Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Vizualizációs megoldás IoT adat elemző rendszerhez

Szakdolgozat

Készítette
Kunkli Richárd

Konzulens
dr. Simon Csaba

2020. november 27.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Kunkli Richárd*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2020. november 27.

Kunkli Richárd
hallgató

Kivonat

Adott egy tanszéken fejlesztett felhő alapú elosztott rendszer, melynek eszközei madárhangok azonosítására képesek. Ha a rendszer úgy észleli, hogy az egyik álatala vezérelt eszköz mikrofonja felvételén madárhang található, akkor riasztást kezdeményez az eszközön ezzel elijesztve a madarat ezáltal megóvva a növényzetet.

A rendszernek több kisebb komponense van, amelyek rengeteg adatot dolgoznak fel és nincs jelenleg egy olyan egységes grafikus felület ahol a rendszer teljes állapotát át lehetne tekinteni, illetve ahol a feldolgozott adatokat vizualizálni lehetne.

A piacon létezik már több olyan szoftver csomag, amely hasonló problémákra próbál megoldást nyújtani, de ezek sem mindig tudják kielégíteni azokat a speciális igényeket, amelyek egy ilyen rendszernél felmerülnek.

Jelen szakdolgozat célja egy olyan vizualizációs megoldás bemutatása, amelynek segítségével a rendszer könnyedén áttekinthető és kezelhető. A tanszéki rendszer által kezelt eszközök a felületen is vezérelhetők és azok működéséről különböző statisztikákat felhasználva egyszerűen értelmezhető diagrammok generálódnak.

A backend megvalósítására az ASP.NET Core-t választottam, mely platformfüggetlen megoldást nyújt a web kérések kiszolgálására. A frontend-et a React.js használatával készítettem, mely segítségével egyszerűen és gyorsan lehet reszponzív felhasználói felületeket készíteni. Dolgozatomban bemutatom a tanszéken fejlesztett rendszert, a mikroszolgáltatások vizualizálásának alternatíváit, ismertetem az általam választott technológiákat és a készített alkalmazás felépítését.

Abstract

There is a department developed cloud-based distributed system whose devices are capable of identifying bird sounds. If the system detects a bird's voice on the recording of a microphone on one of the devices, it will trigger an alarm on the device scaring the bird away thereby protecting the vegetation.

The system has several smaller components that process a lot of data and currently there is no unified graphical user interface where the overall state of the system could be reviewed or where the processed data could be visualized.

There are already several software packages on the market that try to solve similar problems, however they aren't always able to meet the special needs that arise with such a system.

The purpose of this thesis is to present a visualization solution that allows the users to easily review and manage the system. The devices maintained by the department developed system can be controlled on the interface and easy-to-understand diagrams are generated using statistics about their operation.

I chose ASP.NET Core as the backend framework, which provides a platform-independent solution for serving web requests. The frontend was created using React.js, which allows for an easy and quick way to create responsive user interfaces. In my thesis I present the system developed at the department, the alternatives of visualization of microservices, I describe the technologies I have chosen and the structure of the application I have created.

1. fejezet

Bevezetés

Szőlőtulajdonosoknak éves szinten jelentős kárt okoznak a seregélyek, akik előszeretettel választják táplálékul a megtermelt szőlőt. Erre a problémára dolgoztak ki a tanszéken diáktársaim egy felhő alapú konténerizált rendszert, a Birdnetes-t mely a természetben elkelyezett eszközökkel kommunikál, azokat vezérli. Az eszközök bizonyos időközönként hangfelvételt készítenek a környezetükről, majd valamilyen formában elküldik ezeket a felvételeket a központi rendszernek, amely egy erre a célra kifejlesztett mesterséges intelligenciát használva eldönti a felvételről, hogy azon található-e seregély hang vagy sem. Ha igen akkor jelez a felvételt küldő eszköznek, hogy szólaltassa meg a riasztó berendezését, hogy elijessze a madarakat.

1.1. A probléma

A jelen rendszer használata során nincs vizuális visszacsatolás az esetleges riasztásokról azok gyakoriságáról és a rendszer állapotáról sem. Különböző diagnosztikai eszközök ugyan implementálva lettek mint például a logolás vagy a hiba bejelentés, de ezek használata nehézkes, nem kézenfekvő. Szükség van valamire amivel egy helyen és egyszerűen lehet kezelni és felügyelni a rendszer egyes elemeit.

1.2. A megoldás

A jelen szakdolgozat egy olyan webes alkalmazás elkészítését dokumentálja, melyel a felhasználók képesek a természetben elhelyezett eszközök állapotát vizsgálni, azokat akár ki és bekapcsolni igény szerint. Az egyes rendszer eseményeket vizsgálva a szoftver statisztikákat készít, melyeket különböző diagrammokon ábrázolok. Ilyen statisztikák például, hogy időben melyik eszköz mikor észlelt madár hangot, vagy hogy hány hang üzenet érkezik az eszközöktől másodpercenként.

1.3. A szakdolgozat felépítése

A szakdolgozatom első részében, a 2. fejezetben, bemutatom a Birdnetes felépítését, az egyes komponensek közötti kapcsolatokat és a technológiát, amire épült. A 3. fejezetben ismertetem a jelenleg az iparban is használt mikroszolgáltatás működését vizualizáló alternatívákat, majd a saját megoldásom tervezetét, az arra vonatkozó elvárásokat. A 4. fejezetben az alkalmazásom által használt technológiákat mutatom be, ezzel előkészítve az 5. és 6. fejezetet, ahol ismertetem a szerver- és kliensalkalmazások felépítését. A 7. és 8. fejezet az alkalmazás teszteléséről és telepítéséről szól. Az utolsó fejezetben értékelem a munkám eredményét, levonom a tapasztalatokat és bemutatok néhány továbbfejlesztési lehetőséget.

2. fejezet

A Birdnetes részletes bemutatása

Ebben a fejezetben ismertetem a Birdnetes mikroszolgáltatás rendszerének architektúráját. Részletesen kifejtem az alkalmazásom szempontjából fontos komponensek feladatát és működését. Majd egy példával ábrázolom a rendszer hangfelismerő folyamatát.

2.1. Gyors elméleti összefoglaló

Ez a szakasz nem azt a célt szolgálja, hogy minnél részletesebb képet mutasson az itt leírt technológiákról. Ez csupán egy rövid összefoglaló a Birdnetes működésének megértése szempontjából elengedhetetlen technológiákról és elvekről, hogy valamilyen szinten tisztában legyünk a fejezetben elhangzó kifejezésekkel.

2.1.1. Cloud, felhő

A cloud lényegében annyit jelent, hogy a szervert, amin az alkalmazás fut, nem a fejlesztőnek kell üzemeltetnie, hanem valamilyen másik szervezet[1] által vannak karban tartva. Ez több okból is hasznos:

- Olcsóbb. Nem kell berendezéseket vásárolni, nincs üzemeltetési díj. Az egyetlen költség a bérlés, ami általában töredéke annak, amit akkor fizetnénk ha magunk csinálnánk az egészet.
- Gyorsabb fejlesztés. Az alkalmazás futtatására használt szervereket általában a fejlesztő nem látja, ezekkel nem kell foglalkoznia. Ha az alkalmazásnak hirtelen nagyobb erőforrás igénye lesz, a rendszer automatikusan skálázódik.
- Nagyobb megbízhatóság. Az ilyen szolgáltatást nyújtó szervezeteknek ez az egyik legnagyobb feladata. Az alkalmazás bárhol és bármikor elérhető.

2.1.1.1. Mikroszolgáltatások

A mikroszolgáltatások nem sok mindenben különböznek egy általános szolgáltatástól. Ugyan úgy valamilyen kéréseket kiszolgáló egységek, legyen az web kérések kiszolgálása

[1] Ilyenek például a Microsoft Azure, az Amazon Web Services vagy a Google Cloud.

HTTP-n keresztül vagy akár parancssori utasítások feldolgozása. Az egyetlen fő különbség az a szolgáltatások felelősségköre. A mikroszolgáltatások fejlesztésénél a fejlesztők elsősorban arra törekednek, hogy egy komponensnek minnél kevesebb feladata és függősége legyen, ezzel megnő a tesztelhetőség és könyebb a skálázhatóság.

2.1.1.2. Konténerek

A konténer technikailag semmivel sem több mint egy Linux-on futó processz amelyre különböző korlátozásokat szabtak. Ilyen korlátozások lehetnek például, hogy a konténer nem látja a teljes fájlrendszert, annak csak egy kijelölt részét, megadható a konténer által használható processzor és memória igény vagy akár korlátozható az is, hogy a konténer hogyan használhatja a hálózatot. Léteznek eszközök, például a Docker[2], mely lehetővé teszi a fejlesztők számára az ilyen konténerek könnyed létrehozását és futtatását.

2.1.1.3. Kubernetes

A Kubernetes[3] az ilyen komplex konténerizált mikroszolgáltatás rendszerek menedzselésének könnyítését szolgálja. Kihasználja és ötvözi az imént említett technológiák előnyeit, hogy egy robosztus rendszert alkosson. Használatával felgyorsulhat és automatizált lehet az egyes konténerek telepítése, futtatása, de talán a legfőbb előnye, hogy segítségével könnyedén megoldható a rendszert ért terhelési igények szerinti dinamikus skálázódás. Azok a mikroszolgáltatások, amikre a rendszernek épp nincs szüksége, nem futnak, nem igényelnek erőforrást a szerveren, így nem kell utánnuk fizetni sem. Ezzel ellentétben, ha valamely szolgáltatás után hirtelen megnő az igény, akkor az könnyedén duplikálható.

2.1.2. MQTT

Az MQTT (Message Queue Telemetry Transport) az egy kliens-szerver publish/subscribe üzenetküldő protokoll. Könnyű implementálni és alacsony a sávszélesség igénye, mellyel tökéletes jelöltje a Machine to Machine (M2M), illetve az Internet of Things (IoT) kommunikáció megvalósítására. Működéséhez szükség van egy szerverre, amelynek feladata a beérkező üzenetek továbbküldése témák alapján. Egyes kliensek fel tudnak iratkozni bizonyos témákra, míg más kliensek publikálnak és a szerver levezényli a két fél között a kommunikációt.

2.1.3. Open API

Az Open API egy nyilvános alkalmazás-programozási leíró, amely a fejlesztők számára hozzáférést biztosít egy másik alkalmazáshoz. Az API-k lírják és meghatározzák, hogy egy alkalmazás hogyan kommunikálhat egy másikkal, melyet használva a fejlesztők könnyedén képesek a kommunikációra képes kódot írni vagy generálni.

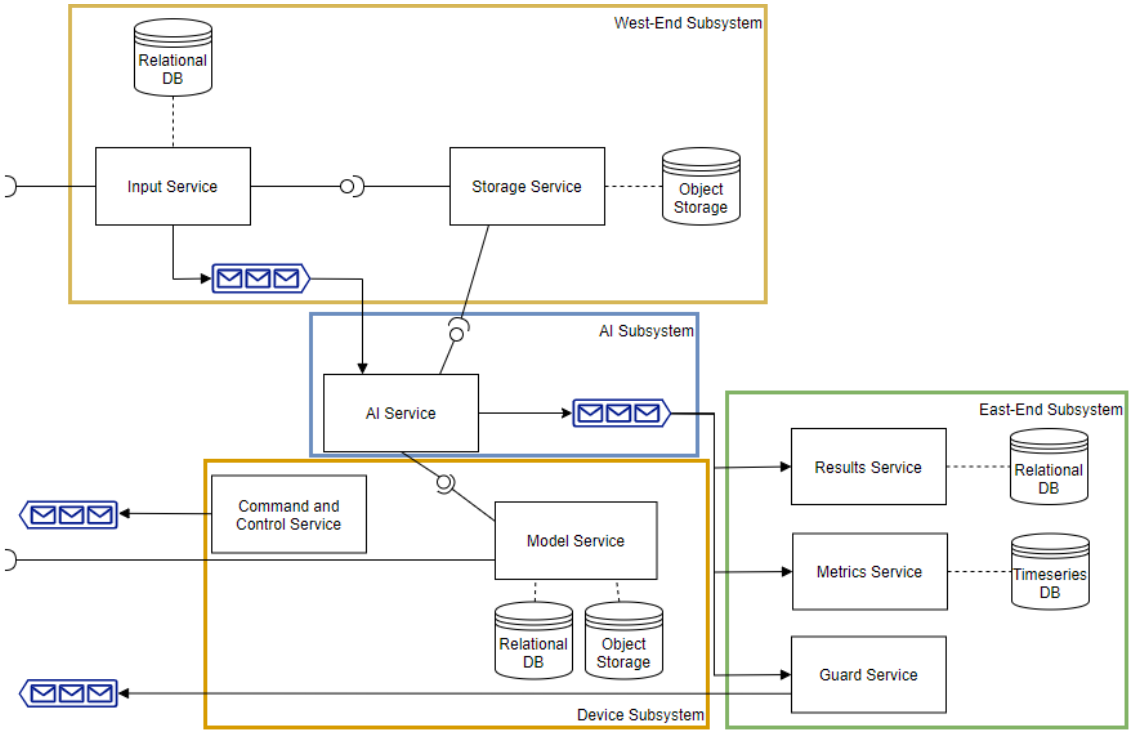
[2]https://www.docker.com/
[3]https://kubernetes.io/

2.2. Rendszerszintű architektúra

A Birdnetes fejlesztése során kifejezetten fontos szerepe volt a mikroszolgáltatás alapú rendszerek elvei követésének. A rendszer egy Kubernetes klaszterben van telepítve és több kisebb komponensből áll, melyek egymás között a HTTP és az MQTT protokollok segítségével kommunikálnak. A rendszer összes szolgáltatásának van egy Open API leírója, melyet használva hamar volt egy olyan kódbázisom, amely képes volt a rendszerrel való kommunikációra.

2.2.1. Főbb komponensek

A 2.1-es ábrán láthatóak a rendszer komponensei, melyek mind egy-egy mikroszolgáltatás. Az egymás mellett lévő kék levélborítékok az MQTT kommunikációt jelölik, amellyel például a természetben elhelyezett eszközök felé irányuló kommunikációja is történik. A következő alszakaszokban bemutatom az alkalmazásom szempontjából fontosabb komponenseket.



2.1. ábra. A Birdnetes rendszer architektúrája

2.2.1.1. Input Service

A kihelyezett IoT eszközök által felvett hangfájlok ezen a komponensen keresztül érkeznek be a rendszerbe. Itt történik a hanganyaghoz tartozó metaadatok lementése az Input Service saját adatbázisába. Ilyenek például a beküldő eszköz azonosítója, a beérkezés dátuma vagy a hangüzenet rendszerszintű egyedi azonosítója. Amint a szolgáltatás a berékezett

üzenettel kapcsolatban elvégezte az összes feladatát, publikál egy üzenetet az MQTT üzenetsorra a többi kliensnek feldolgozásra.

2.2.1.2. AI Service

Az AI Service példányai fogadják az Input Service-től érkező üzeneteket és elkezdik klasszifikálni az abban található hanganyagot. Meghatározzák, hogy a hanganyag mekkora valószínűséggel volt seregély hang vagy sem. Ennek eredményét a hangminta egyedi azonosítójával együtt publikálják egy másik üzenetsoron.

2.2.1.3. Guard Service

A Guard Service feliratkozik az AI Service által publikált üzenetek témájára és valamilyen valószínűségi kritérium alapján eldönti, hogy a hangminta tartalmaz-e seregély hangot. Ha igen, akkor az üzenetsoron küld egy riasztás parancsot a hanganyagot küldő eszköznek.

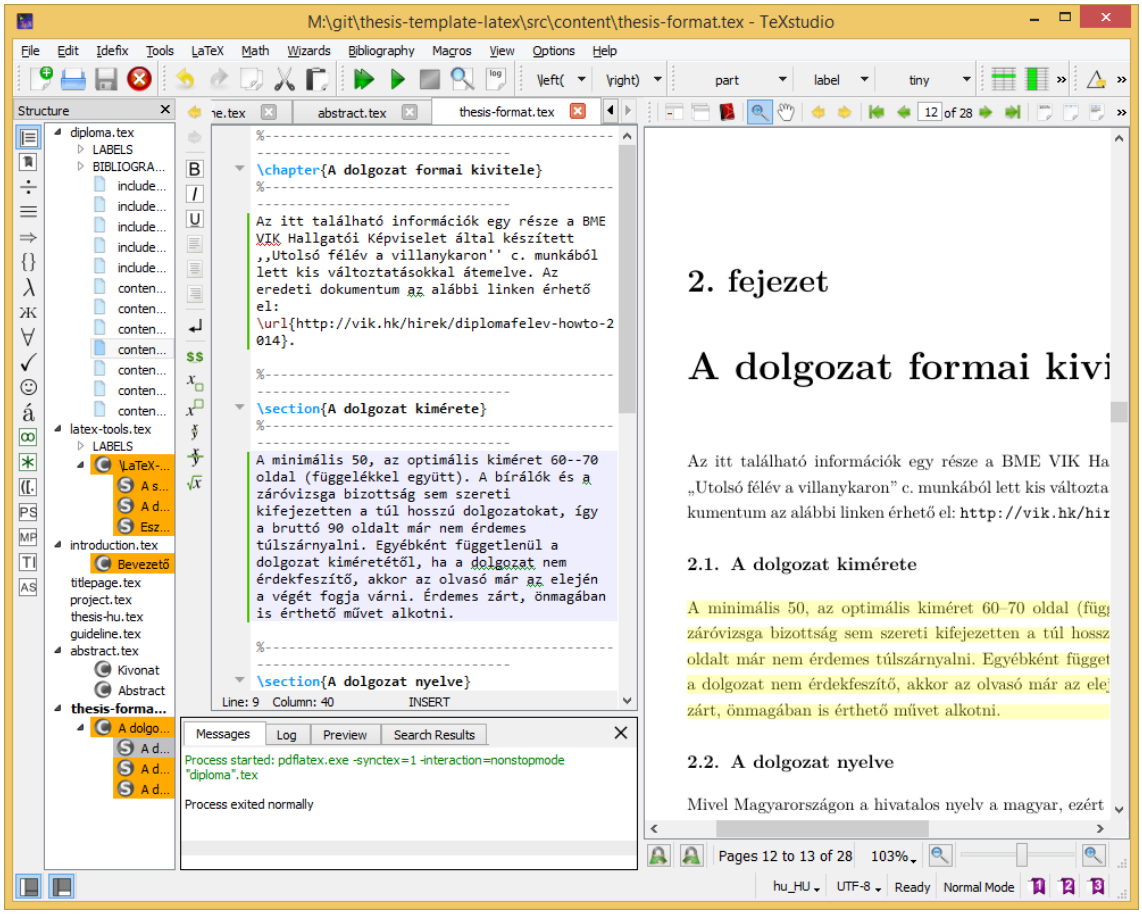
2.2.1.4. Command and Control Service

A Command and Control Service az előzőekkel ellentétben nem vesz részt a minták fogadásában, feldolgozásában vagy kezelésében. Felelősége az eszközök és azok szenzorai állapotának menedzselése és követése. Ezen keresztül lehet az egyes eszközöket ki- és bekapcsolni.

Irodalomjegyzék

Függelék

F.1. A TeXstudio felülete



F.1.1. ábra. A TeXstudio LaTeX-szerkesztő.

F.2. Válasz az „Élet, a világmindenség, meg minden” kérdésére

A Pitagorasz-tételből levezetve

$$c^2 = a^2 + b^2 = 42. \tag{F.2.1}$$

A Faraday-indukciós törvényből levezetve

$$\operatorname{rot} E = -\frac{dB}{dt} \qquad \longrightarrow \qquad U_i = \oint\limits_{\mathbf{L}} \mathbf{Edl} = -\frac{d}{dt} \int\limits_{A} \mathbf{Bda} = 42. \tag{F.2.2}$$