	Politechnika Śląska **Hurtownie Danych i Systemy Eksploracji Danych**			
Rok akademicki	**Rodzaj studiów*: SSI/NSI/NSM**	**Semestr :**	**Grupa**	**Sekcja**
2019/2020	**SSI**	**6**	**BDiS**	**7**
Data i godzina planowana ćwiczenia:	**Poniedziałek - 17:30**	**Przedmiot** :	**HDISED**	
Prowadzący :	**----------**			

Raport z projektu zasadniczego

Temat ćwiczenia:

Rozproszony system ekstrakcji danych z Internetu Rzeczy

Skład sekcji:	1. ---------- 2. ----------

1. Opis projektu

Projekt jaki postanowiliśmy zrealizować w ramach przedmiotu to dwie aplikacje mobilne, jedna na urządzenia iOS, a druga na urządzenia z systemem Android. Obie aplikacje wykorzystują czujniki w które wyposażone są obecne telefony, a następnie zebrane informacje gromadzone są w bazie danych. Jedna z aplikacji wykorzystuje krokomierz, a druga wbudowany w telefon czujnik natężenia światła. Zebrane informacje następnie są gromadzone w bazie danych.

2. Analiza, projektowanie

Przed przystąpieniem do pracy podzieliliśmy realizacja projektu na dwa problemy. Pierwszy z nich to implementacja dostępu do sensorów oraz odczytywanie interesujących nas wartości. W przypadku aplikacji iOS z pomocą przyszedł tutaj moduł CoreMotion, który umożliwia pobieranie danych z czujników takich jak akcelerometr, żyroskop czy też krokomierz. W przypadku aplikacji Android SensorManager do obsługi czujnika natężenia światła, FusedLocationClient do pobrania lokalizacji oraz OkHttp do obsługi połączeń sieciowych.

Kolejnym aspektem projektu było gromadzenie danych. Tutaj dla danych pobranych z krokomierza zdecydowaliśmy się na bardzo wygodne, praktyczne i nowoczesne rozwiązanie jakim jest platforma Firebase zaproponowana przez Google. Dane gromadzone z aplikacji Android posiadają nasze własne rozwiązanie oparte o:

- bazę MSSQL uruchomioną jako kontener Dockera na serwerze z Linuxem
- serwis REST-owy w technologii .NET CORE, których może być uruchomiony na Windowsie, Linuxie jak i MacOS, co pozwala utworzyć wiele endpointów dla aplikacji mobilnych

3. Specyfikacja zewnętrzna

Aplikacje składają się z jednego widoku na którym prezentujemy aktualną wartość pobraną z czujnika oraz tabele, która prezentuje poprzednie wartości.

Widok aplikacji Android

Widok aplikacji iOS

```
SELECT * FROM DataRecords
```

100 %

Results | Messages

	RecordId	Mac	Category	Data	CreationTime	SendTime	DbTime	Position
915	1914	02:00:00:00:00:00	Light	10	2020-09-19 18:20:05.017	2020-09-19 18:20:10.623	2020-09-19 16:20:12.763	null
916	1915	02:00:00:00:00:00	Light	13	2020-09-19 18:20:07.460	2020-09-19 18:20:10.630	2020-09-19 16:20:12.773	null
917	1916	02:00:00:00:00:00	Light	10	2020-09-19 18:20:09.020	2020-09-19 18:20:10.633	2020-09-19 16:20:12.773	null
918	1917	02:00:00:00:00:00	Light	12	2020-09-19 18:20:08.627	2020-09-19 18:20:10.633	2020-09-19 16:20:12.780	null
919	1918	02:00:00:00:00:00	Light	15	2020-09-19 18:20:09.220	2020-09-19 18:20:10.633	2020-09-19 16:20:12.780	null
920	1919	02:00:00:00:00:00	Light	32	2020-09-19 18:20:09.423	2020-09-19 18:20:10.637	2020-09-19 16:20:12.780	null
921	1920	02:00:00:00:00:00	Light	11	2020-09-19 18:20:04.627	2020-09-19 18:20:10.620	2020-09-19 16:20:12.787	null
922	1921	02:00:00:00:00:00	Light	21	2020-09-19 18:20:10.797	2020-09-19 18:20:10.803	2020-09-19 16:20:12.823	null
923	1922	02:00:00:00:00:00	Light	9	2020-09-19 18:16:06.577	2020-09-19 18:20:10.807	2020-09-19 16:20:12.830	null
924	1923	02:00:00:00:00:00	Light	13	2020-09-19 18:17:02.367	2020-09-19 18:20:10.817	2020-09-19 16:20:12.840	null
925	1924	02:00:00:00:00:00	Light	12	2020-09-19 18:16:06.970	2020-09-19 18:20:10.810	2020-09-19 16:20:12.840	null
926	1925	02:00:00:00:00:00	Light	13	2020-09-19 18:16:10.570	2020-09-19 18:20:10.813	2020-09-19 16:20:12.843	null
927	1926	02:00:00:00:00:00	Light	14	2020-09-19 18:16:03.747	2020-09-19 18:20:10.807	2020-09-19 16:20:12.843	null
928	1927	02:00:00:00:00:00	Light	13	2020-09-19 18:19:32.780	2020-09-19 18:20:10.817	2020-09-19 16:20:12.863	null
929	1928	02:00:00:00:00:00	Light	13	2020-09-19 18:17:17.640	2020-09-19 18:20:10.817	2020-09-19 16:20:12.863	null
930	1929	02:00:00:00:00:00	Light	12	2020-09-19 18:19:32.257	2020-09-19 18:20:10.817	2020-09-19 16:20:12.870	null
931	1930	02:00:00:00:00:00	Light	12	2020-09-19 18:19:33.047	2020-09-19 18:20:10.820	2020-09-19 16:20:12.873	null
932	1931	02:00:00:00:00:00	Light	13	2020-09-19 18:16:56.327	2020-09-19 18:20:10.813	2020-09-19 16:20:12.873	null
933	1932	02:00:00:00:00:00	Light	11	2020-09-19 18:20:04.627	2020-09-19 18:20:10.823	2020-09-19 16:20:12.877	null
934	1933	02:00:00:00:00:00	Light	10	2020-09-19 18:20:05.017	2020-09-19 18:20:10.827	2020-09-19 16:20:12.883	null
935	1934	02:00:00:00:00:00	Light	12	2020-09-19 18:20:08.627	2020-09-19 18:20:10.827	2020-09-19 16:20:12.887	null

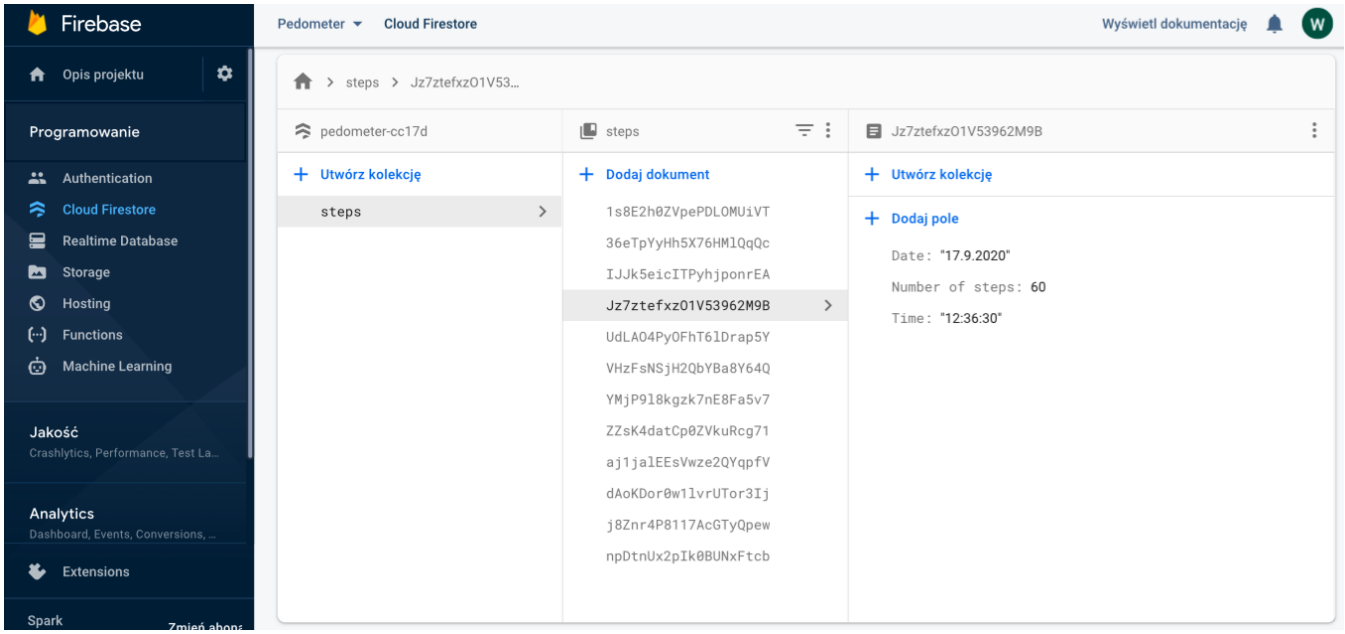
```
SELECT * FROM Devices
```

100 %

Results | Messages

	DeviceId	IpAddress	Category	CreationTime	ModTime	Port
1	1	192.168.55.105	Endpoint	2020-08-13 01:50:34.000	NULL	5001
2	4	192.168.1.113	Endpoint	2020-09-19 18:32:05.333	NULL	5001

Tabele bazy MSSQL dla aplikacji Android



Firebase

Wykorzystując platformę Firebase możemy podglądać nasze dane w przeglądarce internetowej.

4. Specyfikacja wewnętrzna

Aplikacja iOS

W przypadku programowania aplikacji mobilnych na urządzenia z systemem iOS zostaje udostępniony nam wyżej wspomniany moduł CoreMotion. Dzięki niemu możemy w bardzo wygodny sposób możemy odczytać wartość z krokomierza, który jest wbudowany w naszych telefonach. Poniżej prezentuje dwie najważniejsze metody : ***countSteps,*** która jest odpowiedzialna za pobranie informacji z sensora, funkcja ta pobiera dane za każdym razem gdy sensor wyczuje zmianę oraz funkcja ***sendPack,*** która jest odpowiedzialna za wysłanie „paczki" do bazy danych. Paczka to zaimplementowana struktura ***StepData***, która przechowuje ilość kroków oraz datę i godzinę odczytania wartości z czujnika.

```
func countSteps() {
    if CMPedometer.isStepCountingAvailable() {
        self.pedometer.startUpdates(from: Date()) { (data, error) in
            if error == nil {
                if let response = data {
                    DispatchQueue.main.async {
                        print("STEPS : \(response.numberOfSteps)")
                        self.numberOfSteps.text = response.numberOfSteps.stringValue
                        let testPack = self.createPack()
                        print(testPack)
                        self.sendPack(pack: testPack)
                    }
                }
            }
        }
    }
}
```

Funkcja ***countSteps*** odpowiedzialna za pobranie informacji z sensora

```
func sendPack(pack: StepData) {
    db.collection("steps").addDocument(data:
        ["Number of steps": pack.numberOfSteps,
         "Date": pack.date,
         "Time": pack.time
    ]) { (error) in
        if let e = error {
            print("Problem found : \(e)")
        } else {
            print("Data saved !!")
        }
    }
}
}
```

Funkcja ***sendPack*** odpowiedzialna za wysyłanie danych do bazy.

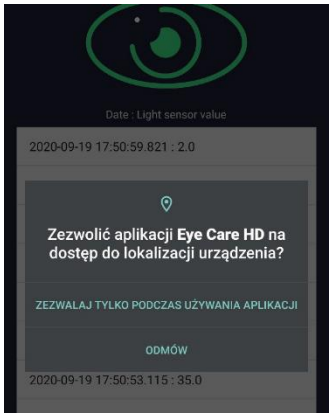
Aplikacja Android

Jest utworzona w bardzo przystępnym języku Kotlin (następca Javy w aplikacjach na platformę Android). Najważniejszymi metodami są odczyty z sensora natężenia światła:

```
override fun onSensorChanged(event: SensorEvent?) {
    var temp: LightPosLog? = null;
    try {
        temp = LightPosLog(getMac( context: this).toString(), this.getTimeStamp(), event!!.values[0], getLastKnownLocation(),  sent2DB: false)
        dataList.put(dataList.count()+1, temp)
    } catch (e: IOException) {
    }
    var counter = dataList.count()
    var listItems = arrayOfNulls<String>(counter)
    dataList.values.forEach { entry ->
        listItems[counter - 1] = "" + entry.time + " : " + entry.light;
        counter--
    }
    val adapter = ArrayAdapter( context: this, android.R.layout.simple_list_item_1, listItems)
    listView.adapter = adapter

    if(sending){
        var result = sendData(temp)
        temp?.sent2DB = result;
        if (temp != null) {
            dataList.replace(dataList.count(), temp)
        }
        sendAllData();
    }
}
```

Za pozyskiwanie danych o lokalizacji urządzenia odpowiada metoda:

```
fun getLastKnownLocation() : Location?{
    var ret : Location? = null
    if (ActivityCompat.checkSelfPermission(
            context: this,
            Manifest.permission.ACCESS_FINE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(
            context: this,
            Manifest.permission.ACCESS_COARSE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED
    ) {
        ActivityCompat.requestPermissions(
            activity: this,
            arrayOf(Manifest.permission.ACCESS_FINE_LOCATION, Manifest.permission.ACCESS_COARSE_LOCATION),
            requestCode: 1
        );
    }
    fusedLocationClient.lastLocation
        .addOnSuccessListener { location : Location? ->
            // Got last known location. In some rare situations this can be null.
            ret = location
        }
    return ret
}
```



Dodatkowo ważnym aspektem jest pozyskanie zgody na dostęp aplikacji do funkcji telefonu takich jak czujnik światła, lokalizacja oraz dostęp do internetu.
Dostępne endpointy pobierane są przy użyciu metody GET dla

```
fun getEndpoints(): String {
    var responseString: String? = null;
    val client = OkHttpClient.Builder()
        .followRedirects( followRedirects: true)
        .build()
        val request = Request.Builder()
            .url( url: "192.168.55.105:5001" + "/api/Devices")
            .build()
    client.newCall(request).execute().use { response -> return response.body!!.string() }
}
```

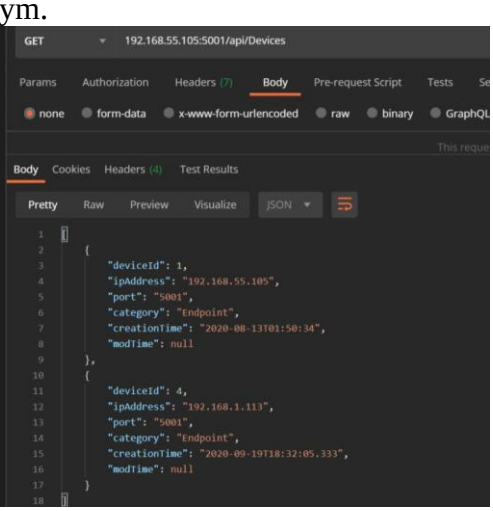
Wysyłanie danych odbywa się przy użyciu funkcji, w której dodatkowo jest mechanizm wykrywania wadliwego połączenia:

```
fun sendData(data: LightPosLog?):Boolean {
    var result = false;
    val currTime = LocalDateTime.now();
    val payload = "{\n" +
            "    \"Mac\": \"${data?.user.toString()}\",\n" +
            "    \"Category\": \"Light\",\n" +
            "    \"Data\": ${data?.light?.toInt()},\n" +
            "    \"Position\": \"${data?.location?.toString()}\",\n" +
            "    \"CreationTime\": \"${data?.time.toString().replace( oldValue: " ", newValue: "T")}\",\n" +
            "    \"SendTime\": \"$currTime\"\n" +
            "}"

    val okHttpClient = OkHttpClient.Builder()
        .followRedirects( followRedirects: true)
        .build()
    var endpointURL = endpointList.get(0); //pobranie adresu Endpointa
    val requestBody = payload.toRequestBody()
    val request = endpointURL?.let { it: String
        Request.Builder()
            .header( name: "Content-Type", value: "application/json")
            .method( method: "POST", requestBody)
            .url(it)
            .build()
    }
    if (request != null) {
        okHttpClient.newCall(request).enqueue(object : Callback {
            override fun onFailure(call: Call, e: IOException) {
                endpointList.remove( key: 0) //usuwanie wadliwego Endpointa
            }
            override fun onResponse(call: Call, response: Response) {
                result = true
            }
        })
    }
    return result
}
```

Serwis REST

Posiada model danych utworzony przy użyciu biblioteki Microsoft .NET CORE Entity Framework na podstawie wcześniej utworzonej bazy MSSQL na zdalnym serwerze. Do obsługi zapytań RESTowych został użyty ASP.NET, zostały utworzone dwa Controllery które posiadają komplet metod http (GET, POST, PUT, DELETE). Dzięki wykorzystaniu technologii .NET Core projekt może być skompliowany na wiele systemów operacyjnych czyniąc go bardzo wszechstronnym.

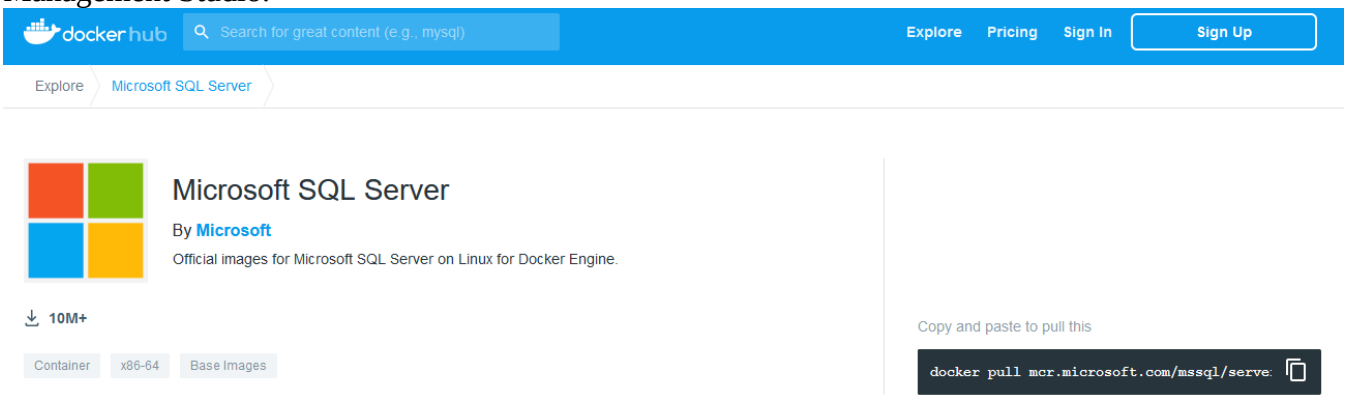
```
[Route("api/[controller]")]
[ApiController]
public class DevicesController : ControllerBase
{
    private readonly HDContext _context;

    public DevicesController(HDContext context)
    {
        _context = context;
    }

    // GET: api/Devices
    [HttpGet]
    public async Task<ActionResult<IEnumerable<Devices>>> GetDevices()
    {
        return await _context.Devices.ToListAsync();
    }
```



Przykładowy kod instrukcji GET oraz jego odpowiedź pobrana z bazy danych.

Baza MSSQL

Została utworzona na maszynie działającej pod kontrolą systemu Linux w kontenerze Dockera z obrazu udostępnionego developerom. Wybór na tą bazę padł dzięki nauce wynikającej z wykorzystywania w projekcie wstepnym przedmiotu HDISED co przyzwyczaiło nas do korzystania z Microsoft SQL Server Management Studio.



Skrypty tworzące bazę:

```
Create Table Devices(
DeviceId Int Identity(1,1) Primary Key,
IpAddress Varchar(16) Not Null,
Port Varchar(6),
Category Varchar(20),
CreationTime DateTime Default(GetDate()) Not Null,
ModTime DateTime
);

Create Table DataRecords(
RecordId Int Identity(1,1) Primary Key,
Mac Varchar(18) Not Null,
Category Varchar(10) Not Null,
Data Int Not Null,
Position Varchar(20),
CreationTime DateTime Default(GetDate()) Not Null,
SendTime DateTime Default(GetDate()) Not Null,
DbTime DateTime Default(GetDate()) Not Null
);
```

5. Testowanie

W celu przetestowania aplikacji zostały one zainstalowane na urządzeniach fizycznych. Programy działają dobrze i spełniają swoją rolę. Dane są wysyłane do bazy pomyślnie. Testowana również była sytuacja w której telefon tracił połączenie z bazą danych podczas działania aplikacji. Korzystając z platformy Firebase scenariusz ten jest już zabezpieczony i dane nie zostają utracone, przy ponownym połączeniu zaległe „paczki" zostają przesłane do bazy danych. W aplikacji Android po utraceniu połączenia z serwisem następuje próba połączenia z inną instancją serwisu pod innym adresem, przy pozytywnym połączeniu dane, które nie zostały wysłane zostają wysłane na bazę.

Wnioski

Mimo panującego zamieszania udało się sprawnie wykonać projekt. Cieszy nas fakt, że mogliśmy go realizować w wybranym przez nas języku programowania oraz tworzyć aplikacje w interesującej nas technologii. Aplikacje działają sprawnie i realizują podstawowe założenia projektu pozostawiając wiele miejsca to rozwoju projektu o analizę lub porównywanie danych. Dodatkowo po pozyskaniu publicznego adresu IP od dostawcy internetowego możliwe byłoby wystawienie endpointów „na świat" umożliwiając działanie poza siecią domową.

Dzięki realizacji tego projektu mogliśmy zgłębić naszą wiedzę na temat programowania urządzeń mobilnych oraz zmierzyć się z zagadnieniem ekstrakcji danych. Wybrane przez nas narzędzia okazały się wygodne w użyciu oraz dobrze spełniły początkowe założenia projektu.