8
EEPROM (kB)	1	4	4
USB – Serial Převodník	ATmega16U2	ATmega16U2	CH340G
USB konektor	USB B	USB B	Micro USB
Vstupy a Výstupy			
Digitální I/O	22	54	54
PWM piny	6	15	14
Analogové Vstupy	6	16	16
DC Proud na pin (mA)	20	20	40
Vstupní napětí (V)	7-12	7-12 (doporučeno)	6-12 (doporučeno)
5 V max Proud (mA)		800	800
3.3 V max Proud (mA)		180	180
Váha (g)	25	37	45
Cena u výrobce	20 € (540 Kč)	35 € (945 Kč)	8,26 $ (200 Kč)

Tabulka 1 - Specifikace mikrokontrolerových desek Arduino

I²C sběrnice

Arduino umožňuje pro připojení některých periferií využít sběrnici I2C. Tato sběrnice umožňuje 7bitové adresování, kdy má každé ze zařízení svou specifickou adresu. Je tak možné na jednu sběrnici připojit až 128 různých zařízení. Jsou zde však určitá omezení v podobě parazitní kapacity linek, a tak je celkový počet připojených zařízení nižší. Pro připojení ke sběrnici je potřeba dvou pinů, SDA pro připojení k datovému vodiči a SCL pro připojení k vodiči hodinovému. Sběrnice funguje na principu master/slave, kdy master (řídící prvek) vyšle zprávu pro zařízení na určité adrese a podle potřeby počká na odpověď řízeného prvku. [19]

2.5.2 Particle

Particle je spolehlivá a bezpečná platforma IoT zařízení umožňující rychlé a jednoduché budování IoT řešení. Do portfolia hardwarových produktů Particle se řadí:

- Argon (Wi-Fi, Bluetooth)
- Boron (Mobilní síť)
- Photon (Wi-Fi)

Zařízení Particle jsou v Evropě hůře dostupná. V České republice je jen jeden oficiální distributor, který nabízí pouze Particle Photon. V rámci této práce bude tedy využit Particle Photon, přičemž Bluetooth funkcionalitu dodá klon HM-10 modulu, konkrétně AT-09 s řídícím čipem CC2541. Tato varianta dokonce snižuje náklady oproti použití Argonu.

Particle Photon

Particle Photon je malé IoT zařízení a stejně jako Arduino jej lze naprogramovat pro širokou škálu aplikací. Deska je založena na procesoru STM32F205RGY6 ARM Cortex M3 s 32-bit architekturou a její součástí je také Wi-Fi modul. Mikrokontroler disponuje 1 MB flash pamětí, jak je ale patrné z následující tabulky, většina paměti je vyhrazená pro jiné služby. Polovina flash paměti slouží k uložení firmwaru firmy Particle, který je rozdělen na dvě samostatné části. OTA (Over the Air) záloha slouží pro aktualizace uživatelského kódu přes síťové spojení. Kód je nahrán do OTA zálohy a po restartu zařízení dojde k přepisu původního kódu. OTA aktualizace Device OS funguje na stejném principu. Každá z části systému má ale 256 kB, pro aktualizace je tedy potřeba využít části paměti pro tovární i OTA zálohu. Pro vlastní kód uživatele je vyhrazeno 128 kB.

Pokud je velikost paměti pro uživatele nedostačující, je možné obětovat oblast paměti pro OTA aktualizace a využít ji pro zvýšení paměti. Toto řešení ale není oficiálně podporováno a jeho aplikací uživatel přijde o jednu z hlavních výhod této mikrokontrolérové desky – vzdálená aktualizace přes síť. Pro jakékoliv aktualizace tak bude potřeba zapojit desku do počítače a nahrát nový firmware a kód ručně.

Vyhrazeno pro	Využitá kapacita
Bootloader	16 kB
DCT1 (Wi-Fi přihlašovací údaje, uložené klíče, výrobní údaje, systémové příznaky)	16 kB
DCT2 (swap oblast pro DCT1)	16 kB
EEPROM1	16 kB
EEPROM2	16 kB
Device OS (Wi-Fi)	256 kB
Device OS (Platforma a služby)	256 kB
Tovární záloha	128 kB
OTA záloha	128 kB
Aplikační software	128 kB

Tabulka 2 - Rozdělení paměti Particle Photon

Device OS

Device OS je nízkoúrovňový firmware, který podporuje základní funkce Particle zařízení. Lze jej považovat za operační systém pro vestavěný hardware zařízení. Particle Device OS je zodpovědný především za tyto 4 oblasti:

- Zabezpečenou komunikaci mezi zařízením a Particle cloudem. Zajišťuje, že jakákoliv komunikace je ověřená a zašifrovaná. V případě Wi-Fi zařízení (Photon, P1, Core) je používán CoAP přes TCP spojení s šifrováním AES a session klíčem. Mobilní zařízení (Electron) používají CoAP s DTLS přes UDP. Oba typy zařízení se s cloudem navzájem ověří pomocí RSA veřejného a privátního klíče.
- Poskytuje jediné sjednocené rozhraní bez ohledu na hardwarovou architekturu. Napsaný kód tak může běžet na kterémkoliv Particle zařízení bez potřeby provádět úpravy.
- Vystavuje API, které lze použít při psaní softwaru pro daná zařízení
- Umožňuje bezdrátovou aktualizaci aplikačního softwaru.

Webhooky

Webhooky představují jednoduchou cestu, kterou lze posílat data z Particle zařízení do jiných aplikací a služeb na internetu. Pomocí webhooků lze ukládat data do databáze, zobrazit data ze senzorů nebo třeba spustit nějakou vzdálenou funkci.

Každé zařízení může vytvořit nějakou událost nebo se přihlásit k odběru určité události, na kterou posléze reaguje. Při vytvoření webhooku je určena specifická událost, na kterou má reagovat. Jakmile zařízení vygeneruje danou událost, webhook odešle webový požadavek na

dříve definovanou URL adresu. Tento požadavek může obsahovat informace o vygenerované události.

Webhook pro Particle cloud podporují metody typu POST, GET, PUT a DELETE. Pokud požadavek webhooku vrací nějaká data, může se zařízení přihlásit k odběru odpovědi.

Komunikace

Device OS podporuje několik různých způsobů, jak může zařízení komunikovat přes cloud s jinými službami a naopak. Základní možnost komunikace probíhá pomocí událostí, zároveň však cloud umožňuje vystavení veřejných REST API pro vzdálené ovládání zařízení.

Přehled funkcí:

- **Particle.publish** – umožňuje poslat event (událost) ze zařízení do cloudu, z cloudu do zařízení nebo mezi jednotlivými zařízeními. Tato metoda se nejvíce uplatní v odesílání dat ze zařízení do cloudu, kde mohou být nadefinované webhooky pro určité události.
- **Particle.subscribe** – přihlásí zařízení k odběru události z jiných zařízení nebo cloudu. Jakmile je událost zaznamenána, dojde v kódu mikrokontroleru k přerušení a zavolá se funkce přiřazená k této události.
- **Particle.variable** – nastavení deklarované proměnné pro Particle cloud umožňuje cloudu zavolat request pro získání hodnoty této proměnné. Zatímco metoda *publish* posílá data s každým svým zavoláním, při použití *variable* je hodnota proměnné stále uložena v zařízení a posílá se pouze na dotaz.

  Nastavení *variable* má i své omezení. Pokud je zařízení offline, nelze získat ani poslední hodnotu proměnné před přechodem do offline režimu.
- **Particle.function –** umožňuje cloudu posílat příkazy jednotlivým zařízením. Slouží především pro ovládání zařízení prostřednictvím cloudu.

2.6 Zvolené technologie

2.6.1 ESP32

ESP32 je dvoujádrový systém založený na Harvardské architektuře vyvíjen firmou Espressif Systems. Systém vychází ze svého předchůdce ESP2866 a přináší řadu vylepšení v podobě většího výkonu, zvýšené kapacity paměti, přesnějšího převodníku analogového signálu na digitální, a především přidání podpory pro Bluetooth. [20]

Systém je postaven nad open-source operačním systémem pro mikrokontrolery FreeRTOS, který přidává další řadu možnosti aplikace a zajišťuje správný běh nahraného softwaru. FreeRTOS se může starat o správu paměti a bezpečné operace napříč různými vlákny programu. Další funkcionalitou je možnost nahrání nového firmwaru pomocí OTA aktualizace.

Existuje několik verzí mikrokontroleru ESP32, s ohledem na snížení nákladů se nejvhodnějším typem zdá využití ESP32-WROOM-32, jehož cena začíná na 3 eurech u zahraničních prodejců. Díky podpoře Bluetooth a Wi-Fi taktéž odpadá nutnost pořízení komponent pro tato připojení, jak by tomu bylo v případě použití mikrokontroleru Arduino Mega2560.

ESP32-WROOM-32

ESP32-WROOM-32 je mikrokontrolérová deska s čipem ESP32-DOWDQ6. Procesor se skládá ze dvou jader, která lze samostatně ovládat, frekvenci procesoru lze nastavit v rozsahu 80 MHz až 240 MHz. V případě potřeby je možné přenechat obsluhu kódu koprocesoru s nízkým výkonem. Díky operačnímu systému FreeRTOS podporuje deska zabezpečené OTA aktualizace firmwaru. V rámci OS využívá ESP vlastní implementaci heap_5 algoritmu pro správu paměti tak, aby nedocházelo k vysoké fragmentaci. [21]

K dispozici jsou 4 MB flash paměti, kterou lze rozdělit dle potřeby na oddíly pro program, aktualizace nebo energeticky nezávislou paměť (NVM).

2.6.2 Flutter

Aby bylo možné regulační jednotku ovládat, je třeba navrhnout a naprogramovat aplikaci. Aplikace by měla podporovat mobilní zařízení i webové rozhraní. Aby nebylo nutné psát aplikaci pro každou platformu zvlášť, je vhodné využít některý z již existujících multiplatformních frameworků, které dovolují programátorovi napsat jeden kód a následně jej převedou do nativních kódu pro dané platformy.

Příklady frameworků

Mezi multiplatformní frameworky lze zařadit React Native, Ionic a Flutter. Úkolem těchto frameworku je umožnit vývojářům vytvářet aplikace, které se chovají a vypadají nativně na různých platformách.

React Native je vyvíjen firmou Facebook. Jedná se o open-source framework založeném na javaSriptové knihovně React. Aplikace se píší kombinací JavaScriptu a XML známé jako JSX.

React Native využívá můstek, který se stará o volání nativních API pro vykreslení. Objective-C u iOS a Java u Androidu. [22]

Ionic Framework je framework pro tvorbu mobilních aplikací za využití webových technologií (HTML, CSS, JavaScript) s podporou dalších populárních frameworků a knihoven (Angular, React) [23]

Flutter je relativně nový framework pro tvorbu výkonných aplikací nejen pro Android a iOS, ale přidává podporu i pro web i desktop za stálého použití pouze jednoho kódu. [24]

Flutter

Alfa verze Flutteru vyšla v květnu 2017. V porovnání s React Native z roku 2015 a Ionic Framework z roku 2013 se tak řadí mezi nejmladší multiplatformní frameworky. I přes tento fakt se Flutter stal na trhu vývoje aplikací velmi populárním. Používá jej například čínská společnost Alibaba Group, aplikace Google Ads, GreenTea nebo Stadia. V rámci této práce byl Flutter vybrán kvůli svému potenciálu a možnosti snadné integrace s dalším funkčním prvkem – Google Firebase. [22]

Aplikace se píší v Dartu, ryze objektově orientovaném programovacím jazyku založeném na syntaxi Javy. Stejně jako Flutter je i Dart vyvíjený firmou Google. Dart lze kompilovat AOT (ahead of time) i JIT (Just in Time) kompilery. AOT kompilace se používá především pro optimalizaci release verze, kdy dojde ke kompilaci do nativního strojového kódu. Naopak JIT kompilace je využívána především v procesu vývoje. Flutter totiž přidává jednu funkčnost pro prototypování aplikací, takzvaný *hot reload*, který umožňuje vývojáři změnit jen část kódu a následně rekompilovat jen změnou dotčené widgety. [25]

Widgety

Widgety jsou základním kamenem každé aplikace psané ve Flutteru. Každý widget je neměnná deklarace části uživatelského prostředí. Zatímco jiné frameworky od sebe oddělují pohledy, řadiče, grafické rozvržení a další vlastnosti, Flutter má jeden konzistentní objektový model – widget. [24]

Widget umožňuje definovat strukturu, styl nebo rozložení. Hierarchie objektů je založena na kompozici. Každý widget je vnořen do jiného widgetu, rodiče. Vnořený widget dědí vlastnosti svého rodiče. Na vnější události lze reagovat například pokynem pro framework, aby nahradil widget v hierarchii jiným widgetem. Framework porovná nové a staré widgety a účinně aktualizuje uživatelské rozhraní. [24]

Widgety jsou velmi často složeny z několika malých jednoúčelových widgetů, které se dají zkombinovat pro vytvoření určitého efektu. Widget tak může být zodpovědný za rozložení, vykreslování, umístění nebo například velikost. Pokud například vývojář chce, aby byl určitý widget vycentrovaný na střed, potom ho vnoří do widgetu *Center*. [24]

Ve Flutteru se widgety dělí na dva typy – Stateless a Stateful. Stateless je widget, který nemění svůj stav a jeho podoba je dána při zavolání svého konstruktoru. Příkladem je třeba Text. Stateful widget je dynamický, uchovává si svůj vnitřní stav a je schopný reagovat na jeho změnu. Stav Stateful widgetu je uložen v objektu typu State. V tomto objektu jsou reprezentována jak data widgetu, tak i jeho vzhled. Když se změní stav widgetu, State objekt zavolá metodu *setState*, čímž předá pokyn frameworku widget překreslit. Třída State obsahuje kromě stavu widgetu i metodu build, ve které probíhá sestavení daného widgetu s ohledem na jeho stav, může se tak úplně změnit obsah celého widgetu. [26] [24]

2.6.3 Firebase

V rámci této práce je třeba zajistit správu uživatelů (registrace, autentizace uživatelů a autorizace přístupu k jejich datům), úložiště jejich dat včetně možnosti uložit obrázky z vizuálního záznamu regulační a ovládací jednotky a také notifikační systém pro uživatele v případě nenadálé události.

Ideálním způsobem i v kombinaci s využitým frameworkem pro tvorbu aplikací je využití Google Firebase, platformy pro vývoj mobilních aplikací, která pomáhá s její tvorbou a rozšířením. Firebase poskytuje širokou škálu nástrojů, které může vývojář použít. Konkrétně nabízí Firebase 17 produktů, které lze rozdělit do tří kategorií – vývoj, kontrola kvality a rozšiřující produkty. Zároveň též nabízí řadu rozšíření pro aplikace, ty jsou však stále v beta verzi.

Firebase poskytuje backendové služby, SDK a již hotové knihovny uživatelského rozhraní pro ověřování uživatelů v rámci produktu Authentication. Pro ukládání dat je možné použít NoSQL cloudovou databázi Cloud Firestore. Vizuální záznam akvárií a terárií lze ukládat pomocí Cloud Storage. Firebase Cloud Messaging (FCM) umožňuje posílat notifikace do aplikace o změně v databázi. Posledním použitým produktem v této práci jsou takzvané Cloud Functions. Tyto cloudové funkce jsou schopné pracovat s ostatními produkty Firebase. Vývojář tak napíše vlastní kód a nahraje jej také do cloudu.

Firebase Authentication

Aby bylo možné bezpečně uchovávat uživatelská data, je třeba nejprve daného uživatele ověřit. Authentication podporuje ověřování pomocí hesla, telefonního čísla nebo pomocí externích zprostředkovatelů ověřování jako jsou například Google nebo Facebook. [27]

Pro přihlášení uživatelů lze použít FirebaseUI jako kompletní vložené řešení v aplikaci nebo využít Firebase Authentication SDK a manuálně implementovat jednu nebo více metod přihlášení. [27]

Přihlášení probíhá v několika krocích:

- Aplikace získá přihlašovací údaje uživatele. Může se jednat o kombinace e-mail/heslo nebo oAuth token od externího zprostředkovatele
- Aplikace předá přihlašovací údaje SDK
- Backendové služby Google ověří tyto údaje a vrátí odpověď aplikaci spolu s tokenem

Po úspěšném přihlášení uživatele získá aplikace uživatelský token, který slouží k přístupu k základním uživatelským informacím nebo autorizaci přístupu k datům v databázi. [27]

Cloud Firestore

Firestore je NoSQL databáze vhodná pro ukládání a synchronizaci dat na straně klienta i serveru. Aplikace k databázi přistupuje prostřednictvím nativního SDK. Zároveň je Cloud Firestore přístupný například v nativních SDK pro Node.js, Javu, Pythonu, ale i pomocí REST a RPC API.

Data jsou ukládána do dokumentů a ty jsou dále organizovány do kolekcí. Každý dokument obsahuje sadu map typu klíč-hodnota. Hodnota může nabývat různých datových typů od textových řetězců přes listy až po další kolekce.

Pro správu přístupu k datům jsou využívána bezpečnostní pravidla. Syntaxe těchto pravidel umožňuje specifikovat přístupy a operace jak pro celé části databáze, tak pro jednotlivé dokumenty. Pravidla se skládají z příkazu *match*, který specifikuje cestu k dokumentu a následnému příkazu *allow*, který určuje podmínky, za kterých lze určitou činnost provést. Veškeré příkazy match musí ukazovat na dokument. Pokud se pravidla vztahují na všechny dokumenty v kolekci, je možné použít zástupný znak.

Allow se vztahuje na základní operace *read* a *write*, zároveň však umožňuje tyto činnosti rozdělit na specifičtější pravidla. *Read* lze rozdělit na *get* a *list*, *write* se dělí na *create*, *update* a *delete*.

Následující příklad převzatý a upravený z oficiální dokumentace Firebase ukazuje způsob implementace bezpečnostních pravidel. Označení služby *cloud.firestore* zabraňuje záměny bezpečnostních pravidel pro Firestore s jinými pravidly, například pro Cloud Storage. V zobrazeném příkladu je využit zástupný znak v podobě *{city}*, bezpečnostní pravidla se tedy vztahují na jakýkoliv dokument v kolekci *cities*.

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /cities/{city} {
      // Použije se při žádosti o čtení jednoho dokumentu
      allow get: if <condition>;

      // Použije se při dotazech (queries) a žádostech o čtení kolekcí
      allow list: if <condition>;
    }

    match /cities/{city} {
      // Použije se při vytvoření nového dokumentu
      allow create: if <condition>;

      // Použije se při úpravě stávajícího dokumentu
      allow update: if <condition>;

      // Použije se při mazání dokumentu
      allow delete: if <condition>;
    }
  }
}
```

Kód 1 - Příklad deklarace bezpečnostních pravidel Cloud Storage

Pro tvorbu této práce je velice prospěšná možnost získávat aktuální informace z databáze v reálném čase. Zavoláním metody *onSnapshot* z aplikace získáme aktuální otisk požadovaného dokumentu či dokumentů. Pokaždé, kdy dojde ke změně těchto dokumentů v databázi, je změněn i tento otisk. V kombinaci s využitím Flutteru tak lze lehce nastavit otisk databáze jako stav widgetu a při každé změně překreslit widget podle dostupných dat.

Realtime database

RTDB je stejně jako Firestore database NoSQL databáze sloužící k synchronizaci dat mezi několika zařízeními naráz. Jedná se o předchůdce Firestore a hodí se především pro aplikace s větším množstvím zápisů malého objemu dat. RTDB umožňuje streamování dat pomocí REST API s podporou Server-Sent Events (SSE) neboli serverem odesílaných událostí. Zařízení se prostřednictvím HTTP požadavku přihlásí k odběru událostí a dále probíhá už jen

jednosměrná komunikace od serveru ke klientu do doby, než je spojení přerušeno nebo dojde k vypršení přihlášení. V takovém případě je obnovení znovu navázáno s novým pověřením. [27]

Aktualizace dat se projevují zasláním dvou eventů (PUT a PATCH) a každí událost obsahuje JSON mající dva klíče: *path* a *data*. *Path* ukazuje cestu v databázi relativní k URL zaslaného požadavku, *data* potom obsahují JSON změn pod danou cestou. Pokud je událost typu PUT, znamená to, že může klient nahradit veškerý obsah pod uvedenou cestou nově přijatými daty. PATCH potom indikuje změnu jen u zaslaných dat a ostatní data na straně klienta by měla zůstat nezměněna. [27]

Cloud Storage

Cloud Storage slouží jako úložiště pro uživatelem vygenerovaný obsah, například fotografie, hudbu nebo videa. Firebase SDK pro Cloud Storage zajišťuje bezpečné nahrávání a stahování dat bez ohledu na kvalitu síťové připojení. Pokud se spojení přeruší, je po jeho obnovení pokračováno dál. Data jsou přístupná nejen z aplikace, ale i z jiných cloudových služeb. To je výhodné zejména v případě, kdy je třeba data po nahrání uživatelem ještě nějakým způsobem modifikovat.

Pro správu přístupu k datům je možné stejně jako v případě Firestore využít Cloud Authentication v kombinaci s bezpečnostními pravidly. Rozdíl oproti bezpečnostním pravidlům produktu Firestore přichází s možností validací dat, která uživatel nahrává. Pravidla mohou omezit typ nahraného souboru nebo jeho velikost.

Cloud Functions

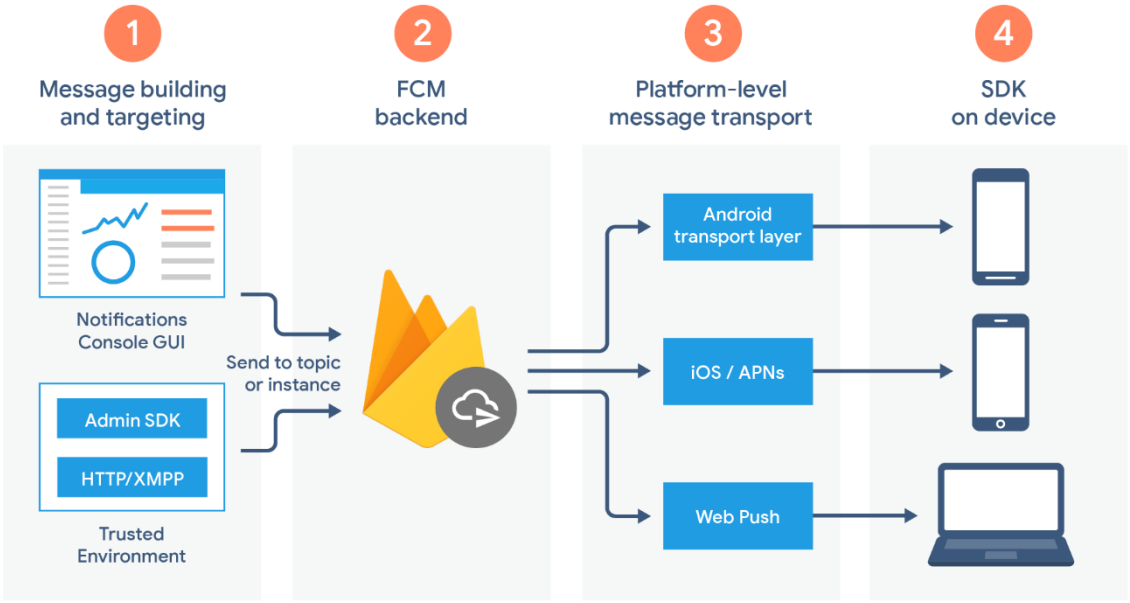
Cloud Functions umožňují vývojáři napsat backendový kód, který by reagoval na události vygenerované produkty Firebase nebo HTTPS dotazy. Kód se nahraje do cloudu a běží v prostředí pod správou Google, takže není třeba spravovat vlastní servery.

Napsané funkce lze spouštět dvěma způsoby. První je přímé volání funkce. To lze provést buď zavoláním HTTP dotazu nebo při využití Firebase SDK přímo z aplikace. Funkce musí mít nastaven poslech na události těchto volání, pro HTTP dotaz se používá metoda *functions.https.onRequest* a pro volání z aplikace slouží *functions.https.onCall*.

Druhým způsobem spouštění funkce jsou reakce na události vzniklé přímo v cloudu Firebase. Funkce může čekat na vytvoření nového či změnu stávajícího dokumentu v Cloud Firestore nebo nahrání nového souboru do Cloud Storage.

Firestore Cloud Messaging

Firebase Cloud Messaging (FCM) je multiplatformní řešení pro spolehlivé zasílání zpráv. Pomocí tohoto produktu lze zasílat notifikace, které se zobrazí uživateli aplikace. Případně lze zaslat data do aplikace a nechat aplikaci rozhodnout, jakým způsobem s daty naloží. Zprávy mohou být poslány jednomu zařízení, skupině zařízení nebo všem zařízením, která jsou přihlášena k odběru určitého tématu. Zprávy je také možno zasílat z aplikace do cloudu.



Obrázek 1 - FCM Architektura

Na obrázku 1 je FCM Architektura z oficiální dokumentace Firebase. Schéma popisuje jednotlivé komponenty účastnící se procesu zasílání zpráv.

1. FCM pro svůj běh potřebuje důvěryhodné prostředí. Je možné využít Cloud Functions, nebo vlastního aplikačního serveru, který vytváří, cílí a odesílá žádosti o zprávy. V rámci této práce bude zprávy odesílat samotné zařízení pomocí servisního účtu.

2. Žádosti o odeslání zprávy je přijímána backendem FCM, který zprávě přiřadí ID a jiná metadata. Tato komponenta se také stará o hromadné zaslání zprávy všem zařízením, která jsou přihlášena k odběru určitého tématu. Zprávy jsou zaslány do transportní vrstvy cílené platformy.

3. Transportní vrstva na úrovni platformy směruje zprávy do cílového zařízení, stará se o doručení a případně použije konfiguraci specifickou pro danou platformu. Android zařízení s Google Play službami používají Android Transport Layer (ATL), pro iOS existuje Apple Push Notification service (APNs) a pro webové aplikace je použit Web Push protokol.
4. Poslední komponentou v architektuře posílání zpráv je FCM SDK v cílovém zařízení. Stará se o zobrazení notifikace, případně o jiné zpracování přijaté zprávy podle toho, zda se aplikace nachází na popředí nebo v pozadí.

3 Design

3.1 Architektura systému

Prvotní návrh architektury systému počítal s využitím mikrokontroleru Arduino Mega2560 jako jádrem ovládací a regulační jednotky. Do tvorby návrhu vstupovala nutnost zajistit pro zařízení zabezpečenou komunikaci pomocí Wi-Fi připojení a taktéž podpora pro Bluetooth. V průběhu vývoje však bylo nalezeno spolehlivější a finančně méně náročné řešení v podobě využití čipu ESP32, které bylo následně implementováno.

3.1.1 Architektura s centrální jednotkou

Navrhovaný systém se skládá z mobilní či webové aplikace, databáze, systémem pro zasílání upozornění, cloudové služby mikrokontroleru Particle Photon, samostatných řídících a regulačních jednotek akvárií a terárií a kamerou.

Systém je s ohledem na bezpečnost navržen tak, aby jediná komunikace mezi řídícími jednotkami a veřejnou síti probíhala pomocí šifrované komunikace mezi mikrokontrolerem firmy Particle a cloudem. Jak již bylo zmíněno v analytické části, Particle využívá pro svou komunikaci mezi mikrokontrolerem a cloudem AES/RSA šifrování a komunikace cloudu s mobilní aplikací je zajištěna pomocí Particle cloud REST API. Kamera v pravidelných intervalech ukládá obraz do Firebase Storage pomocí metody POST protokolu HTTPS s ověřením oAuth 2.

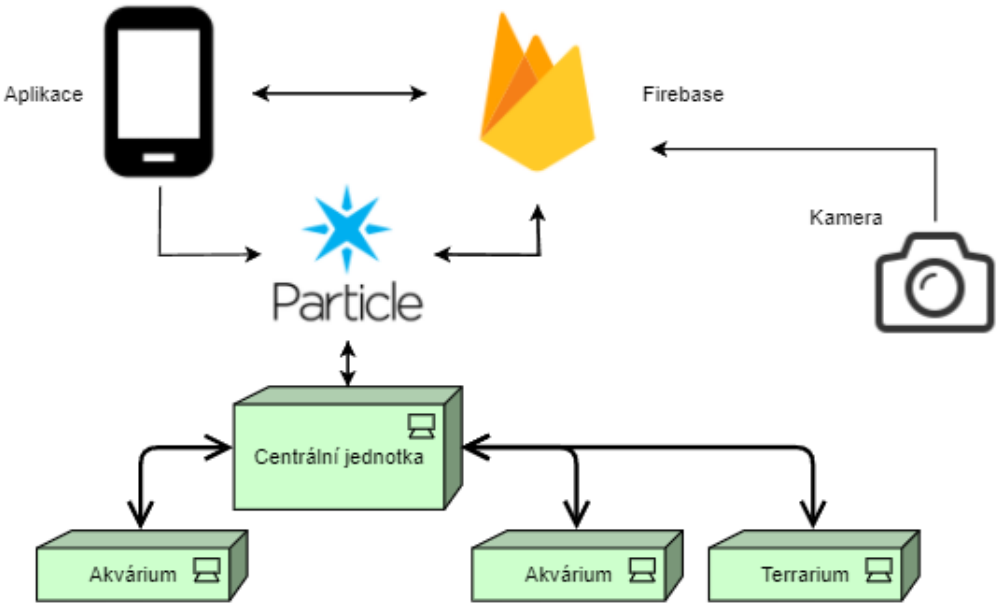


Schéma 1 - Systém s centrální jednotkou

Schéma číslo 1 ukazuje obecný pohled na architekturu systému s využitím centrální jednotky. Jsou zde vidět jednotlivé prvky a komunikační cesty mezi nimi. Zobrazený systém přepokládá, že je uživatel zaregistrován a přiřadil ke svému účtu svá vivária.

3.1.2 Architektura bez centrální jednotky

Navrhovaný systém se skládá z regulační jednotky vivária, aplikace, cloudové služby a samostatné kamery.

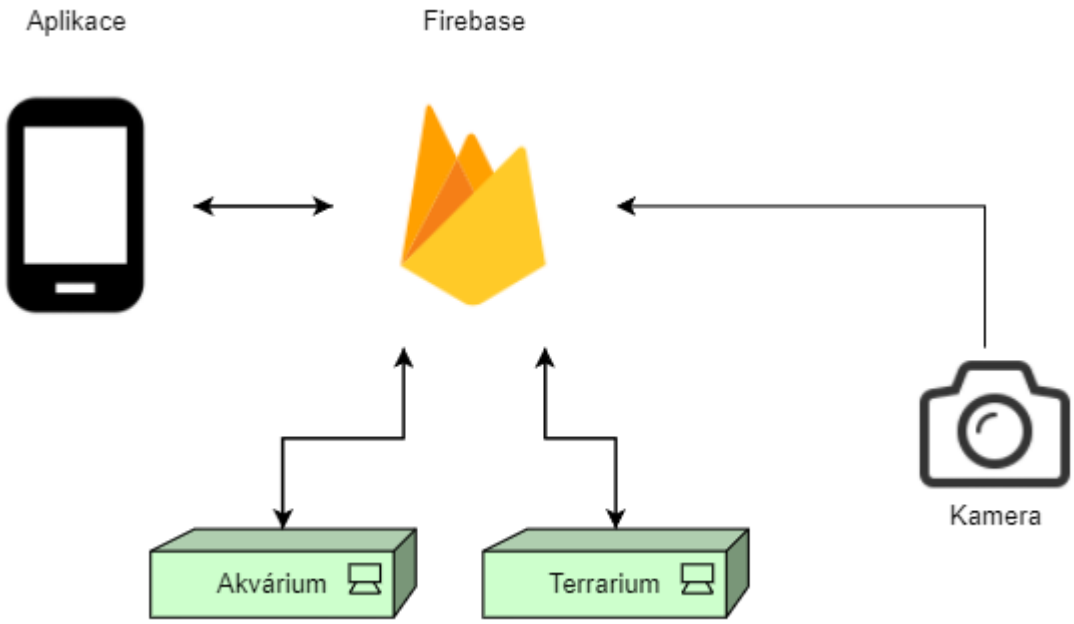


Schéma 2 - Systém bez centrální jednotky

Hlavní komunikace probíhá prostřednictvím Wi-Fi spojení mezi Firebase cloudem a ostatními prvky systému. Základem řídící jednotky vivária je mikrokontroler ESP32, pro který existuje Firebase knihovna pro zabezpečenou komunikaci s Firebase cloudem. Tato knihovna umožňuje streamovat změny dat v databázi pomocí automatických aktualizací ze serveru prostřednictvím HTTP spojením díky SSE technologii. Tento způsob aktualizací je možné použít pro Realtime databázi, novější Firestore databáze podporuje pouze vzdálené volání procedur RPC, což Firebase knihovna pro ESP32 neumožňuje.

V případě potřeby může vivárium pomocí FCM zaslat notifikaci na aplikaci uživatele. Kamera je samostatným prvkem, který lze přiřadit k jakékoliv řídící jednotce.

Systém byl navržen tak, aby umožnil běh vivária v co největším rozsahu bez internetového připojení. Návrh umožňuje nastavit vivárium a získat informace o aktuálním stavu jen prostřednictvím Bluetooth.

3.2 Regulační a ovládací jednotka vivária

Vytvoření regulační a ovládací jednotky je primárním cílem této práce. Tato jednotka má na starost kontrolu stavu vivária, řídí jeho příslušenství a dokáže regulovat charakteristiku vnitřního prostředí.

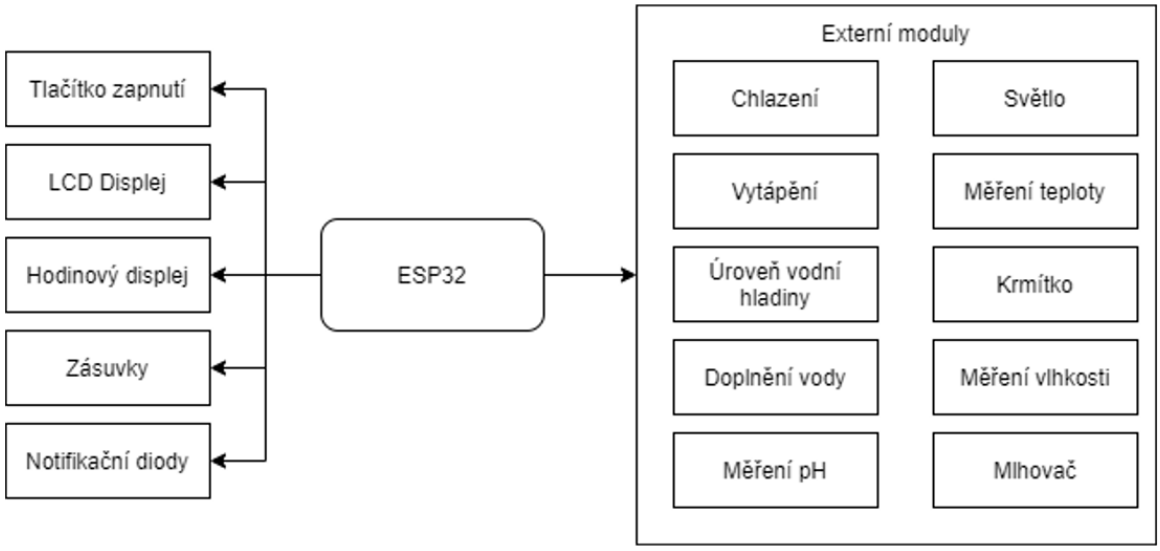


Schéma 3 - Schéma periferií regulační a ovládací jednotky

Blokové schéma představuje připojené periferie k regulační a ovládací jednotce akvária. Je třeba provést analýzu jednotlivých částí a vybrat konkrétní modul s ohledem na cenu a kvalitu.

3.2.1 Interní moduly

Zobrazení informací

Pro zobrazení aktuálního stavu řídící a regulační jednotky je využit 16x2 LCD displej s připojením na sběrnici I2C a k zobrazení času pak hodinový displej s obvodem TM 1637.

Pro ovládání LCD displeje mikrokontrolerem slouží knihovna *LiquidCrystal_I2C*, pomocí které lze ovládat libovolný LCD displej po I2C sběrnici. Při deklaraci LiquidCrystal_I2C objektu se pak definuje nastavená adresa pro I2C sběrnici, počet řádků a počet znaků na řádek. Knihovna poté umožňuje aktualizaci zobrazeného textu po jednotlivých znacích nebo

řetězcích. Podporuje také několik efektů, například posouvání zobrazené zprávy, zobrazení kurzoru nebo jeho posunutí.

Následující kód ukazuje, jakým způsobem lze na displeji vypsat text „Vivarium Control Unit“ na dva řádky.

```
LiquidCrystal_I2C lcd(DISPLAY_ADDRESS, 16, 2

void setup(){
   lcd.begin();                // inicializace
   lcd.setCursor(4, 0);        // sloupec, řádek
   lcd.print("Vivarium");
   lcd.setCursor(0,1);
   lcd.print("Control Unit");
}
```

Kód 2 - Ukázka kódu pro knihovnu LiquidCrystal_I2C

Na displej se v jeden moment nevejdou veškeré informace o stavu vivária, zobrazený text se tedy bude v pravidelném intervalu střídat.

Zobrazení času

Aby měl uživatel přehled, se kterým časovým údajem ovládací jednotka operuje, bude aktuální čas zobrazen na hodinovém displeji složeném z bloku 4 sedmisegmentových displejů. Konkrétně bude využit hodinový displej TM1637 od firmy RobotDyn.

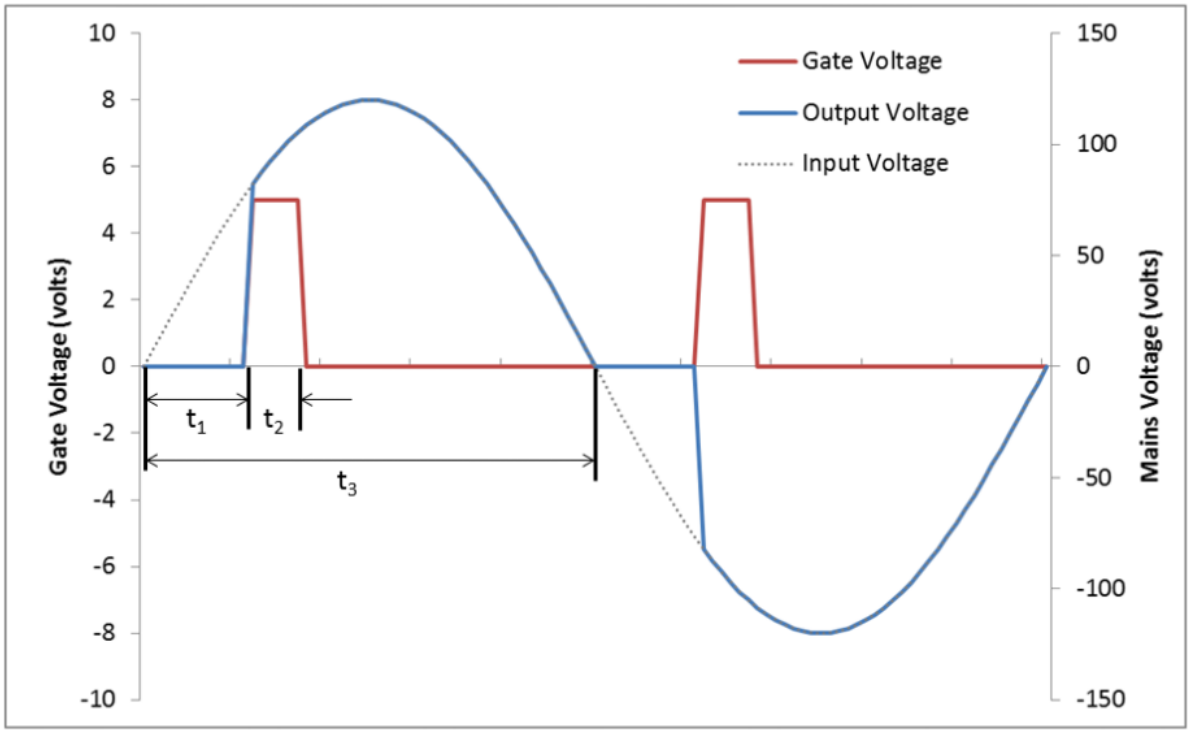
Ovládání zásuvek

Ovládací jednotka disponuje 3 zásuvkami se střídavým napětím 230 V pro zapojení dalších periferií. Spínání dvou zásuvek umožňuje dvoukanálový SSR modul, které je možné ovládat pomocí logiky na 5 V. Uživatel může nechat zásuvku stále zapnutou, případně ji vzdáleně pomocí aplikace vypnout. Výstupní proud na jednom kanále je omezen na 2 A.

Třetí zásuvka je ovládána jednokanálovým stmívačem pro ovládání střídavého napětí. Pro ovládání tohoto stmívače lze použít 3,3 V nebo 5 V logiku. Obvody jsou od sebe odděleny optočlenem a pro ovládání AC napětí je využit triak BTA16-600B, který zvládne pravidelnou zátěž 2 ampér a krátkodobě až 5 ampér.

Stmívač funguje na principu detekce průchodu střídavého napětí nulou a regulaci výsledného napětí, jak je znázorněno na následujícím obrázku. Při každém průchodu nulou je vyslán 5 V pulz, který je zachycen mikrokontrolerem a způsobí zavolání funkce přerušení. Tato funkce nastaví časovač na čas $\mathbf{t_1}$, po jehož uplynutí dojde k přivedení napětí na elektrodu gate

triaku (vyznačeno červeně), čímž dojde k sepnutí obvodu a změně výstupního napětí (vyznačeno modrou křivkou). Triak pak zůstává aktivován bez ohledu na přivedené napětí na gate až do příštího průchodu nulou. Interval pulsu přivedeného napětí na gate t_2 je dán minimálními požadavky triaku. Je třeba pohlídat, aby impuls na gate do konce půlvlny sinusoidy skončil, aby bylo možné proces opakovat. Tímto způsobem dochází k odseknutí části vlny střídavého proudu, a tedy i snížení výkonu zařízení. Čím delší interval t_1 je, tím menší výkon zařízení má. Nedoporučuje se snižovat výkon na velmi nízké hodnoty, aby nedošlo k překrytí průchodu nulou intervalem t_2, v takovém případě zůstane po průchodu triak aktivní a výkon bude na 100 %. [28]



Obrázek 2 - Fázové řízení střídavého proudu [28]

Výrobce pro svůj stmívač vytvořil knihovnu, která se o výše popsaný proces detekce průchodu nulou postará sama. V kódu pak lze lehce přepnout výkon zásuvky, vypnout regulaci úplně nebo uhladit změny výkonu tak, aby byl výkon regulován postupně s časovou prodlevou.

Zdroj

Všechny vypsané komponenty nelze napájet pouze ESP, je potřeba vybrat externí zdroj s odpovídajícím výkonem, proto byla vytvořena následující tabulka s přehledem spotřeby všech komponent.

Komponenta	Napětí	Max proud	Výkon	Kusů	Výkon celkem
LCD displej	5 V	200 mA	1 W	1	1 W
Segmentový displej	5 V	220 mA	1,1 W	1	1,1 W
PH sonda	5 V	10 mA	0,05 W	1	0,05 W
Čidlo teploty	3,3 V	1,5 mA	0,00495 W	1	0,00495 W
Ventilátor	12 V	75 mA	0,9 W	4	3,6 W
Senzor hladiny	5 V	15 mA	0,075 W	1	0,075 W
Čerpadlo vody	5 V	300 mA	1,5 W	1	1,5 W
RGB LED dioda	5 V	60 mA	0,3 W	20	6 W
LED dioda	5 V	20 mA	0,1 W	10	1 W
Krokový motor	5 V	130 mA	0,65 W	1	0,65 W
ESP32	3,3 V	500 mA	1,65 W	1	1,65 W

Tabulka 3 - Přehled spotřeby komponent

Požadovaný výkon zdroje pro napájení všech komponent je přibližně 16,6 W. Pro napájení celé ovládací a regulační jednotky tak byl zvolen zdroj s výstupním napětím 12 V s maximálním odběrem 2 A. ESP je možné napájet skrze Vin pin 5 V, proto byl přidán step-down měnič s řídícím obvodem XL4015 o účinnosti do 85-96 procent, který je schopný kontinuální výstupního proudu 3 A bez potřeby přídavného chlazení.

3.2.2 Externí moduly

Měření PH

Pro měření pH vody v akváriu byla zvolena sonda E-201-C spolu s modulem PH-4502C. Modul pracuje s napětím 5 V a umožňuje měřit pH v rozsahu 0 až 14 s maximální odchylkou 0,5 oběma směry. Na modulu se nachází potenciometr pro kalibraci pH. Ke kalibraci pH sondy je potřeba kalibračních pufrů. Druhý potenciometr slouží pro nastavení hraniční hodnoty, po jehož překročení je přivedena logická jednička na jeden z výstupních pinů modulu. Tato funckionalita je vhodná pro kontinuální měření s implementovaným přerušením na některém z pinů mikrokontroleru. Zvolená sonda není vhodná na neustálé měření a při takovém způsobu měření je potřeba ji velmi často kalibrovat. Je proto lepší variantou je udržovat sondu v neutrálním prostředí a testovat pH vody v akváriu manuálně dle potřeby. I v takovém případě bude třeba provádět kalibrace, interval mezi kalibracemi však bude mnohem delší. Sondy se k modulu připojují pomocí BNC konektoru a uživatel tak má možnost nahradit pH sondu za kvalitnější, program proto bude podporovat oba režimy měření.

Měření teploty vody

K měření teploty bylo využito vodotěsné sondy DS18B20 schopné měřit teplotu v rozsahu od -55 °C do 125 °C s přesností ±0,5 °C v rozsahu -10 °C až 85°C. Pro komunikaci s ESP slouží sběrnice OneWire umožňující komunikaci po jednom vodiči.

K implementaci kódu obsluhujícího teplotní čidla je potřeba využít knihovny OneWire a DallasTemperature. Nejprve dojde k vytvoření instance pro sběrnici OneWire na zvoleném pinu a následnému vytvoření instance agregující teplotní čidla. Při startu mikrokontroleru je třeba inicializovat komunikaci po sběrnici. Zavoláním metody *requestTemperatures* dojde k inicializaci měření všech připojených čidel. K teplotám ve stupních Celsia se pak přistupuje metodou *getTempCByIndex*, jejímž parametrem je pozice zapojeného senzoru.

```
OneWire oneWire1(TEMP_1_PIN);
DallasTemperature tempSensor1(&oneWire1);
double temp;
void setup(){
      tempSensor1.begin();
}
void loop(){
      tempSensor1.requestTemperatures();
      temp = tempSensor1.getTempCByIndex(0);
}
```

Kód 3 - Ukázka kódu pro práci s teplotním čidlem

Senzor vodní hladiny

Voda v akváriu se neustále odpařuje, a proto je jednou z vyžadovaných funkcích kontrola vodní hladiny v akváriu. Byly analyzovány 3 možnosti detekce vodní hladiny a následně vybrána nejvhodnější varianta. Prvním způsobem bylo použití bezkontaktního čidla XKC-Y25-V, které stačí umístit z vnější části akvária a nastavit poslech sestupné hrany na výstupním vodiči s přerušením. Ve chvíli, kdy senzor přestane zaznamenávat vodu v akváriu, bude zavolána funkce s dalším kódem pro varování uživatele akvária.

Na podobném principu fungují i plovákové senzory hladiny. Tyto senzory se skládají z pevné a pohyblivé části a připevňují se vně akvária. Pohyblivý plovák obsahuje magnet, pevná část potom kontakt citlivý na magnetické pole. Jakmile se plovák dostane do stejné úrovně jako kontakt pevné části, obvod se sepne a Arduino zavolá funkci přerušení.

Na výšce vodní hladiny závisí i jiné části ovládací a regulační jednotky, proto je důležitá spolehlivost a univerzálnost. Pokud je bezkontaktní čidlo umístěno nekvalitně, může se

uvolnit a vysílat chybné zprávy. Nevýhodou bezkontaktního čidla i plováku je měření pouze jedné úrovně vodní hladiny. Není možné dále zjistit, jak velký pokles hladiny nastal a jestli tím pádem nedochází k úniku vody z akvária. Lepší variantou je měření výšky vodní hladiny pomocí ultrazvukového senzoru vzdálenosti HC-SR04.

Tento senzor umožňuje detekovat překážky na principu vysílání a zpětné detekce signálu o frekvenci 40 kHz. Zapojuje se pomocí 2 napájecích vodičů a 2 vodičů pro vysílání a zachytávání echa. Pracovní rozsah senzoru je 2 až 450 cm s přesností až na 3 mm. Čím kratší je vzdálenost mezi senzorem a překážkou, tím přesnější hodnoty senzor udává. Při správném umístění nad vodní hladinou akvária a po nastavení optimální výšky vodní hladiny pak lze uživatele upozornit na naměřené výchylky.

Chlazení

V některých případech může nastat situace, kdy je potřeba snížit teplotu vodu v akváriu. Proto byla vytvořena chladící jednotka ze 4 ventilátorů napájených 12 V o rozměrech 40x40mm. Ventilátory jsou zapojeny paralelně a pro regulaci jejich výkonu je použit MOSFET modul s tranzistorem IRF520. Při náhlém rozepnutí obvodu se z rozběhnutých ventilátorů může stát zdroj napět a může dojít k napěťové špičce, která může poškodit řídící zařízení. Do obvodu je proto připojena paralelně zpětná dioda v závěrném směru tak, aby byl jakýkoliv přebytek nepoškodil mikrokontroler. Pro bezpečné zapojení 4 ventilátoru byl navržen a vytisknut 3D model, který je součástí přílohy.

ESP32 má periferní zařízení pro kontrolu LED primárně určenou k nastavení intenzity osvětlení, které ale může vytvářet PWM signál i pro jiné účely. Pro ovládání PWM signálů je možné využít 16 kanálů rozdělených do dvou skupin pracujících v jiných rychlostech. Rychlejší mód využívá hardwarové řešení pro dynamickou změnu cyklu, v případě pomalejšího režimu je poté třeba změny cyklu kontrolovat přímo v programu. V případě ovládání ventilátorů je potřeba využít jeden z prvních osmi kanálů pro automatickou změnu cyklu.

Při vyšší frekvenci PWM je slyšet pískání ventilátorů. Postupným snižováním frekvence až na 40 Hz došlo ke snížení hluku vydávaného ventilátory na minimum. Nejprve je potřeba inicializovat frekvenci a rozlišení PWM signálu pro určitý kanál. Následně je přiřazen daný kanál některému pinu. V následující ukázce je rozlišení nastaveno na 8 bitů, tedy kanál může vykazovat délku cyklu od 0 do 255, kdy 0 znamená vypnutí pinu a 255 úplné zapnutí.

```
#define FAN_PIN 25
#define FAN_RESOLUTION 8
#define FAN_FREQUENCY 40
#define FAN_CHANNEL 0

ledcSetup(FAN_CHANNEL, FAN_FREQUENCY, FAN_RESOLUTION);
ledcAttachPin(FAN_PIN, FAN_CHANNEL);
ledcWrite(FAN_CHANNEL, speed);
```

Kód 4 - Inicializace PWM v ESP

Krmítko

Krmení je jednou z řízených funkcí akvária. V této práci bylo zvoleno přidat podporu pro dva typy krmítek. Jelikož design krmítek není součástí této práce, bylo rozhodnuto pro použití některých z již existujících modelů. Typ krmítka pro dávkování vloček vytvořil Jason Hitesman a zveřejnil na serveru Thingiverse pod licencí CC BY-NC-SA 4.0[1]. Jedná se o krmítko poháněné krokovým motorem 28BYJ-48, které otáčí Archimédovým šroubem. Krmítko má zásobník, do kterého se vločky nasypou. Druhým použitým modelem je dávkovač jednotlivých pelet či granulí jehož autorem je Trevor Kerth. Svůj projekt zveřejnil na serveru Instructables také pod licencí CC BY-NC-SA 4.0[2]. Toto krmítko funguje na principu posunu zásobníku a propadu granulí skrze otvor do akvária. Počet krmení je v tomto modelu dán počtem přihrádek na granule. Pokud uživatel po doplnění krmiva tuto skutečnost zaznamená, může dostávat notifikace o zbývajícím počtu krmení.

2BYJ-48 je krokový motor ovládaný sérií elektromagnetických cívek. Na hřídel jsou připevněny magnety a postupným spínáním okolních cívek je vytvářeno magnetické pole, které přitáhne magnet, čímž dojde k otočení hřídele. Pro ovládání motoru slouží řadič ULN2003, do kterého je třeba zapojit 4 ovládací a 2 napájecí piny. Motor má převod o poměru 1/64 s pootočením o 5,625 stupňů. Napájení motoru je možné 5 V.

Pro ovládání je využita knihovna *Stepper*, ovládání je pak jednoduché. Nejprve je potřeba deklarovat instanci objektu *Stepper*, během níž jsou specifikovány ovládací piny a počet kroků motoru k plnému otočení. Ovládací piny jsou v pořadí 1,3,2,4. Výpočet počtu kroků probíhá pomocí vzorce $K = \frac{360}{U} * \frac{1}{P}$, kde K je počet kroků, U značí úhel skoku a P je převodový poměr. Počet kroků u krokového motoru 2BYJ-48 je tak $\frac{360}{5{,}625} * \frac{1}{\frac{1}{64}} = 64 * 64 = 4096$. Poté lze funkcí *setSpeed(long RPM)* nastavit rychlost v počtu otáček za minutu. Samotné otočení

[1] https://www.thingiverse.com/thing:2959685
[2] https://www.instructables.com/id/Betta-Fish-Feeder/

motoru proběhne zavolání metody *step(int pocetKroku)*. Tato metoda je blokující a je proto vhodné volat ji volat častěji ve smyčce *loop* mikrokontroleru po menším počtu kroků než pomocí jednoho volání pro otočení naráz. Pro otočení opačným směrem pošleme metodě *step* zápornou hodnotu.

```
Stepper feeder = Stepper(pocetKroku, IN_1, IN_3, IN_2, IN_4);
feeder.setSpeed(15);
feeder.step(pocetKroku);
```

Kód 5 - Ukázka ovládání krokového motoru

Vytápění

Teplotu v akváriu lze ovládat dvěma způsoby. Jeden spočívá v zapojení topítka s integrovaným termostatem. V současné době se jedná o nejdostupnější variantu vytápění akvárií, kdy se prodávají varianty s rozdílným výkonem pro různé velikosti akvárií. Uživatel si na topítku nastaví manuální teplotu a termostat se sám stará o vypínání a zapínání vytápění. Tato práce s výše uvedenou možností počítá a umožňuje uživateli zapojení topítka s externím ovládáním teploty.

Druhým způsobem je zapojení topného kabelu či jiného topného tělesa. Kvůli absenci termostatu je potřeba výkon těchto topných těles ovládat mikrokontrolerem. V odstavci věnující se ovládání zásuvek (str. 33) byl popsán princip fungování zásuvky napojené na stmívač. Ten lze využít i v případě regulace napájení těchto topných těles.

ESP může teplotu regulovat pomocí změny výstupního výkonu stmívače. K určení konečného výkonu na výstupu je možné využít princip PID regulace. Proporcionálně integračně derivační regulátor je typem řídícího systému se zpětnou vazbou. Základem systému je řídící veličina (požadovaná hodnota) a regulovaná veličina (skutečná hodnota). Regulátor přijme rozdíl mezi řídící a regulovanou veličinou, který se nazývá regulační odchylka. Regulátor poté na základě této odchylky vypočítá hodnotu akční velečiny, kterou aplikuje na regulovanou soustavu tak, aby se rozdíl mezi řídící a regulovanou veličinou snížil. [29]

PID regulátor je složen ze tří složek:

- Proporcionální – výstup z proporcionální složky regulátoru je přímo úměrný regulační odchylce. Čím větší je rozdíl mezi požadovanou a skutečnou hodnotou veličiny, tím více působí na regulovaný systém.

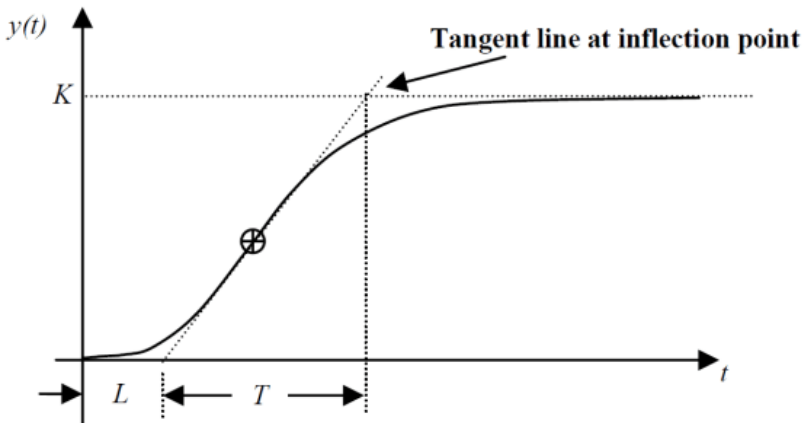
- Integrační – akční veličina je přímo úměrná integrálu regulační odchylky. Integrační složka postupně sčítá chybu v čase a násobí ji definovanou konstantou. Ať je regulační odchylka jakkoliv nízká, nakonec vždy zapůsobí jejich součet a regulátor provede úpravy regulované veličiny.
- Derivační – není samostatně fyzikálně realizovatelná, proto se používá ve složených typech regulátorů typu PD nebo PID. Výstup derivační složky je přímo úměrný derivaci regulační odchylky. Sleduje trend růstu regulované veličiny a dokáže do jisté míry předvídat nastávající změny. [29]

Získanou akční veličinu lze vyjádřit následujícím vztahem:

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dx}$$

kde K_p, K_i, K_d jsou konstanty pro ladění regulátoru a e(t) je regulační odchylka v čase t. [29]

ESP disponuje knihovnou pro PID regulaci, stačí tedy specifikovat ladící konstanty a proměnné pro řídící, řízenou a akční veličinu. Pro jejich určení je možné využít například Ziegler-Nicholsovu metodu přechodové charakteristiky tak, jak ji popisují ve své knize Karl Johan Aström a Richard M. Murray. [30] Metoda je založena na krokové změně výkonu u systému s otevřenou smyčkou. Jsou pozorovány změny měřených hodnot a jejich průběh je aproximován polynomem n-tého stupně.



Graf 1 - Kroková odezva systému s otevřenou smyčkou [31]

Graf 1 ukazuje odezvu systému na změnu vstupu. Je potřeba určit inflexní bod získané funkce a vést jím tečnu, díky které je možné získat zpoždění L a časovou konstantu T. Ty lze využít k výpočtu proporcionálního zisku K_p, integrálního času T_i a derivačního času T_d podle pravidel metody Ziegler-Nicholse popsané v tabulce 4.

Typ kontroléru	K_p	T_i	T_d
P	$\frac{T}{L}$		
PI	$0.9\ \frac{T}{L}$	$\frac{L}{0.3}$	
PID	$1.25\ \frac{T}{L}$	$2L$	$0.5L$

Tabulka 4 - Ziegler-Nicholsova pravidla přechodové charakteristiky

Konstanty K_i a K_d lze poté získat z následujícího vztahu:

- $K_i = \frac{K_p}{T_i}$
- $K_d = K_p * T_d$

Doplňování vody

S výškou vodní hladiny souvisí i možnost doplňování odpařené vody. Jakmile je zaznamenán pokles hladiny, je ztráta vody vykompenzována. Pro tento úkon bylo v této práci využito 5 V ponorné čerpadlo, které přečerpává vodu ze zásobníku vody. Ovládání čerpadla je zajištěno MOSFET modulem s tranzistorem IRF520.

Čerpadlo je spouštěno ve chvíli, kdy je dosažena hraniční hodnota vodní hladiny akvária. Po dorovnání hladiny se čerpadlo zastaví.

Osvětlení

Pro osvětlení byl vybrán LED pásek s RGB diodami WS2812B. Každá z LED diod má vestavěný čip s pamětí 3 bytů pro nastavení intenzity každé ze tří barev. Komunikace po jednokanálovém rozhraní umožňuje adresovat každou z LED diod zvlášť. Spotřeba jedné barvy na LED diodě je 20 mA, každá RGB dioda odebírá 60 mA. Pásek je napájen 5 V.

Existuje několik knihoven pro ovládání LED pásek. Prvotně vybraná knihovna *Adafruit_NeoPixel* však při prvotních testech nesprávně vypínala diody, proto byla pro ovládání použita knihovna *Freenove_ESP32_WS2812*. Při inicializaci je třeba zadat počet pinů, komunikační pin, kanál pro komunikaci s páskem a typ pásku pro kódování barvy. V této práci byl použit typ GRB. Zavoláním funkce *begin* dojde ke konfiguraci knihovny. Funkce *setAllLedsColor(u32 barva)* uloží novou barvu pro všechny diody na pásku a zároveň je zobrazí.

```
#include <Freenove_WS2812_Lib_for_ESP32.h>
#define LED_PIN 26
#define LED_COUNT 20
#define LED_CHANNEL 1

Freenove_ESP32_WS2812 *_strip;

void initializeLed(){
 _strip = new Freenove_ESP32_WS2812(LED_COUNT, LED_PIN, LED_CHANNEL, TYPE_GRB);
 _strip->begin();
 _strip->setAllLedsColor(color);
}
```

Kód 6 - Ukázka použití knihovny Freenove WS2812 Lib for ESP32

Mlhovač

Pro zvýšení vlhkosti v terárium byl zvolen ultrazvukový zvlhčovač pracující s napětím 24 V. Modul je napájen adaptérem zapojeným do 230 V zásuvky. Pro správné fungování tohoto modulu je potřeba zapojit i modul pro měření vlhkosti vzduchu. Uživatel nastaví požadovanou vlhkost a mlhovač porovnává naměřenou vlhkost s požadovanou. Pokud je vlhkost nižší než požadovaná, je mlhovač zapnut.

Teplota a vlhkost vzduchu

Pro měření teploty a vlhkosti je využito DHT 11 modulu pro jehož ovládání slouží stejnojmenná knihovna od firmy Adafruit. Tento modul je možné napájet v rozsahu od 3 do 5.5 V, Teplotu měří v rozsahu 0-50 °C s přesností na 2 °C. Vlhkost je měřena v rozmezí od 20 do 90 % s pětiprocentní odchylkou. Senzor sbírá data v intervalu přibližně 1 vteřiny, dotaz na naměřené hodnoty by tedy měl probíhat nejméně se stejnou frekvencí.

3.2.3 Návrh akvária

Aby bylo možné v pozdější fázi otestovat navrhnutý systém, je potřeba vytvořit funkční regulační jednotku pro určitý typ vivária. Bylo rozhodnuto vytvořit regulační a ovládací jednotku pro akvárium.

Systém je umístěn v plastové elektroinstalační krabici o rozměrech 20x25x10 cm. Pro snazší zapojení jsou všechny moduly umístěné ve víku napájeny jedním konektorem a zbytek komponent je zapojen do navržené PCB desky.

Víko

K víku krabice jsou připojeny oba displeje, zásuvky, stmívač, dvoukanálové SSR relé a I2C rozšiřující modul sloužící k jeho ovládání. K tomuto rozšiřujícímu modulu jsou taktéž připojeny dvě LED diody sloužící k indikaci stavu zásuvek.

Modul	Pin	Připojení na
Stmívač	GND	GND
	VCC	VCC +3.3 V
	Z-C	2
	PWM	15
I/O Rozšíření	GND	GND
	VCC	VCC +5 V
	SCL	SCL (21)
	SDA	SDA (20)
LCD	GND	GND
	VCC	VCC +5 V
	SCL	SCL (21)
	SDA	SDA (20)
Hodinový displej	GND	GND
	VCC	VCC +5 V
	CLK	32
	DIO	34

Tabulka 5 - Přehled zapojení komponent na víku

Na následujícím obrázku je znázorněno schéma PCB desky pro víku vytvořené v programu Fritzing. Na desku jsou připojené piny pro napájení notifikačních diod, I/O rozšíření a SSR relé.

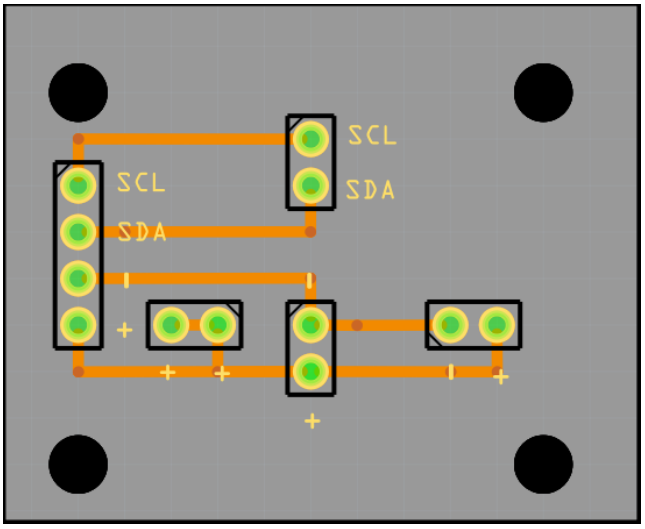


Schéma 4 - Návrh DPS pro víko

Vnitřek boxu

Většina modulů vyplňuje prostor uvnitř boxu. Jedná se o ovládací prvky modulů, konektorové průchodky, zdroj, převodníky napětí, tlačítka modulů, tlačítko pro zapnutí Bluetooth modulu a vypínač napájení.

Modul	Pin	Připojení na
Teplotní čidlo	GND	GND
	VCC	VCC +5 V
	DAT	4
PH senzor	GND	GND
	VCC	VCC +5 V
	pH výstup	39
Ventilátor (MOSFET)	GND	GND
	VCC	VCC +5 V
	SIG	12
Krmítko (Krokový motor)	GND	GND
	VCC	VCC +5 V
	IN 1	0
	IN 2	13
	IN 3	14
	IN 4	27
Čerpadlo (MOSFET)	GND	GND
	VCC	VCC +5 V
	Spínání	26
LED	GND	GND
	VCC	VCC +5 V
	DAT	25
Senzor hladiny	GND	GND
	VCC	VCC +5 V
	Echo	33
	Přerušení	32

Tabulka 6 - Přehled zapojení komponent uvnitř boxu

Je potřeba určit zapojení modulů do desky mikrokontroleru. Předcházející tabulka obsahuje přehled zapojení jednotlivých externích modulů akvária. Kromě těchto modulů je potřeba připojit tlačítka a ovládání stavových LED diod indikující připojení modulu. Senzor tlačítek je napojen na GPIO 36. K ovládání LED diod slouží posuvný registr 74HC959 napojený na piny 19 (Latch), 23 (Clock), 5(data) a 18 (jas). Zvolené diody pracují s napětím 3 voltů a protéká jimi napětí 20 mA, je tedy třeba připojit sériově pro každou diodu 100 Ω rezistor.

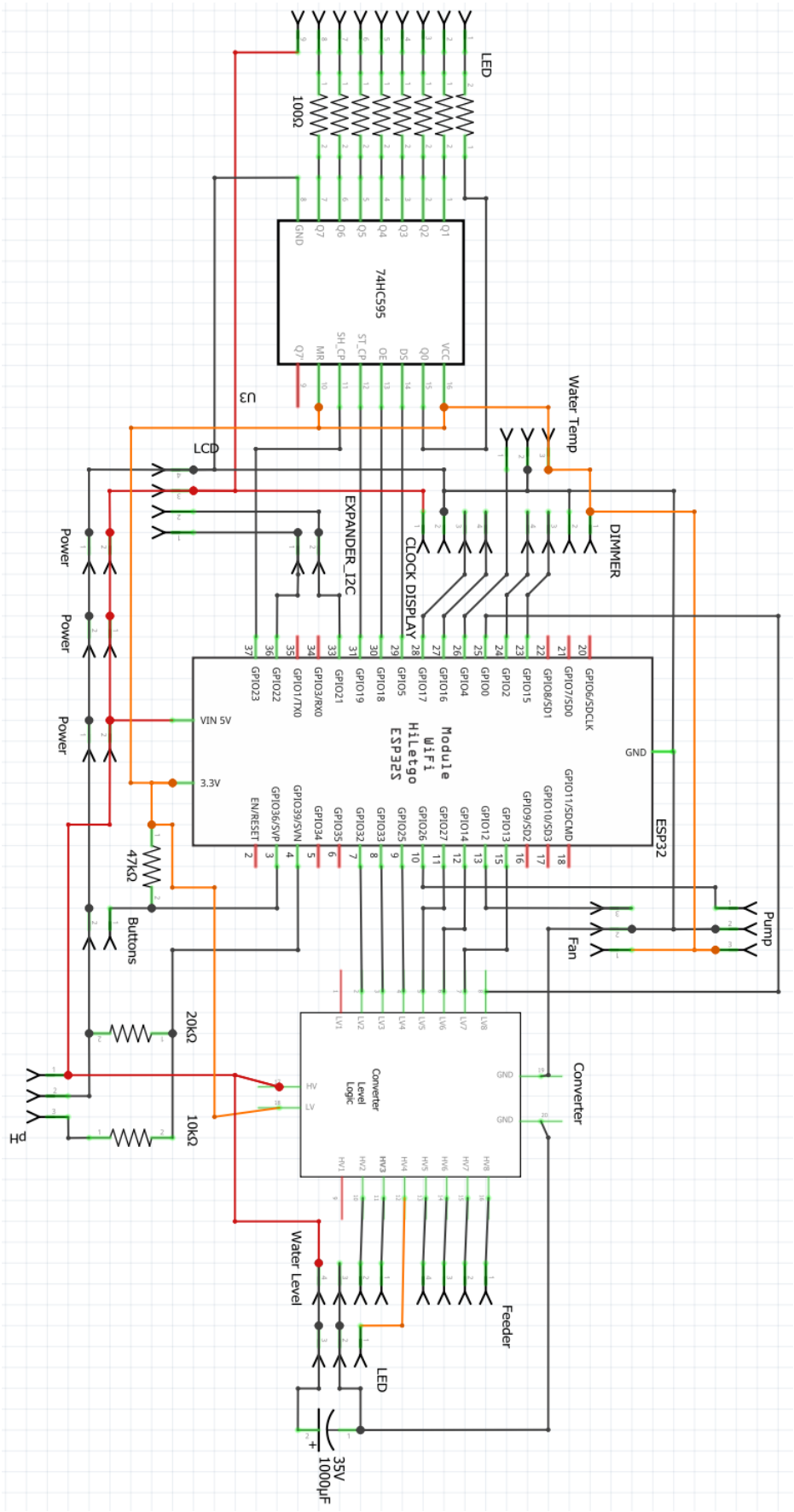


Schéma 5 - Schéma zapojení komponent

Velká část modulů využívá pro komunikaci logickou úroveň 5 voltů, nelze je tedy přímo napojit do ESP32 pracující s 3,3 volty. Proto je použit obousměrný převodník logické úrovně mezi 5 a 3,3 volty.

Následující schéma ukazuje návrh PCB desky pro ovládání akvária vytvořené v programu Fritzing.

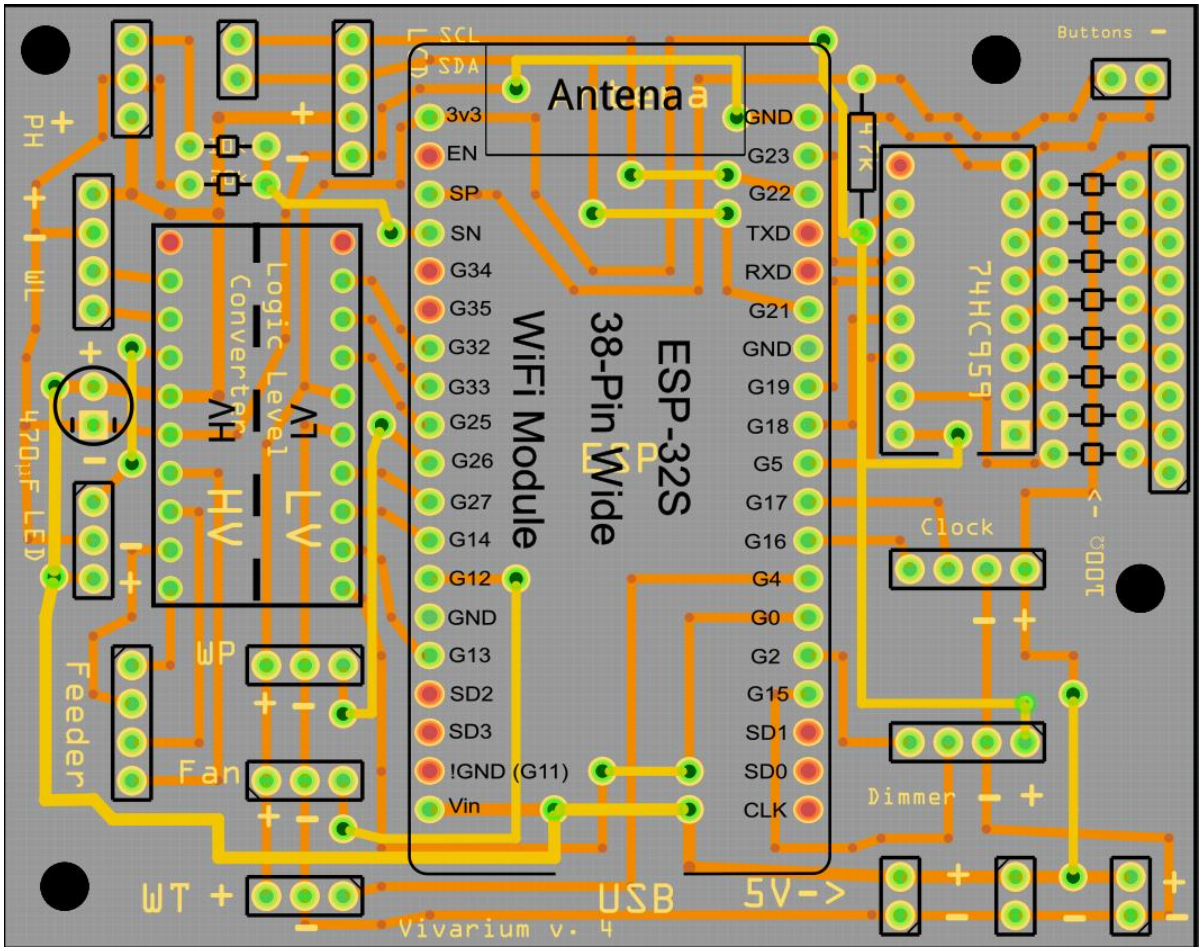


Schéma 6 - Návrh PCB desky pro řídící jednotku Akvária

Finanční náklady

V následující tabulce jsou zahrnuty veškeré náklady spojené s tvorbou chytrého vivária. Výsledná cena vždy závisí na zvoleném typu vivária a jeho hardwarové konfiguraci. V případě akvária bylo vynechán mlhovač a DHT11.

HW	Cena
pH Sonda	798 Kč
Teplotní čidlo DS18B20	75 Kč
HC-SR04	49 Kč
4x ventilátor	60 Kč
Krokový motor	131 Kč
RobotDyn stmívač	108 Kč
Ponorné čerpadlo	68 Kč
Led osvětlení	55 Kč
Mlhovač	299 Kč
Čidlo DHT11	64 Kč
SSR relé	60 Kč
Step-down měnič	95 Kč
I/O expander	98 Kč
ESP32	91 Kč
Zdroj 24 W	75 Kč
LCD displej	68 Kč
Hodinový displej	58 Kč
3x zásuvka	228 Kč
2x IRF520 MOSFET	30 Kč
PCB	250 Kč
Cena všech komponent	**2.760 Kč**
Cena akvária	**2.397 Kč**

Tabulka 7 - Finanční náklady použitých komponent

3.3 Kamera

Při rozhodování o způsobu pořízení obrazového záznamu se bral zřetel na cenu a způsob realizace. Kamera by měla být umístěna nezávisle na pozici řídící jednotky vivária a připojení pomocí kabelu k řídící jednotce je příliš omezující. Obsluha kamery by vyžadovala využití většího množství pinů čímž by se snížil maximální počet připojených modulů.

Finančně stejně náročné, a přitom lépe ovladatelné i spravovatelné řešení, je předání obsluhy obrazového záznamu samostatnému mikrokontroleru ESP32-S na desce ESP32-CAM, který

disponuje Bluetooth modulem, Wi-Fi modulem a konektorem pro připojení kamery. S modulem je dodávaná kamera OV2640 s rozlišením 2 Mpx.

Mikrokontroler bude schopný komunikovat přes Bluetooth s mobilní aplikací. V aplikaci se kameře nastaví SSID a heslo k Wi-Fi sítí, zároveň se pošle i identifikátor uživatele, kterému kamera náleží. Kamera bude posílat v pravidelném intervalu pomocí Wi-Fi obrazový záznam do úložiště Firebase Storage. Cesta uloženého souboru se bude skládat z id uživatele a id vivária, což umožní aplikaci zobrazovat správnou fotografii podle uživatele.

3.4 Flutter aplikace

Aplikace slouží ke správě vivárií vlastněných uživatelem a má dvě hlavní úlohy, které lze následně rozdělit do několika dílčích částí.

Pod správu uživatele spadá:

- Registrace uživatele
- Přihlášení uživatele
- Odhlášení

Pod správu vivárií spadá:

- Přidání vivária
- Odebrání vivária
- Prohlížení stavu
- Změna nastavení
- Přidání kamery
- Odebrání kamery
- Prohlížení obrazového záznamu

3.4.1 Obrazovky aplikace

Na následujícím obrázku lze vidět schéma rozdělení stránek aplikace a způsoby přesunu mezi nimi. Po spuštění aplikace se zobrazí domovská obrazovka *Home*. Pokud není uživatel přihlášen, má možnost přejít na stránku přihlášení nebo registrace. Přihlášený uživatel poté může přejít na stránku *Devices*, kde je mu zobrazen přehled všech vivárií, která vlastní. Odsud

může zahájit sekvenci pro přidání nového zařízení nebo se prokliknout na detail některého již přidaného vivária.

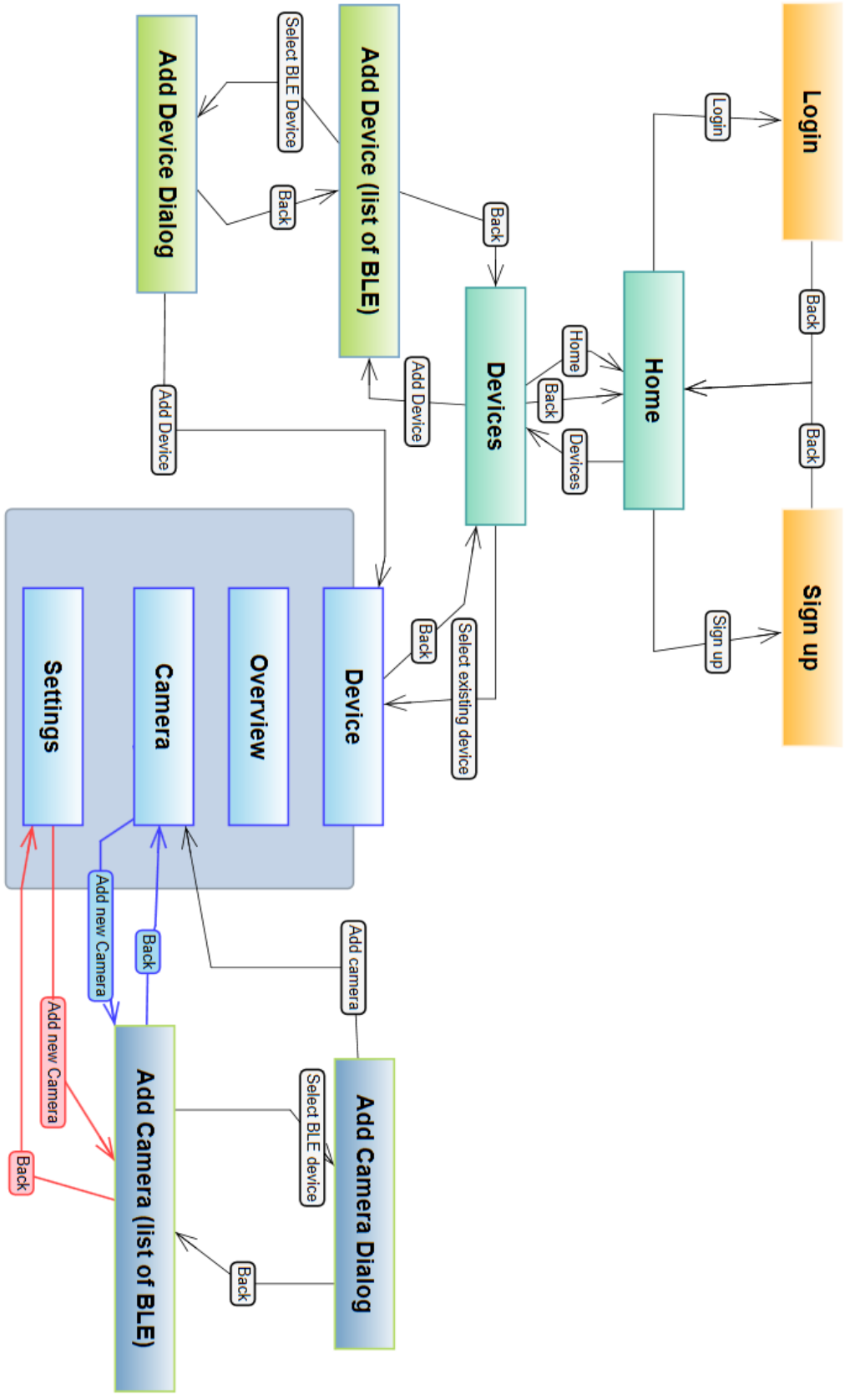


Schéma 7 - Schéma obrazovek mobilní aplikace

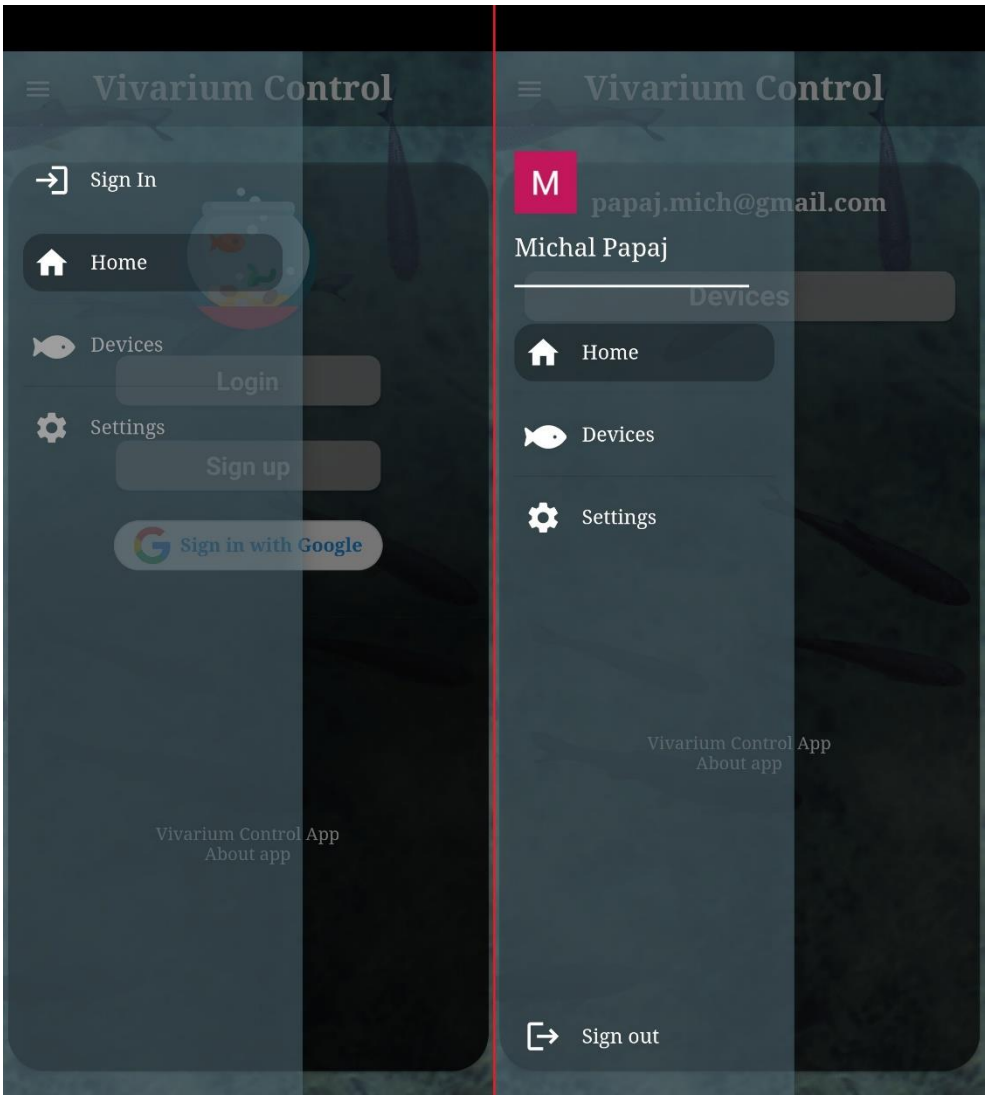
Navigační podokno

Pro přepínání mezi stránkami slouží navigační podokno v podobně postranního šuplíku. Navigace je rozdělena na hlavičku, tělo a patu.

Hlavička ukazuje aktuálního uživatele. Konkrétně je vidět v případě přihlášeného uživatele jméno a profilový obrázek, pokud uživatel pro přihlášení použil Google účet. Pokud uživatel přihlášen není je hlavička nahrazena tlačítkem pro přihlášení.

Tělo obsahuje navigační tlačítko pro přepínání mezi hlavními okny aplikace – *Home* a *Devices*. Okno *Devices* je přístupné pouze pro přihlášené uživatele, proto je nabídka při odhlášeném uživateli zešedlá.

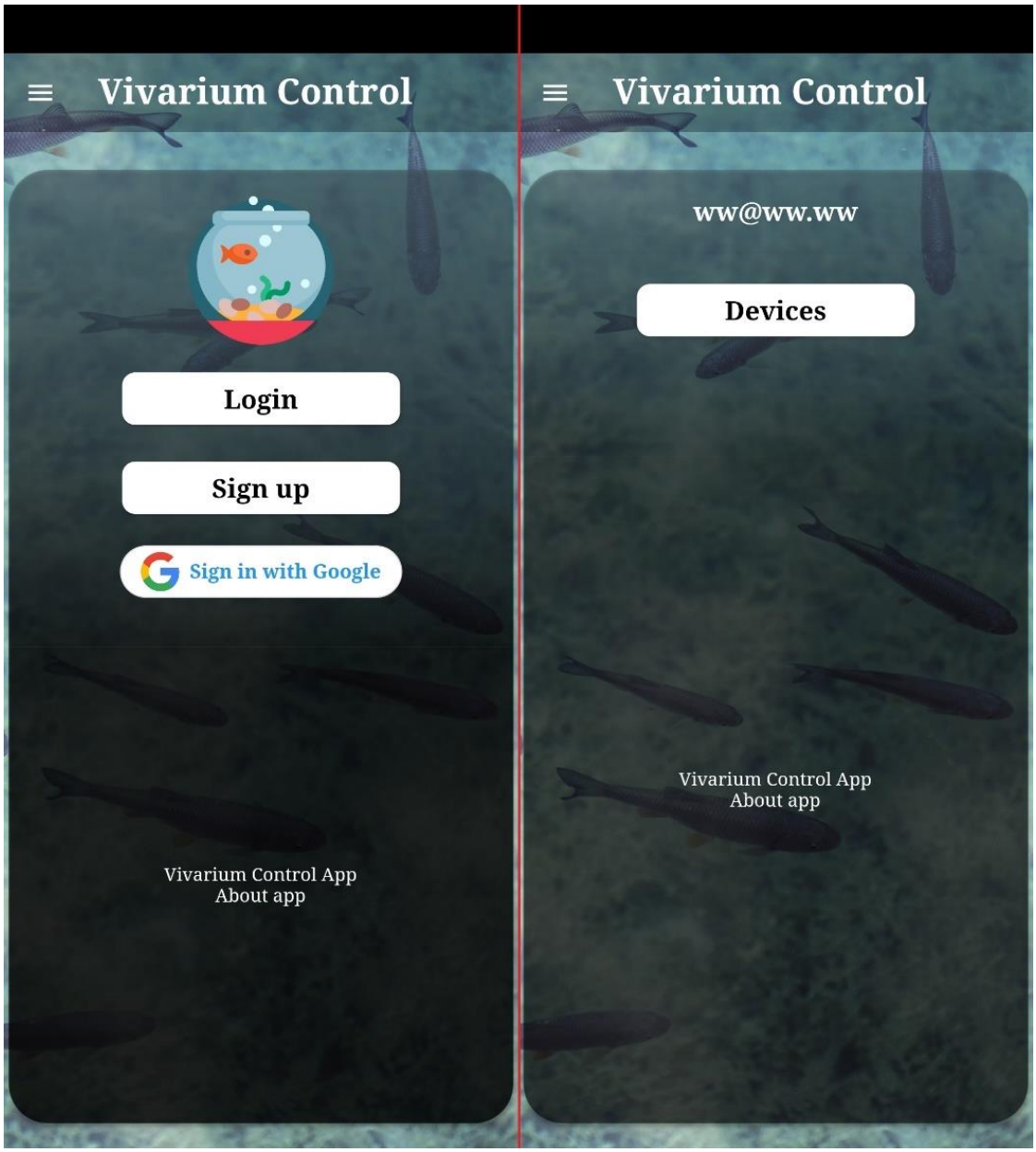
Pata navigačního šuplíku obsahuje jediné tlačítko pro odhlášení, které je viditelné pouze při přihlášeném uživateli.



Obrázek 3 - Navigační okno mobilní aplikace

Home

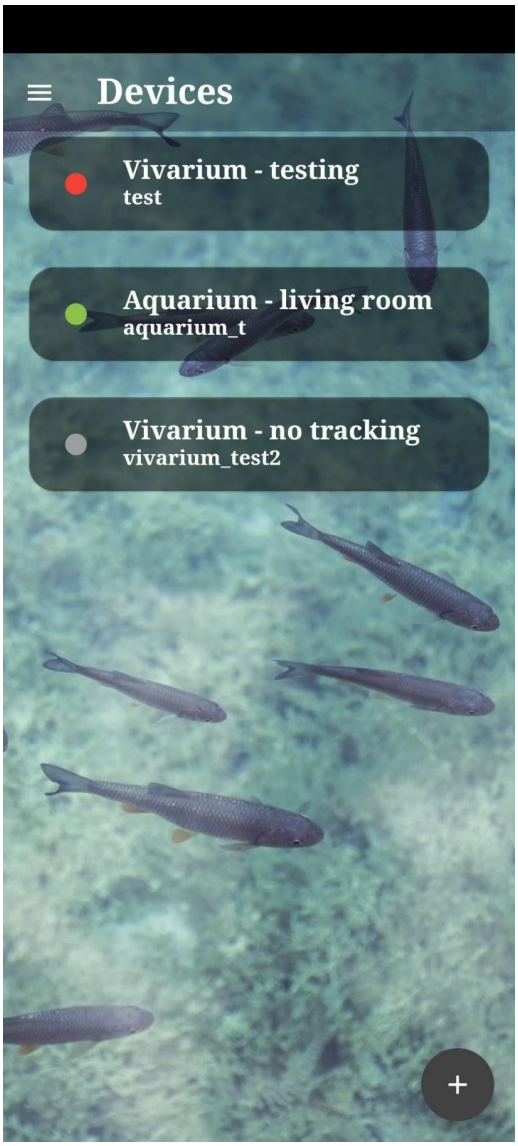
Hlavní stránka zobrazuje různé informace podle stavu přihlášení. Pokud uživatel není přihlášen, zobrazí se mu volby pro přihlášení, registraci či přihlášení prostřednictvím Google účtu. Pokud je již uživatel přihlášen, uvidí na hlavní stránce svou e-mailovou adresu a tlačítko pro zobrazení svých zařízení. V obou případech se ve druhé polovině obrazovky zobrazují základní informace o aplikaci. Oba stavy lze spatřit na následujícím obrázku. Obrazovka nepřihlášeného uživatele se nachází vlevo, stav po přihlášení je napravo.



Obrázek 4 - Přihlašovací obrazovka mobilní aplikace

Devices

Přehled zařízení, která patří přihlášenému uživateli. Každé zařízení je zobrazeno na vlastní kartu. Uživatel vidí jméno zařízení, ID zařízení a jeho online stav. Zelená značí přihlášení před méně než dvěma minutami. Pokud zařízení nekomunikovalo s databází po delší dobu, zobrazí se u zařízení červená barva. Pokud uživatel u některého zařízení vypne kontrolu, zobrazí se šedá. To se hodí, pokud je zařízení umístěno mimo dosah Wi-Fi pro automatickou práci offline. na vlastní kartu spolu se svým názvem. Na stránce se též nachází plovoucí tlačítko se znakem plus pro přidání nového zařízení.



Obrázek 5 - Seznam dostupných zařízení v mobilní aplikaci

Add Device / Camera

Pokud se uživatel rozhodne přidat nové zařízení (vivárium nebo kameru), je přesunut na stránku se seznamem dostupných Bluetooth zařízení v okolí. Každé zařízení se zobrazí na jeden řádek spolu s informacemi o kvalitě signálu, mac adrese a jména zařízení. Po ukončení skenování okolí se zobrazí plovoucí tlačítko pro spuštění nového skenování.

Add Device / Camera Dialog

Tato stránka představuje dialog s formulářem pro přivlastnění nového zařízení (vivária či kamery). Pokud se uživatel pokouší přivlastnit již přivlastněné zařízení, zobrazí se mu upozornění. V opačném případě může upravit jméno zařízení a přidat údaje pro připojení k Wi-Fi síti.

Device

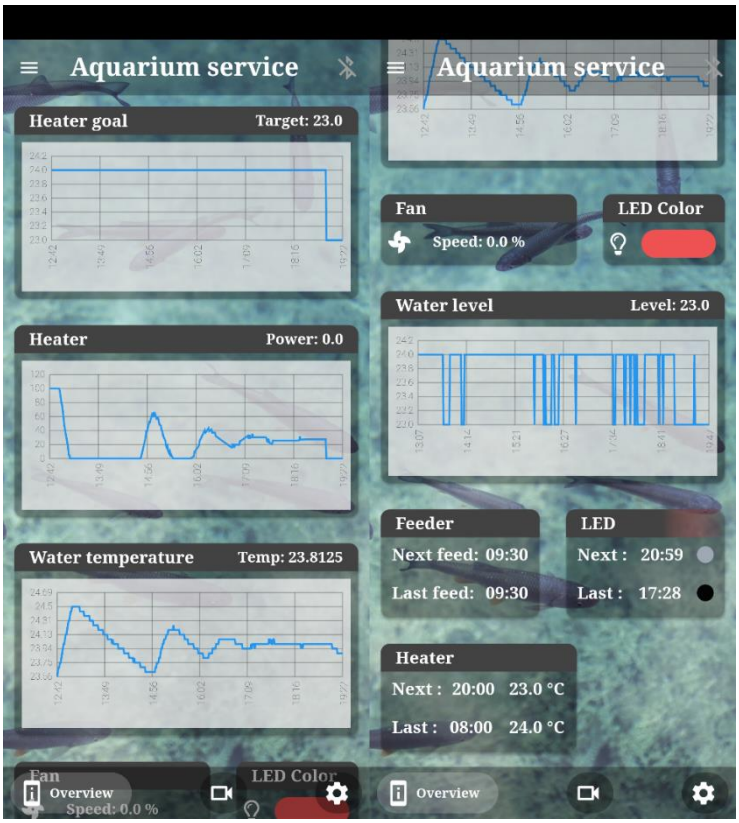
Obrazovka s detaily vivária je rozdělena do tří samostatných oken, mezi nimiž je možné přepínat pomocí navigačního panelu v zápatí stránky, přičemž aktuální volba je zvýrazněna.

V hlavním panelu je kromě názvu zvoleného zařízení umístěno tlačítko pro Bluetooth spojení s daným viváriem. Toto tlačítko se může nacházet v jednom ze tří stavů:

- Bluetooth je vypnuté – ikona je šedá
- Zařízení je odpojené – žlutá ikona
- Zařízení je připojené – zelená ikona

Overview

První okno zobrazuje přehled aktuálního stavu vivária. Informace jsou zobrazeny ve formě samostatných karet, přičemž jednomu modulu vivária lze přiřadit více takovýchto karet.



Obrázek 6 - Přehled zařízení

pH

Ukazuje aktuální hodnotu pH a graf vývoje pH vody. Pokud není dostatek dat pro vykreslení grafu, předává aplikace tuto informaci uživateli.

LED

Pokud má uživatel aktivní LED modul, zobrazí se mu v přehledu dvě karty. Jedna karta ukazuje aktuální barvu LED osvětlení, druhá karta dává informaci o příštím i posledním alarmu pro změnu LED osvětlení, takže má uživatel vždy přehled o změnách barvy

Krmítko

Karta krmítka slouží k zobrazení časového údaje předchozího a následujícího krmení.

Vodní hladina

Karta ukazuje výšku vodní hladiny a při dostatečném množství dat i graf vývoje vodní hladiny akvária.

Teplota vody

Teplotu vody lze spatřit v pravém horním rohu této karty spolu s grafem, který je viditelný při dostatku dat.

Teplota vzduchu a vlhkost vzduchu

Třebaže tento modul zahrnuje čtení dvou charakteristik prostředí, je potřeba je v přehledu zařízení oddělit od sebe. Z tohoto důvodu jsou karty pro vlhkost a teplotu vzduchu zvlášť. Obě mohou při dostatku dat ukázat graf vývoje veličin.

Ventilátor

Ukazuje rychlost otáček ventilátoru v procentech.

Čerpadlo

Na této kartě lze zkontrolovat, zda je čerpadlo zapnuté či vypnuté.

Topítko

Topítko je rozděleno na dvě karty s historií. První karta zobrazuje výkon topítka a jeho změny v čase, druhá karta se poté zaměřuje na cílovou teplotu a pamatuje předchozí změny této veličiny.

Mlhovač

Karta Mlhovače ukazuje stav zapnuty / vypnuto na kartě přehledu

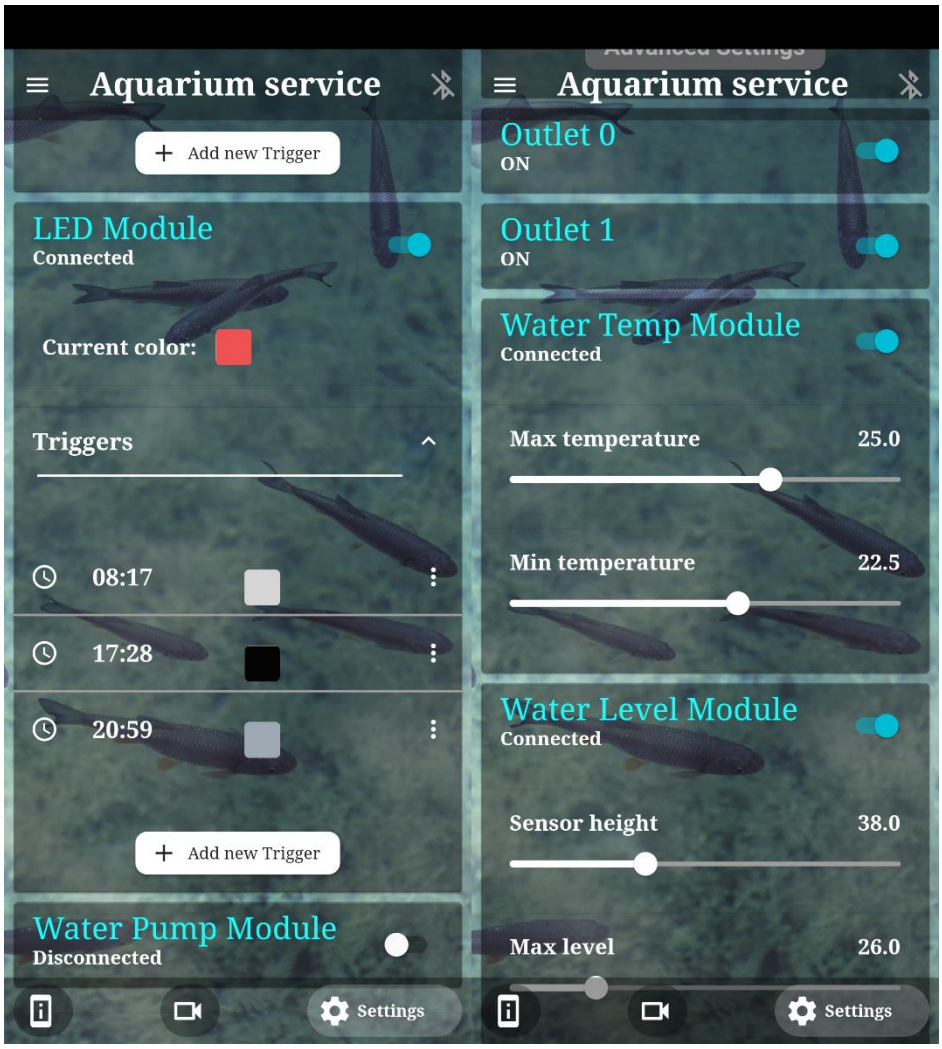
Kamera

Záznam z kamery je zobrazován na samostatném panelu v detailu vivária. Pokud žádná kamera není k viváriu přiřazena, nachází se na místě fotografie tlačítko pro její přidání. Po stisku tohoto tlačítka je uživatel přesunut na výše popsanou obrazovku *Add Device / Camera*.

Uživatel může poklepnutím na fotografii přepnout zobrazení na celou obrazovku, kdy dojde ke změněně orientace zařízení z portrétu na šířku. V tomto režimu se zobrazuje pouze fotografie se zachováním poměru stran, zbytek obrazovky je vyplněn pozadím aplikace. Další poklepnutí přepne telefon zpět do orientace na výšku.

Settings

Stránka nastavení je rozdělena na několik oblastí. První oblast slouží pro základní nastavení vivária – jméno zařízení a tlačítko pro úplné odstranění vivária. Pokud je uživatel napřímo přihlášen k viváriu prostřednictvím Bluetooth, nabídka se rozroste o možnost změnit přihlašovací údaje k Wi-Fi. Druhá oblast přidává možnost přidat či odebrat kameru. V poslední části se nachází nastavení samostatných modulů vivária. Každý modul lze aktivovat pomocí posuvníku v záhlaví karty nastavení. Po aktivaci modulu se karta nastavení rozšíří a zobrazí se možnosti nastavení daného modulu.



Obrázek 7 - Nastavení zařízení

pH

Modul pro měření pH umožňuje nastavit horní a dolní hranici stupnice, při které dostane uživatel upozornění o překročení limitu. Zároveň může přepínat režim měření pH mezi jednorázovým měřením, po němž se pH sonda vypne nebo soustavným měřením, kdy je měření prováděno v pravidelném intervalu. Interval měření si může nastavit na posuvníku.

LED

U osvětlení lze nastavit aktuální barvu a taktéž naplánovat čas přepnutí osvětlení v určitý čas dne. Pro přidání stiskne uživatel tlačítko „*Add new Trigger*“. Po stisku je přidán do karty modulu nový řádek, kde si může zvolit barvu a čas, kdy má k přepnutí dojít.

Ventilátor

U ventilátoru lze nastavit teplotu spouštění a teplotu, při které ventilátor dosáhne maximálních otáček.

Krmítko

Krmítko umožňuje nastavit svůj typ a také časovač spouštění. Pro nový časovač lze použít tlačítko „*Add new Trigger*“ a poté na nově vytvořeném řádku časovače změnit čas spouštění.

Vodní hladina

Modul kontrolující výšku vodní hladiny lze nastavit třemi posuvníky. Kromě výšky umístění měřící senzoru jsou dva posuvníky určeny pro nastavení maximálního a minimálního stavu vody při jejichž překročení přijde uživateli upozornění.

Teplota vody

Vystavuje posuvníky pro nastavení horního a dolního limitu teploty vody po jejichž překročení přijde uživateli upozornění.

Teplota a vlhkost vzduchu

Nastavení tohoto modulu slouží taktéž čistě pro odchozí upozornění uživateli při překročení limitů. Posuvníky jsou zde čtyři pro kontrolu teploty i vlhkosti.

Mlhovač

Obsahuje jeden posuvník pro nastavení požadované vlhkosti vzduchu.

Čerpadlo

Slouží k nastavení požadované výšky vodní hladiny.

Topítko

Uživatel může pomocí rozevíracího tlačítka změnit typ topítka. Pokud vybere možnost s interním ovládáním topítka, zobrazí se mu posuvná lišta pro nastavení požadované teploty. Lze taktéž vytvořit časovače pro změny teploty v průběhu dne. Pokud je vybrán automatické řízení teploty, lze přejít na stránku s nastavením parametrů PID regulátoru.

4 Implementace

4.1 Firebase

4.1.1 Realtime Database

Ukládání dat v rámci jednoho JSONu s sebou přináší problémy se stahováním aktuálně nepotřebnými daty. Z toho důvodu jsou data rozdělena hned v první úrovni pro snížení jejich hloubky.

Device

Struktura zařízení je rozdělena na několik částí – camera, info, sensorData, settings a state.

Objekt *camera* uchovává časové razítko posledního nahrání fotky a informaci o tom, zda je kamera připojená k vivárium. *Info* obsahuje základní informaci o vivárium, *sensorData* aktuální informace ze senzorů, *settings* nastavení všech modulů a *state* jejich aktuální stav.

```
"info": {
  "active": true,
  "condition": 0,
  "firmware": "5_3",
  "id": "aquarium_1",
  "macAddress": "10:52:1C:66:2C:BA",
  "name": "Aquarium home",
  "owner": "T4ZZ6aF8jrUDqkuPbuMpSRPpxwB3"
},
```

Kód 7 - JSON základních informací o vivárium v RTDB

Spouštění

Vivárium po každém spuštění loguje tuto informaci do databáze v */start/{deviceId}/{timestamp}*

Uživatelé

Uživatelské nastavení lze nalézt v cestě */users/{userId}/settings* a tokeny pro FCM se nachází v /users/{userId}/tokens.

Historie dat

Mikrokontroler v pravidelném intervalu nahrává objekt reprezentující aktuální stav vivária do cesty */sensorData/{deviceId}/{timestamp}*. Obsah nahraných dat záleží na aktuálně zapnutých

modulech. Databáze neumožňuje prázdný JSON objekt, proto nedochází při všech vypnutých modulech k žádném nahrání nového stavu.

Firmware

Databáze ukládá název souboru každé verze firmwaru. Cesta k názvu je */firmware/{verze}/filename*

Krmení

Aby bylo možné ověřit, že modul krmítka probíhá v pořádku, ukládá se při každém spuštění krmítka časové razítko do databáze ve tvaru */feed/{deviceId}/{timestamp}*.

Pravidla

Firebase umožňuje nastavit pravidla přístupu k datům v databázi. Pravidla jsou strukturována stejným způsobem jako je struktura databáze. Je možné nastavit pravidlo pro celou databázi nebo jakékoliv vnořené umístění.

Použitá pravidla jsou vidět na následujícím obrázku s číslem 8. Základem je vytvořit obecný zákaz práce s databází komukoli, kdo nemá administrátorské oprávnění. Toto pravidlo zaručuje, že všechny dotazy kromě explicitně povolených jsou běžnému uživateli zakázány. Pravidlo upravující vstup do */users/{$uid}/* začínající na řádce 7 povoluje uživatelům přístup pouze ke svým vlastním datům. Firebase umožňuje pravidla omezit na přesně definovaný dotaz do databáze. Na 14 je nastaveno, že seznam vivárií lze získat pouze dotazem na vivária, jejichž majitel je zároveň dotazující uživatel. Číst detaily zařízení, vytvořit nové zařízení nebo změnit některou z uložených hodnot může pouze jejich majitel.

```
{
    "rules": {
        ".read":"auth.uid!=null
                        && auth.uid!='' && root.child('admins').child(auth.uid).exists()",
        ".write":"auth.uid!=null
                            && auth.uid!='' && root.child('admins').child(auth.uid).exists()",
        "users": {
            "$uid": {
                ".write": "$uid === auth.uid",
                ".read": "$uid === auth.uid"
            }
        },
        "devices":{
            ".read": "auth.uid!=null &&
                                query.orderByChild == 'info/owner' &&
                                query.equalTo == auth.uid",
            ".indexOn": "info/owner",
            "$deviceId": {
                                ".read":  "auth.uid !=null &&
                                    data.child('info').child('owner').val() === auth.uid",
                                ".write": "auth.uid != null &&
                                    (!data.exists() ||
                               data.child('info').child('owner').val() === auth.uid) &&
                                    root.child('users').child(auth.uid).exists()",
            }
        },
        "sensorData":{
          "$deviceId":{
            ".read": "auth.uid !=null &&
                                root.child('devices').child($deviceId).child('info/owner').val() === auth.uid"
          }
        }
    }
}
```

Kód 8 - Bezpečností pravidla RTDB

4.1.2 Storage

Ve Firebase Storage se ukládají fotografie z kamer a firmware aplikace. Firmware je ukládán do složky */firmware*, název programu je zakončen sufixem s verzí firmwaru pro snadnou orientaci ve verzích.

Fotografie jsou ukládaný do samostatných složek podle uživatele a vivária, ke kterému kamera patří. Konkrétní cesta k fotografiím je */camera/{userId}/{deviceId}/photo.jpg*.

Pravidla

Mikrokontrolery se přihlašují pod speciálně vytvořeným účtem. Je tedy potřeba přizpůsobit pravidla Firebase Storage tak, aby měl daný účet povolený přístup pro nahrávání fotografií a stahování nové verze firmwaru. Bezpečností pravidla neumožňují posílat dotazy do RTDB pro ověření oprávnění, proto bylo rozhodnuto o zapsání ID speciálního uživatele přímo do pravidel služby. Aplikace stahují fotky po přihlášení uživatele, a proto je přístup omezen pouze do složky, která patří aktuálně přihlášenému uživateli.

```
rules_version = '2';
service firebase.storage {
  match /b/{bucket}/o {
    allow read: if false;
    allow write: if false;

    match /camera/{userId} {
      match /{cameras=**}{
        allow read: if request.auth.uid == userId;
        allow write: if request.auth.uid =="UMEYityTDDYrFBkMNQDHaV2igf83"
      }
    }
    match /firmware/{file=**}{
    allow read: if request.auth.uid == "UMEYityTDDYrFBkMNQDHaV2igf83"
    }
  }
}
```

Kód 9 - Bezpečností pravidla Firebase Storage

4.1.3 Testy bezpečnostních pravidel

K ověření přístupu k datům uložených ve Firebase Storage a RTDB byly napsány napsat unit testy za využití JavaScriptového frameworku Mocha, který umožňuje nastavit opakovatelné úkony spouštěné před a po každém testu. Pro samotné testy je použit oficiální npm balíček *@firebase/rules-unit-testing*.

Pro testování je potřeba spustit na lokálním uložišti emulátory testovaných služeb. V samotném testu je nejprve vytvořeno testovací prostředí manipulující s těmito lokálními službami. Před každým testem jsou v rámci tohoto prostředí smazána všechna data, která mohla zůstat z předchozího testu. Testy jsou zaměřené na autorizaci čtení a zápisů podle úrovně uživatele – neautentizovaný uživatel, autentizovaný uživatel a admin.

```
describe("Sensor data", () => {
    it("should prevent READ any data from unauthenticated user", async () => {
        const unDB = testEnv.unauthenticatedContext().database();
        await assertFails(get(databaseRef(unDB, 'sensorData/device_1/1636201621/data')));
    });

    it("should allow READ data of authenticated user", async () => {
        await testEnv.withSecurityRulesDisabled(async context => {
            let db = context.database();
            await db.ref('sensorData/device_1/1636201621').set({'data': 'value'});
            await db.ref('devices/device_1/info').set({'owner': 'user_123'});
        });
        const db = testEnv.authenticatedContext('user_123').database();
        await assertSucceeds(get(databaseRef(db, 'sensorData/device_1/1636201621/data')));
    });

    it("should prevent READ of foreign data", async () => {
        await testEnv.withSecurityRulesDisabled(async context => {
            let db = context.database();
            await db.ref('sensorData/device_1/1636201621').set({'data': 'value'});
            await db.ref('devices/device_1/info').set({'owner': 'user_777'});
        });
        const db = testEnv.authenticatedContext('user_111').database();
        await assertFails(get(databaseRef(db, 'sensorData/device_1/1636201621/data')));
    });


    it("should prevent WRITE of any authenticated user", async () => {
        await testEnv.withSecurityRulesDisabled(async context => {
            let db = context.database();
            await db.ref('sensorData/device_1/1636201621').set({'data': 'value'});
            await db.ref('devices/device_1/info').set({'owner': 'user_123'});
        });
        const db = testEnv.authenticatedContext('user_123').database();
        await assertFails(set(databaseRef(db, 'sensorData/device_1/1636201621/data'), {'data':
'data1'}));
    })
});
```

Kód 10 - Testování bezpečnostních pravidel

4.2 Flutter aplikace

Existuje vícero návrhů architektury pro tvorbu aplikace pomocí frameworku Flutter. V rámci této práce bylo zvolen způsob přepínání mezi stránkami pomocí takzvaných *Named Routes* a využití stavového manažeru *Provider*. Stejný kód byl použit pro mobilní i webovou aplikaci.

Přidání a odebrání vivária a jeho kamery vyžaduje přístup k Bluetooth. Použité pluginy pro práci s Bluetooth zatím nepodporují webovou platformu, proto je tato funkcionalita dostupná pouze pro mobilní aplikaci.

Zároveň bylo potřeba nadefinovat nastavení specifické pro použité platformy.

4.2.1 Nastavení mobilní aplikace

Mobilní aplikace vyžaduje úpravu souboru *androidManifext.xml*. Je nutné specifikovat kanál pro zobrazení notifikací a vytvořit seznam všech povolení, která aplikace v průběhu použití potřebuje. Tato povolení jsou potřeba pro správné fungování použitých knihoven aplikace,

aby bylo možné využít funkce mobilního zařízení. Pro připojení k Firebase službám je potřeba přidat vygenerovaný konfigurační soubor *google-services.json* do adresáře *android/app*.

```
<application
   <meta-data
     android:name="com.google.firebase.messaging.default_notification_channel_id"
     android:value="vivarium_channel" />
</application>

<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
<uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Kód 11 - Nastavení Android aplikace v souboru AndroidManifest.xml

4.2.2 Nastavení webové stránky

Většina pluginů nepotřebuje žádné další nastavení pro podporu webu. Výjimkou jsou Firebase služby, které vyžadují načtení Firebase SDK uvnitř *index.html* souboru a následnou inicializaci Firebase pomocí konfiguračního JSONu.

```
<script src="https://www.gstatic.com/firebasejs/8.6.1/firebase-app.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.6.1/firebase-analytics.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.6.1/firebase-auth.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.6.1/firebase-firestore.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.6.1/firebase-messaging.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.6.1/firebase-database.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.6.1/firebase-storage.js"></script>
<script>
    firebase.initializeApp({
        "apiKey": "AIzaSyCrVwfunuauskErYhbScfK4rXh2RTG4v0o",
        "authDomain": "vivarium-control-unit.firebaseapp.com",
        "databaseURL": "https://vivarium-control-unit.firebaseio.com",
        "projectId": "vivarium-control-unit",
        "storageBucket": "vivarium-control-unit.appspot.com",
        "messagingSenderId": "937863307915",
        "appId": "1:937863307915:web:8a2f15de4cf6d1ff206a8e",
        "measurementId": "G-1KW94F1JLT"
    });
</script>
```

Kód 12 - Nastavení webové aplikace

4.2.3 Named Routes

Zobrazované stránky v aplikaci jsou ukládány do zásobníku. Při spuštění aplikace se v zásobníku nachází pouze hlavní stránka. Pro navigaci mezi stránkami slouží Navigator, což je widget starající se o hierarchii stránek v zásobníku a jejich přidání či odebrání. Pro přechod na jinou stránku můžeme použít funkci *Navigaro.push*. V zásobníku pak budou uloženy dvě

stránky a uživatel uvidí tu, která je aktuálně na vrchu. Pro odebrání stránky ze zásobníku slouží funkce *Navigator.pop*. Tento způsob přesouvání mezi stránkami je u menších aplikací dostačující. Spolu s růstem aplikace se však zvyšuje počet stránek, a tedy údržba kódu, kdy je složitější ohlídat veškeré přechody. Mnohdy také nastávají duplicity kódu.

Tuto nevýhodu je možné odstranit využitím Named Routes. Základem je definování obrazovek a cest k nim tak, jak je vidět na následujícím výňatku z kódu.

```
class Routes {
  static const String home = '/';
  static const String login = '/login';
  static const String register = '/register';
  static const String userDevices = '/userDevices';
  static const String settings = '/settings';
  static const String locations = '/locations';
  static const String addDeviceList = '/addDeviceList';
  static const String addCameraList = '/addCameraList';
  static const String addDeviceDialog = '/d+';
  static const String addCameraDialog = '/c+';
  static const String device = '/device';
}
```

Kód 13 - Seznam definovaných cest ke stránkám aplikace

Po definici cest k obrazovkám je potřeba definovat mapu, která přiřadí k cestám novou obrazovku.

```
Map<String, WidgetBuilder> defaultRoutes() => {
    Routes.home: (context) => HomePage(),
    Routes.register: (context) => RegisterPage(),
    Routes.login: (context) => LoginPage(),
    Routes.settings: (context) => SettingsPage(),
    Routes.userDevices: (context) => DevicesPage(),
    Routes.addDeviceList: (context) => AddDevicePage()
  };
```

Kód 14 - Základní cesty aplikace

Jak je z předchozího kódu patrné, mapa nevrací všechny stránky. Jiným stránkám je potřeba předat při přechodu určité argumenty, což tento konstrukt definovat neumí. Naštěstí s tím framework počítá a je možné implementovat funkci, která se postará o vygenerování nové cesty, pokud dosud nebyla definována.

Tato funkce přijímá jako argument objekt *RouteSettings*, který obsahuje dvě informace – název cesty a argumenty předané nově vytvořené stránce. V aplikaci byla tato funkce použita pro generování obrazovek pro přidání vivária, kamery, které potřebovali pro správnou funkcionalitu detaily o Bluetooth zařízení a pro stránku detailu vivária, která potřebovala vědět ID vivária, které zobrazuje. Tuto metodu lze také využít pro zpracování

neznámé cesty. V tomto případě dojde k zobrazení prázdné stránky s informací, že uvedená cesta neexistuje.

```
static Route<dynamic> generateRoute(RouteSettings settings) {
  switch (settings.name) {
    case Routes.addCameraList:
      return MaterialPageRoute(
        builder: (_) => (settings.arguments == null)
            ? defaultPage(settings.name)
            : AddCameraPage(deviceId: settings.arguments),
      );

    case Routes.addDeviceDialog:
      return MaterialPageRoute(
          builder: (_) => (settings.arguments == null)
              ? defaultPage(settings.name)
              : AddDeviceDialog(device: settings.arguments));

    case Routes.addCameraDialog:
      return MaterialPageRoute(
          builder: (_) => (settings.arguments == null)
              ? defaultPage(settings.name)
              : AddCameraDialog(device: settings.arguments));

    case Routes.device:
      return MaterialPageRoute(
          builder: (_) => (settings.arguments == null)
              ? defaultPage(settings.name)
              : DevicePageRoot(device: settings.arguments));

    default:
      return MaterialPageRoute(builder: (_) => defaultPage(settings.name));
  }
```

Kód 15 - Generování nových cest aplikace s parametry

Pro přechod stránek je používáno vícero funkcí:

- *PushNamed* přidá novu stránku do zásobníku
- *PushNamedAndRemoveUntil* přidá novou stránku do zásobníku, ale zároveň odebere ze zásobníku všechny vrchní stránky až po zadanou v parametru. Tato funkce je vhodná například při odhlášení uživatele, kdy je třeba ze zásobníku odebrat všechny stránky, ke kterým nepřihlášený uživatel nemá přístup.
- *PopUntil* odebere ze zásobníku všechny vrchní stránky až po zadanou v parametru. Využití najde například při dokončení přidání kamery, kdy by se měl uživatel vrátit zpět na přehled kamery v detailu zařízení. *PushNamedAndRemoveUntil* v takovém případě nelze využít, neboť by se uživatel neobjevil na panelu kamery, ale na prvním panelu detailu obsahující přehled stavu vivária.

4.2.4 Provider

Každá stránka je definována stromem widgetů, přičemž v případě stateful widgetů zná widget pouze svůj stav. Pokud bychom chtěli definovat nějakou vlastnost widgetu, je možné předat parametry při volání konstruktoru v rodiči. Tento způsob předávání parametrů je běžný, ale mohou nastat situace, kdy je třeba předat informaci widgetu o několik úrovní hlouběji, případně je třeba udržovat stav widgetu mimo stromovou strukturu úplně. Je sice možné vytvořit konstruktor pro všechny následující widgety, ale tento způsob přináší několik nepříjemných jevů. V první řadě přestává být kód přehledný, neboť není jasné, zda widget s daty opravdu pracuje nebo je předává dál. Další problém je výkon, kdy jakákoliv změna v kořenu stromu nutí nově překreslit veškeré nekonstantní potomky. Tyto problémy řeší stavové manažery. Flutter umožňuje využití několika stavových manažerů, mezi nimž jsou *Redux, Rx, hooks, BloC, Provider* a jiné. Dokumentace Flutteru označuje Provider jako základní stavový manažer, jehož koncept je aplikovatelný i na jiné manažery. Jelikož ani po porovnání navrhovaných manažerů nebyl nalezen důvod vybrat jiný, bylo rozhodnuto ve využití právě Provideru. [24]

Provider poskytuje několik způsobů, jak předávat informace o stavu objektu. V aplikaci byly použity následující metody:

Provider

Základní forma provideru pro předání hodnoty hlouběji do stromové struktury. Potomek je obalen Providerem a parametru *value* je přiřazena hodnota, která má být přístupná kterémukoliv z potomků.

```
Provider<NavigationPage>.value(
    value: navigationPage, child: NavigationDrawer())
```

Kód 16 - Ukázka využití Provideru

StreamProvider

Třída *StreamProvider* umožňuje přihlásit se k odběru streamu dat a vystavit svým potomkům poslední získanou hodnotu. Do parametru *value* je vložen stream a *initialData* slouží k nastavení prvotní hodnoty předtím, než stream nějakou poskytne.

```
StreamProvider<BluetoothDeviceEncapsulation>.value(
  value: _bluetoothService.devices,
  initialData: BluetoothDeviceEncapsulation(devices: []),
  child: ResultList(),
)
```

Kód 17 - Ukázka využití StreamProvideru

FutureProvider

Tento druh provideru vezme jako počáteční vstup funkci vracející *Future*. Po dokončení funkce je hodnota poskytnuta potomkům. *FutureProvider* je použit v aplikaci pouze pro stáhnutí nové fotografie. Může se stát, že se fotografie nepodaří nahrát, proto je vhodné přidat parametr *catchError* s návratovou hodnotou *null*, aby potomek *ImageView* věděl, že žádný obrázek nemá zobrazit.

```
FutureProvider<CameraImage>.value(
    value: StorageService()
        .getImage(deviceId: device.info.id, userId: user.userId)
        .then((value) => CameraImage(
            updated: device.camera.updated, data: value)),
    catchError: (context, error) {
      print(error);
      return null;
    },
    initialData: null,
    child: ImageView(),
  )
```

Kód 18 - Ukázka využití FutureProvideru

Pro získání informací v potomkovi je možné použít dvou konstruktů:

- *Provider.of<T>(context, listen: false)* – *Provider.of* spolu s parametrem *listen* nastaveným na *false* slouží k jednorázovému získání aktuální hodnoty. To může přijít vhod například při volání funkcí aplikace.
- *Consumer<T>()* – Consumer slouží k získání sdíleného objektu. Má dva parametry – child a builder. Parametr *child* slouží k definování struktury, která není změnou získaného objektu nijak ovlivněna. To je vhodné zejména pokud je potřeba změnit stav některého widgetu, ale není třeba znovu postavit ostatní potomky ve stromové struktuře.

4.2.5 Použité knihovny

V aplikaci bylo využito několika externích knihoven (v rámci Dartu nazývaných balíčky) pro snazší implementaci aplikace. Přidání balíčků do projektu je snadné, stačí připsat do souboru

pubspec.yaml název balíčku a jeho verzi a příkazem *„flutter pub get“* zavolat funkci, která soubor zkontroluje a stáhne potřebné balíčky. Tento příkaz také zkontroluje kompatibilitu mezi knihovnami a v případě problému upozorní vývojáře. Verzi lze specifikovat dvěma způsoby – zadat přesnou verzi, která má být použita, nebo před číslo verze přidat znak stříšky ^. Pokud je před verzí stříška, program bude používat nejnovější verzi, která neobsahuje žádnou z takzvaných *breaking change* (změn, které zásadně mění fungování některé funkce knihovny).

Firebase knihovny

Základem aplikace je komunikace s platformou Firebase. Pro každou službu existuje vlastní knihovna. V tomto konkrétním projektu je třeba použít firebase_core (v. 1.10.0), firebase_auth (3.20.0), firebase_messaging (1.1.0), firebase_database (8.1.0) a firebase_storage (10.1.0).

Notifikace

I když firebase_messaging zpracovává notifikace, umožňuje zobrazení notifikací pouze v případě, že je aplikace na pozadí nebo ve stavu *Terminated*. Proto je třeba použít knihovnu pro zpracování a zobrazení notifikace pro případ, kdy se aplikace nachází na popředí. Pro tuto funkci je zvolena knihovna flutter_local_notifications (9.1.2), která je určena pro spolupráci s balíčkem Firebase.

Pro správné nastavení je potřeba nejprve v AndroidManifest.xml nadefinovat vlastní kanál pro notifikace přidáním *meta-data* tagu.

Při startu aplikace je poté potřeba v kódu vytvořit objekt se stejným id a tento nově vytvořený objekt předat instanci *FlutterLocalNotificationsPlugin*

```
/// Definice nového kanálu v AndroidManifest.xml

<meta-data
    android:name="com.google.firebase.messaging.default_notification_channel_id"
    android:value="vivarium_channel" />

/// Registrace kanálu po spuštění aplikace

static final AndroidNotificationChannel _channel = AndroidNotificationChannel(
  'vivarium_channel', // id
  'Vivarium Module Notifications', // title
  'This channel is used for notifications from vivarium modules.', // description
  importance: Importance.high,
);

await _flutterLocalNotificationsPlugin
    .resolvePlatformSpecificImplementation<
        AndroidFlutterLocalNotificationsPlugin>()
    ?.createNotificationChannel(_channel);

/// Zobrazení notifikace skrze vytvořený kanál

_flutterLocalNotificationsPlugin.show(
    notification.hashCode,
    notification.title,
    notification.body,
    NotificationDetails(
      android: AndroidNotificationDetails(
        _channel.id,
        _channel.name,
        _channel.description,
        icon: 'launch_background',
      ),
    ));
```

Kód 19 - Registrace kanálu pro FCM

Google Sign In

Pro přihlášení prostřednictvím Google je možné využit knihovnu *google_sign_in*, V aplikace se vytvoří instance třídy *GoogleSignIn* a zavolá se funkce *signIn*, která inicializuje dialog pro přihlášení. Veškeré další kroky jsou pak v režii samotné knihovny. Funkce vrací objekt *GoogleSignInAccount*. Při neúspěšném přihlášení je vrácená hodnota *null*.

Bluetooth

Pro práci s Bluetooth jsou požity dvě knihovny. Knihovna *bluetooth_enable* umožňuje vytvořit dialog pro aktivaci Bluetooth v zařízení a *flutter_reactive_ble* zajišťuje připojení a komunikaci s Bluetooth zařízením prostřednictvím Bluetooth Low Energy.

Toast

Toast je základní informační dialog operačního systému Android. Pro zobrazení toastů je využita knihovna *fluttertoast*, která umožňuje zobrazení těchto dialogů i na platformě iOS a webové verzi aplikace. Výhodou této knihovny je možnost zobrazení toastů bez popisovače umístění *BuildContext*. Toast je tedy možné zobrazit odkudkoli, čehož lze využít při zpracování FCM notifikací.

Dialog pro výběr barvy

Aplikace umožňuje výběr barvy podsvícení vivária. Ideální způsob výběru barvy je zobrazení dialogu při stisku ikony. Knihovna *flex_color_picker* umožňuje široký způsob výběru barvy spolu s možností dát uživateli na výběr mezi několika typy výběru. V této práci je využita metoda barveného kola. Výběr probíhá formou dialogu, který vrací hodnotu *true* při potvrzení výběru a *false* při přerušení výběru. Je tedy třeba hlídat aktuálně zvolenou barvou pomocí změny anonymní vnitřní funkce s voláním *setState* uvnitř *stateful* widgetu.

Dialog pro zvolení času

Vhodnou knihovnou pro volbu času k alarmům je *date_time_picker*. Widget *DateTimePicker* lze umístit na požadované místo ve stromu widgetů a uživateli je zobrazen čas se zvolenou ikonou. Po kliknutí do oblasti času je uživateli zobrazen dialog pro výběr jiného času. Jelikož tento widget pracuje s časem v podobě textového řetězce String a v databázi je čas uložen do proměnné typu int, bylo třeba napsat pomocnou funkci pro vlastní převod mezi oběma formami.

Graf

Zobrazení historie je provedeno pomocí grafu. Jako nejvhodnější byla vybrána knihovna *fl_animated_linechart*, která umí dobře pracovat s časovou osou. Konstruktor AnimatedChart přijímá jako parametr list linií pro vykreslení v podobě mapy data a hodnoty: *List<Map<DateTime, double>>*.

Potvrzující dialog

Napříč aplikací může uživatel provést volbu vyžadující jeho potvrzení. Alternativou k vlastní implementaci bylo využití knihovny *confirm_dialog*, která umožňuje editovat titulek a dialogové možnosti.

Linter

Nejedná se o knihovnu rozšiřující kód aplikace, ale o nástroj pro analýzu psaného kódu a pro kontrolu stylistických pravidel. Google zveřejňuje svá interní pravidla pro psaní kódu v podobě balíčku *pedantic*. Pravidla lze volně měnit vytvořením souboru *analysis_options.yaml* v kořenovém adresáři projektu. Oproti originálním pravidlům z knihovny *pedantic* byla přidána preference použití *void* místo *null*, kdekoliv to je možné (*prefer_void_to_null*) a kontrolu přiřazení do *void* (*void_checks*). Během práce byla podpora tohoto linteru přerušena na úkor balíčku *lints*.

Ostatní knihovny

V aplikaci byly použity ikony z knihovny *font_awesome_flutter*, navigační panel *bottom_navy_bar*, *flutter_staggered_grid_view* pro rozložení widgetů na stránce do mřížky a *app_settings* pro přechod do nastavení aplikace v zařízení.

4.2.6 Struktura kódu

Kód aplikace je rozdělen do tří částí:

- Models – Reprezentace dat, se kterými aplikace pracuje
- UI – Uživatelské rozhraní rozdělené podle zobrazovaných obrazovek
- Utils – Logika aplikace a pomocné funkce

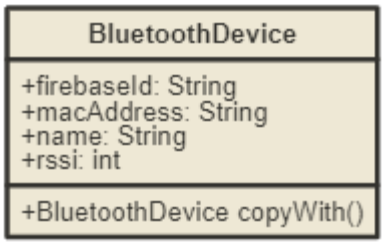
4.2.7 Models

Aplikace pracuje se šesti modely, které jsou popsány budou nyní popsány.

BluetoothDevice

Objekt BluetoothDevice reprezentuje Bluetooth zařízení nalezené při skenování okolí. Aplikace jej využívá při přidání kamery nebo vivária, kdy je tento objekt předán při přechodu ze seznamu Bluetooth zařízení na stránku přidání konkrétního zařízení. Objekt udržuje

informace o názvu zařízení, jeho MAC adrese a hodnotu úrovně signálu RSSI. Pokud je objekt použit k zobrazení dialogu pro přidání kamery, je potřeba přidat informaci o ID vivária.



Obrázek 8 - Třída BluetoothDevice

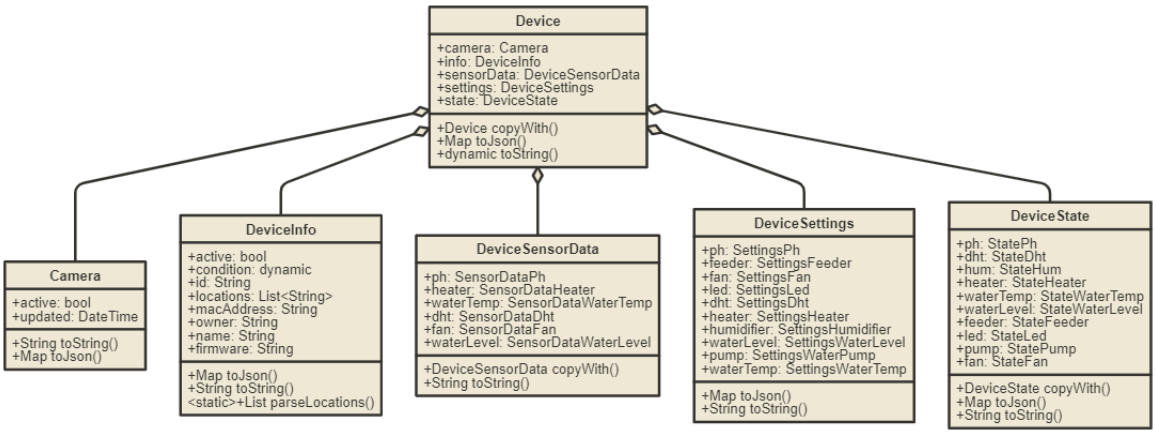
CameraImage

Uchovává binární data stažené fotografie a datum nahrání.

Obrázek 9 - Třída CameraImage

Device

Objekt reprezentující datový záznam z RTDB slouží pro synchronizování dat vivária s aplikací. Svou strukturou odpovídá objekt struktuře RTDB.

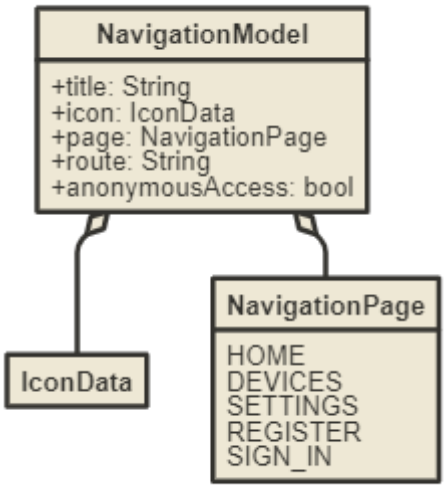


Obrázek 10 - Třída Device

NavigationModel

Definuje položku v navigačním panelu aplikace. Umožňuje snadnou správu obrazovek aplikace a přidává snadnou rozšiřitelnost aplikace. Boolean hodnota proměnné *anonymousAccess* určuje přístupnost dané stránky bez přihlášení. Pokud je hodnota *false*,

potom je položka v navigačním panelu neaktivní. NavigationPage je výčtový typ který umožňuje detekovat aktuálně zvolenou stránku pro grafické zvýraznění položky.



Obrázek 11 - Třída NavigationModel

VivariumUser

Objekt udržující informace o přihlášeném uživateli. *StreamProvider* poskytuje tento objekt pro celou aplikaci. Pokud je objekt *null*, znamená to nepřihlášeného uživatele. *ImageUrl* je dostupné pouze v případě přihlášení prostřednictvím Google, v opačném případě se rovná *null*.

```
class VivariumUser {
  final String userName;
  final String userEmail;
  final String userId;
  final String imageUrl;

  VivariumUser({this.imageUrl, this.userEmail, this.userId, this.userName});
}
```

Kód 20 - Třída VivariumUser

SensorDataHistory

Důležitý model pro zobrazování historie v detailu zařízení. Zatímco mikrokontrolér ukládá informace z modulů do databáze v podobě jednoho JSONu pod jedním časovým razítkem, aplikace potřebuje tyto informace rozdělit podle jednotlivých modulů. Historie je uchována ve struktuře *SplayTreeMap*, která umožňuje ukládat hodnoty v samovyvažovacím binárním stromu. Jako klíč pro vkládání do této mapy slouží časové razítko, což zaručuje, že jsou data ve struktuře vždy řazena chronologicky.

```dart
class SensorDataHistory {
  final SplayTreeMap<DateTime, SensorDataPh> phMap;
  final SplayTreeMap<DateTime, SensorDataWaterTemp> waterTempMap;
  final SplayTreeMap<DateTime, SensorDataHeater> heaterMap;
  final SplayTreeMap<DateTime, SensorDataDht> dhtMap;
  final SplayTreeMap<DateTime, SensorDataWaterLevel> waterLevelMap;

  /// Následuje konstruktor a gettery pro historii měřených hodnot
}
```

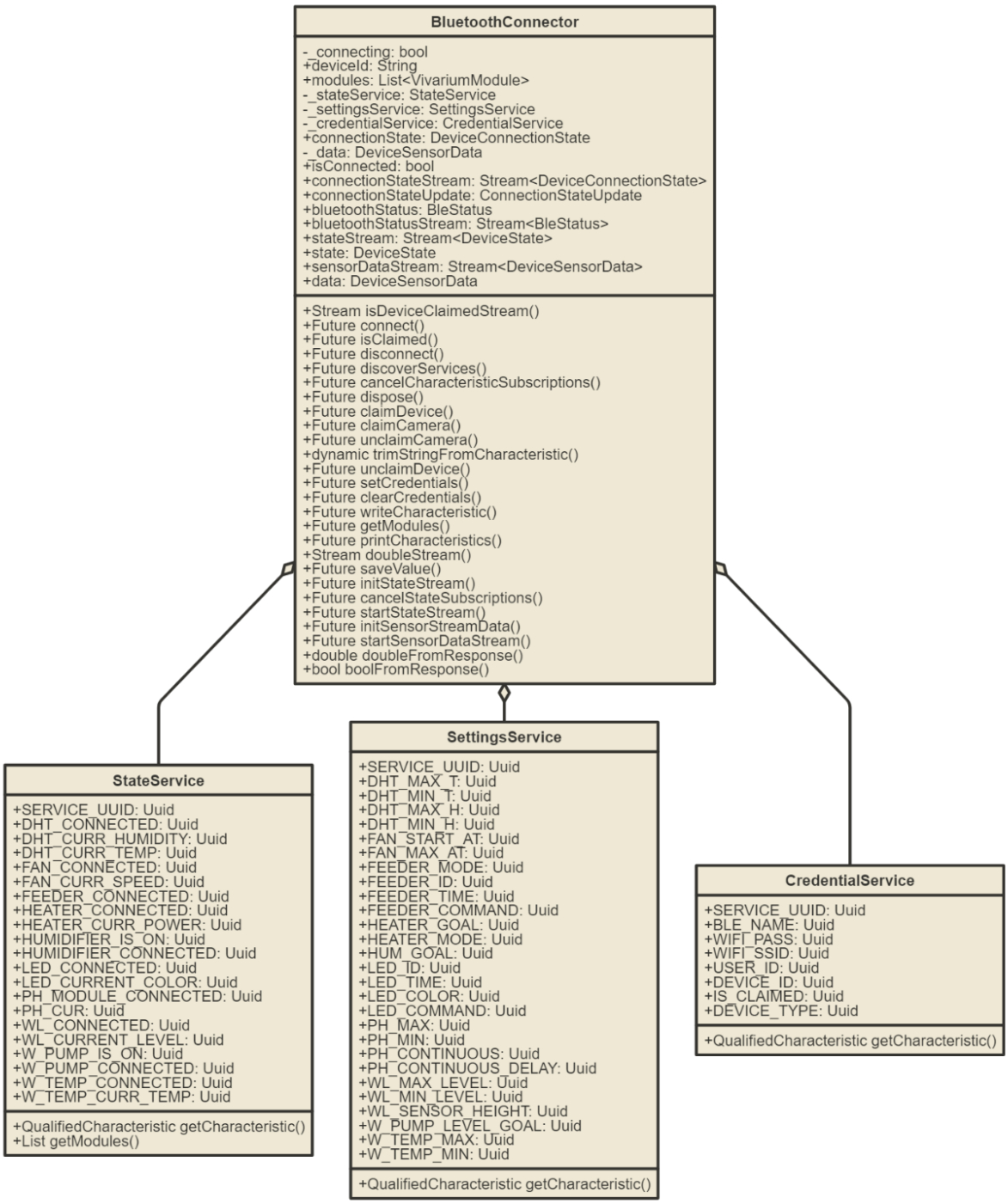
Kód 21 - Třída SensorDataHistory

4.2.8 Bluetooth

Pro připojení k Bluetooth byla vytvořena třída *BluetoothConnector*. Tato třída obstarává komunikaci mezi aplikací a Bluetooth zařízením, stará se o přihlášení k odběru charakteristik a jejich změny.

Konstruktor přijímá jako parametr MAC adresu Bluetooth zařízení, jedna instance tohoto objektu je tedy vytvářena pro každé zařízení, se kterým aplikace komunikuje.

Bluetooth zařízení inzeruje tři služby (State, Settings, Credentials) a každé této službě odpovídá jedna třída v kódu aplikace.



Kód 22 - Diagram třídy BluetoothConnector

Připojení k zařízení

Pro připojení k zařízení slouží funkce *connect*. Jedná se o asynchronní operaci a je potřeba zajistit, aby v jednu chvíli probíhala pouze jedna sekvence přihlášení. Funkce *connectToDevice* vystavená knihovnou *flutter_reactive_ble* vrací stream stavu přihlášení. Pro správnou funkcionalitu je vhodné v některých částech aplikace zareagovat ihned po úspěšném připojení. K tomuto účelu slouží funkce *onConnectionCallback* předaná v parametru.

```
Future<void> connect({VoidCallback onConnectionCallback}) async {
  if (_connecting) return;
  _connecting = true;

  await _connectionSubscription?.cancel();

  _connectionSubscription = _flutterReactiveBle
      .connectToDevice(id: deviceId, connectionTimeout: Duration(seconds: 5))
      .listen((event) async {
    connectionStateUpdate = event;
    if (event.connectionState == DeviceConnectionState.connected) {
      onConnectionCallback();
    }
  }, onError: (e) => print('Error = $e'));
  _connecting = false;
}
```

Kód 23 - Připojení pomocí Bluetooth

Přidání zařízení

Přidání zařízení probíhá v podobě nastavení uživatelského ID a přihlašovacích údajů pro Wi-Fi. Při zápisu do Bluetooth charakteristik se posílá naráz 20 bytů. ID uživatele může být delší než 20 znaků, proto je do řetězce přidán ukončující znak ‚*‘. Vivárium potom ukládá ID uživatele do bufferu do doby, než načte tento ukončující znak. V posledním kroku je dotaz na ID vivária, který také funkce vrátí. Pokud někde v průběhu procesu přiřazení vivária nastane chyba, vrátí funkce *null*.

```
Future<String> claimDevice(
    {@required String userId,
    @required String deviceId,
    @required String ssid,
    @required String password,
    @required String name}) async {
  try {

    await _flutterReactiveBle.writeCharacteristicWithResponse(
        _credentialService.getCharacteristic(
            deviceId, _credentialService.USER_ID),
        value: (userId + '*').codeUnits);

    await setCredentials(ssid: ssid, pass: password);

    await _flutterReactiveBle.writeCharacteristicWithResponse(
        _credentialService.getCharacteristic(
            deviceId, _credentialService.BLE_NAME),
        value: name.codeUnits);

    return trimStringFromCharacteristic(
        await _flutterReactiveBle.readCharacteristic(_credentialService
            .getCharacteristic(deviceId, _credentialService.DEVICE_ID)));
  } catch (e) {
    print(e);
    return null;
  }
}
```

Kód 24 - Spárování řídící jednotky s uživatelem

Přidání kamery

Přidání kamery probíhá stejným způsobem jako přidání vivária s tím rozdílem, že je kameře potřeba před navíc i ID vivária. Pod tímto ID pak ukládá pořízené fotografie, čímže je zajištěn bezpečný přístup pouze k relevantním datům. Pokud má kamera již nastaveného uživatele, nelze ji přiřadit nikomu jinému. Uživatel nejprve musí kameru uvést do továrního nastavení.

Změna hodnoty

Pro změnu charakteristiky slouží funkce *saveValue*. Zařízení Nejprve se podle klíče získá požadovaná Bluetooth charakteristiku a následně je do ní zapsána hodnota v podobě pole bytů.

```
Future<void> saveValue({String key, dynamic value}) async {
  var characteristic = getCharacteristic(key: key, deviceId: deviceId);
  if (characteristic != null)  {
    var values = value.toString().codeUnits;

    if (values.isNotEmpty) {
      await _flutterReactiveBle
          .writeCharacteristicWithResponse(characteristic, value: values);
    }
}
```

Kód 25 - Nastavení Bluetooth charakteristiky

4.2.9 Správa uživatelů

Registrace

Uživatel se může registrovat pomocí kombinace e-mailu a hesla nebo prostřednictvím svého Google účtu.

Při registraci pomocí e-mailu vrací metoda *createUserWithEmailAndPassword* objekt typu *UserCredentials*. Z toho objektu je získáno id uživatele, které je využito pro zapsání uživatelských dat do RTDB.

Přihlášení prostřednictvím Google probíhá ve dvou fázích. Nejprve se zavolá metoda *signIn*, která vrací *GoogleSignInAccount*. Na vráceném objektu je možné potom zavolat metodu pro autentizaci po níž dojde k získání *OpenID* tokenu a *OAuth2* přístupovému tokenu. Tyto dva tokeny je poté možné využít k přihlášení do Firebase. Pro zjištění prvního přihlášení je možné porovnat datum posledního přihlášení s datem vytvoření Firebase uživatele.

```
Future<VivariumUser> signInWithCredentialObj(Credentials credentials) async {
  if (credentials == null) return null;
  try {
    final authResult = await _firebaseAuth.signInWithCredential(
        GoogleAuthProvider.credential(
            accessToken: credentials.accessToken,
            idToken: credentials.idToken));
    final user = authResult.user;

    if (user.metadata.creationTime == user.metadata.lastSignInTime){
      _dbService.onNewUser(user.uid);
    }
    return _parseFromFirebaseUser(user);
  } catch (e) {
    return null;
  }
}
```

Kód 26 - Přihlášení uživatele prostřednictvím Google účtu

Přihlášení

V případě přihlášení prostřednictvím Google je scénář stejný jako při registraci. Pro použití e-mailu stačí zavolat *signInWithEmailAndPassword*.

Při každém přihlášení je potřeba do databáze uložit nový FCM token. Zároveň probíhá kontrola povolení zobrazení notifikací na zařízení.

Odhlášení

Odhlášení probíhá ve třech fázích. Nejprve je odhlášen uživatel přímo ze služby Firebase, poté se zkontroluje, zda není uživatel přihlášen též ke svému Google účtu a pokud ano, je taktéž odhlášen. V závěru je třeba odstranit z RTDB token pro FCM notifikace.

4.2.10 Správa vivárií

Pro práci s viváriem byla implementována třída *DeviceProvider*, která odděluje uživatelské rozhraní od logiky na pozadí aplikace. Změny se primárně propisují do databáze. Pokud je aplikace offline, zapisují se změny do paměti a po připojení je databáze aktualizována. Pokud je aplikace připojena k vivárium napřímo skrze Bluetooth, zapisují se změny přímo do vivária.

Přidání zařízení

Pro přidání vivária je potřeba mít k dispozici vivárium se spuštěným Bluetooth. Nejprve dojde ke spárování vivária s uživatelem prostřednictvím Bluetooth jak je popsáno výše v kapitole *Bluetooth*. Pokud proběhne proces v pořádku a aplikace získala validní ID vivária, je možné vytvořit záznam v RTDB. Nejprve dojde k získání seznamu všech modulů, kterými vivárium disponuje, poté je upravená verze výchozího objektu po nahrána do databáze. Pokud se nahrání nezdaří, zavolá se metoda, která uvede vivárium zpět do výchozího stavu.

Nastavení zařízení

Pro každou nastavitelnou položku vivária je v aplikaci přiřazen unikátní řetězec, který slouží jako klíč pro ukládání dat do databáze a zařízení skrze Bluetooth. Obsah řetězce odpovídá pozici v databázi, ale pro ukládání dat do vivária je potřeba konverze z klíče na UUID Bluetooth charakteristiky. Z uživatelského rozhraní je zavolána metoda *saveValue,* která zkontroluje stav Bluetooth spojení a případně uloží hodnotu do charakteristiky. Funkce končí nahráním dat do databáze bez ohledu na stav Bluetooth zařízení.

```
Future<bool> saveValue({String key, dynamic value}) async {
  if (_device == null) return false;


  if (_bluetoothConnector.isConnected) {
    await _bluetoothConnector.saveValue(key: key, value: value);
  }

  return await _databaseService.saveItem(
      value: value, deviceId: _device.info.id, key: key);
}
```

Kód 27 - Nastavení parametru vivária

Zobrazení dat

Získávání dat a jejich zobrazení uživateli je komplikovanější oproti jejich zápisu. Je potřeba zajistit změnu toku dat při připojení skrze Bluetooth. Data jsou uchovávána v objektu třídy *Device* a *DeviceProvider* poskytuje stream tohoto objektu.

Pokud není Bluetooth spojení aktivní, používá se aktivní stream z RTDB. Při připojení k viváriu napřímo jsou aktivovány streamy jednotlivých charakteristik. Jelikož každá charakteristika mění jen část objektu *Device*, byla implementována funkce *copyWith*, která umožnuje vygenerovat nový objekt z původního za využití nepovinných parametrů pro přepis jen části dat. Takto nově vytvořený objekt je předán streamu. Generování nového objektu je také potřeba kvůli způsobu, jakým stream funguje. Pokud je do streamu vložen původní objekt, nedojde k upozornění žádného z posluchačů streamu a žádné změny se tak v uživatelském rozhraní neprojeví.

Kontrola online stavu

Aplikace umožňuje zároveň sledovat online stav registrovaných zařízení a informovat o jejich stavu uživatele. Tato funkce by měla být nezávislá na hlavním programu, je tedy potřeba vytvořit samostatnou službu.

Dokumentace pro Android specifikuje tři typy služeb: [32]

- *Foreground* – služba je viditelná pro uživatele, po dobu běhu služby je zobrazena notifikace v informační liště. Služba běží i po zavření aplikace. Notifikace nelze z informační lišty odebrat ručně, odebere se automaticky buď vypnutím služby nebo jejím přesunem do pozadí.
- *Background* – běží na pozadí bez vědomí uživatele. Android verze 8.0 ukládá nová pravidla pro běh služby na pozadí pro zlepšení optimalizace baterie. Služba na pozadí může volně běžet pouze pokud je aplikace na popředí. Jakmile ji uživatel zavře, může systém značně omezit a případně i odložit přidělení času procesoru. Požadovaný proces tak může proběhnout se zpožděním až několik hodin.
- *Bound* – služba vázáná na aplikace. Běží, dokud existuje alespoň jedna aktivní komponenta svázaná s touto službou.

Kontrola internetové konektivity zařízení je kritická funkce pro uživatele a měla by běžet bez přerušení i po zavření aplikace. Z tohoto důvodu bylo rozhodnuto implementovat službu běžící na pozadí se zobrazenou notifikací.

Pro vytvoření servisy byl použit plugin *flutter_background_service*, který usnadňuje zavedení a obsluhu služby na popředí, zajišťuje posílání dat mezi hlavní aplikací a službou a umožňuje editovat vystavenou notifikaci.

Služba je vytvořena při spouštění aplikace. Dojde k inicializaci vlastní *FirebaseApp* instance, poté jsou vytvořeny odběry dat ze streamu pro posílání dat z hlavního vlákna a stavu připojení k internetu. Nakonec je přidán časovač, který v pravidelném intervalu spouští hlavní kód služby. V něm je zkontrolován stav připojení mobilního zařízení k internetu, zda je uživatel v aplikaci přihlášen, a nakonec jsou zkontrolována vivária. Uživateli je poté zobrazen jejich stav.

4.2.11 Testování

Flutter umožňuje snadné testování kódu pomocí *flutter_test* knihovny. Každý test začíná zavoláním funkce *testWidgets*, která jako parametry přijímá popis testu a anonymní vnitřní funkce, kde probíhá testování. Funkce pro testování přijímá jako parametr objekt typu *WidgetTester*, který slouží pro nastavení testovaného widgetu, změnu jeho stavu a následný test.

Na začátku testu je nutné nakonfigurovat emulátor testovaného zařízení včetně velikosti displeje. Poté se předá testovaný objekt metodou *pumpWidget* zavolanou nad objektem *WidgetTester*. K testování slouží dvě třídy – *Matcher* a *Finder*. *Finder* vyhledává widgety ve struktuře testovaného objektu podle zadaných podmínek a *Matcher* pak slouží ke kontrole nalezených hodnot.

```
void testHomePage(Size size) {
  testWidgets('HomePage Without User', (WidgetTester tester) async {
    final TestWidgetsFlutterBinding binding =
        TestWidgetsFlutterBinding.ensureInitialized();
    await binding.setSurfaceSize(size);

    await tester.pumpWidget(MediaQuery(
        data: MediaQueryData(),
        child: MaterialApp(
            home:
                Provider<VivariumUser>.value(value: null, child: HomePage()))));

    expect(find.text('Sign in with Google'), findsOneWidget);
    expect(find.text('Devices'), findsNothing);
  });

  testWidgets('HomePage With User', (WidgetTester tester) async {
    final TestWidgetsFlutterBinding binding =
        TestWidgetsFlutterBinding.ensureInitialized();
    await binding.setSurfaceSize(size);

    await tester.pumpWidget(MediaQuery(
        data: MediaQueryData(),
        child: MaterialApp(
            home: Provider<VivariumUser>.value(
                value: VivariumUser(userEmail: 'user@email.cz'),
                child: HomePage()))));

    expect(find.text('Sign in with Google'), findsNothing);
    expect(find.text('Devices'), findsOneWidget);
    expect(find.text('user@email.cz'), findsOneWidget);
  });
}
```

Kód 28 - Ukázka testování widgetů ve Flutteru

4.3 Kamera

Pro programování kamery na čipu ESP32-S bylo použito jádro Arduino pro ESP32 vyvíjené firmou Espressif Systems, které umožňuje psát kód v jazyce Wiring stejně jako pro kteroukoliv desku Arduino.

Firebase nepodporuje streamování obrazu a nebyla nalezena žádná vyhovující alternativa, proto bylo rozhodnuto ke snímání fotografie v pravidelném intervalu a jejímu nahrání do Firebase Storage, odkud si ji aplikace uživatele může stahovat pro následné zobrazení.

Původní návrh počítal s paralelním během Bluetooth i Wi-Fi, ale ukázalo se, že v takovém režimu není dostatek paměti pro spolehlivé odesílání pořízených fotografií. Důvodem byla vysoká paměťová náročnost šifrovaného spojení s Firebase, kdy objekt udržující informace o SSL připojení může mít velikost až 100 kB. Inicializovaný Bluetooth stack je taktéž paměťově náročný. Přestože bylo místo původního Bluedroidu pro ESP využito úspornějšího NimBLE stacku, který snižuje využitou RAM o 100 KB, stále neměl mikrokontroler dostatek volné paměti. Odeslání jedné fotografie o velikosti 300 kB trvalo 15 vteřin a většinou

nahrávání fotografie skončilo neúspěšně. Bylo tedy rozhodnuto, že v jednu chvíli může být spuštěna jen jedna služba a přepínání mezi těmito službami je zajištěno tlačítkem. Po implementaci přepínání mezi Wi-Fi a Bluetooth se nahrání zdařilo téměř vždy a doba nahrání se zkrátila na 3–4 vteřiny.

4.3.1 Běh programu

Po svém spuštění kamera ověří, zda má v paměti uložená data nastavení. Pokud žádná data nenajde, zapne se Bluetooth server. V případě úspěšného načtení svého nastavení se kamera pokusí přihlásit k Wi-Fi a zahájí cyklus nahrávání fotografií. Kamera se v pravidelných intervalech dotazuje Firebase databáze, zda nedošlo ke vzdálenému vymazání údajů uživatelem skrze aplikaci. V takovém případě smaže uložené údaje a přepne se do režimu Bluetooth.

4.3.2 Kód

Následující schéma zobrazuje strukturu kódu a závislostí jednotlivých částí. Hlavní objektem je *VivariumCamera*, který se stará o nastavení kamery a inicializaci ostatních objektů.

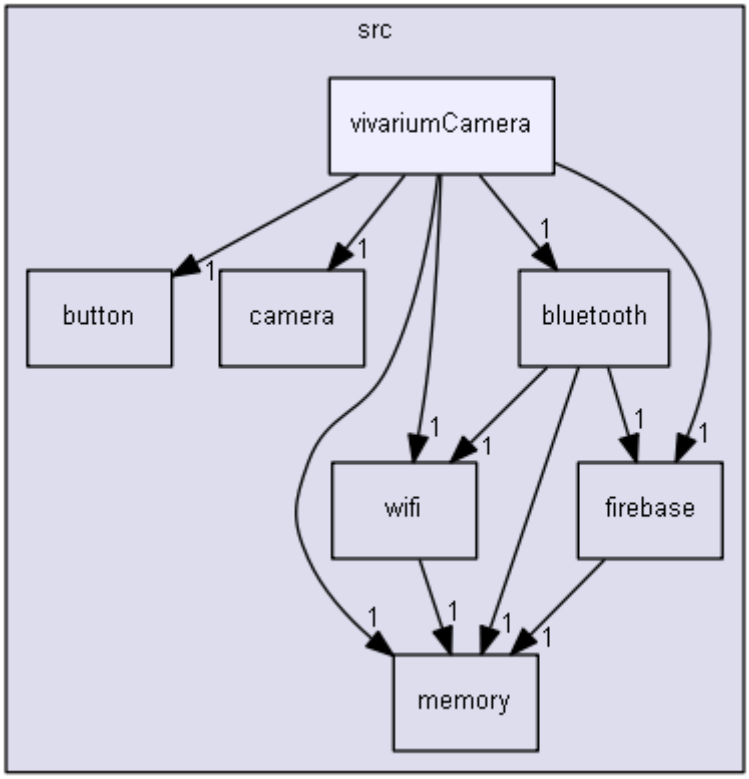


Schéma 8 - Schéma závislostí tříd pro firmware kamery

VivariumCamera

Hlavní třída kamery má dvě veřejné funkce: *setup* a *onLoop*. První funkce slouží pro inicializaci ostatních objektů kamery, jak je vidět z následující ukázky kódu. Pokud se mikrokontroleru nepodaří připojit k Wi-Fi, je automaticky zapnuto Bluetooth pro připojení aplikace a zadání přihlašovacích údajů.

```
void VivariumCamera::onLoop()
{
    checkClearButton();
    checkToggleButton();
    checkCamera();
    if (_bleService->checkDisconnected())
    {
        if (_wifiProvider->setup())
        {
            _firebaseService->start();
        }
        else
        {
            _bleService->setup();
        }
    }
}
```

Kód 29 - Hlavní smyčka programu kamery

Metoda *onLoop* je pak volána pravidelně z metody *loop* mikrokotroleru a má tři hlavní funkce:

- Zeptá se objektu třídy *ButtonController*, zda od posledního dotazu nedošlo ke stisku tlačítka pro úplný reset a zapomenutí přihlašovacích údajů. Pokud ano, řekne objektům tříd *WiFi* a *FirebaseService*, aby smazaly svá data.
- Zeptá se objektu třídy *ButtonController*, zda od posledního dotazu nedošlo ke stisku tlačítko pro přepnutí mezi Bluetooth a WiFi. V případě kladné odpovědi řekne objektům spravující tato vysílání, aby přepnuly svůj stav.
- V pravidelném intervalu si nechá od objektu třídy *CameraHandler* vygenerovat novou fotku, kterou posléze předá do instance *FirebaseService* pro další zpracování

Memory

Třída pro správu přístupových údajů. Prostřednictvím metody *saveString(String klíč, String hodnota)* ukládá řetězce do nevolatilní paměti, odkud mohou být pomocí metody

getString(String klíč, String defaultní_hodnota) vráceny. Pro úplné smazání klíče z nevolatilní paměti slouží metoda *clearKey(String klíč)*.

WifiProvider

Třída pro obsluhu Wi-Fi. Pracuje s přístupovými údaji k AP. Po zavolání funkce *setup* získá přístupové údaje od třídy *Memory* a následně se pokusí přihlásit. Funkce vrací informaci o tom, zda přihlášení k Wi-Fi proběhlo v pořádku.

```
bool WifiProvider::setup()
{
    loadCredentials();
    return startWifi() == WL_CONNECTED;
}
```

Kód 30 - Nastavení Wi-Fi u kamery

CameraHandler

CameraHandler má tři veřejné metody. Ve funkci *setup* dochází k inicializaci ESP kamery. Funkce *getPhoto* se stará o naplnění bufferu daty z pořízené fotografie. Při úspěšném pořízení fotografie je vrácena hodnota true.

```
void VivariumCamera::checkCamera()
{
    if (millis() - last_capture_millis > capture_interval && _firebaseService->isRunning())
    {
        last_capture_millis = millis();
        camera_fb_t *buffer = nullptr;
        if (_cameraHandler->getPhoto(&buffer))
        {
            _firebaseService->uploadPhoto(buffer);
        }
        _cameraHandler->clearBuffer();
        checkCameraActive();
    }
}
```

Kód 31 - Pořízení fotografie

Aby nedošlo k zaplnění paměti, je třeba po zpracování obsah bufferu vyčistit, k tomu slouží metoda *clearBuffer*.

ButtonController

V této třídě je spravován stav tlačítek, která jsou prezentována následující strukturou:

```
struct Button
{
    int pin;
    bool pressed;
    int last_val;
    unsigned long last_time;
};
```

Kód 32 - Struktura tlačítka

Proměnná *last_time* je potřeba pro takzvaný odskok tlačítka. Tlačítko po svém stisku může způsobit několikanásobnou změnu stavu na pinu mikrokontroleru a je tedy žádoucí po určitý čas od změny stavu jakékoliv následující stavy ignorovat. V rámci této práce je interval odskoku nastaven na 200 milisekund.

Po zavolání veřejné metody *setup* dojde k vytvoření nové úlohy běžící v samostatném vlákně. Tato úloha neustále kontroluje digitální hodnotu na pinech připojeným k tlačítkům a při změně hodnoty aktualizuje stav tlačítka.

BluetoothService

BluetoothService se stará o Bluetooth připojení. Základní metodou této třídy je *setup*, uvnitř které dojde k inicializaci Bluetooth serveru a služeb spolu s jejími charakteristikami. Server vystavuje jednu službu se šesti charakteristikami. Jedna charakteristika je pouze pro čtení a jsou jí tedy přiřazeny vlastnosti pro šifrované ověřené čtení a pro notifikaci, což umožní mobilní aplikaci přihlásit se k odběru stavu této charakteristiky. Jedná se o charakteristiku popisují, zda již byla kamera přivlastněna některým uživatelem. Zbylé charakteristiky (ID uživatele, ID zařízení, Wi-Fi SSID a Wi-Fi heslo) jsou pouze pro šifrovaný a ověřený zápis bez možnosti čtení.

FirebaseService

FirebaseService využívá knihovnu *Firebase_ESP_Client* pro připojení ke cloudovým službám Firebase. Aby bylo připojení zabezpečené, je potřeba správně knihovnu nakonfigurovat. Mikrokontroler vykonává jen specifické úkony na Firebase cloudu, přičemž k Firebase se přihlašuje prostřednictvím administrátorského účtu. Aby bylo možné šifrovaně komunikovat se službami Firebase Storage a RTDB, je potřeba poskytnout platný SSL certifikát. Jelikož tyto služby mají rozdílné certifikáty, je potřeba nastavovat *FirebaseData* objektu správný certifikát za běhu programu podle potřeby.

```
    firebaseAuth.user.email = USER_EMAIL;
    firebaseAuth.user.password = USER_PASSWORD;

    firebaseConfig.api_key = API_KEY;
    firebaseConfig.time_zone = 2;
    firebaseConfig.database_url = FIREBASE_HOST;
    firebaseConfig.token_status_callback = tokenStatusCallback;
    firebaseConfig.max_token_generation_retry = 5;
    Firebase.reconnectWiFi(true);

    Firebase.begin(&firebaseConfig, &firebaseAuth);
```

Kód 33 - Inicializace Firebase

Třída *FirebaseService* slouží k nahrání pořízené fotografie a kontrole přivlastnění kamery. Mikrokontroler se v pravidelném intervalu dotazuje na hodnotu v databázi na cestě */devices/{deviceId}/camera/active*. Pokud je hodnota nastavena na *false*, znamená to, že uživatel v aplikaci odstranil tuto kameru, a dojde tedy k vymazání přístupových údajů, neboť další běh tohoto zařízení není potřeba. Pro další fungování kamery je potřeba ji znovu přihlásit.

4.4 Vivárium

Kód je navržen s důrazem na co největší univerzálnost ovládací jednotky. V praxi to znamená, že stejný kód je využitelný pro jakýkoliv druh vivária bez ohledu na použitelné moduly. Každý modul je reprezentován jednou třídou, je tak možné vytvořit různé hardwarové variace kontrolní jednotky a v metodě *setup* inicializovat objekty použitých modulů. Zbytek kódu se potom nemění. Tento způsob umožňuje případné přidání dalších modulů do budoucna podle potřeby, aniž by bylo třeba zasáhnout do kódu aplikace jinak než přidáním nové třídy a její inicializace.

Jádrem řídící jednotky je obsluha Wi-Fi a Bluetooth spolu s kontrolou aktualizací firmwaru. Navrženy jsou taktéž pomocné třídy pro kontrolu paměti, stavu programu a ovládání tlačítek kontrolního panelu.

4.4.1 Bluetooth

Bluetooth je jednou z nejdůležitějších částí vivária. Ovládání Bluetooth je zajištěno třídou *BleController*. Povinnosti této třídy se dají rozdělit na několik částí:

- Inicializace a deinicializace Bluetooth serveru
- Vytvoření služeb pro moduly vivária
- Udržovat charakteristik všech modulů a upozorňovat je na změny připojení

Inicializace

Při inicializaci dojde k vytvoření Bluetooth serveru s nutností zabezpečeného spárování. Upřesněním možností vstupů a výstupu zařízení na hodnotu BLE_HS_IO_DISPLAY_ONLY je zajištěno, že bude ke správnému spárování zařízení potřeba opsat vygenerovaný PIN zobrazený na LCD obrazovce vivária.

```
void BLEController::init()
{
    _textOutput->setText({"Bluetooth", "Initializing."});
    _bluetoothName = _memoryProvider->loadString(BLE_NAME_KEY, DEFAULT_BLE_NAME);

    NimBLEDevice::init(_bluetoothName.c_str());
    NimBLEDevice::setPower(ESP_PWR_LVL_P9);
    NimBLEDevice::setSecurityIOCap(BLE_HS_IO_DISPLAY_ONLY);
    NimBLEDevice::setSecurityAuth(BLE_SM_PAIR_AUTHREQ_SC | BLE_SM_PAIR_AUTHREQ_BOND);

    // Create the BLE Server
    pServer = NimBLEDevice::createServer();
    pServer->setCallbacks(new VivariumServerCallbacks());

    _textOutput->setText({"Bluetooth", "Initializing."});
    setupModuleServices();
    _textOutput->setText({"Bluetooth", "Starting.."});
    pServer->getAdvertising()->start();

    _running = true;
    _initialized = true;

    _ledControl->setLedOn(BLUETOOTH_BUTTON);

    _textOutput->setText({"Bluetooth ON", "Ready..."});
}
```

Kód 34 - Inicializace Bluetooth serveru

Tvorba charakteristik

Před spuštěním samotného serveru je potřeba vytvořit službu a jejich charakteristiky v závislosti na zvolených modulech. Kromě zabezpečení samostatného serveru lze nastavit přístup k jednotlivým charakteristikám. Aby bylo dosaženo jednotného zabezpečení napříč všemi moduly, došlo k implementaci dvou funkcí podle Bluetooth služeb *createStateCharacteristic* a *createSettingsCharacteristic*. Obě funkce přijímají jako parametr ukazatel na instanci *NimBLEService*, UUID nově vytvářené charakteristiky a volitelně i třídu implementující zpětná volání na změny dané charakteristiky. Charakteristiky obou služeb mají společné čtení v šifrovaném spojení autentizovaném pomocí hesla. Zároveň umožňují posílat informace o změnách připojeným klientům. Služba pro nastavení mikrokontroleru navíc přidává možnost zapisovat do svých charakteristik pomocí šifrovaného a autentizovaného spojení.

```
NimBLECharacteristic *IBluetooth::createSettingsCharacteristic(NimBLEService *service,
const char *uuid,
NimBLECharacteristicCallbacks *callbacks)
{
    NimBLECharacteristic *characteristic = service->createCharacteristic(uuid,
                                                          NIMBLE_PROPERTY::READ |
                                                          NIMBLE_PROPERTY::READ_AUTHEN |
                                                          NIMBLE_PROPERTY::READ_ENC |
                                                          NIMBLE_PROPERTY::WRITE |
                                                          NIMBLE_PROPERTY::WRITE_AUTHEN |
                                                          NIMBLE_PROPERTY::WRITE_ENC |
                                                          NIMBLE_PROPERTY::NOTIFY |
                                                          NIMBLE_PROPERTY::INDICATE);
    characteristic->setCallbacks(callbacks);
    return characteristic;
}

NimBLECharacteristic *IBluetooth::createStateCharacteristic(NimBLEService *service, const char
*uuid, NimBLECharacteristicCallbacks *callbacks)
{
    NimBLECharacteristic *characteristic = service->createCharacteristic(uuid,
                                                          NIMBLE_PROPERTY::READ |
                                                          NIMBLE_PROPERTY::READ_AUTHEN |
                                                          NIMBLE_PROPERTY::READ_ENC |
                                                          NIMBLE_PROPERTY::NOTIFY |
                                                          NIMBLE_PROPERTY::INDICATE);
    characteristic->setCallbacks(callbacks);
    return characteristic;
}
```

Kód 35 - Tvorba BLE charakteristik

4.4.2 Wi-Fi

Třída *WiFiProvider* má na starosti připojení k internetu. Přihlašovací údaje k Wi-Fi se nastavují skrze Bluetooth, proto třída dědí z *IBluetooth* a přidává dvě charakteristiky pro SSID a heslo.

Připojení k internetu zajišťuje funkce *setupWiFi*. Pokud má *WiFiProvider* uložené SSID sítě, pokusí se přihlásit. Přihlášení probíhá v samostatné úloze v režii operačního systému. Z toho důvodu následuje za přihlášením smyčka, při které je pravidelně volán dotaz na stav připojení. Smyčka skončí buď vypršením limitu nebo připojením k Wi-Fi.

4.4.3 Firebase

Třída Firebase s pomocí knihovny Firebase-ESP-Client zajišťuje komunikaci s Firebase cloudovými službami – autentifikace, databáze a FCM.

Knihovna umožňuje přihlášení pomocí servisního nebo uživatelského účtu. Servisní účet vyžaduje e-mailovou adresu spolu s vygenerovaným soukromým klíčem. Uživatel se pak přihlašuje prostřednictvím kombinace e-mailu a hesla. V obou případech dochází k vygenerování přístupového tokenu, který je používám pro další volání. Token má platnost 60 minut, proto je každých 57 minut obnovován. Při generování tokenu pro servisní účet docházelo k občasným chybám na straně Firebase, které knihovna nedokázala zachytit a způsobovala zamrzání mikrokontroleru. Ani oprava knihovny jejím autorem neodstranila veškeré problémy, proto bylo přistoupeno k vytvoření speciálního uživatele.

Stream dat

Pro sdílení dat mezi mikrokontrolerem a Firebase databází je potřeba využít dva *FirebaseData* objekty, každý pro jeden směr komunikace. *FirebaseStreamBdo* reprezentuje vivárium v databázi. Změny v databázi streamovány do tohoto objektu a při každé změně je zavolána callback funkce, kde je možno na tyto změny reagovat. Kontroluje se přepínač do továrního nastavení, verze firmwaru, názvu Bluetooth zařízení a nastavení jednotlivých modulů. Každá změna je reprezentována hodnotou uloženou v řetězci, datovým typem a cestou k uložené hodnotě v databázi. Kromě primitivních datových typů může být změna reprezentována též JSON objektem. Jelikož změny v JSON objektu mohou reprezentovat nastavení více modulů, je potřeba rozdělit následující zpracování do dvou odlišných funkcí: *jsonCallback* a *valueCallback*.

```
void streamCallback(MultiPathStream stream)
{
    checkActiveStatus(stream);
    checkFirmwareVersion(stream);
    checkBluetoothName(stream);
    checkModules(stream);
}

void checkModules(MultiPathStream stream)
{
    for (int i = 0; i < 2; i++)
    {
        stream.get(firebaseService->childPaths[i]);
        if (stream.type == "json")
        {
            FirebaseJson json;
            json.setJsonData(stream.value);
            firebaseService->jsonCallback(&json, stream.dataPath);
        }
        else if (stream.type != "")
        {
            firebaseService->valueCallback(stream.dataPath, stream.type, stream.value);
        }
    }
}
```

Kód 36 - Stream dat do vivária

Odesílání dat

Aby byl zajištěn plynulý chod vivária, bylo odesílání dat do Firebase přesunuto do samostatného vlákna. Knihovna Firebase-ESP-Client nabízí možnost asynchronní odesílání dat bez ověřování odpovědi serveru, tato metoda se však ukázala jako nespolehlivá a odchozí data se do databáze nenahrála. Proto byla vytvořena třída *FirebaseSender* přijímající data pro odeslání. Data jsou ukládána do fronty, odkud se pravidelně vybírají a posílají do databáze. Velikost zásobníku paměti tohoto vlákna byla nastavena na 6500 bytů. Odesílání dat může trvat i několik vteřin, a proto bylo potřeba v rámci vlákna prodloužit WDT, který by se jinak spouštěl předčasně. V rámci testování se ukázalo 20 vteřin jako dostačující. Pro zajištění vícevláknového přístupu k frontě dat je využit zámek v podobě semaforu. Semafor je zajištěný operačním systémem FreeRTOS.

Data jsou do fronty přidávána dvěma způsoby. Pokud proběhne změna v modulu, může okamžitě předat informaci třídě *FirebaseService*, která přidá nové údaje do fronty. Zároveň je

v pravidelném intervalu vytvářen JSON objekt, do kterého jednotlivé moduly postupně do zapíší svůj aktuální stav. Tento objekt je poté poslán do databáze, kde se ukládá pod časovou známkou tak, aby byla uchována historie stavu vivária.

Odesílání FCM

Třída *FirebaseSender* slouží také k odeslání FCM zpráv. Princip je stejný jako v případě odesílání dat. Modul předá zprávu *MessagingService*, kde jsou zkontrolovány podmínky pro odeslání zprávy. Jsou definovány 3 typy zpráv podle různých událostí: spuštění alarmu, překročení limitu a připojení/odpojení modulu. Uživatel si může vybrat, na které z těchto událostí chce získávat notifikace v zařízení a jak často se mají tyto notifikace zobrazovat. Pokud zpráva projde validací podmínek, je předána třídě *FirebaseSender*, která ji zařadí do fronty pro odeslání zpráv.

4.4.4 Paralelní vlákno

Během vývoje nastala potřeba oddělit několik dílčí částí kódu do samostatného vlákna. Jednalo se především o podpůrné části monitorující stav mikrokontroleru. S ohledem na omezenou paměť mikrokontroleru bylo rozhodnuto pro vytvoření jedné třídy *ParallelRunner*, která může obsluhovat několik činností podle aktuální potřeby. Třída spustí jediné vlákno pravidelně se dotazující na dostupné činnosti.

```
#define MAX_CALLBACKS 3
typedef void (*callback)(void);
class ParallelRunner
{
public:
    ParallelRunner();
    bool addCallback(callback);
    void startRunner();
    callback getNextCallback();
private:
    int _lastCallback = 0;
    int _callbackCount = 0;
    callback _callbacks[MAX_CALLBACKS];
};
extern ParallelRunner runner;
```

Kód 37 - Deklarace třídy ParallelRunner

Při inicializaci mikrokontroleru jsou instanci této třídy přiřazeny callback funkce. Následně je zavolána funkce *startRunner*, která spouští novou RTOS úlohu.

4.4.5 Rozhraní modulů

Každý modul je představován svou vlastní třídou, která dědí z několika abstraktních tříd podle potřebných vlastností.

IModule

Každý modul je rozšiřován touto abstraktní třídou. Konstruktor vyžaduje řetěz pro načtení stavu připojení z paměti, pořadí konektoru modulu na ovládacím panelu a ukazatel na abstraktní třídu *MemoryProvider*, která slouží pro komunikaci s NVM pamětí. Třída se stará o následující:

- Uchování informací o stavu zapojení modulu včetně načtení poslední hodnoty z paměti
- Zajišťuje komunikaci se třídou *LedControl*, která obsluhuje indikační diody ovládacího panelu

Jednotlivé moduly musí ve svém kódu obsahovat implementaci čistě virtuálních funkcí *onConnectionChange, onLoop* a volitelně také implementaci virtuální funkce *beforeShutdown*. Funkce *onConnectionChange* slouží k reakci modulu na změnu připojení, ať akcí uživatele stiskem tlačítka na ovládacím panelu, nebo prostřednictvím aplikace. Funkce *onLoop* je volána při každé iteraci ze třídy *ModuleControl* a obsahuje hlavní logiku každého modulu. Implementace *beforeShutdown* je volitelná. Pokud se program dostane do stavu, kdy je potřeba provést řízený restart zařízení, dojde k zavolání této metody na všech dostupných modulech. Toho lze využít například při PID řízení teploty, kdy je potřeba uložit naměřenou hodnotu regulátoru, aby ji bylo možné po spuštění znovu načíst.

IBluetooth

Třída rozšiřující modul o možnost komunikovat prostřednictvím Bluetooth. Pro vytvoření Bluetooth serveru slouží dříve zmíněná třída *BLEController*, která dopředu potřebuje znát počty charakteristik všech modulů, aby pro ně mohla rezervovat dostatečnou velikou část paměti. Každý modul rozšiřující třídu *IBluetooth* tedy implementuje čistě virtuální funkci *getHandlesCount*, jejíž parametry jsou tři ukazatele na proměnné typu int. Pro vlastní inicializaci Bluetooth charakteristik je možné podle potřeby implementovat až tři virtuální funkce – *setupBLECredentials, setupBLESettings, setupBLEState*. V těchto metodách probíhá vytvoření vlastních charakteristik pro příslušné Bluetooth služby.

Aby mohl modul reagovat na změny v Bluetooth připojení, musí obsahovat implementaci čistě virtuálních funkcí *onBLEDisconnect* a *onBLEConnect*. Tyto funkce slouží většinou pro uložení změn v nastavení do paměti mikrokontroleru po ukončení Bluetooth spojení a pro aktualizaci hodnot v Bluetooth charakteristikách při navázání nového spojení.

IFirebaseModule

IFirebaseModule rozšiřuje moduly o možnost zasílání notifikací na mobilní aplikaci uživateli a taktéž komunikaci s databází. Obsahuje deklaraci čtyř čistě virtuální funkcí. Funkce *parseJson* a *parseValue* slouží pro zpracování přijatých dat z databáze. JSON objekt je přijat v případě většího množství změn naráz. Pokud nastane změna pouze u jedné hodnoty, je z databáze přijata jen dvojice cesta-hodnota. Vrácený řetězec funkce *getSettingsKey* slouží k určení správného příjemce získaných dat. K pravidelnému posílání dat do databáze slouží *updateSensorData*. Modul přijme ukazatel na JSON objekt, do kterého může zapsat aktuální hodnoty ze senzorů.

```
class IFirebaseModule
{
public:
    IFirebaseModule(){};
    virtual ~IFirebaseModule(){};

    virtual void parseJson(FirebaseJson *json, String path) = 0;
    virtual String getSettingKey() = 0;
    virtual void parseValue(String key, String value) = 0;
    virtual bool updateSensorData(FirebaseJson *json) = 0;
    void addFirebaseService(FirebaseService *service) { firebaseService = service; };
    void addMessagingService(MessagingService *service) { messagingService = service; };

protected:
    FirebaseService *firebaseService = nullptr;
    MessagingService *messagingService = nullptr;

    void sendConnectionChangeNotification(String module, bool connected)
    {
        String title = "Module " + module;
        if (messagingService != nullptr)
            messagingService->sendFCM(title, connected ? "Module connected " : "Module
disconnected ", FCM_TYPE::CONNECTION, module);
    }
};
```

Kód 38 - IFirebaseModule

TextModule

Slouží pro tisknutí textů na displej vivária. Nekomunikuje s displejem napřímo, ale využívá implementaci třídy *TextOutput*, což zajišťuje oddělení logiky od hardwarového řešení. Modul může zobrazit text z vlastní iniciativy. Například pokud uživatel ručně vyvolá čtení ze senzoru, modul zobrazí aktuální hodnu a po určitou dobu pozastaví změnu textu na displeji.

Čistě virtuální funkce *getText* slouží pro pravidelné střídání informací o jednotlivých modulech. Třída obsluhující displej v pravidelném intervalu postupně volá tuto funkci nad každým modulem a získanou informaci posléze zobrazí.

Alarm

Některé moduly využívají plánovaných akcí v určitý čas, například přepínání osvětlení, krmení či vytápění. Vyvstala proto potřeba vytvořit systém pro spínání těchto událostí.

Základní informace o budíku je uložena ve struktuře *Trigger*, která uchovává čas spouštění, ID pro uložení do NVS paměti, ID pro práci s knihovnou *TimeAlarms* a klíč pro práci s databází. Pokud je pro správnou funkci potřeba další údaj, je možné využit rozšíření *PayloadTrigger*. Rozšíření *PayloadTrigger* lze využít například u LED osvětlení, kdy je uložena hodnota nové barvy osvětlení.

```
struct Trigger
{
    int storageId = INVALID_MEMORY_ID;
    int hour;
    int minute;
    AlarmId id;
    String firebaseKey;
};

template <typename T>
struct PayloadTrigger : public Trigger
{
    T payload{};
};
```

Kód 39 - deklarace struktur Trigger a PayloadTrigger

Pro maximální univerzálnost bylo rozhodnuto vytvořit abstraktní šablonu třídy *BaseAlarm<T>*, která udržuje informace o všech událostech pro daný modul. Tyto události jsou spouštěny formou budíků, kdy je při každé iteraci hlavní smyčky vznesen dotaz na třídu *TimeAlarmsClass* z knihovny *TimeAlarms*. V případě, že aktuální čas překročí čas budíku, je

zavolána dříve přiřazená callback funkce. *BaseAlarm* udržuje budíky v asociativním poli, kdy je klíčem řetězec shodný s *firebaseKey* a hodnotou samotný budík.

Kvůli své komplexitě bylo rozhodnuto pro omezení nastavení pouze skrze aplikaci prostřednictvím Firebase. Změny probíhají dvěma způsoby – aktualizací jedné hodnoty či získání celého JSON objektu. Získaná hodnota je využita pro aktualizaci již existujícího budíku. JSON objekt je využit pro vytvoření nového budíku, případně pro aktualizaci celého asociativního pole.

Vzhledem k rozdílu mezi *PayloadTrigger* a *Trigger* je potřeba individuální přístup pro správu budíků. Proto byly vytvořeny třídy *PayloadAlarm* a *PlainAlarm* rozšiřující základní třídu *BaseAlarm*.

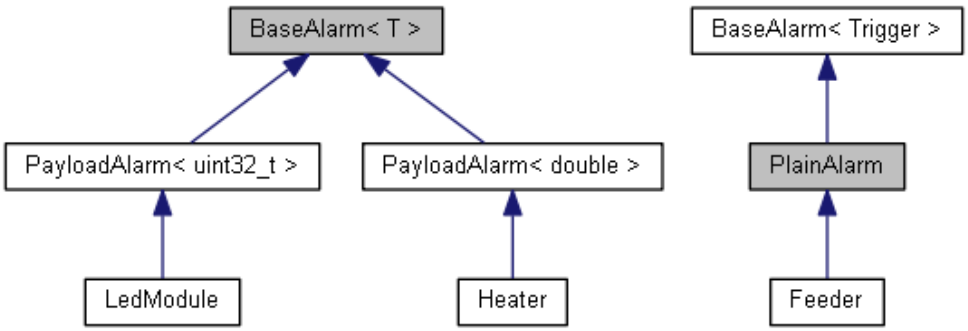


Schéma 9 - Dědičnost třídy BaseAlarm

Třídy byly napsány takovým způsobem, aby byl počet čistě virtuálních funkcí, které je potřeba implementovat přímo v modulu, co nejmenší. Při použití *PlainAlarm* byla tato potřeba eliminována, moduly rozšiřující *PayloadAlarm* musí obsahovat implementaci pouze dvou funkcí: *getPayloadFromJson* a *getPayloadFromValue*. Následující schéma ukazuje životní cyklus změny budíku prostřednictvím streamu z RTDB.

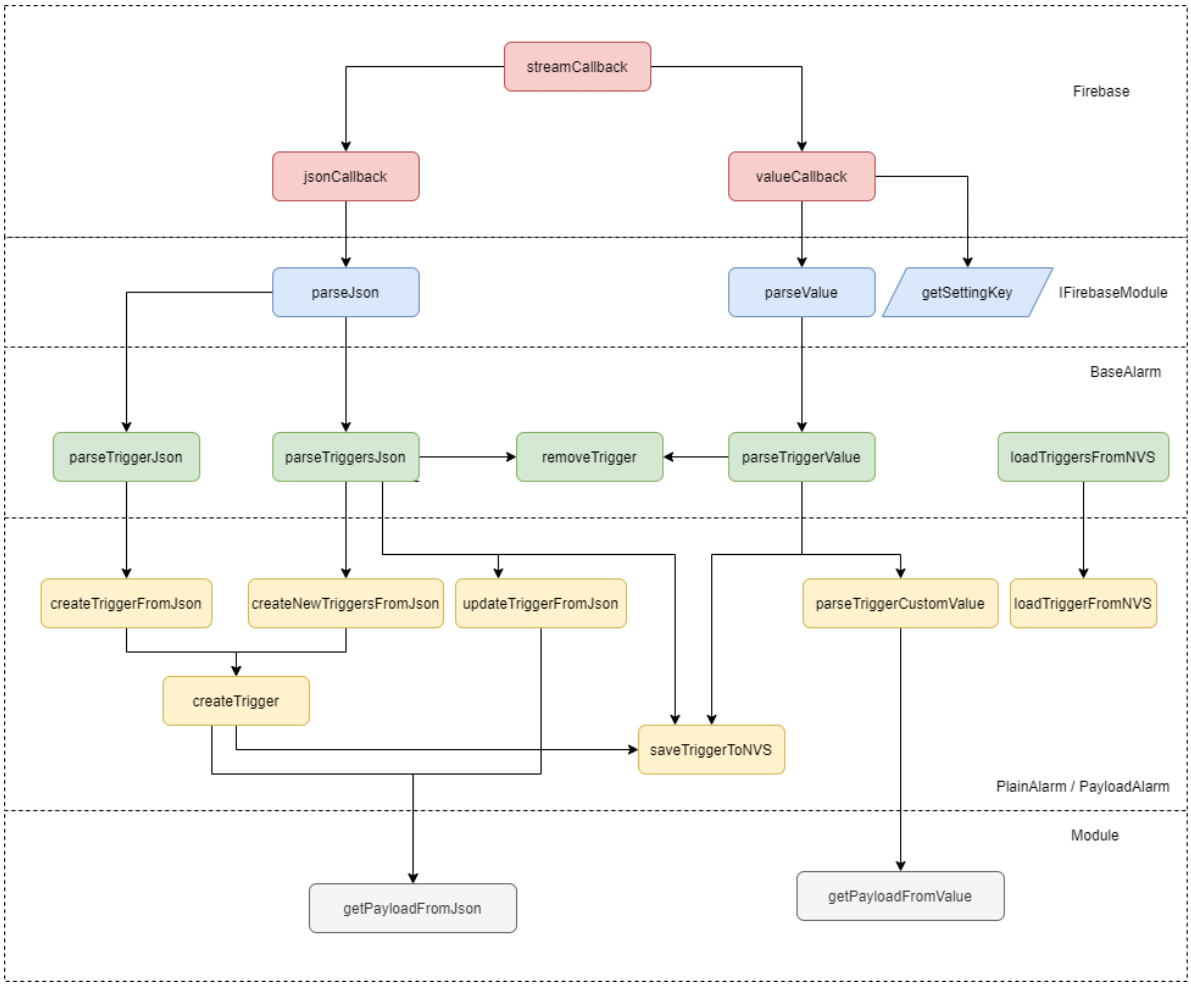


Schéma 10 - Provolávání funkcí při změně budíku

4.4.6 Ovládací panel

Ovládací panel viváriа se skládá z konektorů modulů, tlačítky pro přepnutí stavu modulů a notifikačních diod značící stav jejich zapojení. Pro ovládání tlačítek byl použit upravený rezistorový žebřík podle článku *The perfect multi-button input resistor ladder*. [33]

Rezistory jsou zapojeny se vzestupnou hodnotou odporu sériově za sebou. Hodnota za prvním rezistorem je čtena na analogovém pinu mikrokontroleru. Za každým dalším rezistorem je pak připojeno jedno tlačítko spojující obvod se zemí. Postupně vzrůstající odpor obvodu po sepnutí způsobuje, že se na vstupu mikrokontroleru objevují hodnoty s pravidelným rozestupem. To umožňuje napsat jednotný kód pro rozpoznání stisknutého tlačítka.

Pro kontrolu aktuální hodnoty na vstupním pinu mikrokontroleru probíhá pomocí callback funkce přidané do paralelního vlákna třídy *ParallelRunner*. Tato funkce převádí analogovou hodnotu na číslo tlačítka, které přijímá třída *ButtonControl* v podobě parametru funkce *buttonPressed*. Zde jsou rozlišeny tlačítka modulů (0-6) a tlačítko přepínání mezi Wi-Fi a Bluetooth (7). Úloha nemá paměťové prostředky na přímé spuštění Bluetooth serveru nebo

Firebase, nastaví tedy přepínač ve třídách *BLEController* a *FirebaseService* na přepnutí svého stavu v budoucnosti a nechá na hlavním vláknu, aby se o správné přepnutí postaralo.

4.4.7 Aktualizace firmwaru

Aby bylo možné vivárium testovat a provádět úpravy v reálném provozu, nahrává se napsaný firmware do Firebase Storage. Mikrokontroler pak kontroluje svou aktuální verzi s verzí zapsanou v databázi pod svým ID. Pokud je verze v databázi rozdílná, vygeneruje si prostřednictvím knihovny *Firebase_ESP_Client* URL pro stažení nové verze firmwaru spolu s přístupovým tokenem a ten poté předá aktualizační třídě *HttpsOTAUpdate*, která spustí novou úlohu pro aktualizaci.

Aktualizace pomocí OTA jsou umožněny rozdělením programového prostoru na dva stejně velké oddíly. Pokud dojde ke spuštění aktualizace, nový firmware se zapíše do nevyužívaného oddílu. Pokud je firmware úspěšně stažen, je třeba mikrokontroler restartovat, při svém startu totiž mikrokontroler zapíná aplikaci z oddílu, který byl přepsán jako poslední.

Kontroly nové verze probíhají ve dvou samostatných fázích:

Kontrola při startu

Po spuštění mikrokontroleru se spolu s inicializací jiných služeb zavolá metoda *begin*, při které se zkontroluje, zda před předchozím vypnutím mikrokontroleru nedošlo k zapsání nové verze do paměti. Pokud ano, program se pokusí načíst novou adresu z paměti a spustí aktualizaci.

```
void OtaService::begin()
{
  _firmwareVersion = memoryProvider.loadString(FIRMWARE_VERSION_KEY, "0");
  _newFirmwareVersion = memoryProvider.loadString(FIRMWARE_NEW_VERSION_KEY, "0");
  if (_firmwareVersion != _newFirmwareVersion)
  {
    downloadUrl = memoryProvider.loadString(FIRMWARE_DOWNLOAD_URL_KEY, "");
    startUpdate();
  }
}
```

Kód 40 - Ověření verze firmwaru

V metodě *startUpdate* je zavolána metoda pro zobrazení nové verze na displeji. Poté je spuštěno zabezpečené stažení nové verze firmwaru z nově získané adresy.

```
void OtaService::startUpdate()
{
  _textOutput->setText({"FW update", _newFirmwareVersion});
  _firmwareUpdateRunning = true;
  HttpsOTA.begin(downloadUrl.c_str(), https_cert, false);
}
```

Kód 41 - Spuštění OTA aktualizace firmwaru

Kontrola při běhu programu

Pro kontrolu při běhu programu slouží streamování změn v databázi pomocí knihovny *Firebase_ESP_Client*. Změna verze je zachycena callback funkcí a nová hodnota poté předána aktualizační službě. Nejprve dojde ke kontrole, zda je nová verze rozdílná od současné, poté dojde k ověření, že existuje k nové verzi platná URL adresa pro stažení. Pokud je vše v pořádku, dojde k zastavení Firebase i Bluetooth a zahájí se aktualizace.

```
/**
 * @brief Get download URL for given firmware version
 *
 * @param version Firmware version
 * @return String Download URL of the new firmware, empty URL if no download link found
 */
String FirebaseService::getFirmwareDownloadUrl(String version)
{
    String fileName = getFirmwareName(version);
    if (fileName == "")
        return "";
    String filePath = "firmware/" + fileName;
    String url;
    firebaseSemaphore.lockSemaphore("getFirmwareDownloadUrl");
    firebaseBdo->setCert(storage_cert);
    if (Firebase.Storage.getMetadata(firebaseBdo, STORAGE_BUCKET_ID, filePath.c_str()))
    {
        url = firebaseBdo->downloadURL();
    }
    else
    {
        url = "";
        printE("getFirmwareDownloadUrl request failed:");
        printlnE(firebaseBdo->errorReason());
    }
    firebaseBdo->setCert(rtdb_cert);
    firebaseSemaphore.unlockSemaphore();
    return url;
}
```

Kód 42 - Získání URL pro stažení nové verze firmwaru

4.4.8 Třída Vivarium

Třída *Vivarium* slouží pro ovládání vivária. Program je navržen tak, aby bylo potřeba v hlavním souboru projektu načíst pouze hlavičkový soubor této třídy. Tato třída má dvě základní funkce – prvotní nastavení vivária a poté jeho kontrolu ve smyčce programu.

```
#include <Arduino.h>
#include "src/vivarium/vivarium.h"

Vivarium vivarium;

void setup()
{
  vivarium.setup(2, "aquarium_t");
  vivarium.createModule(ModuleType::WATER_LEVEL, 0);
  vivarium.createModule(ModuleType::HUMIDIFIER, 1);
  vivarium.createModule(ModuleType::WATER_PUMP, 2);
  vivarium.finalize();
}

void loop()
{
  vivarium.onLoop();
}
```

Kód 43 - Příklad kódu hlavního souboru vivária

Spouštění

Spuštění mikrokontroleru probíhá ve dvou fázích. Nejprve je potřeba iniciovat veškeré služby vivária zavolám funkce *setup*. V této funkci dochází i inicializaci sériové linky, vytváří se instance všech tříd potřebných pro správný běh vivária – ovládání obrazovky, hodinového displeje, Firebase a Bluetooth. Mikrokontroler se taktéž pokusí přihlásit k Wi-Fi, pokud v paměti nalezne uložené přihlašovací údaje. Pokud je připojení k Wi-Fi úspěšné, aktualizuje se systémový čas mikrokontroleru podle získané hodnoty z NTP serveru. Zároveň dojde k zapsání tohoto času do RTC modulu. Pokud se připojení k Wi-Fi nezdaří, je systémový čas převzat z RTC modulu. Mezi poslední instrukce patří kontrola dostupnosti OTA aktualizace. Pokud instance třídy *OtaService* v paměti nalezne uloženou adresu pro stažení nové verze, zahájí se aktualizace.

Druhým krokem je nastavení externích modulů. Pro tento úkon byl vytvořen výčtový typ *ModuleType*, který obsahuje názvy všech implementovaných modulů. Třída *Vivarium* implementuje funkci *createModule*, která v parametru přijímá typ modulu a pozici tlačítka na

ovládacím panelu. Ve funkci *createModule* je poté podle typu modulu vytvořena instance odpovídající třídy. Ukazatel na nově vytvořený objekt je předán řídícím objektům – *ModuleControl, FirebaseService, BLEController* a *LCDDisplay*.

Nakonec je zavolána funkce *finalize*. Pokud neběží aktualizace firmwaru, zkontroluje se stav Wi-Fi připojení. Pokud je mikrokontroler připojen, připojí se k Firebase cloudu. V opačném případě je vytvořen Bluetooth server. Je taktéž zapnut běh kontrolních služeb sledujících stav mikrokontroleru.

Hlavní smyčka

Při implementace hlavní smyčky byl důraz kladen na zajištění plynulého průchodu kódem bez nutnosti pozastavovat hlavní vlákno mikrokontroleru. Speciálním případem je aktualizace firmwaru. Všechny služby jsou pozastaveny a hlavní smyčka pouze kontroluje stav aktualizace a vypisuje jej na LCD obrazovku.

V klasickém režimu je průběh hlavní smyčky následující:

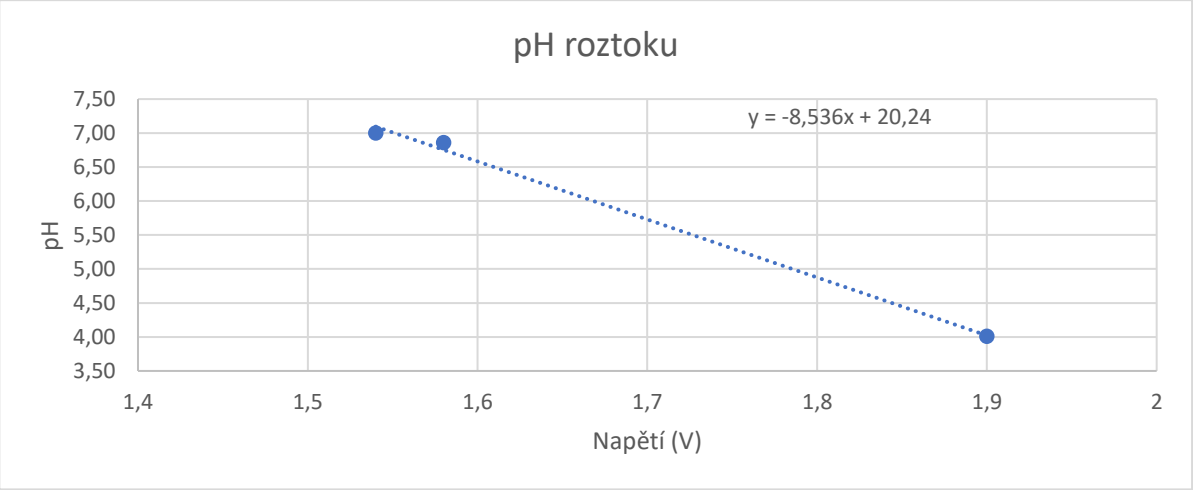
- Aktualizace hodinového displeje v určitém intervalu
- Kontrola přepínače mezi Bluetooth Firebase – kontrola vypnutí obou služeb
- Kontrola Wi-Fi připojení a případně znovu připojení
- Kontrola zamrznutí Firebase a případné restartování služby
- Aktualizace indikačních diod zásuvek
- Kontrola stavu připojení k Bluetooth serveru
- Aktualizace LCD displeje
- Předání obsluhy všem modulům
- Kontrola změny Wi-Fi údajů
- Kontrola *FirebaseService* – upload nových dat v určitém intervalu + obnovení FCM tokenů
- Kontrola nastavených budíků

4.4.9 Vybrané moduly

Všechny moduly dědí ze tříd *IModule*, *IBluetooth*, *IFirebase* a *TextModule*, čímž je zajištěna jejich univerzálnost pro nastavení pomocí Bluetooth i Firebase. Zároveň mohou díky třídě *TextModule* využít tisku textu na LCD obrazovku. Většina modulů má velmi podobnou implementaci lišící se nastavením, použitou knihovnou a měřenou hodnotou, proto je v následujících odstavcích zmíněn pouze výběr z nich.

pH Modul

Implementace ovládání pH senzoru umožňuje jednorázové i průběžné měření. Pro správný převod naměřené analogové hodnoty bylo potřeba senzor zkalibrovat. Ke kalibraci posloužila destilovaná voda a dva kalibrační pufry o hodnotě 4.01 a 6.86 rozpuštěné v destilované vodě. Byla naměřena hodnota napětí na analogovém pinu po ponoření sondy do každého z těchto roztoků. Hodnoty byly zadány do grafu a aproximovány lineární funkcí. Vzorec výsledné funkce je $y = -8.536x + 20.24$.



Graf 2 - Aproximace naměřených hodnot pH sondy lineární funkcí

Čtení aktuální hodnoty pH probíhá ve funkci *_readPh*. Aby byla eliminována možnost chybného čtení, provádí se načtení analogové hodnoty desetkrát za sebou v intervalu po 30 milisekundách. Naměřené hodnoty jsou seřazeny a následně je vypočítán průměr středních hodnot, 2 nejvyšší a 2 nejnižší hodnoty jsou z výpočtu vyřazeny. Nakonec je použit dříve získaný vzorec pro převod napětí na skutečnou hodnotu pH.

Vrácená hodnota funkce *_readPh* je poté porovnána s předchozí naměřenou hodnotou. Pokud se hodnota liší a běží Bluetooth server, je změněna hodnota Bluetooth charakteristiky. Zároveň je změna nahrána do RTDB a jsou zkontrolovány uživatelem nastavené limity při jejichž překročení je uživatel informován prostřednictvím FCM.

```
float PhModule::_readPh()
{
    for (int i = 0; i < PH_READING_COUNT; i++)
    {
        _phReadBuffer[i] = analogRead(_pin);
        delay(30);
    }
    qsort(_phReadBuffer, PH_READING_COUNT, sizeof(float), compareFloat);
    float phAvgValue = 0;
    for (int i = PH_VALUES_CUT; i < PH_READING_COUNT - PH_VALUES_CUT; i++)
    {
        phAvgValue += _phReadBuffer[i];
    }
    phAvgValue /= PH_READING_COUNT - (2 * PH_VALUES_CUT);
    float Po = (float)phAvgValue * PH_VOLTAGE / PH_PRECISION;
    float phValue = PH_VOLTAGE_CHANGE * Po * PH_CALIBRATION;
    if (phValue < 0 || phValue > 14)
    {
        phValue = PH_INVALID_VALUE;
    }
    return phValue;
}
```

Kód 44 - Čtení hodnoty pH

Vodní hladina

Testování ukázalo nedostatky při použití původně zvolené knihovny *NewPing*. Měření ultrazvukovým senzorem může narušit zvlněná hladina akvária, v takovém případě jsou patrné časté výkyvy, v některých případech se může i signál odrazit špatným směrem a echo se nezaznamená. Knihovna sice umožňovala více měření po sobě a vrácení mediánové hodnoty, nicméně špatné odrazy hladiny mohly způsobit zastavení hlavního vlákna na více než 10 vteřin. S ohledem na plynulý běh hlavního vlákna byla napsána vlastní třída pro asynchronní měření vzdálenosti za pomoci obsluhy přerušení ISR.

Pro zpřesnění hodnoty probíhá čtení 10x asynchronně za sebou, přičemž jedna iterace čtení probíhá v následujících krocích:

- Funkce *sendPulse* odešle puls HIGH s trváním 10 mikrosekund
- Funkce *_echo_isr* zachytí změnu HIGH na echo pinu a uloží čas začátku pulsu v mikrosekundách
- Funkce _echo_isr zachytí změnu LOW na echo pinu po přečtení odraženého pulsu a uloží čas ukončení pulsu v mikrosekundách

- Hlavní smyčka se pravidelně dotazuje na ukončení pulsu, jakmile je puls ukončen nebo došlo k vypršení časového limitu pro návrat signálu, je spočítána vzdálenost.

Po skončení všech iterací je získána skutečná vzdálenost. Vyfiltrují se neplatná měření, odstraní se nejextrémnější hodnoty a vypočítá se průměr ze zbývajících hodnot.

```
int HC_SR04::getRange()
{
    if (_valid_it == 0)
        return INVALID_VALUE;

    std::vector<int> validReadings;

    for (int i = 0; i < ITERATION_COUNT; i++)
    {
        if (_readings[i] != INVALID_VALUE)
        {
            validReadings.push_back(_readings[i]);
        }
    }

    if (validReadings.size() == 1)
        return validReadings.at(0);

    if (validReadings.size() > 3)
    {
        std::sort(validReadings.begin(), validReadings.end());
        validReadings.erase(validReadings.begin());
        validReadings.pop_back();
    }
    return (std::accumulate(validReadings.begin(), validReadings.end(), 0.0)
            / (validReadings.size()) + 0.5);
}
```

Kód 45 - Výpočet vzdálenosti

Ani takto filtrované výsledky však nevykazovaly stabilitu v měření, proto bylo rozhodnuto výsledky ještě porovnávat s předchozími měřeními. Byl implementován čítač sledující počet různých měření. Aby byla změna naměřené hodnoty propsána, musí se lišit po dobu 7 měření. Pokud je naměřena znovu původní hodnota, čítač se resetuje. Měření probíhá každých 700 milisekund, změna se tedy může projevit po pěti vteřinách. Vzhledem k nízkému výkonu čerpadla je tento interval postačující, zároveň došlo ke stabilizaci naměřených hodnot.

Doplňování vody

Funkce vodního čerpadla závisí na naměřené hodnotě výšky vodní hladiny. Pro lepší bezpečnost byly implementovány dva kontrolní mechanismy. První mechanismus hlídá stav připojení senzoru vodní hladiny, pokud dojde k jeho odpojení, automaticky je odpojeno i čerpadlo. Druhý mechanismus využívá nezávislého hardwarového časovače tak, aby nebyla funkce čerpadla ohrožena případným zamrznutím hlavního vlákna. Hlavní vlákno v pravidelném intervalu porovnává výšku hladiny s požadovanou výškou. Pokud čerpadlo sepne, je zapnut hardwarový časovač, který je odpovědný za vypnutí čerpadla po dvou vteřinách. Časovač je z hlavního vlákna pravidelně resetován, takže do správného chodu hlavního vlákna nezasahuje.

```
void WaterPump::onLoop()
{
    checkConnectionChange();
    if (_settingsChanged)
    {
        saveSettings();
    }
    if (isConnected())
    {
        bool wlConnected;
        stateStorage.getValue(STATE_WATER_LEVEL_CONNECTED, &wlConnected);
        if (!wlConnected)
        {
            stopPump();
            setConnected(false, true);
            return;
        }
        if (_failSafeTriggered)
        {
            stopPump();
            _failSafeTriggered = false;
        }
        uint32_t level;
        if (stateStorage.getValue(STATE_WATER_LEVEL, &level))
        {
            if (level == INVALID_VALUE)
                return;
            if (_levelGoal != INVALID_VALUE && level < _levelGoal)
            {
                startPump();
            }
            else
            {
                stopPump();
            }
        }
    }
}
```

Kód 46 - Ovládání čerpadla

Ohřívání vody

Regulace teploty vody je jednou z nejdůležitějších částí vivária. Ovládací a regulační jednotka podporuje tři režimy regulace vody: Dvoustavový (ON/OFF), PID a předání regulace externímu zařízení.

Externí regulace

Vivárium nastaví výkon zásuvky na 100 % a přenechá řízení teploty připojenému zařízení. Tento režim je vhodný při využití topítka s integrovaným termostatem. Nevýhodou tohoto řešení je absence vzdáleného ovládání teploty, pokud jím nedisponuje pořízené topítko.

Dvoustavový termostat

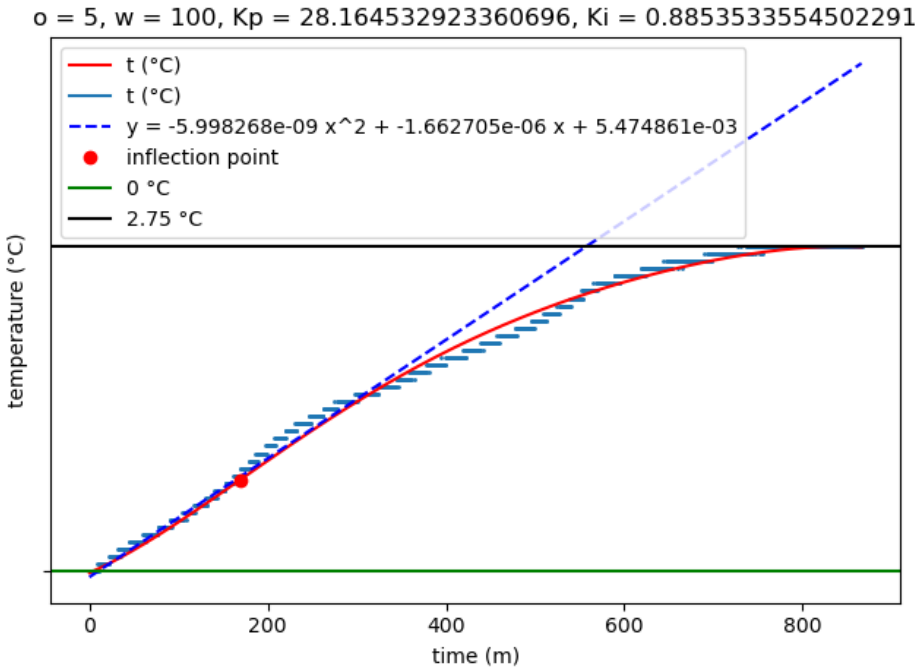
Řídící jednotka v pravidelném intervalu porovnává naměřenou hodnotu teploty s požadovanou teplotou. Pokud je aktuální teplota nižší než požadovaná, zůstává obvod sepnutý. Jakmile teplota překročí požadovanou hodnotu, topítko je vypnuto. Pro spínání bylo využito SSR relé neobsahující žádné mechanické prvky, které je vhodné pro spínání v krátkých intervalech.

Systém zároveň ověřuje stav čidla teploty. Pokud čidlo vrátí chybnou hodnotu, je porovnání přeskočeno. Pokud čidlo po dobu 10 vteřin nevrátí validní hodnotu, je vytápění vypnuto a uživatel upozorněn pomocí notifikace.

PID

Regulátor byl implementován tak, aby umožňoval nastavení pásma proporcionality. Pracuje tak ve dvou fázích. Pokud je rozdíl aktuální a požadované teploty větší než polovina pásma proporcionality, je výkon topítka nastaven na maximální, případně minimální, hodnotu. To zaručuje rychlou reakci na větší změnu požadované teploty. Jakmile se aktuální teplota dostane do proporcionálního pásma, začne fungovat PID regulátor.

Pro nastavení PID regulátoru byla nejprve vyzkoušena Ziegler-Nicholsova metoda směrodatné odchylky. Topítko bylo nastaveno na výkon 33,5 %, při kterém se dostalo do stabilního stavu o teplotě 25 °C. Nejprve došlo k pokusu mírně zvýšit výkon v řádu desetin procent. Změny teploty však byly příliš pomalé a čidlo nedokázalo zaznamenat nárůst teploty s jemnější přesností než 0,0625°C. Výkon topítka byl tedy navýšen o 5 procentních bodů, teplota se po 13 hodinách ustálila na 27,75 °C. Naměřené hodnoty byly normalizovány pro snadný výpočet konstant regulátoru. Vývoj teploty lze spatřit v grafu číslo 3. Změna teploty byla aproximována polynomem čtvrtého řádu, byl nalezen inflexní bod a tečna procházející tímto bodem.



Graf 3 - Kroková odezva akvária

Dle autorů knihy *Controller Tuning for Capacity and Dead Time Processes* [34] je potřeba zohlednit reakční rychlost systému. Z toho důvodu jsou vypočítány následující parametry:

- $L = 9{,}54$ (zpoždění systému)
- $T = 547{,}53$ (časová konstanta)
- $P = 5$ (procentuální vyjádření počáteční změny na vstupu)
- $\Delta C_p = \frac{2.75}{30} * 100 = 9{,}17$ (procentuální vyjádření změny regulované veličiny)
- $N = \frac{\Delta C_P}{T} = \frac{9{,}17}{547{,}53} = 0{,}0167$ (reakční rychlost systému)

Získané parametry je možné dle pravidel Ziegler-Nicholsovy metody využít ke konstrukci PID a PI regulátoru:

- $K_p = \frac{0.9P}{NL} = 28{,}16$; $T_i = \frac{L}{0{,}3} = 31{,}81$; $K_i = \frac{K_c}{T_i} = 0{,}89$ (PI)
- $K_p = \frac{1{,}25P}{NL} = 39{,}12$; $T_i = 2L = 19{,}087$; $T_d = \frac{L}{2} = 4{,}77$; $K_i = \frac{K_c}{T_i} = 2{,}05$; $K_d = K_c T_d = 186{,}6024$ (PID)

Pro porovnání získaných výsledků došlo taktéž k ručnímu ladění konstant PID regulátoru. Konstanty byly vybrány s ohledem na snížení či úplnou eliminaci překmitu již rozběhlého topítka. Nejprve byla zvolena dostatečně vysoká hodnota K_P tak, aby bylo možné uvést systém do stabilního stavu. Následně byla nastavena hodnota integrační konstanty pro zrychlení reakce systému.

Při optimalizaci se ukázalo, že má derivační konstanta velmi nízký vliv na funkci systému. Použitá knihovna *QuickPID* umožňuje nastavit využití proporcionální složky a určit, zda a v jakém poměru bude proporcionální složka využita pro násobek chybové odchylky současné hodnoty od požadované či zda bude zahrnuta do výpočtu se změnou naměřené hodnoty. V rámci testování bylo zjištěno, že je vhodné nastavit hodnotu proporcionální složky *POn* na 40 %. Jen čtyři desetiny hodnoty konstanty K_p jsou tak využity pro reakci systému na odchylku oproti požadovanému stavu, zbylých 60 % hodnoty slouží k vyrovnání změn na vstupu. Reakce systému je tímto nastavením mírně zpomalena, zároveň však dojde ke zmírnění překmitů a stabilního stavu je dosáhnuto rychleji, protože systém tolik neosciluje.

Hodnoty regulátoru byly nastaveny následovně:

$$K_p = 80 \quad K_i = 0{,}08 \quad K_d = 0.8 \quad POn = 0.4$$

Regulační jednotka taktéž umožňuje vzdálenou konfiguraci PID parametrů. Nastavení regulátoru tak lze měnit prostřednictvím aplikace.

4.4.10 Testování

Pro účely testování byla použita knihovna *AUnit* inspirovaná knihovnami *ArduinoUnit* a *Google Test*. Testy jsou spouštěny přímo na mikrokontroleru a pomáhají tak detekovat problémy specifické pro danou platformu. Pro testování byly vybrány klíčové části firmwaru včetně vybraných externích modulů. Bylo vytvořeno celkem 62 testovacích scénářů. Testovány byly řídící třídy třídy *ModuleControl* a *MemoryProvider*, dále pak třídy rozšiřující externí moduly *IModule*, *BaseAlarm*, *PlainAlarm*, *PayloadAlarm*. Zároveň byly k testování vybrány taktéž moduly vyžadující přesnou funkcionalitu, ať již reakce na změny jiných modulů, sepnutí časovačů, nebo ochranné mechanismy – *Feeder*, *LedModule*, *WaterPump*, *Heater*.

```
testF(TestPlainAlarmOnce, create_trigger)
{
    ESP32Time rtc;
    rtc.setTime(0, 0, 1, 1, 1, 2021);

    assertEqual(alarm->getTriggersCount(), 0);
    alarm->createTrigger(256, "first");
    assertEqual(alarm->getTriggersCount(), 1);

    std::shared_ptr<Trigger> tt = alarm->getNextTrigger();
    assertTrue(tt != nullptr);

    int ti;
    assertTrue(alarm->getNextTriggerTime(&ti));
    assertEqual(ti, 256);

    std::shared_ptr<Trigger> t = alarm->getNextTrigger();
    assertEqual(256, getTime(t->hour, t->minute));
}
```

Kód 47 - Příklad testování třídy PlainAlarm

5 Testovací provoz

Pro otestování systému byla vytvořena ovládací a regulační jednotka akvária s 8 moduly: Měření vodní hladiny, LED osvětlení, čerpadlo, krmítko, chladič, pH sonda, teplotní čidlo a topítko.

```
void setup()
{
  vivarium.setup(2, "aquarium_t");
  vivarium.createModule(ModuleType::WATER_LEVEL, 0);
  vivarium.createModule(ModuleType::LED, 1);
  vivarium.createModule(ModuleType::WATER_PUMP, 2);
  vivarium.createModule(ModuleType::FEEDER, 3);
  vivarium.createModule(ModuleType::FAN, 4);
  vivarium.createModule(ModuleType::PH_PROBE, 5);
  vivarium.createModule(ModuleType::WATER_TEMPERATURE, 6);
  vivarium.createModule(ModuleType::HEATER, -1);
  vivarium.finalize();
}
```

Kód 48 - Nastavení akvária

5.1 Nastavení a ověření běhu programu

Zapojení akvária a nastavení modulů lze shrnout do následujících kroků:

- Zapojení regulační a ovládací jednotky do elektrické sítě (spustí se v Bluetooth režimu)
- Přihlášení uživatele v aplikaci (pro testování byl použit účet Google)
- Spárování nového zařízení s uživatelem pomocí mobilní aplikace
- Zapnutí zásuvek řídící jednotky
- Připojení kamery a její spárování s regulační a ovládací jednotkou
- Zapojení a nastavení jednotlivých modulů

Cílem prvního nastavení je ověřit schopnost regulační a ovládací jednotky bezchybně běžet a správně reagovat na všechny pokyny uživatele.

```
{
  "fan" : {
    "maxAt" : 27,
    "startAt" : 25.5
  },
  "feeder" : {
    "triggers" : {
      "-MjnzfYacpBP8zMA-g6z" : {
        "time" : 2334
      }
    },
    "type" : 0
  },
  "heater" : {
    "mode" : 0,
    "triggers" : {
      "-MkifyzWE8Cb_rN-Jkgd" : {
        "goal" : 23,
        "time" : 5120
      },
      "-Ml-bam_PgWzDclRx5QK" : {
        "goal" : 24,
        "time" : 2048
      }
    }
  },
  "led" : {
    "triggers" : {
      "-MjaxEbPdxCL0t5eW0sk" : {
        "color" : 14079702,
        "time" : 2065
      },
      "-MjaxHKHh9FNx63i4hfj" : {
        "color" : 10398132,
        "time" : 5179
      },
      "-MmVKbv4T0UHL1lrc6ul" : {
        "color" : 328708,
        "time" : 4380
      }
    }
  },
  "ph" : {
    "continuous" : false,
    "continuousDelay" : 15,
    "maxPh" : 4,
    "minPh" : 3
  },
  "pump" : {
    "levelGoal" : 25
  },
  "waterTemp" : {
    "maxTemp" : 25,
    "minTemp" : 22.5
  },
  "wl" : {
    "maxL" : 26,
    "minL" : 24,
    "sensorHeight" : 38
  }
}
```


Kód 49 - Nastavení akvária v RTDB

5.2 Nalezené problémy a jejich řešení

5.2.1 Únik paměti

V průběhu testování nastal problém s častým padáním aplikace kvůli únikům v paměti, které se při krátkodobých testech během vývoje neprojevily. Po přezkoumání problému bylo zjištěno, že knihovna *Firebase-ESP-Client* ve verzi 2.0.3 způsobuje nepracuje s pamětí správně a po zavolání funkce *Firebase.begin* začne FreeRTOS detekovat únik paměti, který později vyústí v pád aplikace. Problém byl vyřešen upozorněním autora knihovny, který chybu opravil v další verzi knihovny.

5.2.2 SSL Certifikáty

Komunikace s Firebase probíhá prostřednictvím REST API dvou služeb. Na začátku testování měly obě služby stejnou kořenový certifikát a mikrokontroler se bezpečně připojoval k oběma službám. V průběhu testování ovšem došlo ke změně certifikátů pro RTDB a Firebase Storage. Po zkontrolování platných certifikátů se ukázalo, že GlobalSign Root CA chybí a služby nemají stejný řetěz certifikátů. Knihovna Firebase-ESP-Client však počítala s využitím jednoho certifikátu pro veškerou komunikaci. Autor knihovny proto po vzneseném návrhu přidal možnost měnit certifikát podle typu služby, se kterou mikrokontroler komunikoval.

Poslední změna s certifikáty způsobila problémy s přihlášením mikrokontroleru k Firebase prostřednictvím servisního účtu a privátního klíče. Řešením bylo připojení k Firebase prostřednictvím e-mailové adresy a hesla nově vytvořeného administrátorského účtu a úprava pravidel pro přístup k datům v databázi a ve Firebase Storage.

5.2.3 Zamrzávání hlavního vlákna

Při dlouhodobějším běhu bylo detekování zamrzání hlavního vlákna aplikace. Jiná podpůrná vlákna nejsou tímto zamrznutím ovlivněna, ale zamrznutí hlavního vlákna způsobí, že řídící jednotka přestane reagovat na jakýkoliv podnět. Po hlubším prozkoumání problému bylo zjištěno, že se hlavní vlákno po přibližně jedné hodině znovu rozběhne. Příčinu tohoto zamrznutí se nepodařilo objevit, ale podezření padá na vznik deadlocku při obnovení přístupového tokenu uživatele pro přístup k Firebase, jehož platnost je 60 minut a který je každých 59 minut obnovován. Při dalším pokusu o obnovení tokenu by mohlo dojít k dokončení operace a tím uvolnění kritické sekce, což by vysvětlovalo rozběhnutí hlavního vlákna v následující hodině. Tento proces je v režii knihovny *Firebase-ESP-Client* a bylo by komplikované tuto teorii potvrdit. Místo toho bylo rozhodnuto o vytvoření detekce zamrznutí

hlavního vlákna a případného restartu celého zařízení. Byly vytvořeny třídy *Watchdog* a *TaskHealth.* Třída *TaskHealth* rozšiřuje monitorované třídy a umožňuje udržovat informace o poslední aktivitě a jejich stavu. Instance třídy *Watchdog* hlídá stav třídy *Vivarium* a pokud zaznamená zamrznutí hlavního vlákna, zavolá funkci pro restart zařízení.

Je nutné zmínit, že vyřešení problému s SSL certifikáty přechodem na přihlašování kombinací e-mailové adresy a hesla zároveň odstranilo většinu problému se zamrzáním hlavního vlákna. K zamrzání v průběhu testovacího provozu docházelo velmi nepravidelně. Nejkratší doba od spuštění zařízení byla několik hodin a nejdelší nepřerušený průběh byl více než tři dny. Důkladné testy ukázaly, že příležitostné zamrznutí hlavního vlákna způsobuje čtení teploty vody na čidle DS18B20. Zamrznutí nastává při získávání teplot na úrovni knihovny *OneWire*.

5.3 Testování vybraných modulů v praxi

5.3.1 Výška vodní hladiny

Po nastavení výšky umístění senzoru HC-SR04 na 38 cm došlo k postupnému zaplňování akvária vodou. Čtení výšky vodní hladiny bylo stabilní, dokud vzdálenost mezi senzorem a vodní hladinou neklesla pod 7 cm. Při této vzdálenosti se začaly objevovat výkyvy ve čtení vzdáleností. Četnost těchto výkyvů se s klesající vzdáleností zvyšovala. Tyto nepřesnosti mohou způsobit nesprávné chování vodního čerpadla. V kódu bylo přidáno jednoduché odfiltrování chybných hodnot, aby byla jakákoliv hodnota zaznamenána, musí se několikrát po sobě lišit od původní hodnoty. Takto upravené ovládání senzoru vodní hladiny dostalo stabilnější výsledky, stále však není doporučeno umístit senzor méně než 5 cm nad vodní hladinu.

Čerpadlo doplňující odpařenou vodu se po nových úpravách spouští správně. Tento test však ukázal, že napojení čerpadla na nestabilní senzor nemusí být bezpečné. V rámci této práce byly použity pouze senzory HC-SR04, je možné, že jiné ultrazvukové senzory by dosáhly lepších výsledků. Větší bezpečnost by v tomto případě nejspíš napomohlo napojení čerpadla na okruh s plovákovým senzorem a ultrazvukový senzor zanechat pro zobrazování dat uživateli.

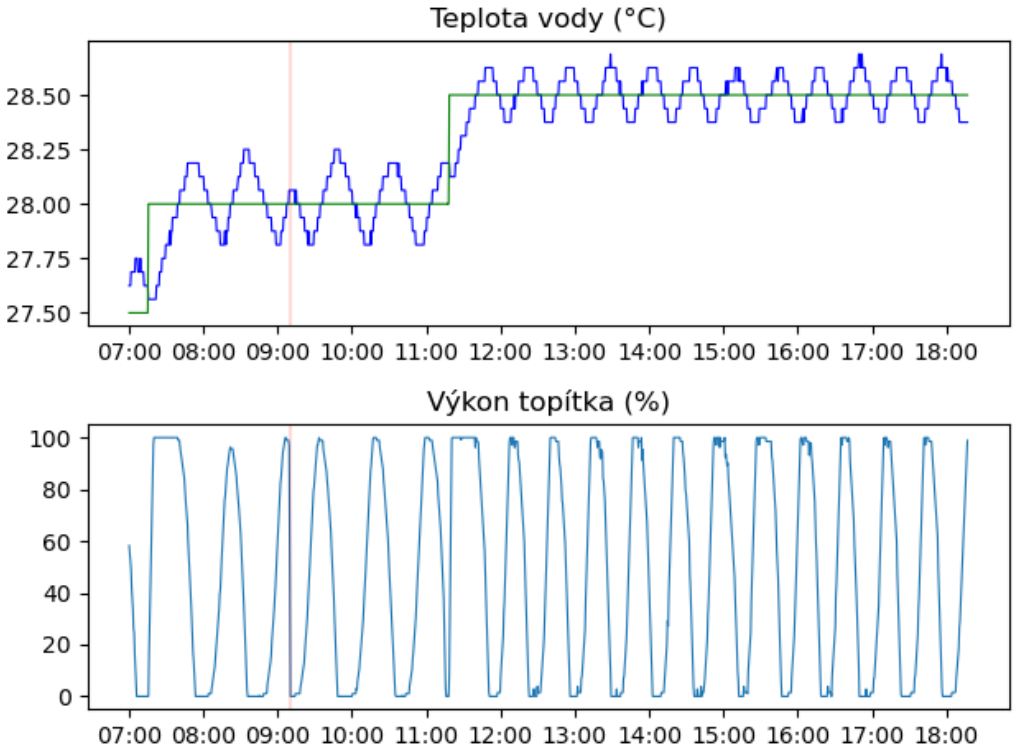
5.3.2 PID regulace teploty

V rámci testování byly porovnány konfigurace PI a PID regulátoru získané v rámci kapitoly PID (str. 106). Konkrétně se jedná o tyto tři konfigurace:

PI (1)	P = 28,16	I = 0, 89		
PID (2)	P = 39,12	I = 2,05		
PID (3)	P = 80	I = 0,08	D = 0,8	POn = 0,4

PI (1) a PID (2) jsou hodnoty získané Z-N metodou přechodové charakteristiky, PID (3) používá ručně nastavené parametry. Všechny regulátory umožňovaly rychlé přiblížení k nastavené teplotě, lišily se však ve stabilitě výstupu.

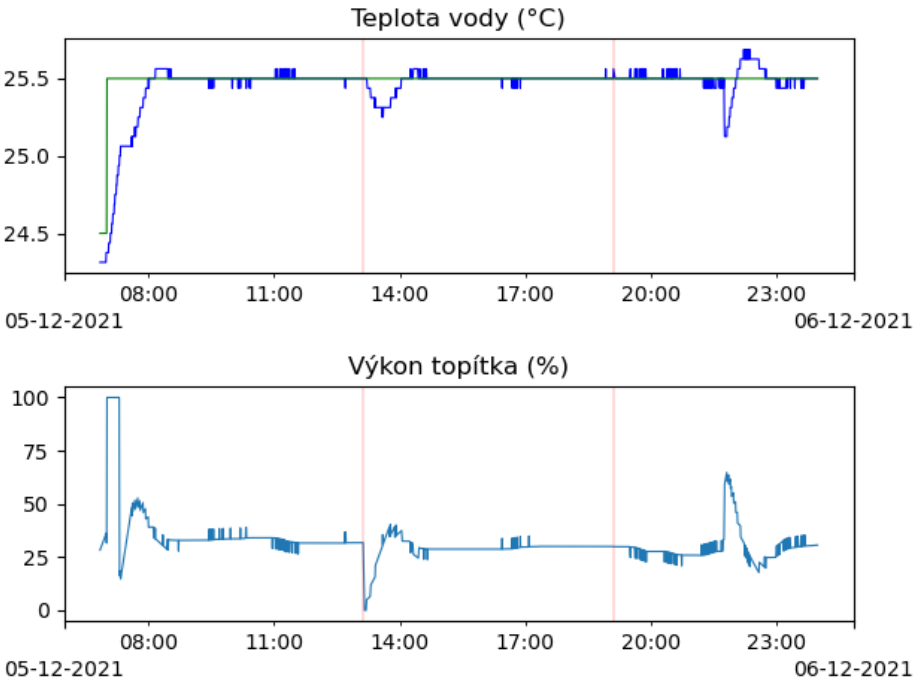
Z grafu číslo 4 lze vyčíst výstup regulátoru PI (1) a PID (2). V 7:10 byl nastaven PI (1) regulátor. Z grafu je patrná vysoká míra oscilace výstupní teploty lišící se od požadované až o 0,2 °C.



Graf 4 - Výsledek regulátoru nastaveného pomocí metody Z-N

V 11:15 byl nastaven regulátor PID (2) spolu s jiným cílem teploty. Oscilace byla v tomto případě nižší, pohybovala se s maximálním překmitem 0,12 °C.

Ručně nastavený PID (3) regulátor dosahoval lepších výsledků, což je patrné z grafu číslo 5, který ukazuje průběh teploty hlídané po dobu 17 hodin. V tomto časovém úseku proběhly 4 testy – rychlost a přesnost stabilizace, reakce na restart bez uložení aktuálního výstupu regulátoru, reakce na restart s uloženým výstupem regulátoru a schopnost reagovat na vnější zásah do systému.



Graf 5 - Testování PID regulátoru

První test byl zaměřen na rychlost a přesnost ohřátí vody. Počáteční teplota byla 23,31 °C. V sedm hodin ráno došlo k přepnutí cílové teploty na 25,5 stupňů, přičemž trvalo 56 minut přiblížit se požadované teplotě s přesností méně než 0,2 °C a dalších 43 minut pro dosažení optimálního výkonu topítka. Překmit regulátoru je minimální, na ukázaném příkladu byla hodnota překmitu 0,0625 °C, což je nejnižší rozlišitelná hodnota tepelného čidla.

Ve 13:07 proběhl řízený restart zařízení bez uložení aktuální hodnoty regulátoru simulující neočekávaný restart. Teplota poklesla o 0,19 °C a systém se vrátil na původní hodnoty po 55 minutách. Tento test ukázal, že je systém schopen se rychle zotavit i v případě neočekávaného restartu zařízení.

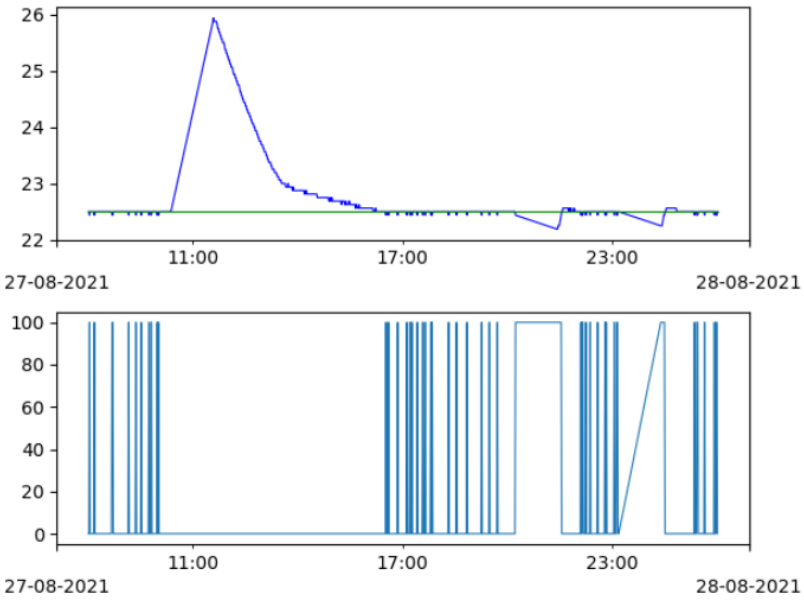
V 19:08 proběhl druhý řízený restart, tentokrát s uložením aktuální hodnoty regulátoru. Restart se podle předpokladů na systému nijak neprojevil.

Poslední test proběhl ve 21:43, kdy byla do akvária doplněna voda o nižší teplotě. Celková teplota vody v akváriu poklesla na 25,12 °C. Maximální teplota při překmitu dosáhla 25,69 °C a ke stabilizaci výkonu došlo ve 22:45, tedy 1 hodinu a 2 minuty po vychýlení.

Testy ukázaly, že je regulátor schopen reagovat na změny prostředí v intervalu přibližně jedné hodiny. Překmit byl vždy menší než 0,2 °C, což je pro chované rybičky dostačující. V průběhu dne na akvárium působily i vnější vlivy, které byly téměř hned vykompenzovány. Odchylky způsobené vnějšími činiteli byly nejvýše 0,0625 °C.

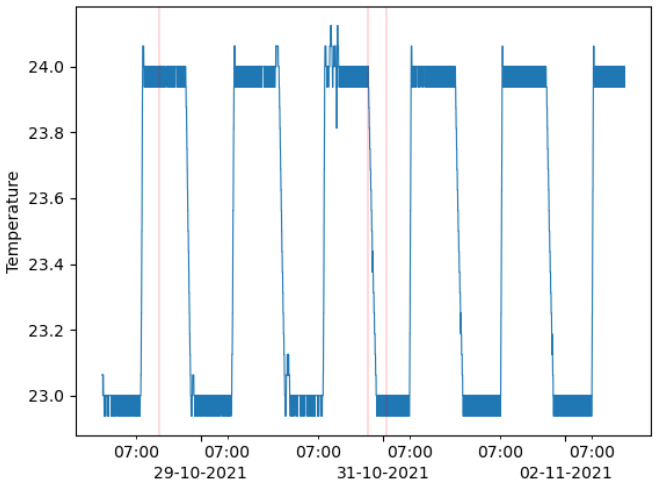
5.3.3 Použití dvoustavového přepínače

Další možností ovládání topítka je aplikace dvoustavového ON/OFF přepínače. Princip testování přepínače je totožný s testováním PID regulátoru. Během testování tohoto režimu bylo zjištěno již dříve popsané zamrzání hlavního vlákna aplikace. Následující graf číslo 6 ukazuje efekt zamrznutí mikrokontroleru na teplotu vody v akváriu. Hlavní vlákno zamrzlo v době zapnutého topítka, což způsobilo zvýšení teploty během jedné hodiny o 3,5 °C. Tento problém byl odstraněn implementací detekce zamrznutí a následného restartu aplikace.



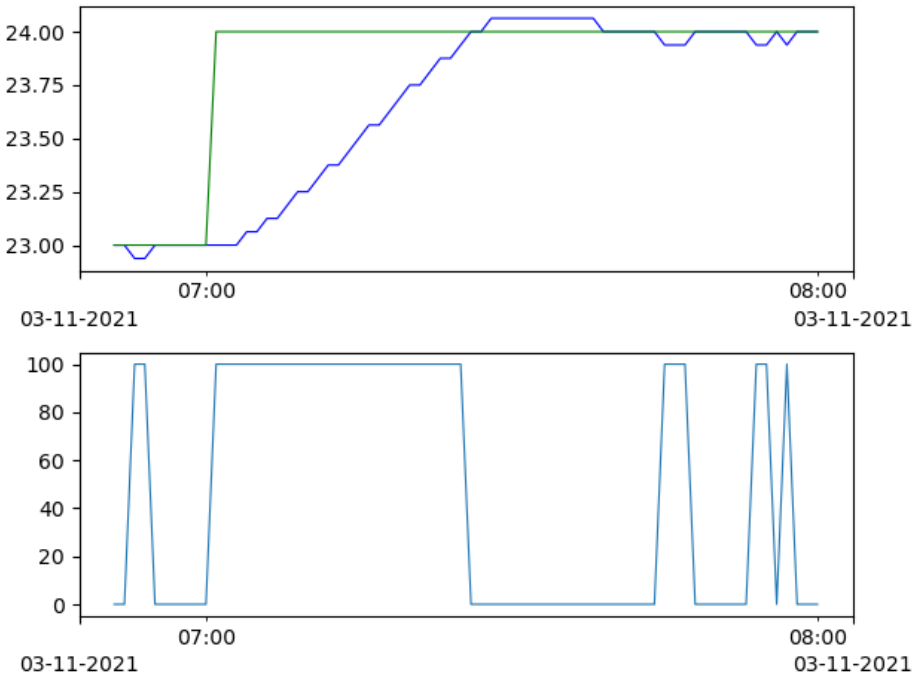
Graf 6 - Ovlivnění termostatu zamrzajícím vláknem

Po odstranění problému s hlavním vláknem bylo možné vyzkoušet běh regulátoru při delším období. Graf číslo 7 ukazuje vývoj teploty během 6 dní. Červené diagonální čáry značí restarty zařízení. I přes jejich výskyt byla teplota stabilní a po většinu času se držela v rozmezí několik setin stupně Celsia od požadované teploty. Drobné výkyvy mohly být způsobeny vlivem vnějšího prostředí.



Graf 7 - Vývoj teploty za použití termostatu během 6 dní

Poslední graf číslo 8 ukazuje rychlost stabilizace požadované teploty za použití termostatu. Po zvýšení cíle o jeden stupeň Celsia trvalo systému přibližně 20 minut ke stabilizaci teploty. Hodnota překmitu byla 0,0625 °C.



Graf 8 - Stabilizace nové teploty při použití termostatu

5.3.4 Chlazení

Při správné funkci topítka nepřekročila teplota vody kritickou hranici pro automatické spuštění ventilátorů. Testování proto probíhalo uměle při vypnutém topítku. Nejprve bylo nutné počkat, až se teplota vody sama stabilizuje a poté byl spuštěn ventilátor. Tento pokus proběhl opakovaně. Ukázalo se, že je možné teplotu vodu snížit o 2 až 3 °C oproti stabilizovanému stavu.

6 Závěr

Cílem práce bylo vytvoření modulární regulační a ovládací jednotky pro akvária a terária. Po analýze chovu živočichů ve viváriích byly definovány požadavky na regulované veličiny a možnosti jejich řízení. Uživatelské scénáře umožnily lépe specifikovat finální podobu aplikace pro ovládání regulační a ovládací jednotky.

Zevrubná analýza dostupných technologií napomohla při návrhu architektury systému. Byly navrženy dva systémy, přičemž jako spolehlivější a cenově výhodnější se ukázal systém s mikrokontrolerem ESP32 pro jádro regulační a ovládací jednotky. Návrh spočíval ve vytvoření jednoho kódu pro řídící jednotku bez ohledu na použitý hardware a počet modulů. Pro testování v reálném provozu byl vytvořen návrh akvária spočívající ve výběru modulů a vytvoření plošných spojů pro zapojení veškerého hardwaru. Fotodokumentace zařízení lze nalézt v příloze číslo 4. Byl taktéž vytvořen návrh aplikace, která umožňuje správu jednoho či více vivárií.

V průběhu vývoje byl kód vivária postupně upraven do podoby umožňující nejen snadnou modularitu podle použitého hardwaru, ale také snadnou rozšiřitelnost pro přidání dalších modulů nesouvisejících s ovládáním vivárií. Stejný kód tak lze s lehkou úpravou připsáním nové třídy pro nový modul použít například pro ovládání chytré zásuvky, květináče nebo samostatné kamery.

Kód aplikace byl pro tuto rozšiřitelnost napsán stejným způsobem tak, aby nebylo nutné provádět zásahy do jádra aplikace pro přidání dalších chytrých zařízení. Při výběru frameworku hrála svou roli i možnost vytvoření webové verze aplikace. Mobilní aplikace umožňuje ovládání vivária pomocí Bluetooth nebo internetu. Použitá knihovna pro Bluetooth nepodporuje web, proto webová verze aplikace umožňuje ovládat zařízení jen přes internet. Byly napsány testy pro kritické části vivária, aplikace a bezpečnostní pravidla Firebase.

Při instalaci hardwaru byla zjištěna omezení některých pinů pro plynulý chod mikrokontroleru a bylo třeba změnit diagram zapojení tak, aby se piny nenacházely při spouštění mikrokontroleru ve stavu znemožňující jeho start.

Testovací provoz odhalil některé chyby v logice firmwaru, jež byly následně opraveny. U některých modulů se nízká cena projevila na jejich spolehlivosti. Kód byl napsán tak, aby většinu nedostatků těchto modulů vykompenzoval. Dalším řešením by bylo nahradit tyto moduly kvalitnější variantou, což by se projevilo na ceně celého systému.

Projekt lze v budoucnu snadno rozšířit o další moduly a aplikaci tak využít pro ovládání chytré domácnosti. Na základě získaných poznatků lze taktéž navrhnout spolehlivější architekturu systému, kde by byl každý modul ovládán vlastním, méně výkonným čipem. Ovládací jednotka by pak zajišťovala obsluhu těchto modulů a předávala jim potřebné informace. Propojení modulů k řídící jednotce by bylo zajištěno pomocí univerzálního konektoru. Tento způsob by umožnil vytvoření jednoho hardwaru řídící a regulační jednotky bez nutnosti jej upravovat pro různé moduly.

7 Bibliografie

[1] HES, Ondřej. *Podmínky chovu plazů v zajetí: včetně velikosti a základního vybavení chovného zařízení, způsobu chovu, výživy, odchytu a transportu*. Praha: Ministerstvo zemědělství ČR, 2003. ISBN 80-708-4383-7.

[2] PETRÁČKOVÁ, Věra a Jiří KRAUS. *Akademický slovník cizích slov: [A-Ž]*. Praha: Academia, 1997. ISBN 80-200-0607-9.

[3] Understanding UVA, UVB, and UVC Reptile Lighting. In: *Zilla* [online]. [cit. 2019-10-15]. Dostupné z: https://www.zillarules.com/articles/understanding-uva-uvb-uvc-reptile-lighting

[4] HUDEČKOVÁ, Monika. *Teorie barveného vidění*. Brno, 2017. Bakalářská práce. Masarykova univerzita, Lékařská fakulta.

[5] ZWART, Peernel a R. KORBEI. Ophthalmology in Reptiles: Anatomy and visual perception of the reptilian eye with special attention to management and animal welfare. In: *Fd*. Mnichov, 2014.

[6] Ultraviolet Light in the Vivarium: The Benefits and the Risks. In: *UV Guide UK* [online]. [cit. 2019-10-17]. Dostupné z: http://www.uvguide.co.uk/uvinviv.htm

[7] UHROVÁ, Iva. *Termoregulace obratlovců*. Brno, 2008. Diplomová práce. Masarykova univerzita, Přírodovědecká fakulta. Vedoucí práce Mgr. Jiří Pacherník, Ph.D.

[8] *How to Set Up a Thermal Gradient for Thermoregulation in Pet Reptiles* [online]. [cit. 2019-10-17]. Dostupné z: https://www.thesprucepets.com/how-to-set-up-a-thermal-gradient-1239118

[9] BURGER, R. Michael. Reptile Water Requirements. *Reptiles* [online]. [cit. 2019-10-17]. Dostupné z: http://www.reptilesmagazine.com/Reptile-Health/Habitats-Care/Reptile-Water-Requirements/

[10] FÜRBACH, Jakub. *Regulátor prostředí terária*. Pardubice, 2014. Bakalářská práce (Bc.). Univerzita Pardubice, Fakulta elektrotechniky a informatiky.

[11] STANTON, Christopher G. Vivarium Temperature and Humidity Control with Arduino. In: *Hackster* [online]. 2017 [cit. 2019-10-18]. Dostupné z: https://www.hackster.io/stanto/vivarium-temperature-and-humidity-control-with-arduino-6aa38a

[12] *FishLore.com's Freshwater Aquarium e-Book* [online]. 2013 [cit. 2021-11-04]. Dostupné z: https://www.fishlore.com/freshwater-aquarium-book.pdf

[13] *Basic aquarium guide* [online]. Montreal: Rolf C. Hagen Inc. [cit. 2021-11-04]. Dostupné z: http://www.hagen.com/pdf/aquatic/bag.pdf

[14] KEPPLER, Rainer, Michael KOŠŤÁL. JBL GMBH & CO. KG. *První akvárium: Užitečné rady začínajícím akvaristům* [online]. 5. přepracované vydání. 2014, 39 s. [cit. 2021-11-07]. EAN:

4014162060617. Dostupné z:
https://www.jbl.de/cs/sthnout/11653/Ratgeber/JBL_Prvi_akvarium_cz.pdf

[15] Beat the Heat: Aquarium Cooling Methods. *Reefkeeping Magazine... an online magazine for the marine aquarist* [online]. [cit. 2021-11-07]. Dostupné z: http://reefkeeping.com/issues/2003-06/nftt/

[16] GUILLEN, Guillermo Perez. *Aquarium Monitoring with AWS-Seeed-Soracom* [online]. [cit. 2021-11-03]. Dostupné z: https://www.hackster.io/guillengap/aquarium-monitoring-with-aws-seeed-soracom-a361eb

[17] Introduction. *Arduino* [online]. [cit. 2020-04-08]. Dostupné z: https://www.arduino.cc/en/Guide/Introduction

[18] Arduino Products. *Arduino* [online]. [cit. 2020-04-13]. Dostupné z: https://www.arduino.cc/en/Main/Products

[19] PRAX, Jakub. *Využití sběrnice I2C pro komunikaci s externím zařízením*. Antonínská 548/1, 601 90 Brno, 2007. Diplomová práce. Vysoké učení technické v Brně. Vedoucí práce Ing. Tomáš Frýza Ph.D.

[20] *ESP32: Technical Reference Manual*. V4.4. Espressif Systems, 2021.

[21] *ESP32-WROOM-32: Datasheet*. Version 3.1. Espressif Systems, 2021.

[22] YADAV, Vikram. React Native vs.Flutter vs. Ionic Which is Challenging Platform for Your App. *Hackernoon* [online]. [cit. 2020-04-11]. Dostupné z: https://hackernoon.com/react-native-vsflutter-vs-ionic-which-is-challenging-platform-for-your-app-623qp3yqm

[23] *Ionic Framework* [online]. 2. 4. 2020 [cit. 2020-04-11]. Dostupné z: https://ionicframework.com/docs

[24] *Flutter Documentation* [online]. [cit. 2020-04-10]. Dostupné z: https://flutter.dev/docs

[25] LERER, Wm. Why Flutter Uses Dart. *Hackernoon* [online]. 2017 [cit. 2020-04-10]. Dostupné z: https://hackernoon.com/why-flutter-uses-dart-dd635a054ebf

[26] ZVĚŘINA, Tomáš. Flutter.io – mobilní aplikace, znovu a lépe. *Zdroják.cz* [online]. 2017 [cit. 2020-04-10]. Dostupné z: https://www.zdrojak.cz/clanky/flutter-io-mobilni-aplikace-lepe/

[27] *Firebase Documentation* [online]. [cit. 2020-04-12]. Dostupné z: https://firebase.google.com/docs

[28] AC Phase Control. In: *Arduino Playground* [online]. [cit. 2020-04-19]. Dostupné z: https://playground.arduino.cc/Main/ACPhaseControl/

[29] OŽANA, Štěpán. *Navrhování a realizace regulátorů: učební text*. Ostrava: Vysoká škola báňská - Technická univerzita, 2012. ISBN 978-80-248-2605-9.

[30] ASTRÖM, Karl Johan a Richard M. MURRAY. *Feedback Systems: An Introduction for Scientists and Engineers*. V2.10a. Princeton, New Jersey: Princeton University Press, 2008. ISBN 978-0-691-13576-2.

[31] ÅSTRÖM, Karl Johan a Tore HÄGGLUND. *PID Controllers: Theory, Design, and Tuning*. 2nd. Research Triangle Park, North Carolina: ISA - The Instrumentation, Systems and Automation Society, 1995, 343 s. ISBN 1-55617-516-7.

[32] Services overview. *Android Developers* [online]. [cit. 2021-11-14]. Dostupné z: https://developer.android.com/guide/components/services

[33] *The perfect multi-button input resistor ladder* [online]. [cit. 2021-10-25]. Dostupné z: http://www.ignorantofthings.com/2018/07/the-perfect-multi-button-input-resistor.html

8 Seznam použitých symbolů a zkratek

API	Aplication programming interface
CoAP	Constrained Application Protocol
FCM	Firebase Cloud Messaging
GPIO	Generic-Purpose Input/Output
HTTP	Hypertxt Transfer Protocol
IoT	Internet of Things
ISR	Interrupt Service Routine
JSON	JavaScript Object Notation
NPM	Node Package Manager
NVM	Non-Volatile Memory
OTA	Over The Air
REST	Representation State Transfer
RPC	Remote Procedure Call
RTC	Real-Time Clock
RTDB	Realtime Database
RTOS	Real Time Operating System
SSE	Server-Sent Events
UUID	Universally Unique Identifier
WDT	Watchdog Timer
Z-N	Ziegler-Nichols

Seznam tabulek

Tabulka 1 - Specifikace mikrokontrolerových desek Arduino 18
Tabulka 2 - Rozdělení paměti Particle Photon 20
Tabulka 3 - Přehled spotřeby komponent 35
Tabulka 4 - Ziegler-Nicholsova pravidla přechodové charakteristiky 41
Tabulka 5 - Přehled zapojení komponent na víku 43
Tabulka 6 - Přehled zapojení komponent uvnitř boxu 44
Tabulka 7 - Finanční náklady použitých komponent 47

Seznam schémat

Schéma 1 - Systém s centrální jednotkou 30
Schéma 2 - Systém bez centrální jednotky 31
Schéma 3 - Schéma periferií regulační a ovládací jednotky 32
Schéma 4 - Návrh DPS pro víko 43
Schéma 5 - Schéma zapojení komponent 45
Schéma 6 - Návrh PCB desky pro řídící jednotku Akvária 46
Schéma 7 - Schéma obrazovek mobilní aplikace 49
Schéma 8 - Schéma závislostí tříd pro firmware kamery 83
Schéma 9 - Dědičnost třídy BaseAlarm 96
Schéma 10 - Provolávání funkcí při změně budíku 97

Seznam grafů

Graf 1 - Kroková odezva systému s otevřenou smyčkou [31] 40
Graf 2 - Aproximace naměřených hodnot pH sondy lineární funkcí 102
Graf 3 - Kroková odezva akvária 107
Graf 4 - Výsledek regulátoru nastaveného pomocí metody Z-N 114
Graf 5 - Testování PID regulátoru 115
Graf 6 - Ovlivnění termostatu zamrzajícím vláknem 116
Graf 7 - Vývoj teploty za použití termostatu během 6 dní 117
Graf 8 - Stabilizace nové teploty při použití termostatu 117

Seznam kódu

Kód 1 - Příklad deklarace bezpečnostních pravidel Cloud Storage 26
Kód 2 - Ukázka kódu pro knihovnu LiquidCrystal_I2C 33
Kód 3 - Ukázka kódu pro práci s teplotním čidlem 36
Kód 4 - Inicializace PWM v ESP 38
Kód 5 - Ukázka ovládání krokového motoru 39
Kód 6 - Ukázka použití knihovny Freenove WS2812 Lib for ESP32 42
Kód 7 - JSON základních informací o viváriu v RTDB 58
Kód 8 - Bezpečností pravidla RTDB 60
Kód 9 - Bezpečností pravidla Firebase Storage 61
Kód 10 - Testování bezpečnostních pravidel 62

Kód 11 - Nastavení Android aplikace v souboru AndroidManifest.xml 63
Kód 12 - Nastavení webové aplikace 63
Kód 13 - Seznam definovaných cest ke stránkám aplikace 64
Kód 14 - Základní cesty aplikace 64
Kód 15 - Generování nových cest aplikace s parametry 65
Kód 16 - Ukázka využití Provideru 66
Kód 17 - Ukázka využití StreamProvideru 67
Kód 18 - Ukázka využití FutureProvideru 67
Kód 19 - Registrace kanálu pro FCM 69
Kód 20 - Třída VivariumUser 73
Kód 21 - Třída SensorDataHistory 74
Kód 22 - Diagram třídy BluetoothConnector 75
Kód 23 - Připojení pomocí Bluetooth 76
Kód 24 - Spárování řídící jednotky s uživatelem 77
Kód 25 - Nastavení Bluetooth charakteristiky 77
Kód 26 - Přihlášení uživatele prostřednictvím Google účtu 78
Kód 27 - Nastavení parametru vivária 79
Kód 28 - Ukázka testování widgetů ve Flutteru 82
Kód 29 - Hlavní smyčka programu kamery 84
Kód 30 - Nastavení Wi-Fi u kamery 85
Kód 31 - Pořízení fotografie 85
Kód 32 - Struktura tlačítka 86
Kód 33 - Inicializace Firebase 87
Kód 34 - Inicializace Bluetooth serveru 88
Kód 35 - Tvorba BLE charakteristik 89
Kód 36 - Stream dat do vivária 91
Kód 37 - Deklarace třídy ParallelRunner 92
Kód 38 - IFirebaseModule 94
Kód 39 - deklarace struktur Trigger a PayloadTrigger 95
Kód 40 - Ověření verze firmwaru 98
Kód 41 - Spuštění OTA aktualizace firmwaru 99
Kód 42 - Získání URL pro stažení nové verze firmwaru 99
Kód 43 - Příklad kódu hlavního souboru vivária 100
Kód 44 - Čtení hodnoty pH 103
Kód 45 - Výpočet vzdálenosti 104
Kód 46 - Ovládání čerpadla 105
Kód 47 - Příklad testování třídy PlainAlarm 109
Kód 48 - Nastavení akvária 110
Kód 49 - Nastavení akvária v RTDB 111

Seznam obrázků

Obrázek 1 - FCM Architektura 28
Obrázek 2 - Fázové řízení střídavého proudu [28] 34
Obrázek 3 - Navigační okno mobilní aplikace 50
Obrázek 4 - Přihlašovací obrazovka mobilní aplikace 51
Obrázek 5 - Seznam dostupných zařízení v mobilní aplikaci 52
Obrázek 6 - Přehled zařízení 54
Obrázek 7 - Nastavení zařízení 56
Obrázek 8 - Třída BluetoothDevice 72
Obrázek 9 - Třída CameraImage 72
Obrázek 10 - Třída Device 72
Obrázek 11 - Třída NavigationModel 73

Seznam příloh

Příloha č. 1 – Software

Příloha č. 2 – Schémata elektroniky

Příloha č. 3 – Zdrojová data 3D modelu

Příloha č. 4 – Fotodokumentace fyzického zařízení