Projekt

Statystyka w Informatyce 1

Wydział Elektrotechniki, Automatyki i Informatyki

Politechnika Świętokrzyska

Studia: **Stacjonarne II Stopnia**	Kierunek: **Informatyka**
Grupa: **1ID22B**	Skład zespołu: **Zakrzewski Mateusz** **Zachariasz Łukasz**

Temat projektu:

Opracowanie projektu i aplikacji do analizy statystycznej i wizualizacji danych dotyczących pomiarów warunków atmosferycznych.

1. WSTĘP

Głównym celem projektu było wykonanie oprogramowania, które wykorzystuje metody statystyczne do analizy danych. Dane te poddane analizie przez system następnie wizualizuje.

System został zrealizowany w architekturze webowej aplikacji klient-serwer, wykorzystano framework Spring oraz Angular 8.

Danymi bazowymi jest zbiór informacji o pogodzie z trzech stacji badawczych. Dane zawierają informacje m.in. o temperaturze powietrza, cieśnieniu atmosferycznym, poziomie opadów czy prędkości wiatru w danym momencie czasu. W kolejnym rozdziale dane te zostały szczegółowo opisane.

Rozdział trzeci poświęcony jest przybliżeniu wykorzystywanych technologii, opisane zostały podstawowe cechy każdej z nich i rola za jaką są w projekcie aplikacji odpowiedzialne. Opisano również instrukcję uruchomienia każdej z aplikacji.

Rozdział czwarty to opis implementacji aplikacji klienta i serwera, przedstawiono najważniejsze fragmenty kodu realizujące zadane operacje statystyczne pobierania danych. Przedstawiono strukturę projektu aplikacji klienta i serwera.

Kolejny, piąty rozdział to przedstawienie warstwy prezentacji aplikacji – klienta, w tym rozdziale znajduje się instrukcja obsługi aplikacji dla użytkownika, możliwości systemu analizy oraz przedstawienie wyników działania w postaci zrzutów ekranu.

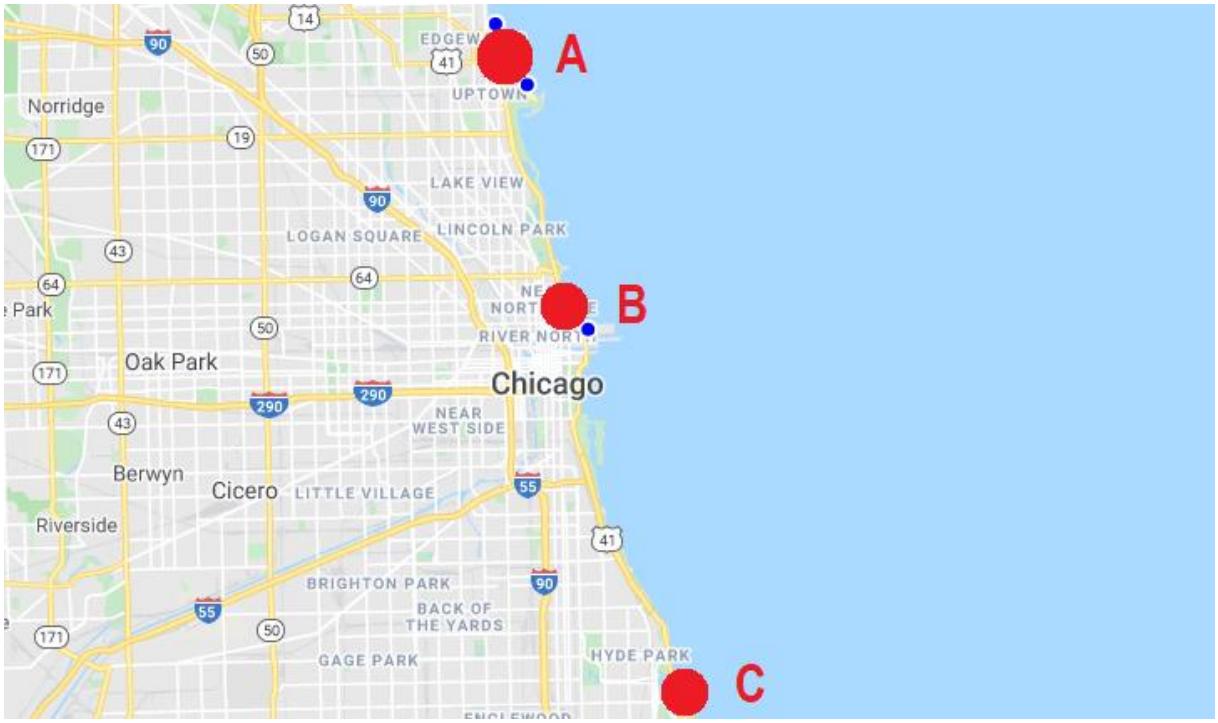
Rozdział szósty to podsumowanie i wnioski jakie uzyskano z tworzenia projektu aplikacji oraz otrzymanych rezultatów.

W ostatnim, siódmym rozdziale umieszczona została bibliografia, która zawiera najważniejsze materiały wykorzystane przy budowie projektu.

2. ZBIÓR DANYCH

Na potrzeby systemu analizy wybrano zbiór danych zawierający historyczne dane pogodowe rejestrowane przez trzy stacje badawcze nad brzegiem jeziora w Chicago w stanie Michigan. Rozmieszczenie zaprezentowano na poniższym fragmencie mapy.

A. Foster Weather Station,
B. Oak Street Weather Station,
C. 63rd Street Weather Station.



Dane dostępne są w formacie .CSV na stronie *Chicago Data Portal*. W dniu pobierania zestawu danych na potrzeby projektu plik zawiera 97 tysięcy rekordów oraz 18 kolumn atrybutów – do analizy zostały przekazane wybrane, najistotniejsze z atrybutów. Czujniki rejestrują na ogół pomiar co godzinę. Dane gromadzone są od 22 Maja 2015 roku i stale aktualizowane [1]. Udostępnione zostało również zewnętrzne API dla danych, przez które można pobierać odczyty na bieżąco.

Zestaw danych wykorzystany na potrzeby projektu został dołączony w postaci pliku .CSV pod nazwą *Beach_Weather_Stations_-_Automated_Sensors* i znajduje się w głównym katalogu projektu.

[1] 14 stycznia 2020 roku – Data tworzenia tego dokumentu.

Atrybuty zbioru danych

Poniższa tabela zawiera spis atrybutów. Wyszczególniono i opisano te atrybuty, które zostały szczególnie poddane analizie przez aplikację.

Nazwa atrybutu	Typ danych	Opis
Station Name	Tekst	Nazwa stacji badawczej.
Measurement Timestamp	Data	Data wykonania pomiaru.
Air Temperature	Liczba	Temperatura powietrza w stopniach Celsjusza.
Wet Bulb Temperature	Liczba	Temperatura mokrego termometru w stopniach Celsjusza.
Humidity	Liczba	Procentowa wilgotność względna.
Rain Intensity	Liczba	Intensywność deszczu w mm na godzinę.
Interval Rain	Liczba	Deszcz od ostatniego pomiaru co godzinę, w mm.
Total Rain	Liczba	Całkowity deszcz od północy w mm.
Precipitation Type	Liczba	0 = Bez opadów 60 = Opady płynne, np. Deszcz - Lód, grad i deszcz ze śniegiem są przekazywane jako deszcz (60). 70 = Opady stałe, np. Śnieg 40 = opady nieokreślone.
Wind Direction	Liczba	Kierunek wiatru w stopniach.
Wind Speed	Liczba	Prędkość wiatru w metrach na sekundę w momencie zgłoszenia zapisu.
Maximum Wind Speed	Liczba	Maksymalna prędkość wiatru w dwuminutowym okresie bezpośrednio poprzedzającym czas zgłoszenia rekordu.
Barometric Pressure	Liczba	Ciśnienie barometryczne w hPa.
Solar Radiation	Liczba	Promieniowanie słoneczne w watach na metr kwadratowy.
Heading	Liczba	Aktualny kurs jednostki pomiaru wiatru. Idealną wartością do uzyskania najdokładniejszych pomiarów jest prawdziwa północ (0 stopni), a jednostka jest dostosowywana ręcznie, w razie potrzeby, aby utrzymać ją blisko tej pozycji
Battery Life	Liczba	Napięcie baterii, wskaźnik pozostałej żywotności baterii wykorzystywany przez dzielnicę Chicago Park, aby wiedzieć, kiedy należy wymienić baterie.
Measurement Timestamp Label	Tekst	Ostatnia aktualizacja wartości w formacie tekstowym.
Measurement ID	Tekst	Unikalny identyfikator rekordu złożony z nazwy stacji i znacznika czasu pomiaru.

3. WYKORZYSTANE TECHNOLOGIE I INSTRUKCJE URUCHOMIENIA APLIKACJI SERWERA I KLIENTA

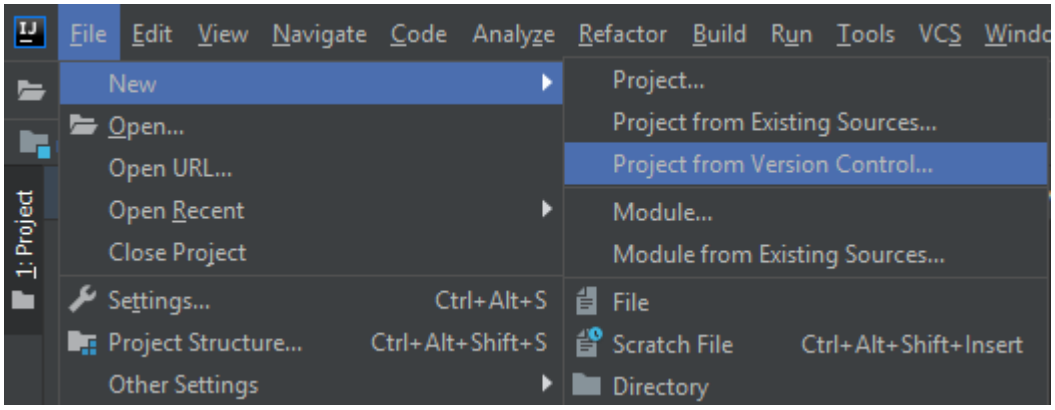
W tym rozdziale przedstawiono technologie, na których zbudowano projekt aplikacji serwera oraz klienta, przedstawiono również opis uruchomienia każdej z aplikacji.

Aplikacja Serwera

W projekcie serwera wykorzystano framework Spring oraz zależności (*dependency)* odpowiedzialne m.in za współpracę z bazą danych czy parsowanie danych z pliku. Serwer został napisany w języku Java 11. Do przechowywania danych wykorzystano bazę NoSQL *MongoDB*. Parsowanie danych z pliku .CSV zrealizowano z wykorzystaniem biblioteki *FasterXML Jackson Dataformat.*

Za kontrolowanie zewnętrznie dostępnych zależności odpowiada *Maven*, które automatyzuje budowę aplikacji na platformę Java. Środowisko programistyczne, w którym zrealizowano serwera to IntelliJ IDEA 2019.3. Do uruchomienia projektu wymagane jest również JDK w wersji 11.

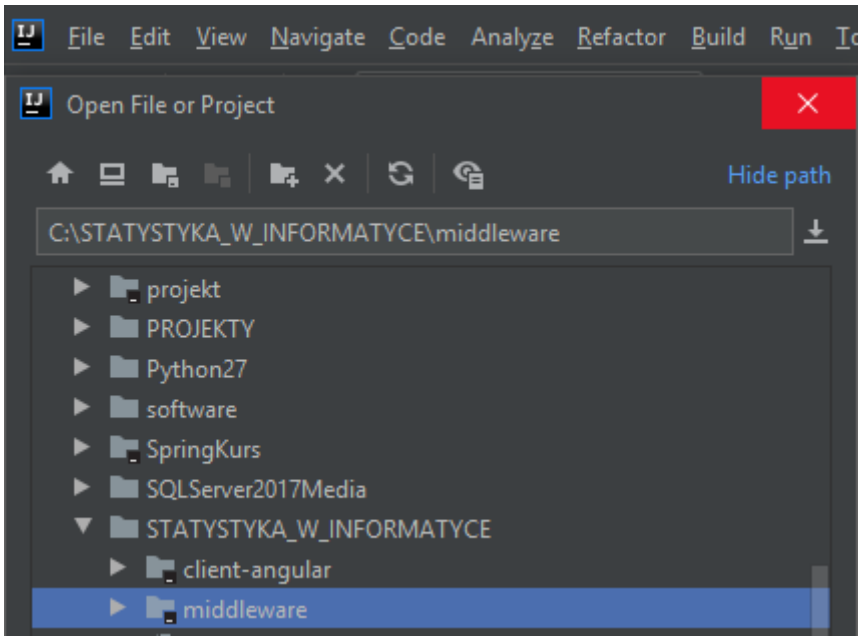
Aby uruchomić aplikację serwera można skorzystać w dwóch sposobów. Pierwszym jest wybranie w środowisku IDE opcji *Get from Version Control*, z menu *File*, co przedstawia poniższy zrzut ekranu.



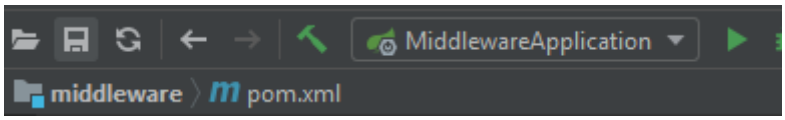
Następnie należy podać URL do repozytorium zdalnego GitHub, pod którym umieszczono projekt aplikacji:

https://github.com/LukaszZachariasz/SWI.git

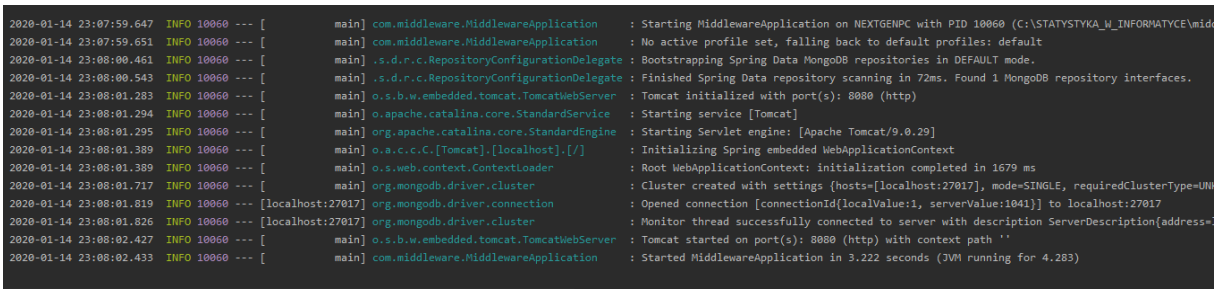
Drugim sposobem jest pobranie z repozytorium paczki .zip, rozpakowanie jej, a następnie wybraniem lokalizacji poprzez opcję: *File -> Open,* następnie należy wybrać lokalizację folderu z rozpakowanym projektem.



Gdy już projekt zostanie przez środowisko IDE zainicjowany, a Maven automatycznie pobierze wszystkie zdefiniowane w pliku *pom.xml* biblioteki zależności, można uruchomić aplikację wybierając komendę Run oznaczoną ikoną zielonego przycisku *Play*.



Potwierdzeniem uruchomienia jest wypisanie w oknie terminala następujących informacji, spośród których najważniejszą jest że aplikacja wystartowała.



Domyślnie aplikacja serwera działa na porcie 8080, na potrzeby prezentacji została uruchomiona lokalnie więc jej adres to: *http://localhost:8080/*

Aplikacja Klienta

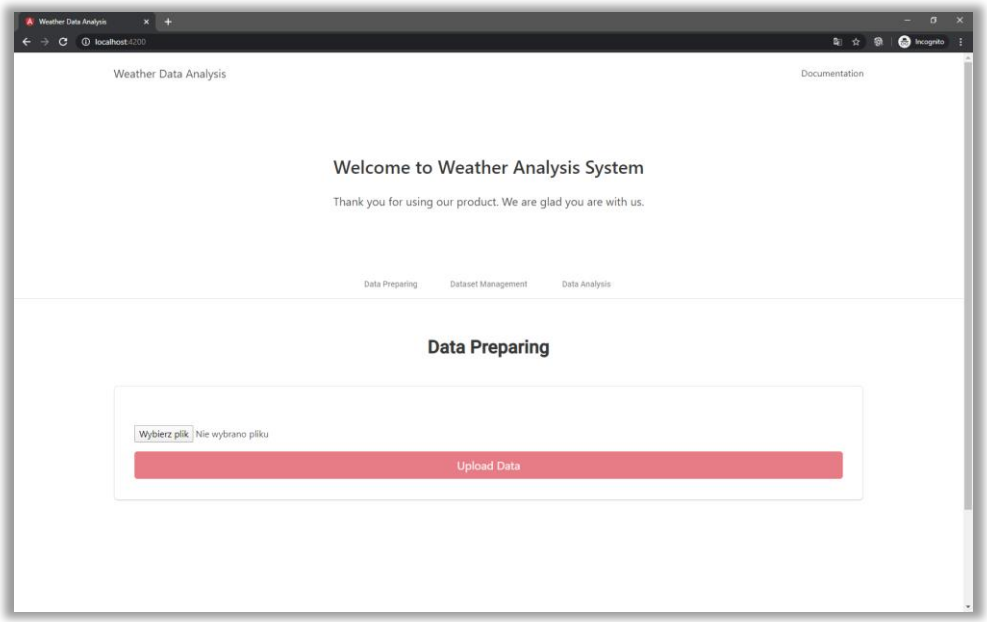
Aplikacja klienta jest z punktu widzenia środowiska odrębnym projektem, która może być uruchomiona na innym komputerze i jedynie komunikować się za pomocą sieci z usługami wystawianymi zdalnie przez aplikację serwera.

Projekt klienta powstał w języku *TypeScript* jako wiodący dla frameworka *Angular 8*. Dodatkowo dla zapewnienia estetycznego wyglądu komponentów graficznych wykorzystano biblioteki *Angular Material*, *Bootstrap, BULMA* czy *ChartJS* do prezentacji danych.

Również aplikacja klienta została wykonana przy wykorzystaniu środowiska IDE IntelliJ IDEA. Do prawidłowego działania aplikacji opartej o framework Angular 8 wymagana jest instalacja środowiska NodeJS.

Uruchomienie aplikacji wygląda analogicznie tak jak dla serwera. Po wybraniu projektu i wczytaniu do środowiska, zainicjuje ono pobieranie bibliotek lub wyświetli zapytanie czy może pobrać wymagane zależności. Gdy wszystko zostanie pobrane a aplikacja zostanie zidentyfikowana jako projekt *Angular CLI*, można wybrać zieloną strzałkę *Run* i projekt powinien uruchomić się domyślnie na adresie: *http://localhost:4200/*

Po uruchomieniu przeglądarki na tym adresie, zostanie uruchomiona aplikacja klienta, której ekran prezentuje poniższy zrzut.

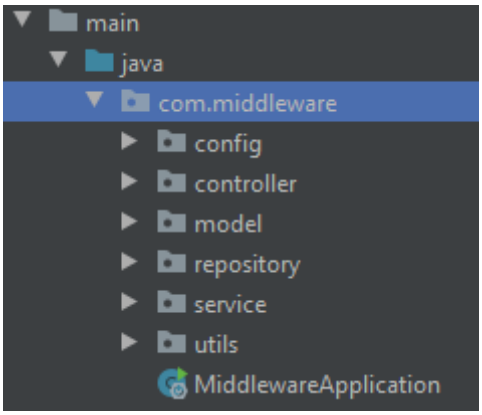


4. STRUKTURA I IMPLEMENTACJA APLIKACJI SERWERA I KLIENTA

W tym rozdziale przedstawiono ogólną strukturę budowy projektu zarówno aplikacji serwera jak i klienta, przedstawiono najważniejsze fragmenty kodu oraz opis działania.

Aplikacja Serwera – Struktura

Poniższy zrzut ekranu prezentuje strukturę oraz podział kodu źródłowego na odpowiedzialne za konkretne zadania moduły. Struktura składa się z 6 głównych modułów.

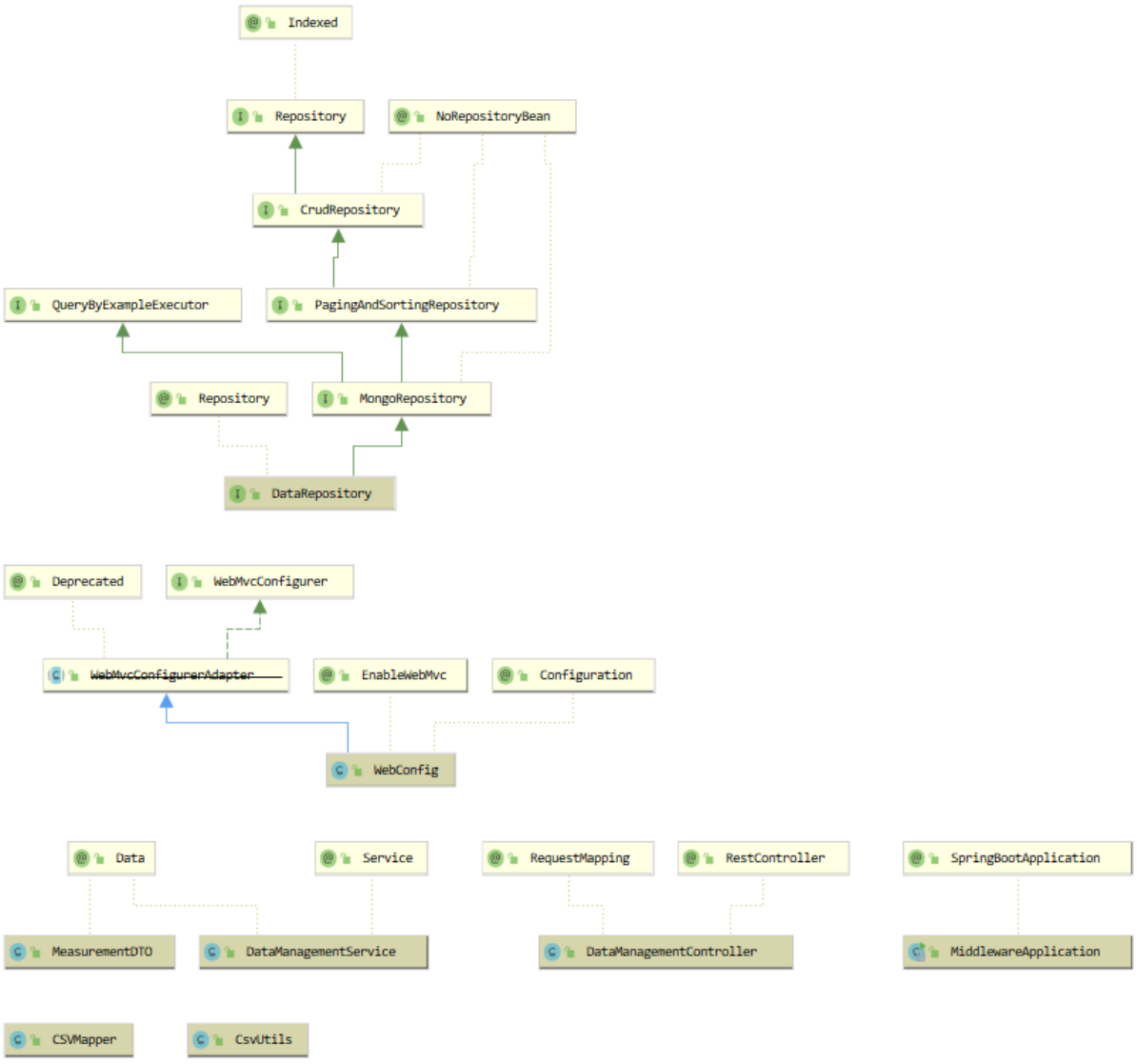


W pakiecie *config* umieszczona jest kod odpowiedzialny za filtrowanie metod protokołu HTTP wykorzystywanych przez klientów na serwerze takich jak: **POST**, **DELETE**, **GET** CZY **PUT.** Skonfigurowane zostały również między innymi mechanizmy CORS Policy.

W pakiecie *controller* umieszczony został kod odpowiedzialny za udostępnienie pod odpowiednimi *endpointami* aplikacji REST zaimplementowanych mechanizmów do zwracania wyników analizy danych bądź obsługujących ładowanie ich do bazy czy sterowanie nimi np.: usunięcia. Główny Controller udostępnia następujące punkty końcowe:

- ***/getAllData*** – zwraca odpowiedź w postaci wszystkich rekordów załadowanych do bazy.
- ***/getAvgValuesInDataRange*** – zwraca średnie wartości dla zdefiniowanych atrybutów w okresie czasu zdefiniowanym argumentami: daty początkowej (*startDate)* i końcowej (*endDate*) z wybranego urządzenia pomiarowego przekazywanego jako trzeci argument (*pickedDevice*).
- ***/getMinValuesInDataRange*** – zwraca minimalne wartości dla zdefiniowanych atrybutów w okresie czasu zdefiniowanym argumentami: daty początkowej (*startDate)* i końcowej (*endDate*) z wybranego urządzenia pomiarowego przekazywanego jako trzeci argument (*pickedDevice*).
- ***/getMaxValuesInDataRange*** – zwraca minimalne wartości dla zdefiniowanych atrybutów w okresie czasu zdefiniowanym argumentami: daty początkowej (*startDate)* i końcowej (*endDate*) z wybranego urządzenia pomiarowego przekazywanego jako trzeci argument (*pickedDevice*).
- ***/uploadFile*** – punkt wykorzystuje Obiekt *DataManagementService* do zapisu danych do bazy z pliku przekazanego w argumencie w postaci *multipart/form-data*.
- ***/getAllValuesInDataRange*** – zwraca wszystkie wartości dla zdefiniowanych atrybutów w okresie czasu zdefiniowanym argumentami: daty początkowej (*startDate)* i końcowej (*endDate*) z wybranego urządzenia pomiarowego przekazywanego jako trzeci argument (*pickedDevice*).
- ***/getValuesByPage*** – punkt końcowy wspomagający ładowanie danych do tabeli aplikacji klienta porcjami, w argumencie przekazywana jest wartość strony tabeli oraz rozmiar w postaci ilości rekordów.
- ***/getDevices*** – zwraca listę unikalnych wartości z kolumny *Station Name* co czyni aplikację responsywną w momencie gdyby do zbioru danych dodano kolejne urządzenie pomiarowe.
- ***/removeAllData*** – punkt pozwalający usunąć wszystkie załadowane do bazy rekordy pomiarów.

Poniższy schemat prezentuje strukturę związków pomiędzy klasami w projekcie wraz z zależnościami.



Aplikacja Serwera – Implementacja

Poniżej przedstawiono wybrane fragmenty kodu serwera odpowiedzialne za operowanie na danych pobieranych z bazy MongoDB.

Poniższy kod w tabeli odpowiedzialny jest za pobieranie z bazy danych średnich wartości wyszczególnionych atrybutów w zadanym zakresie czasu. W metodzie przekazywane są argumenty opisujące czas początkowy, czas końcowy zakresu oraz urządzenie, z którego pobierane są dane.

Przy wykorzystaniu mechanizmów klasy *Criteria* można zbudować zapytanie wyszukujące i grupujące po odpowiedniej wartości daty wartości jej wyższe lub/oraz niższe, te z kolei są agregowane do jednego rekordu, którego każdy atrybut to średnia z zadanej kolumny całego zbioru w zadanym zakresie.

```
@RequestMapping(value = "/getAvgValuesInDataRange", method = RequestMethod.GET)
public ResponseEntity getAverageValuesInRange(@RequestParam("startDate") String
startDate, @RequestParam("endDate") String endDate, RequestParam("pickedDevice")
String pickedDevice) throws ParseException {
    LOG.info("getAvgValuesInDataRange");
    SimpleDateFormat format = new SimpleDateFormat("dd/MM/yy");
    Date d1 = format.parse(startDate);
    Date d2 = format.parse(endDate);
    LocalDateTime localDateTime1 = d1.toInstant()
            .atZone(ZoneId.systemDefault())
            .toLocalDateTime();
    LocalDateTime localDateTime2 = d2.toInstant()
            .atZone(ZoneId.systemDefault())
            .toLocalDateTime();
    TypedAggregation<MeasurementDTO> agg = newAggregation(
            MeasurementDTO.class,
       match(Criteria.where("measurementDate").gte(localDateTime1)
      .lt(localDateTime2).and("stationName").is(pickedDevice)),
            group("stationName")
                    .avg("airTemperature").as("avgAirTemperature")
                    .avg("humidity").as("avgHumidity")
                    .avg("rainIntensity").as("avgRainIntensity")
                    .avg("totalRain").as("avgTotalRain")
                    .avg("windSpeed").as("avgWindSpeed")
                    .avg("barometricPressure").as("avgBarometricPressure")
                    .avg("solarRadiation").as("avgSolarRadiation")
    );

    AggregationResults<DBObject> result
    = mongoTemplate.aggregate(agg, DBObject.class);
    List<DBObject> resultList = result.getMappedResults();

    return ResponseEntity.ok(resultList);
}
```

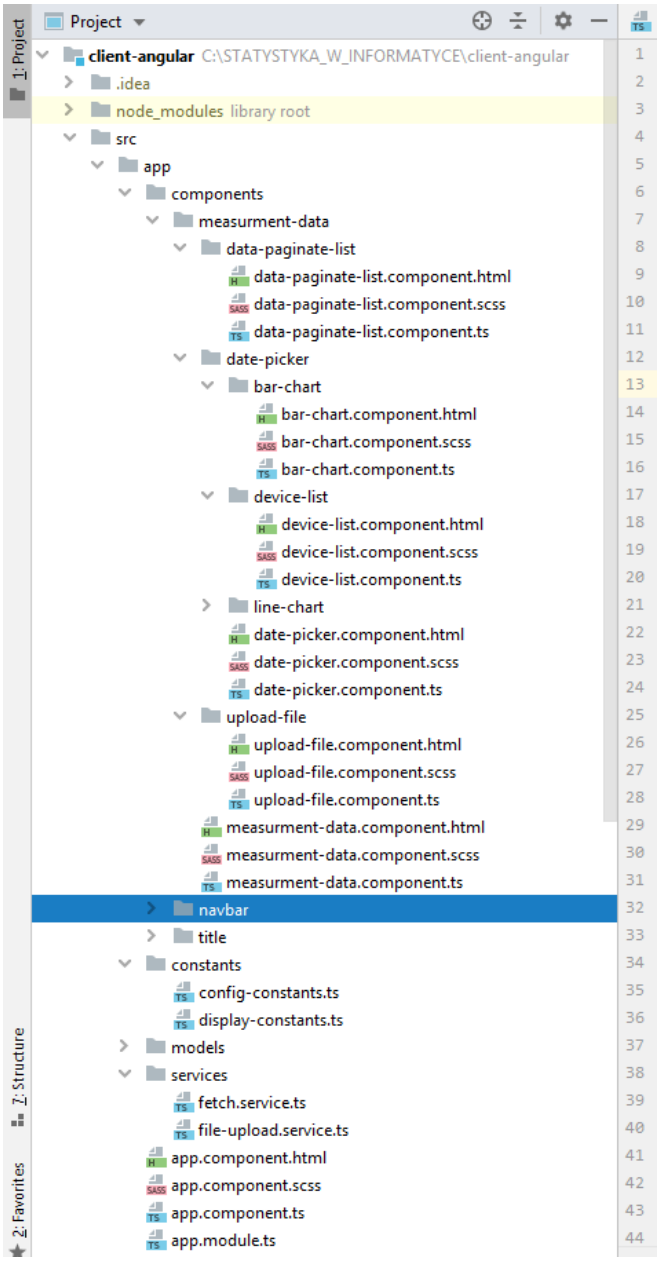
```
@RequestMapping(value = "/getMinValuesInDataRange", method = RequestMethod.GET)
public ResponseEntity getMinInDateRange(@RequestParam("startDate") String
startDate, @RequestParam("endDate") String endDate, RequestParam("pickedDevice")
String pickedDevice) throws ParseException {
    LOG.info("getMinValuesInDataRange");
    SimpleDateFormat format = new SimpleDateFormat("dd/MM/yy");
    Date d1 = format.parse(startDate);
    Date d2 = format.parse(endDate);
    LocalDateTime localDateTime1 = d1.toInstant()
            .atZone(ZoneId.systemDefault())
            .toLocalDateTime();
    LocalDateTime localDateTime2 = d2.toInstant()
            .atZone(ZoneId.systemDefault())
            .toLocalDateTime();
    TypedAggregation<MeasurementDTO> agg = newAggregation(
            MeasurementDTO.class,

     match(Criteria.where("measurementDate")
    .gte(localDateTime1).lt(localDateTime2)
    .and("stationName").is(pickedDevice)),
            group("stationName")
                    .min("airTemperature").as("minAirTemperature")
                    .min("humidity").as("minHumidity")
                    .min("rainIntensity").as("minRainIntensity")
                    .min("totalRain").as("minTotalRain")
                    .min("windSpeed").as("minWindSpeed")
                    .min("barometricPressure").as("minBarometricPressure")
                    .min("solarRadiation").as("minSolarRadiation")
    );
    AggregationResults<DBObject> result
    = mongoTemplate.aggregate(agg, DBObject.class);
    List<DBObject> resultList = result.getMappedResults();

    return ResponseEntity.ok(resultList);
}
```

Powyższy kod w tabeli odpowiedzialny jest analogicznie za pobieranie z bazy danych tym razem minimalnych wartości wyszczególnionych atrybutów w zadanym zakresie czasu. W metodzie przekazywane są argumenty opisujące czas początkowy, czas końcowy zakresu oraz urządzenie, z którego pobierane są dane.

Za parsowanie danych podczas wczytywania z pliku do odpowiednich pól obiektu DTO Modelu odpowiedzialny jest poniższy kod.

```
public class CsvUtils {
    private static final CsvMapper mapper = new CsvMapper();
    public static <T> List<T> read(Class<T> clazz, InputStream stream)
    throws IOException {
        CsvSchema schema
        = mapper.schemaFor(clazz).withHeader().withColumnReordering(true);
        ObjectReader reader = mapper.readerFor(clazz).with(schema);
        return reader.<T>readValues(stream).readAll();
    }
}
```

Aplikacja Klienta – Struktura



Struktura przedstawiona obok prezentuje najważniejsze komponenty aplikacji klienta dla systemu.

Można wyszczególnić najważniejsze katalogi zawierające komponenty odpowiedzialne za odrębne role:

- ***Components*** – zawierają definicję komponentów graficznych aplikacji oraz klasy dla każdego z nich określające ich zachowanie.
- ***Constants*** – w tym folderze zdefiniowano klasy przechowujące stałe, takie jak statyczne teksty, podpisy, kolory elementów, czy globalne opcje dla komponentów.
- ***Models*** – zawiera klasy obiektów DTO.
- ***Services*** – klasy odpowiedzialne za komunikację z serwerem.

Aplikacja Klienta – Implementacja

W ramach aplikacji klienta zrealizowano również normalizację danych uzyskanych z serwera, co prezentuje poniższy kod w języku TypeScript.

```
normalizeDataSetValues(array: Array<number>) {
  let maxVal = 0;
  let minVal = 0;
  array.forEach(element => {
    if (maxVal < element) {
      maxVal = element;
    }
    if (minVal > element) {
      minVal = element;
    }
  });
  const normalizedArray: Array<number> = [];
  array.forEach(element => {
    normalizedArray.push(this.normalize(element, maxVal, minVal));
  });
  return normalizedArray;
}

normalize(val, max, min) {
  return (val - min) / (max - min);
}
```

Poniższe fragmenty kodu prezentują Service odpowiedzialny za pozyskiwanie wyników z serwera i usuwanie danych.

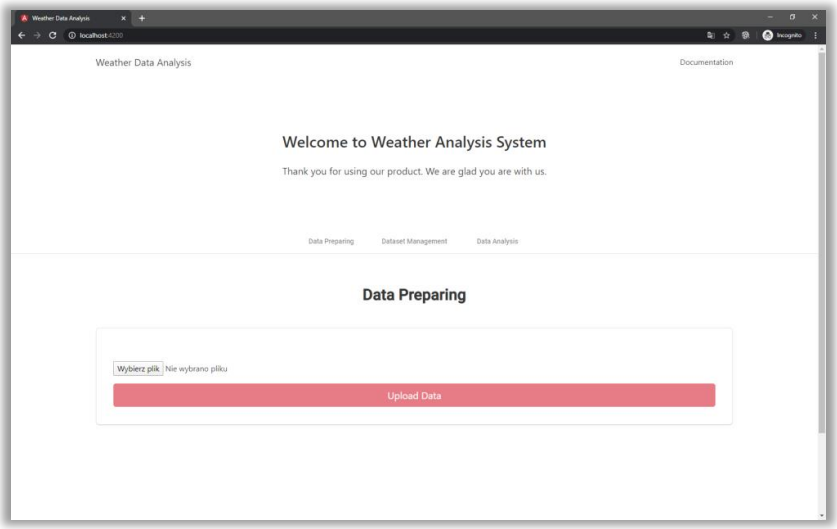
```
@Injectable()
export class FetchService {

  constructor(private http: HttpClient) {
  }

  fetchDevices(): Observable<object> {
    const headers = new HttpHeaders();
    headers.append('content', 'application/json');

    return this.http.get('http://localhost:8080/getDevices', {headers});
  }

  fetchAllValuesInDateRange(pickedStartDate: string, pickedEndDate:
  string, pickedDevice: string): Observable<object> {
    const headers = new HttpHeaders();
    headers.append('content', 'application/json');
    const params = new HttpParams().set('startDate', pickedStartDate)
    .set('endDate', pickedEndDate).set('pickedDevice', pickedDevice);
    console.log(pickedStartDate + '  ' + pickedEndDate + '  ' + pickedDevice);
    return this.http.get('http://localhost:8080/getAllValuesInDateRange',
    {headers, params});
  }
```

```
  fetchAllPagedValues(pageNumber: number, pageSize: number): Observable<object> {
    const headers = new HttpHeaders();
    headers.append('content', 'application/json');
    const params = new HttpParams().set('pageNumber',
    pageNumber.toString()).set('pageSize', pageSize.toString());
    return this.http.get('http://localhost:8080/getValuesByPage',
    {headers, params});
  }

  fetchAvgValuesInDateRange(pickedStartDate: string, pickedEndDate:
  string, pickedDevice: string): Observable<object> {
    const headers = new HttpHeaders();
    headers.append('content', 'application/json');
    const params = new HttpParams().set('startDate',
    pickedStartDate).set('endDate', pickedEndDate).set('pickedDevice',
    pickedDevice);
    return this.http.get('http://localhost:8080/getAvgValuesInDataRange',
    {headers, params});
  }

  fetchMinValuesInDateRange(pickedStartDate: string, pickedEndDate:
    string, pickedDevice: string): Observable<object> {
    const headers = new HttpHeaders();
    headers.append('content', 'application/json');
    const params = new HttpParams().set('startDate',
    pickedStartDate).set('endDate', pickedEndDate)
    .set('pickedDevice', pickedDevice);
    return this.http.get('http://localhost:8080/getMinValuesInDataRange',
    {headers, params});
  }

  fetchMaxValuesInDateRange(pickedStartDate: string, pickedEndDate:
    string, pickedDevice: string): Observable<object> {
    const headers = new HttpHeaders();
    headers.append('content', 'application/json');
    const params = new HttpParams().set('startDate',
    pickedStartDate).set('endDate', pickedEndDate).set('pickedDevice',
    pickedDevice);
    return
    this.http.get('http://localhost:8080/getMaxInDateRange', {headers, params});
  }

  removeAllData(): Observable<object> {
    const headers = new HttpHeaders();
    headers.append('content', 'application/json');
    return
    this.http.post('http://localhost:8080/removeAllData',
    {headers, responseType: 'text'});
  }

}
```

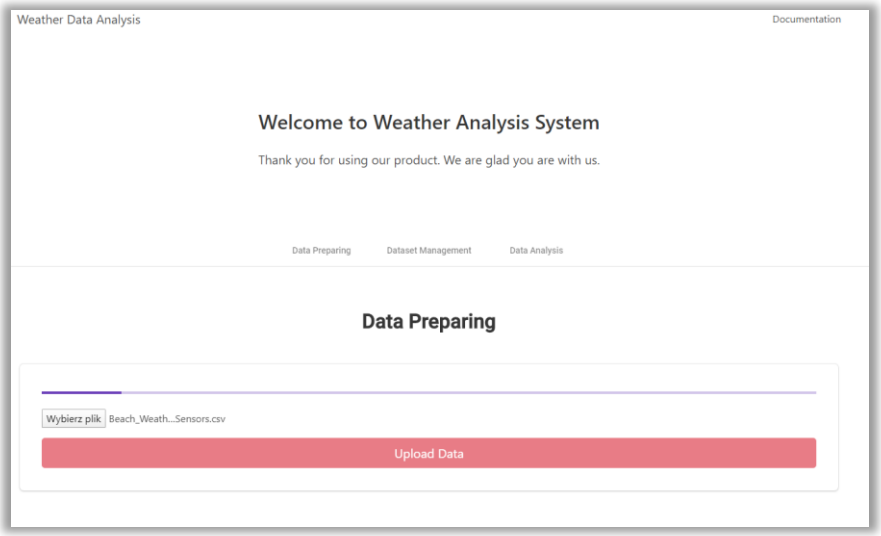
5. PREZENTACJA APLIKACJI

Aplikacja klienta przeznaczona dla użytkownika składa się ze strony zarządzania danymi (*Dataset Management*), ładowania ich (*Data Preparing*) oraz wykonywania operacji analizy (Data Analysis), kolejno nawigowane przez trzy zakładki.

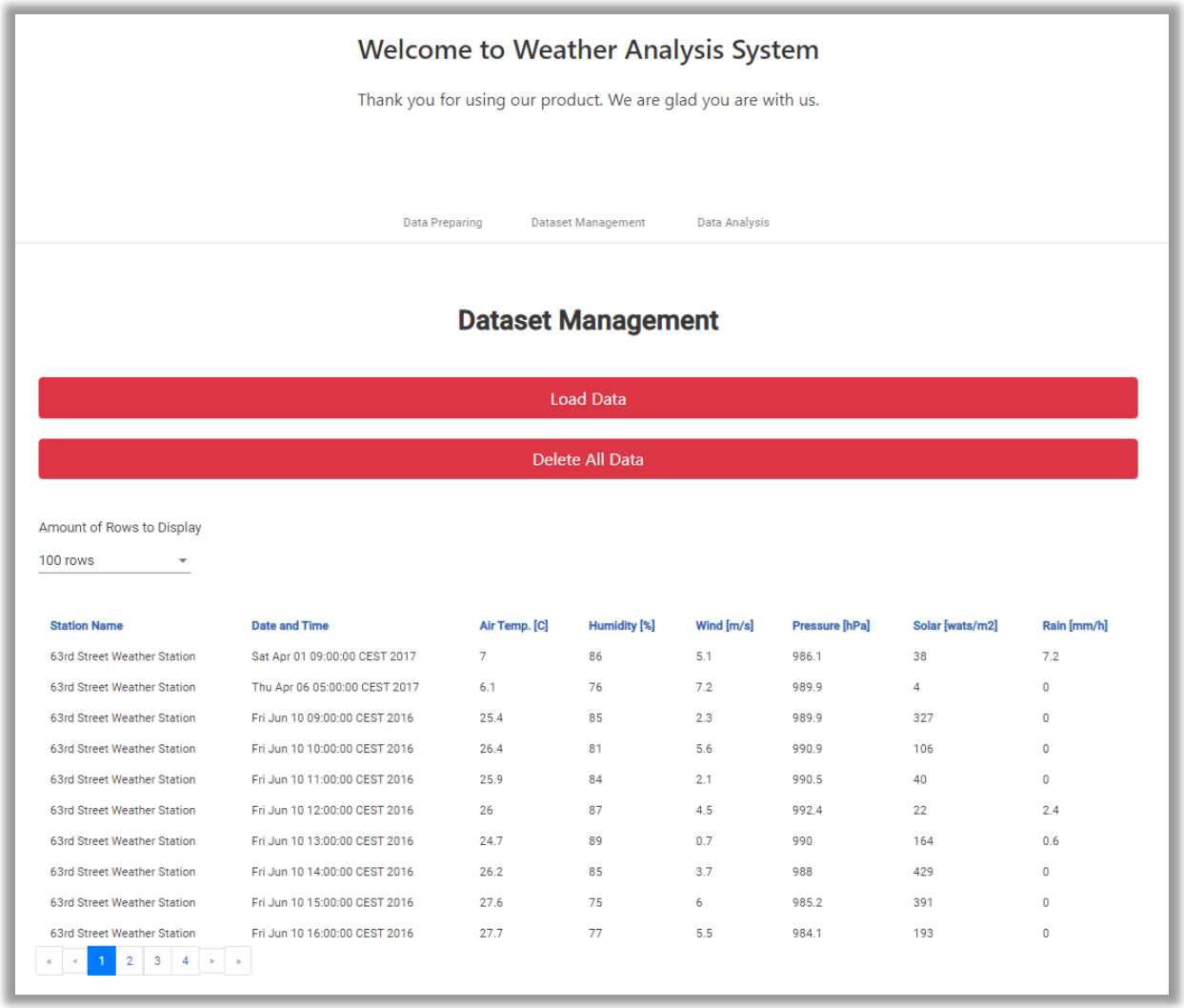
Po uruchomieniu aplikacji i wejściu pod adres serwera klienta użytkownik zobaczy następujący ekran (przytoczony już wcześniej).



Na ekranie początkowym powinien wybrać plik z danymi w formacie .CSV a następnie wcisnąć przycisk ***Upload Data.*** Rozpocznie się ładowanie danych za pośrednictwem klienta, poprzez serwer aż do bazy. Użytkownik ujrzy odpowiednio pasek ładowania.



Po załadowaniu danych można przejść do ekranu wyświetlania zestawu danych, który prezentuje się następująco.

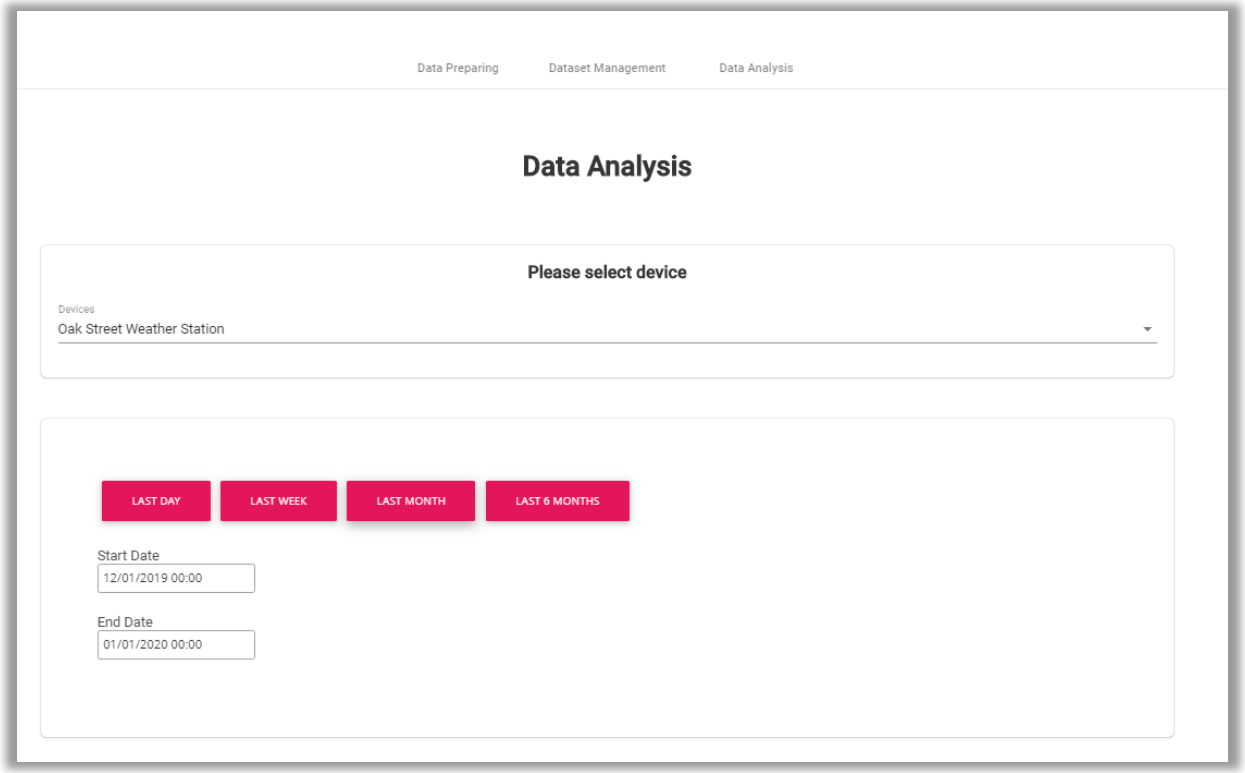


Ekran pozwala na wyświetlenie wszystkich rekordów z wybranymi atrybutami, zdefiniowano możliwość usunięcia wszystkich danych oraz liczby rekordów które zostaną załadowane. Każdy atrybut otrzymał podpis w jakiej jednostce jest wyświetlana wielkość oraz możliwość posortowania rekordów bo wybranym atrybucie rosnąco lub malejąco.

Najistotniejszą jest jednak zakładka *Data Analysis*, której zawartość pozwala skupić się na wybranym urządzeniu (punkcie) pomiarowym, a następnie wybrać zadany okres czasu z którego zostaną przeanalizowane rekordy. Datę można wybrać spośród

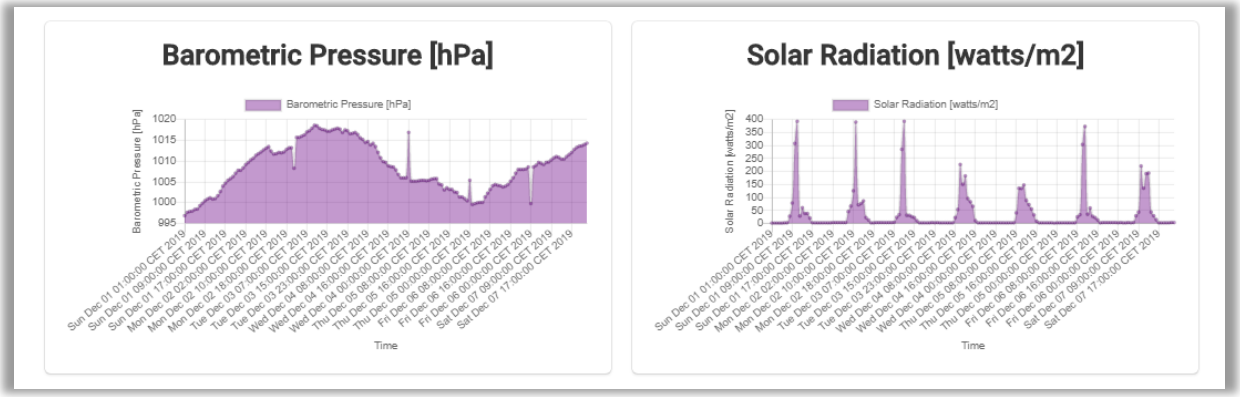
zdefiniowanych zakresów w postaci ostatniego dnia, tygodnia miesiąca bądź ostatniej połowy roku.

Pierwsza część tego ekranu prezentuje się następująco.

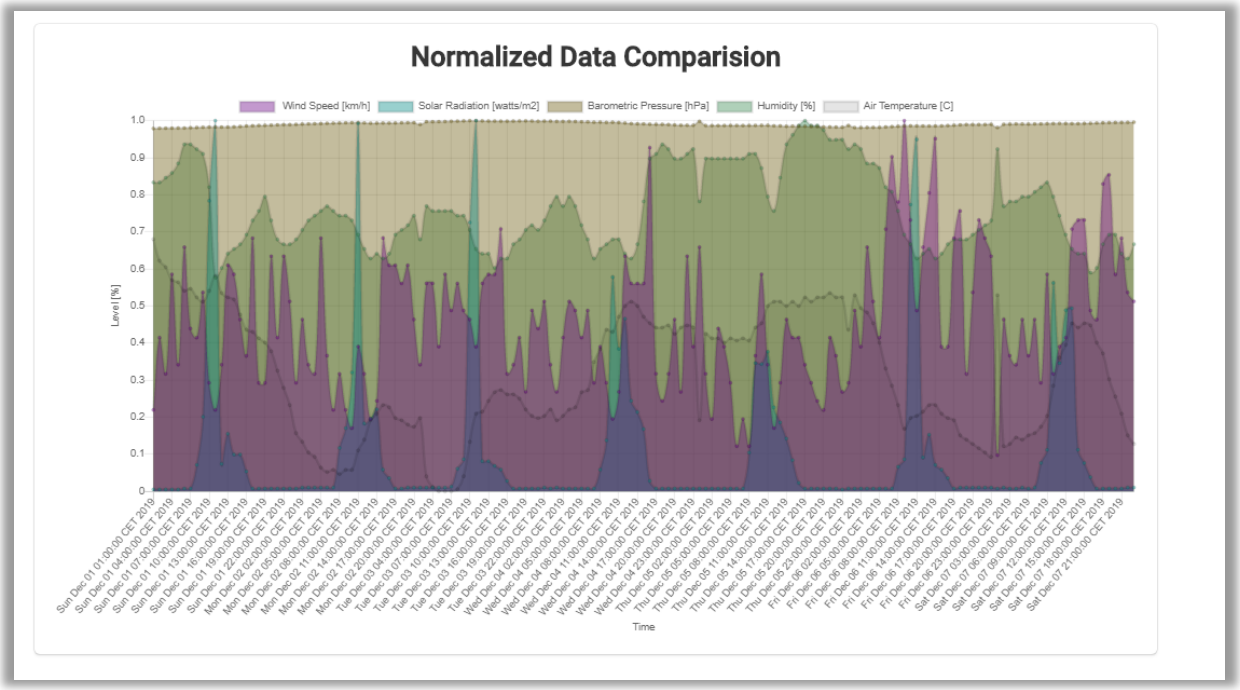


Jak widać w pierwszej kolejności użytkownik może określić urządzenie, z którego serwer zwróci przeanalizowane dane historyczne pomiarów. W kolejnym kroku należy określić przedział czasu, gdy użytkownik tego dokona dalej zostaną wyświetlone wykresy przedstawione na kolejnych zrzutach ekranu.

Wykresy prezentujące wartości temperatury oraz wilgotności

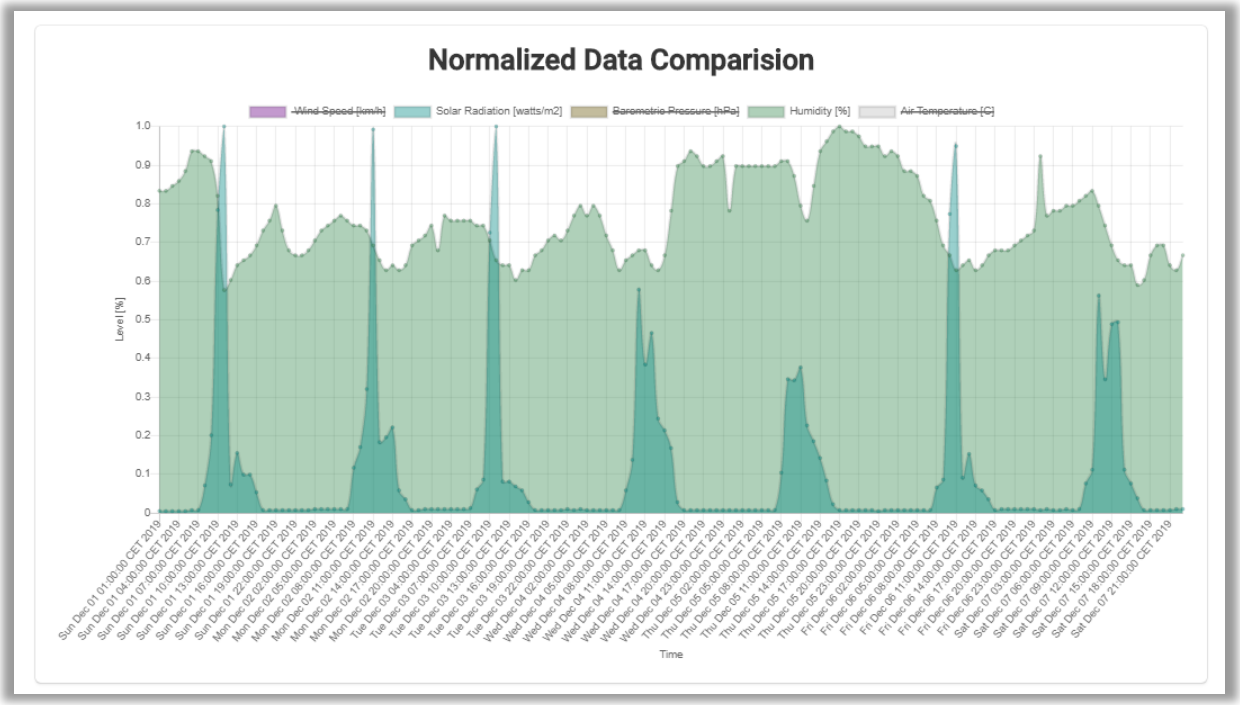


Wykresy prezentujące ciśnienie atmosferyczne oraz promieniowanie słoneczne



Wykresy prezentujący znormalizowany zestaw danych (pięć wybranych atrybutów)

Na wykresie wszystkie z wybranych wartości atrybutów rekordów za wybranego odcinka czasu zostały znormalizowane do wartości 0 – 1. Wykres umożliwia wybranie interesujących atrybutów i prezentacji wybranych, tych, które interesują użytkownika z punktu widzenia porównania zależności pomiędzy poziomami poszczególnych wartości w celu znalezienia zależności.

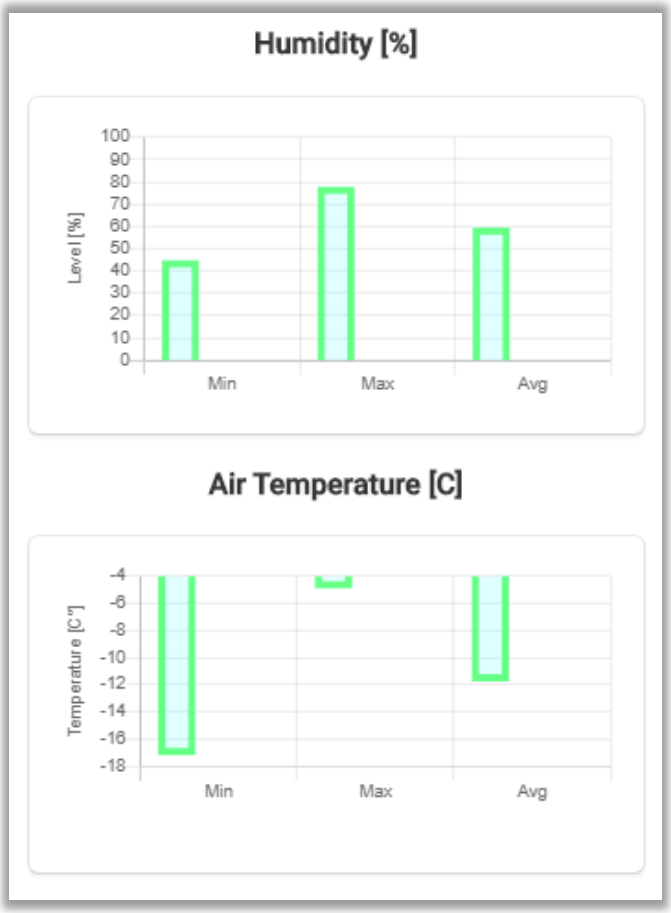


Wykresy prezentujący znormalizowany zestaw danych (promieniowanie słoneczne i wilgotność)

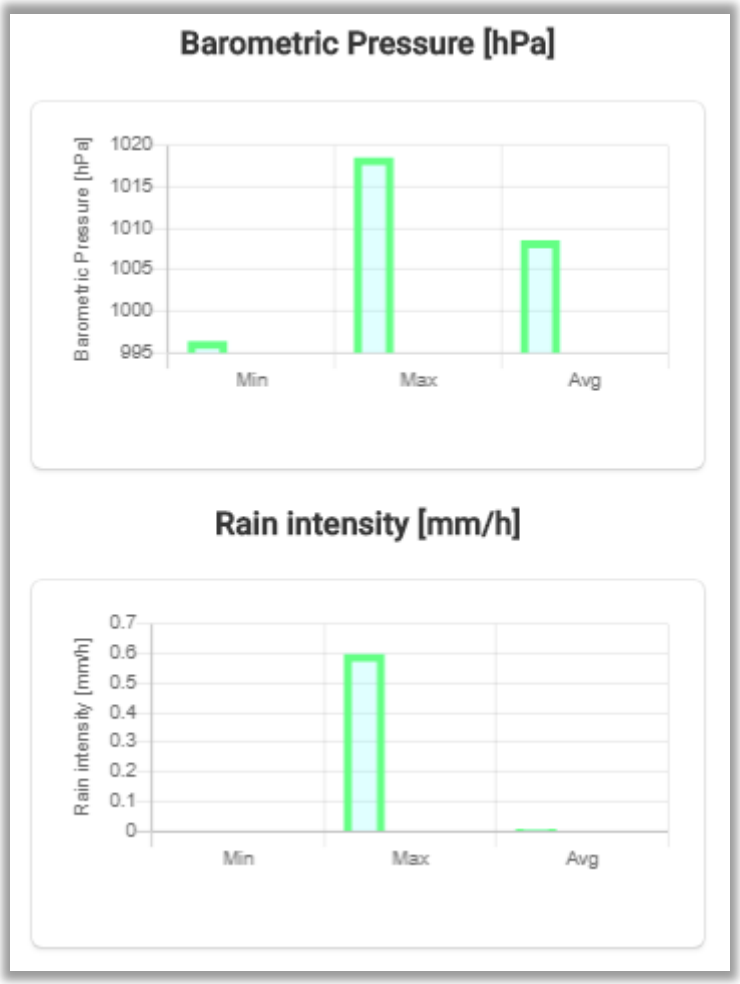
Na przykładzie powyższego wykresu wybranych atrybutów w postaci promieniowania słonecznego i wilgotności można zaobserwować prawidłowość w danych w postaci spadku wilgotności powietrza w czasie, w którym słońce świeciło najmocniej.

Kolejnym elementem analizy danych jest zestaw wykresów słupkowych określających minimalną, maksymalną oraz średnią wartość każdego z atrybutów. Wszystkie wartości dotyczą zestawu danych również z wybranego wcześniej zakresu czasu.

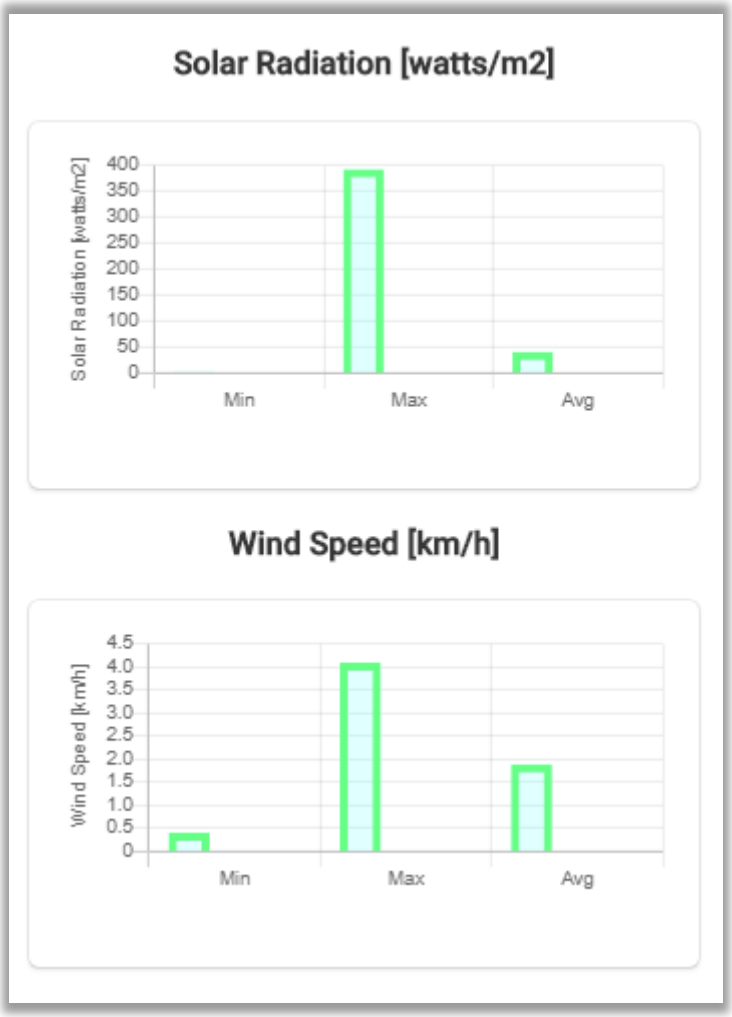
Wyznaczone dla przykładowego zakresu czasu wartości zaprezentowane zostały na wykresach poniżej.



Wykresy prezentujący wartości, minimalne, maksymalne oraz średnie atrybutu wilgotności oraz temperatury powietrza



Wykresy prezentujący wartości, minimalne, maksymalne oraz średnie atrybutu ciśnienia atmosferycznego oraz opadów deszczu



Wykresy prezentujący wartości, minimalne, maksymalne oraz średnie atrybutu promieniowania słonecznego oraz prędkości wiatru.

6. WNIOSKI

Tworząc projekt udało się zrealizować podstawową analizę danych historycznych, na podstawie których zaprezentowano zależności oraz wyniki odnoszące się do wybranych atrybutów zbioru. W Sprawozdaniu przedstawiono strukturę aplikacji, wybrane fragmenty implementacji oraz instrukcję obsługi.

Wybrane technologie w pełni pozwoliły stworzyć zamierzoną aplikację pozwalają również ją rozwijać. Frameworki Spring oraz Angular 8 są dość nowymi oraz popularnymi technologiami wykorzystywanymi w projektowaniu aplikacji z niemal każdej dziedziny.

Wybrany zbiór danych został uzyskany z aktualnych źródeł, które umożliwiają dokładną analizę. W zbiorze można było zauważyć korelacje pomiędzy atrybutami, bez problemu udało się stworzyć wizualizację tych danych oraz obsługę zarówno po stronie serwera jak i aplikacji klienta.

W bibliografii umieszczono odnośniki do zbioru danych, dokumentacji oraz narzędzi wykorzystanych na potrzeby opisanego projektu.

7. BIBLIOGRAFIA

[1] Beach Weather Stations - Automated Sensors [online]
[ostatni dostęp: 15.01.2020], Dostępny w Internecie:
https://data.cityofchicago.org/Parks-Recreation/Beach-Weather-Stations-Automated-Sensors/k7hf-8y75

[2] Dokumentacja biblioteki ChartJS [online]
[ostatni dostęp: 15.01.2020], Dostępny w Internecie:
https://www.chartjs.org/

[3] Dokumentacja Frameworka Angular [online],
[ostatni dostęp: 15.01.2020], Dostępny w Internecie:
https://angular.io/docs

[4] Dokumentacja Frameworka Spring [online],
[ostatni dostęp: 15.01.2020], Dostępny w Internecie:
https://spring.io/guides

[5] Dokumentacja Angular Material [online],
[ostatni dostęp: 15.01.2020], Dostępny w Internecie:
https://material.angular.io/guides

[6] Dokumentacja BULMA [online],
[ostatni dostęp: 15.01.2020], Dostępny w Internecie:
https://bulma.io/documentation/

[7] Dokumentacja MongoDB [online],
[ostatni dostęp: 15.01.2020], Dostępny w Internecie:
https://docs.mongodb.com/