Chytrý úl

Chytré zařízení sbírající informace o ekosystému úlu

Vojtěch Ištvánik

Maturitní práce

Obsah

[Anotace 3](#_Toc128681908)

[Koncept 3](#_Toc128681909)

[Chytrý úl 3](#_Toc128681910)

[Webová aplikace 5](#_Toc128681911)

[Databáze 6](#_Toc128681912)

[Postup 7](#_Toc128681913)

[Webová aplikace a databáze 8](#_Toc128681914)

[Databáze a její propojení s webovou aplikací 8](#_Toc128681915)

[Přidání MVC architektury, rozdělení aplikace na oblasti a testování směrování 9](#_Toc128681916)

[Uživatel 11](#_Toc128681917)

[Admin 12](#_Toc128681918)

[Virtuální zařízení a jejich zobrazení 12](#_Toc128681919)

[Rest API 14](#_Toc128681920)

[Úl 14](#_Toc128681921)

[Propojení s arduino 14](#_Toc128681922)

[Sběr informací 14](#_Toc128681923)

[Databáze 14](#_Toc128681924)

[Poslání informací 14](#_Toc128681925)

[Display 14](#_Toc128681926)

[Kamera 14](#_Toc128681927)

[Závěr 15](#_Toc128681928)

[Zdroje 15](#_Toc128681929)

# Anotace

Smyslem projektu je sestavit zařízení, které bude v základu zaznamenávat a ukládat elementární měřitelné informace úlu. Mezi tyto údaje patří váha, teplota, vlhkost a tlak uvnitř našeho objektu. Záznamy z tohoto pozorování mohou být v budoucnu použity pro výzkum, strojové učení, nebo pro objevení nežádoucích událostí, jako je například napadení úlu škůdcem nebo nemocí jenom s pomocí webové aplikace.

# Koncept

Projekt se skládá z a chytrého úlu a serveru. Úl má za úkol sbírat informace uvnitř a venku úlu. Naměřené hodnoty následně po určitém intervalu odešle webové aplikaci, ze které si uživatel bude moci zobrazit informace z konkrétního úlu.

## Chytrý úl

Hlavní myšlenkou chytrého úlu je sběr informací, které lze využít pro diagnostiku stavu včelstva. Nasbírané informace se také mohou využít pro další výzkum včel. Celá inteligentní část úlu se nachází uvnitř krabičky, kterou lze přidělat na venek úlu.

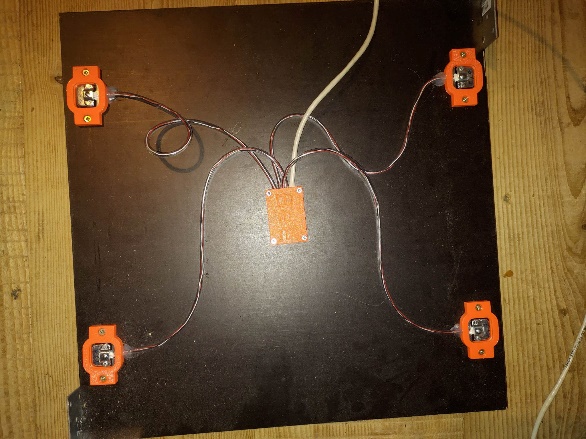
Primárním komponentem chytrého úlu je mikropočítač Raspberry Pi 4. Mikropočítač v úlu se využívá na několik činností:

1. Dočasné uložení nasbíraných informací
2. Bezdrátová komunikace
3. Odesílání nasbíraných informací na webový server

V mikropočítači je využívaný databázový systém PostgreSQL jako uložiště informací.

V mikropočítači běží program, který provádí 3 procesy najednou. První proces má za úkol opakovaně získávat data z mikrokontroleru a ukládat je do databáze. Druhý proces uložená data odesílá na server. Třetí proces odesílá obrázky získané z kamery. Program je možné ovládat pomocí uživatelského rozhraní na dotykové obrazovce úlu. Uživatelské rozhraní používá knihovnu TKinter.

Mikrokontroler Arduino Nano slouží jako sběratel informací. K mikrokontroleru jsou připojeny čidla pro naměření váhy úlu, vnitřní a venkovní teploty, vnitřní vlhkosti a tlaku. Na příkaz od mikropočítače, mikrokontroler nasbírá data ze senzorů, převede je do pochopitelných informací, a odešle je zpět mikropočítači. Mikropočítač komunikuje s mikrokontrolerem za pomocí sériové komunikace.

V úlu jsou použity senzory BMP280 (obr. 2) pro měření hodnot venkovní teploty a tlaku, DHT11 (obr. 3) pro vnitřní vlhkost a teplotu a soustavy pro měření váhy složené ze čtyř 50 kg váhových senzorů a jednoho AD převodníku HX711 (obr. 4).

Obrázek 3

Obrázek 2

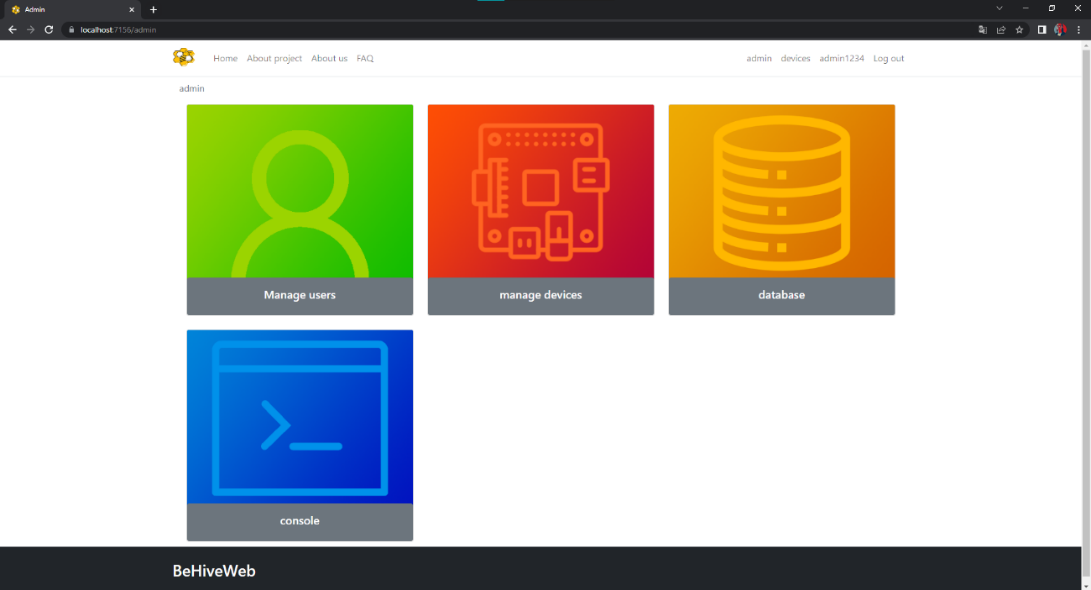
Obrázek

Napájení úlu je řešeno za pomocí USB kabelu, který se připojí do zdroje. Tímto způsobem lze napájet úl například z nabíječky, počítače, nebo solárního panelu a baterie.

K úlu je připojený i dotyková obrazovka MSP2807, přes kterou je možné chytrý úl ovládat.

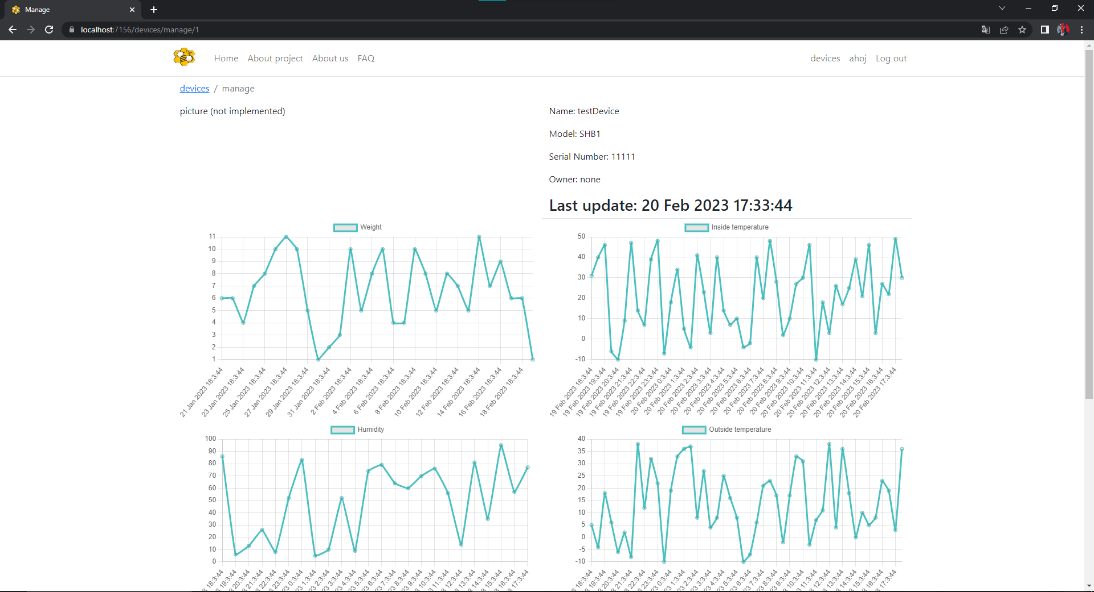
## Webová aplikace

Webová aplikace běží na frameworku ASP.NET Core pracujícím v C# 6.0. Webová aplikace je navržená tak, aby bylo možné v budoucnosti jednoduše přidat další druhy zařízení. Pro komunikaci s databází se používá knihovna Entity Framework Core. Aplikace podporuje https. Pro autorizaci, autentifikaci a uživatele se používá knihovna ASP.NET Core Identity. Aplikace se skládá z několika oblastí, které mají každá jiný účel.

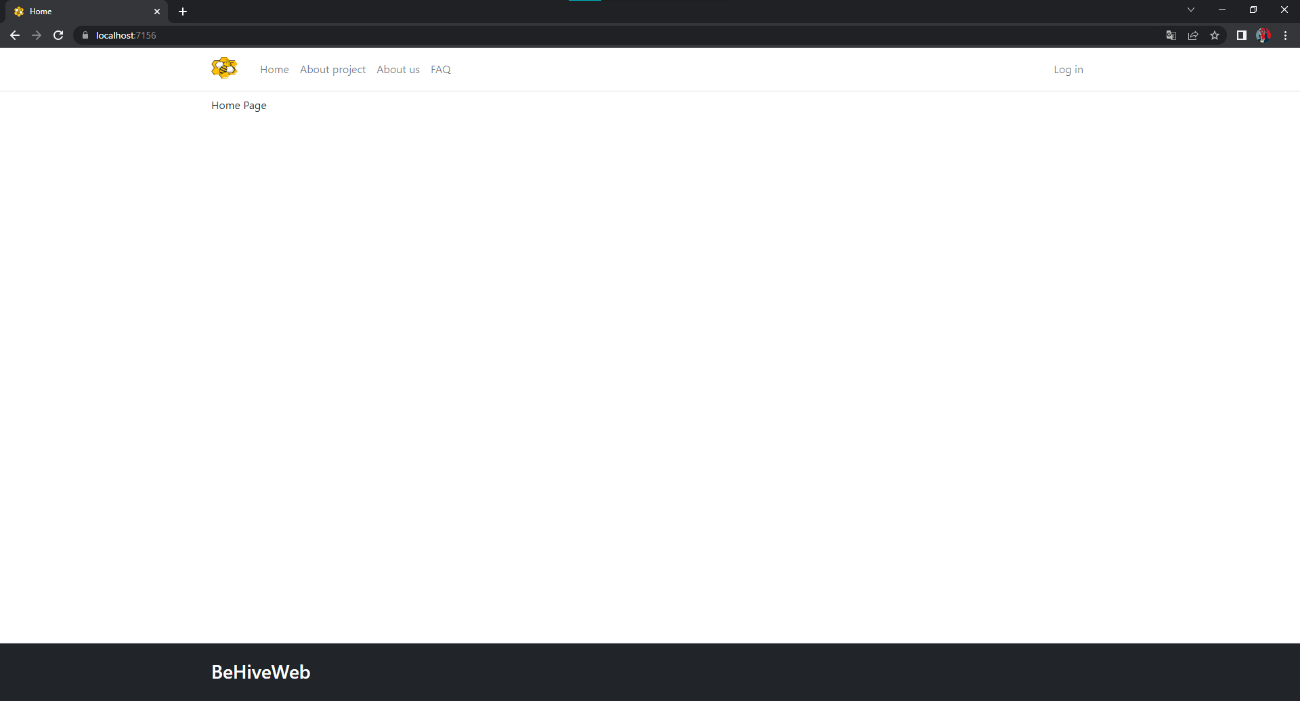
Oblast AdminArea je využívána pro administrátorské potřeby. Nachází se zde základní nástroje pro správu uživatelů a zařízení. Do této oblasti má přístup uživatel s rolí admin.

Obrázek

V oblasti RestArea se nachází akce pro vyřešení požadavků od IoT zařízení.

Oblast UserArea je pro přihlášené uživatele. Jsou zde akce pro autentifikaci a zobrazení všech přidaných zařízení a jejich vlastností a dat.

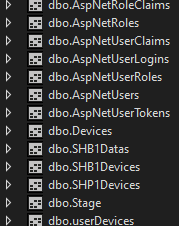
Obrázek

Je zde také nejmenovaná oblast pro základní stránky (jako home page) a pro layouty.

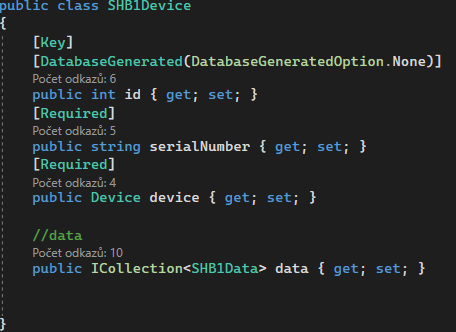
Obrázek

## Databáze

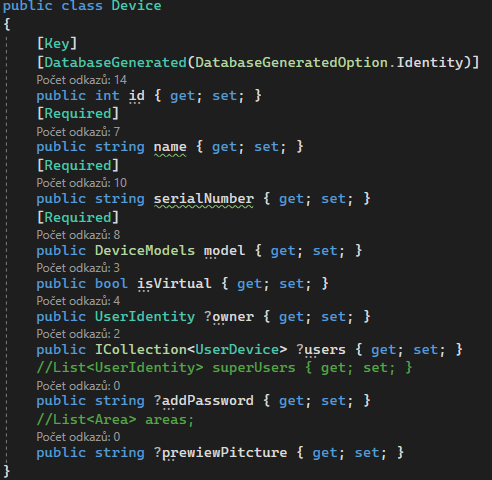
Pro databázi se používá databázový systém MSSQL. Databáze se nachází na serveru, na kterém je i webová aplikace. Databáze, pokud neexistuje, je automaticky vytvořena při spuštění webové aplikace.



Obrázek

Struktura databáze byla navržena tak, aby se všechna zařízení mohla nacházet ve stejné tabulce. Zařízení zde mají vždy dva osobní záznamy. V tabulce Devices se nachází jejich hlavní záznam (obr. 10). Nachází se zde hlavně údaje o jejich modelu a uživatelích. Druhý záznam (obr. 9) je modelový (nachází se v tabulce stejné s názvem jejich modelu) a je závislí na záznamu z tabulky Devices. Zde jsou odkazy na záznamy získaných informací. Oba záznamy mají stejné identifikační číslo.

Obrázek (model)



Obrázek (model)

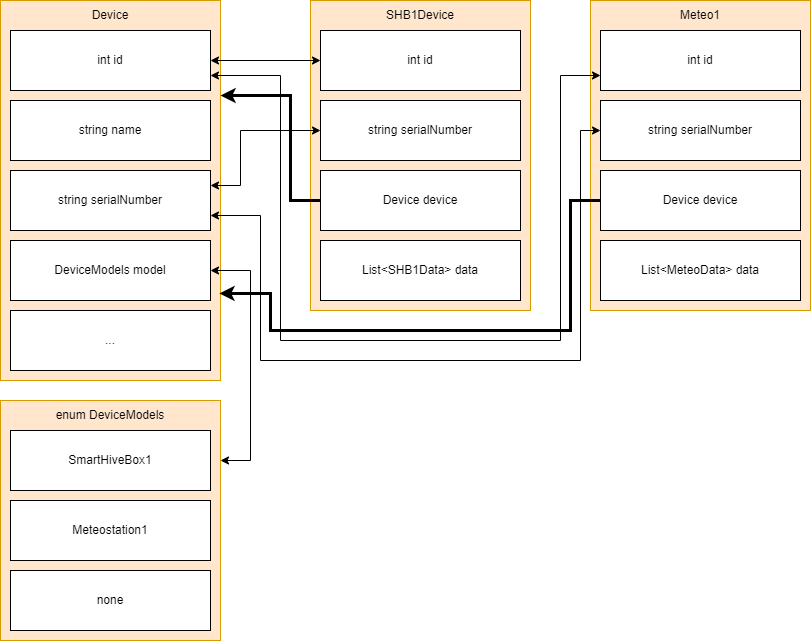
Uživatelé se nacházejí ve stejné databázi jako zařízení. Hesla uživatelů jsou šifrována. Záznamy uživatelů jsou rozšířeny o vlastněné úly a členství v úlech. Uživatelům může být přiřazena role admin.

# Postup

Tento projekt je dokončení již rozpracovaného projektu, na kterém se pracovalo v minulém roce (2022) jako projekt na krajskou soutěž Dobrá škola – Moderní škola 4.0. Úl je již v podstatě fyzicky hotový. Především se bude pracovat na jeho obecném vylepšení a opravu nevyřešených problémů.

## Webová aplikace a databáze

### Databáze a její propojení s webovou aplikací

Začal jsem s výrobou nové webové aplikace, která nahradí starou. Nová webová aplikace by měla být dělaná tak, aby nebylo tak složité přidat další nová zařízení. Všechna zařízení mají nově záznam ve společné tabulce **Devices**. Záznamy ve společné tabulce umožňuje jednoduché vypisování zařízení do jednoho listu nehledě na to, jakého jsou druhu. Každé zařízení má tedy dva záznamy. Hlavní záznam je ten, který se nachází v tabulce **Devices**. Hlavní záznam obsahuje informace o unikátním náhodně vygenerovaném sériovém číslu (toto číslo má mnoho využití především při identifikaci), modelu zařízení, uživatelích, majiteli a dalších vedlejších informací. Sekundární záznam je v tabulce přímo určené pro daný model zařízení. Sekundární záznam v základu obsahuje sériové číslo a odkaz na tabulku obsahující nasbírané informace ze zařízení. Sekundární záznam ukazuje na primární záznam hodnotou **device**. Primární záznam v databázi nijak neukazuje. Za pomocí primárního záznamu je sekundární tabulka vytažena až po určení modelu zařízení. Po určení modelu si na sekundární záznam můžeme ukázat za pomocí hodnot **id** a **serialNumber**, které mají oba záznamy stejné.

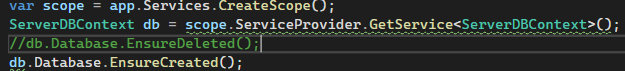
Obrázek

Je potřeba informace získané ze zařízení nějak ukládat do databáze. Zvolil jsem službu Entity Framework Core, která mi umožňuje vytvořit jednoduché a rychlé řešení mého problému. Služba umí tzv. objektově relační mapování (ORM), které při práci s databází dokáže převádět záznamy v databázi na objekty stejného modelu v programu, a naopak převést požadavek z programu na provedení nějaké akce se záznamy přímo na určitou akci v databázi (toto byl jen jednoduchý popis). Tato služba mi tedy umožňuje získat záznam jen za pomocí jednoduchého volání funkce (obr. 12) bez potřeby vytvářet vlastní query string, který bych musel v průběhu vývoje neustále měnit.



Obrázek

Entity Framework dále umožňuje manipulaci z databází. Díky této možnosti jsem přidal do programu kód, který v případě absence databáze vytvoří novou databázi.



Obrázek

Pro připojení do databáze potřebujeme tzv. connection string. Databázový systém, který jsem se rozhodl použít, je Microsoft SQL (MSSQL). Na internetu jsem si našel stránku [www.**connectionstrings.com**](http://www.connectionstrings.com), na které jsem si našel connection string přímo určený pro MSSQL. Connection string jsem si upravil pro svojí aplikaci, a vložil jsem ho do nastavení webové aplikace (obr. 14).



Obrázek

Pro otestování funkčnosti Entity Frameworku jsem si vytvořil vlastní službu **TestingDeviceCreator**, která mi umožňuje vygenerovat entitu virtuálního zařízení, kterému budou přiděleny informace obsahující virtuální hodnoty.

### Přidání MVC architektury, rozdělení aplikace na oblasti a testování směrování

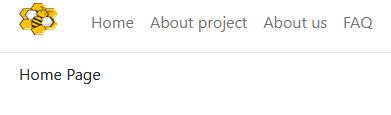
Propojení s databází je hotové. Je načase se pustit do vizuálního zobrazení webové aplikace. První jsem potřeboval přidat službu, která by mi umožňovala využívat architekturu MVC (Obr. 15).



Obrázek

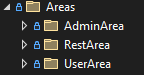
MVC architektura v ASP.NET CORE potřebuje zvláštní strukturu adresářů. Do kořenového adresáře aplikace jsem tedy přidal adresář Models, který obsahuje všechny základní modely, které se budou používat v celé aplikaci, adresář Controllers, který bude obsahovat controllery pro zobrazení, která nebudou potřebovat uživatele, který by byl přihlášený, a adresář Views, kde se nachází zobrazení pro akce v controllerech. Adresář Views také obsahuje layout stránek.

Vytvořil jsem controller Main, do kterého jsem přidal akci index. Tato akce by mi měla vrátit zobrazení hlavní stránky webové aplikace. Akce index aby mohla něco vrátit, tak potřebuje něco, co by mohla vrátit. Přidal jsem do aplikace zobrazení **index.cshtml**. Zobrazení obsahuje pouze text „Home Page“ pro otestování funkčnosti. Stále ale nelze zobrazit dané zobrazené, protože v projektu chybí definovaná cesta, která by na danou akci v controlleru ukázala. Přidal jsem tedy do směrování cestu k danému controlleru. Nyní si lze zobrazit hlavní stránku aplikace. Do aplikace jsem přidal ještě layout, který se bude chovat jako forma pro stránky webové aplikace.



Obrázek

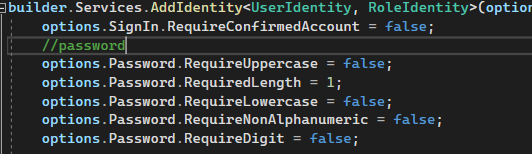
Je načase rozdělit projekt do oblastí, které budou mít každá vlastní účel. Využil jsem generátoru položek, který je jedním z nástrojů Visual studia, aby vytvořil tři oblasti. Vytvořil jsem oblast AdminArea, která je určená pro administrativní účely, dále oblast RestArea, která bude obsahovat rozhraní pro rest požadavky, a oblast UserArea, která bude pouze pro přihlášené uživatele. Otestoval jsem směrování do každé z nich.



Obrázek

### Uživatelé, přihlašování a autorizace

Abych uživatelům zamezil volný přístup tam, kam se jim zachce, přidám do projektu službu, která bude schopna spravovat uživatelské účty a poskytne mi nástroje pro nastavení autorizace uživatelů do určitých akcí. Zvolil jsem proto službu ASP.NET Core Identity, která tyto požadavky splňuje. Službu jsem nastavil tak, aby nezpomalovala práci při vývoji aplikace.



Obrázek

Nyní, když byla služba přidána, je potřeba upravit třídu ServerDBContext, který reprezentuje tzv. **seassion** mezi databází a webovou aplikací, aby osahovala vše potřebné pro službu Identity. Třídě jsem tedy upravil dědičnost na IdentityDbContext. Nyní vytváříme i tabulky, které budou obsahovat uživatele, jejich role a vše ostatní potřebné pro správné fungování služby.



Obrázek

Uživatele jsem upravil tak, aby mohly obsahovat vlastněná zařízení a zařízení, do kterých mají přístup. Protože se bude pro přístup do zařízení ukazovat na obou dvou stranách několikrát na několik objektů (tzv. many-to-many), vytvořil jsem pro ukazování třídu UserDevice. Přidal jsem službu RoleCreator, která automaticky vytvoří roli admin, která bude mít více oprávnění. Vytvořil jsem službu DefaultAdminCreator, která automaticky vytvoří defaultní administrátorský účet na webové aplikaci.

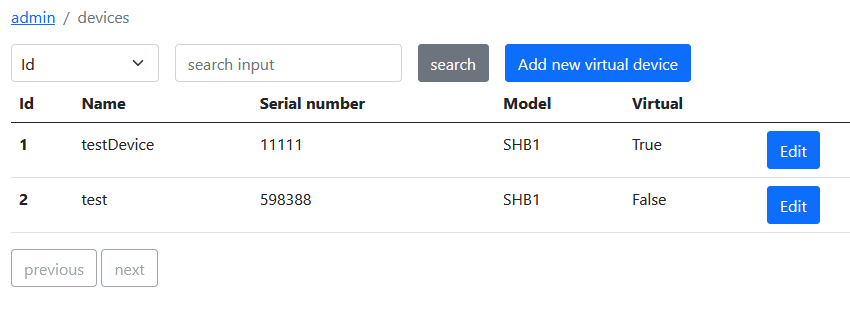
Pustil jsem se do vytváření přihlašování uživatelů. Přihlašování se nachází v oblasti UserArea v controlleru Authentication. Uživatelé jsou nyní schopni se přihlásit. Přidal jsem také akci logout, díky které se uživatelé mohou odhlásit. Na to, abychom se dostali na tyto stránky, je potřeba upravit layout, do kterého jsem přidal odkazy na přihlašování a odhlašování. Tyto odkazy se zobrazí jen v určitých případech, ve kterých uživatel buď je, a nebo není přihlášen.

### Admin

Admin má pro sebe určenou vlastní oblast, ve které se budou nacházet nástroje pro správu webové aplikace. Admin by měl být schopen si zobrazit aktuální log webové aplikace, spravovat uživatele, spravovat zařízení a spravovat samotnou databázi. Potřebuju ale jen malý základ pro vývoj aplikace, takže vytvořím jen jednoduchou správu zařízení.

Začal jsem s vytvářením hlavní stránky admina. Na této stránce se bude moci uživatel dostat na stránky určené pro konkrétní správní akce. Stránka má vlastní controller HomeAdmin. Stránka osahuje čtyři ikonky, které nás přesměruj na vyžádané míso (obr. 5 na str. 5).

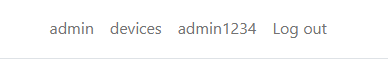
Základem správy zařízení je stránka, která obsahuje přehled všech zařízení na webové aplikaci. Na této stránce máme možnost vytvořit nové zařízení, vyhledat si zařízení, nebo dokonce editovat zařízení (tato funkce nebyla implementována).



Obrázek

### 

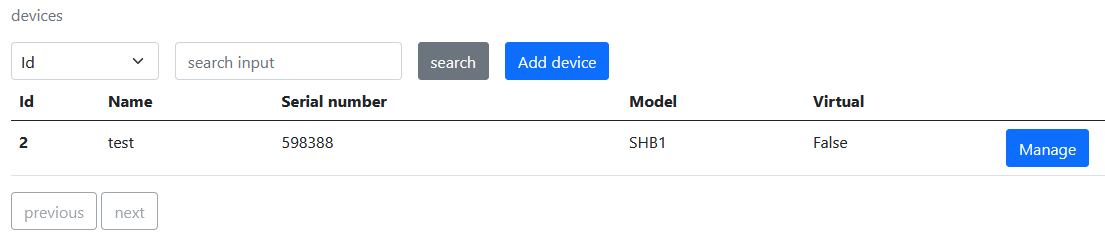
Na to, abych se dostal na hlavní stránku, je potřeba upravit layout. Přidal jsem do něj odkaz, která se zobrazí jen v případě, že uživatel je součástí skupiny admin.



Obrázek

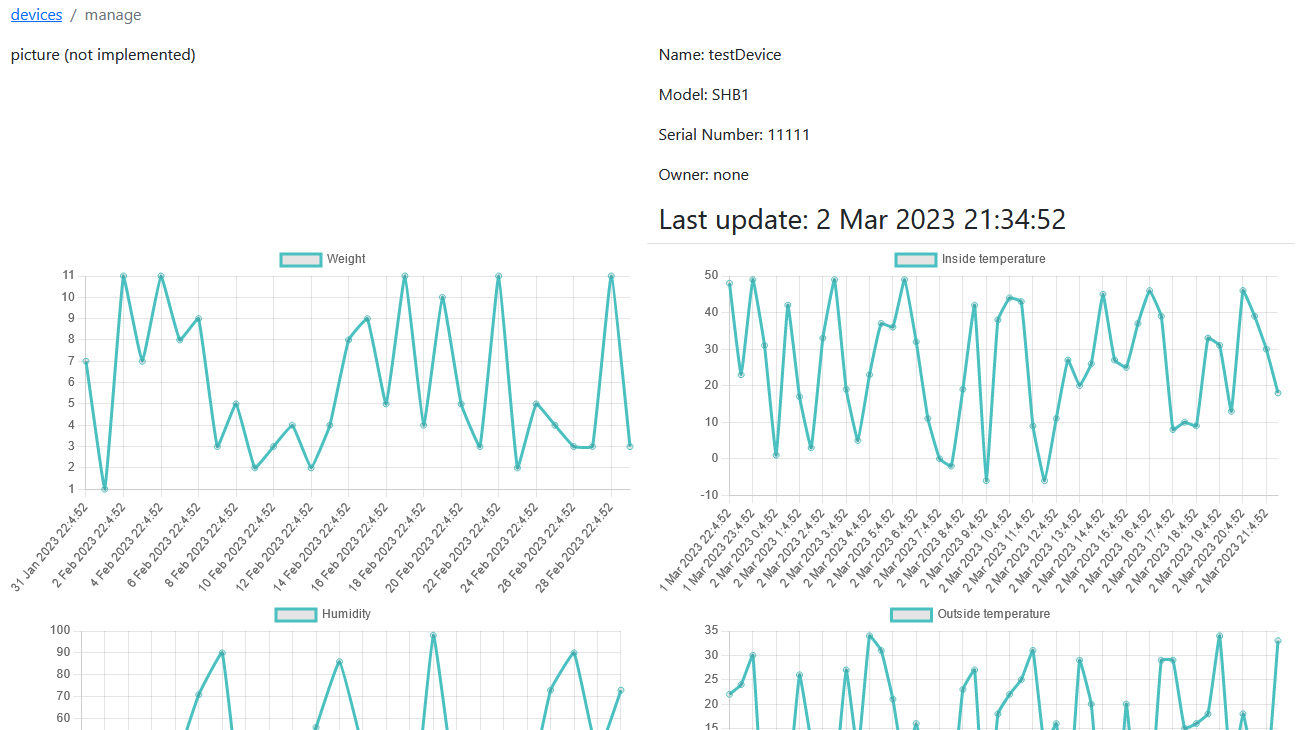
### Zobrazení pro zařízení

Nasbírané informace je potřeba si někde zobrazit, proto jsem se přesunul do oblasti UserArea, ve které vytvořím akce pro zobrazení listu zařízení a detailů zařízení. Vytvořil jsem controller Device, který je pro tyto akce určen. Hlavní stránka je skoro stejná, jako ta v adminovy, její dostupné funkce se ale každopádně liší. Na stránce si lze zobrazit záznamy všech zařízení, která uživatel vlastní, a vyhledávat určitá zařízení. Zařízení si lze i přidat, na to potřebujeme znát sériové číslo zařízení, které je náhodně generováno při jeho vytvoření. Pro daná zařízení je i odkaz, který nás přesměruje k jejich podrobnějšímu popisnému zobrazení.



Obrázek

Když už se umíme orientovat mezi našimi zařízeními, můžeme přidat akce určené pro podrobné zobrazení zařízení. Na tuto stránku se dostaneme jen za pomocí modelu, pro který je stránku určen. Stránka obsahuje všechny dostupné vlastnosti o zařízení, a zároveň obsahuje informace, které zařízení nasbíralo. Pro otestování tohoto zobrazení bylo vytvořeno virtuální zařízení se 43200 záznamy s náhodnými zařízenímy.



Obrázek (informace jsou náhodně vygenerovány)

Pro vykreslování grafů jsem využil javascriptové knihovny Chart.JS. Pro stylizaci stránek byl použit frontendový framework Bootstrap.

### Rest API

Pro REST pro zařízení je v oblasti RestArea jediný controller RestAPIDevice, který obsahuje funkce pro příjem informací ve formátu JSON. Informace s se zde mohou posílat jen v případě, že známe id a serialNumber daného zařízení.

Pro otestování funkčnosti akcí jsem vytvořil virtuální zařízení, které v určitých intervalech odesílá náhodné hodnoty na server. Modelem tohoto fyzicky neexistujícího zařízení je SmartHiveBox1 (SHB1).

## Úl

### Revize úlu

Úl jsem si propůjčil fyzicky již hotový, úl ale nebyl ze softwareové stránky vůbec schopný provozu. Jelikož se heslo stalo zapomenutým, jsem nový image Raspberry Pi OS do microSD karty pro raspberry pi. Nyní jsem schopný se do Raspberry Pi přihlásit. Po instalaci jsem aktivoval službu VNC a povolil sériovou komunikaci. Bezdrátově jsem se připojil na hotspot mého notebooku, abych mohl s úlem manipulovat i mimo domácí Wi-Fi síť.

I přes to, že jsem na úlu sám pracoval, tak jsem se pustil do přezkoumání toho, jak úl fungoval, a jak je zapojen. Zapojení, i když se to může přes směs zamotaných kabelů zdát, je poměrně jednoduché. Komunikace mezi Raspberry Pi a Arduinem probíha přes dva převodníky logické úrovně, které řeší problém mezi logickým napětím Raspberry Pi, které pracuje na 3,3V, a logickým napětím u Arduina, které má logické napětí 5V. Komunikace mezi nimi probíhá pomocí sériové komunikace. Pomocí I2C arduino komunikuje se senzorem BMP280. K Arduinu je ještě připojen senzor DHT11 a váha. U váhy nastal problém, a tím problémem byla její nefunkčnost. Váha vůbec nekomunikovala a generovala překvapivě vysoké teploty. Po rozebrání váhy jsem zjistil, že kabely VCC a GND byly prohozeny. Po přepojení kabelů na jejich místa váha začala fungovat bez problémů. K Raspberry Pi je ještě připojena kamera, kterou ale prozatím ponechám nevyužitou, protože pro ní nemám zobrazení na webové aplikaci.

### Přepsání programu Raspberry Pi a kódu Arduina

Hlavním kamenem úrazu je nefunkční a nestabilní komunikace mezi Arduinem a Raspberry Pi. Program nacházející se na Raspberry Pi má po neúspěšné komunikaci tendenci ukončit se kvůli chybě. Přidal jsem tedy výjimku, která tento problém vyřešila.

Sběr informací probíhá na příkaz z Raspberry Pi. Raspberry Pi odešle charakter ‚1‘ který Arduinu říká, aby nasbíral informace a odeslal je Raspberry Pi ve formátu JSON. Rozhodl jsem se tento způsob trochu upravit. Raspberry Pi nyní místo charakteru ‚1‘ odesílá zprávu „ACTION 1;“. Každá zpráva je zakončená středníkem, který určuje konec zprávy. Zprávám odeslaným z Arduina byl také přidám na konec středník. Program na Raspberry Pi jsem upravil tak, aby četl zprávu z Arduina do středníku. Úprava programu překvapivě efektivně opravila komunikaci mezi Arduniem a Raspberry Pi.

K odesílání informací se používaly seassiony kvůli bezpečnosti. Nová verze tuto funkci ale nepodporuje, takže jsem jí mohl klidně vymazat. Pro komunikaci s webovou aplikací jsou nyní potřeba jen hodnoty id a serial.

Přidal jsem kontrolu připojení s internetem a se serverem, která využívá příkaz ping. Když jeden z testů připojení selže, zařízení se nepokusí odeslat informace na server.

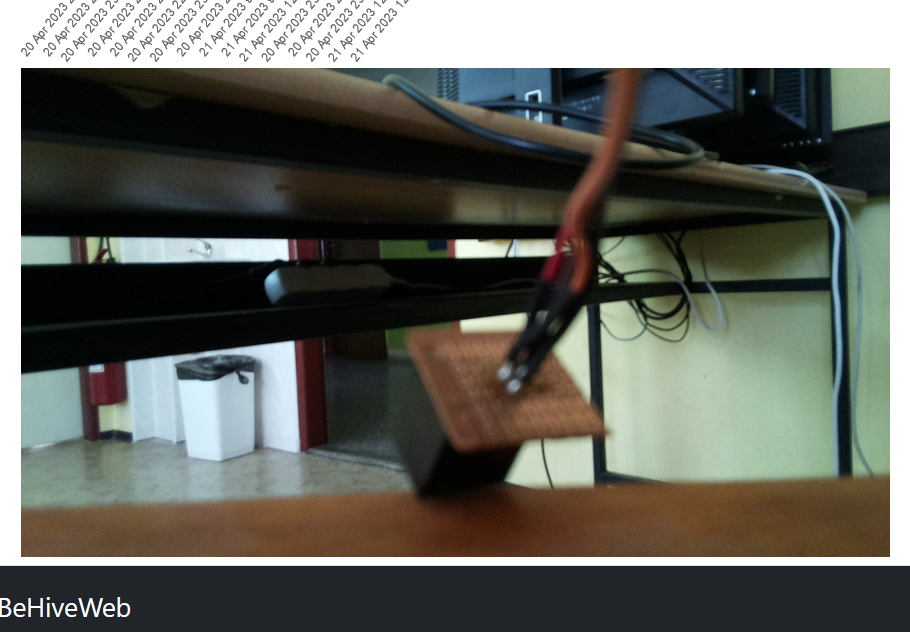
### Databáze

Přidal jsem na Raspberry Pi databázový klient PostgreSQL, kde si Raspberry PI bude do databáze bude dočasně ukládat nasbírané informace. Pro komunikaci programu s databází jsem si ale chtěl ušetřit práci s vypisováním sql požadavků, a tak jsem využil knihovny SQLAlchemy, se kterou mohu pracovat jako s knihovnou Entity Framevork na webové aplikaci.

### Kamera

Na spodku úlu je připevněna kamera Rasberry Pi kamera V2, která se dívá na vchod do úlu. Aktivoval jsem kameru v nastavení raspi-config a napsal kód pro odesílání kamerou zabraných fotek na webovou aplikaci.

Na webové aplikaci jsem použil REST API pro vytvoření rozhraní pro příjem obrázků od úlu. Obrázek z úlu je poté zobrazen na stránce úlu (obr. 21).



Obrázek

### Procesy

Program se často „zasekával“ na časově náročných procesech (sběr dat, odesílání dat, odesílání obrázků). Rozhodl jsem se tedy, že z částí kódu, které se budou neustále opakovat, vytvořím procesy.

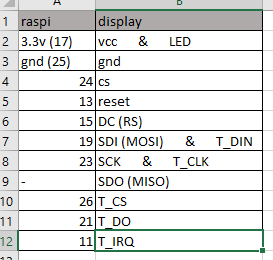
K tomuto úkolu jsem využil knihovnu threading, která mi tento pro tento problém nabízí jednoduché řešení.

Z částí kódu pro sběr dat, odesílání dat a sběr obrázků jsem vytvořil procesy, které umožňují plynulý běh programu.

### Display

Interakce s úlem je velice omezená, proto jsem se rozhodl přidat dotykoví display MSP2807.

Narazil jsem na problém. Display nemá žádnou podporu pro Raspberry Pi. Naštěstí je display velice podobný obrazovce MPI2418, takže mi stačilo vymyslet vlastní zapojení do mikropočítače Raspberry Pi (obr. 22).



Obrázek (tabulka použitá pro orientaci při zapojování)

Nainstaloval jsem drivery pro display MPI2418 a po restartu display začal fungovat.

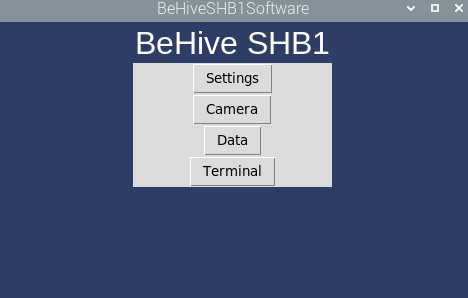
Nainstaloval jsem virtuální klávesnici Matchbox-keyboard.

Display z nějakého neznámého důvodu bránil kameře ve správném fungování. Problém jsem vyřešil zakázáním a následným povolením kamery v nastavení raspi-config.

### GUI

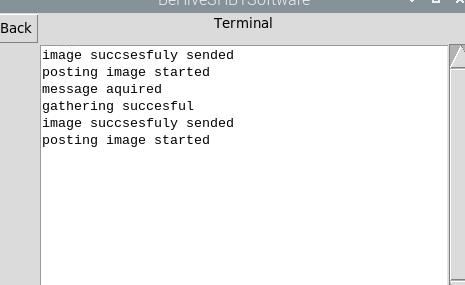
Pro ovládání programu jsem se rozhodl vytvořit GUI za pomocí knihovny TKinter.

Začal jsem s hlavním framem, který bude odkazovat na další framy (obr. 23).



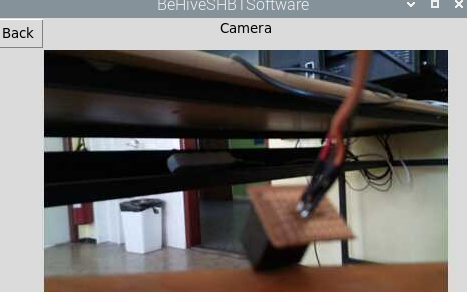
Obrázek

Začal jsem s framem terminál, který bude vypisovat output terminálu (obr. 24). Vypisování jsem dosáhl za pomocí funkce sys.stdout.



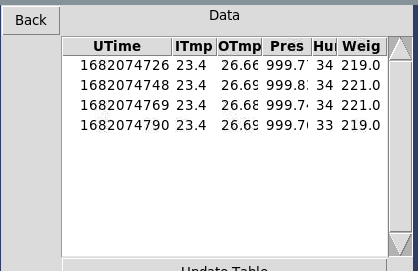
Obrázek

Dále jsem udělal frame camera, který bude zobrazovat nejnovější pořízený obrázek kamerou (obr. 25).



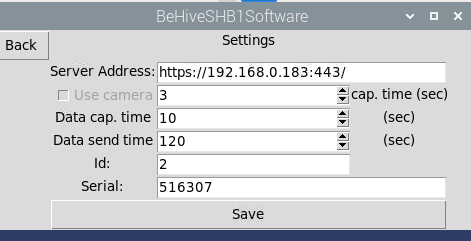
Obrázek

Poté jsem přidal frame, který mi ukáže všechna uložená data v databází (obr. 26). Přídáním jsem zároveň nalezl chybu s časem, kterou jsem také ihned opravil.



Obrázek

Skončil jsem nakonec u frameu settings, který mi umožňuje nastavovat si pracovní hodnoty úlu (obr. 27).



Obrázek

### Soubor config

Přidal jsem automaticky vytvářející se soubor, do kterého se budou ukládat pracovní hodnoty úlu. Při spuštění programu se hodnoty z tohoto souboru načtou do programu. Hodnoty uložené ve frameu settings se ukládají do tohoto souboru.

# Závěr

Vytvořil jsem novou webovou aplikaci. Chytrý úl jsem upravil tak, aby byl schopný pracovat s novou webovou aplikací. Komunikace u chytrého úlu mezi Arduinem a Raspberry Pi nyní po úpravě již nedělá takové problémy jako dříve.

Zprovoznil jsem kameru na úlu.

Zoptimalizoval jsem program s pomocí knihovny threading.

Přidal jsem display, přes který lze úl ovládat.

Přidal jsem GUI, přes které uživatel může ovládat program.

Přidal jsem konfigurační soubor, ze kterého se budou načítat pracovní hodnoty úlu (adresa serveru, id, serial, …).

# Zdroje

<https://github.com/MeivrCZ/BeHiveV2Server>

<https://learn.microsoft.com/cs-cz/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0>

<https://learn.microsoft.com/en-us/ef/>

<https://docs.sqlalchemy.org/en/20/>

<https://www.connectionstrings.com/>

<https://pyserial.readthedocs.io/en/latest/>

<https://arduinojson.org/>

<https://circuitjournal.com/50kg-load-cells-with-HX711>

<https://www.arduino.cc/reference/en/language/functions/communication/serial/>

<https://roboticsbackend.com/raspberry-pi-arduino-serial-communication/>

<https://projects.raspberrypi.org/en/projects/getting-started-with-picamera>

<https://ozzmaker.com/virtual-keyboard-for-the-raspberry-pi/>

<http://www.lcdwiki.com/2.4inch_RPi_Display>

<http://www.lcdwiki.com/2.8inch_SPI_Module_ILI9341_SKU:MSP2807>

<https://tkdocs.com/>