

# Aplikacja do sprawdzania pogody + fakty o kotach

Język Java + łączność z wykorzystaniem API

Krystian Graba

1. Celem projektu jest stworzenie desktopowej aplikacji która umożliwi odpytanie zewnętrznego API serwisu pogodowego i wyświetlenie otrzymanych danych. Aplikacja będzie aplikacją wieloplatformową pozwalającą na jej uruchomienie zarówno w systemach z rodziny Windows jak i Linux.

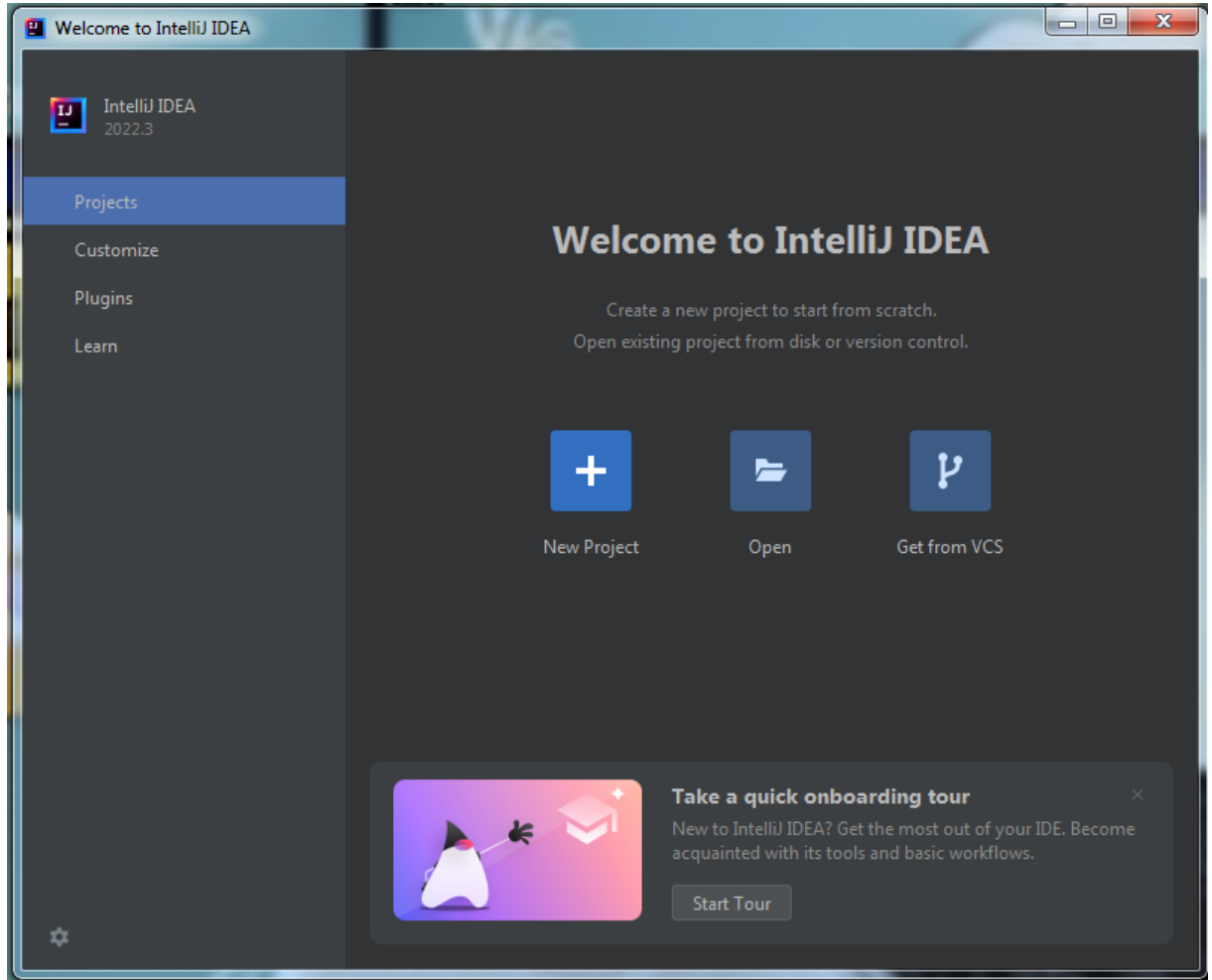
Jako środowisko programistyczne w projekcie wykorzystano produkt firmy JetBrains w postaci oprogramowania IntelliJ IDEA w wersji 2022.3. Po pomyślnej instalacji środowiska należy zresetować komputer.



W projekcie skorzystano z języka Java w wersji x oraz modułów X

2. Należy pobrać środowisko programistyczne ze strony producenta <https://www.jetbrains.com/idea/download> i zainstalować je korzystając z domyślnych ustawień instalatora.

3. Po pomyślnej instalacji należy włączyć oprogramowanie i utworzyć nowy projekt



Z listy w lewej części ekranu należy wybrać JavaFX oraz w głównym oknie sprecyzować dane projektu zgodnie z podanymi poniżej:

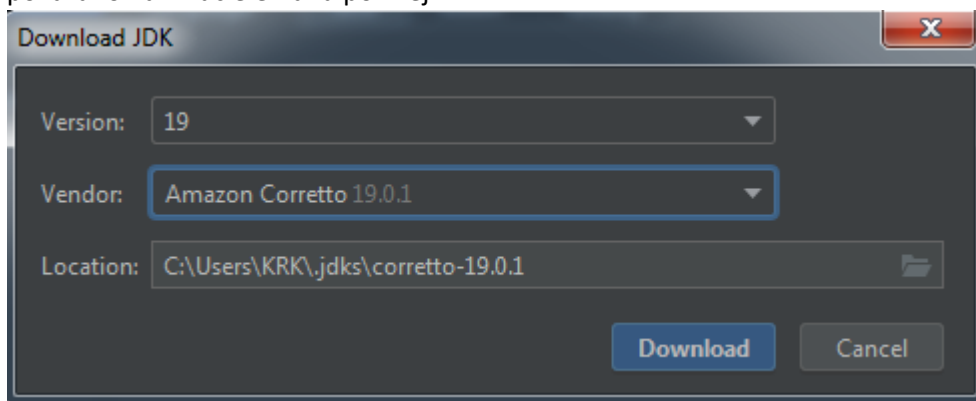
Nazwa: Pogoda

Język: Java

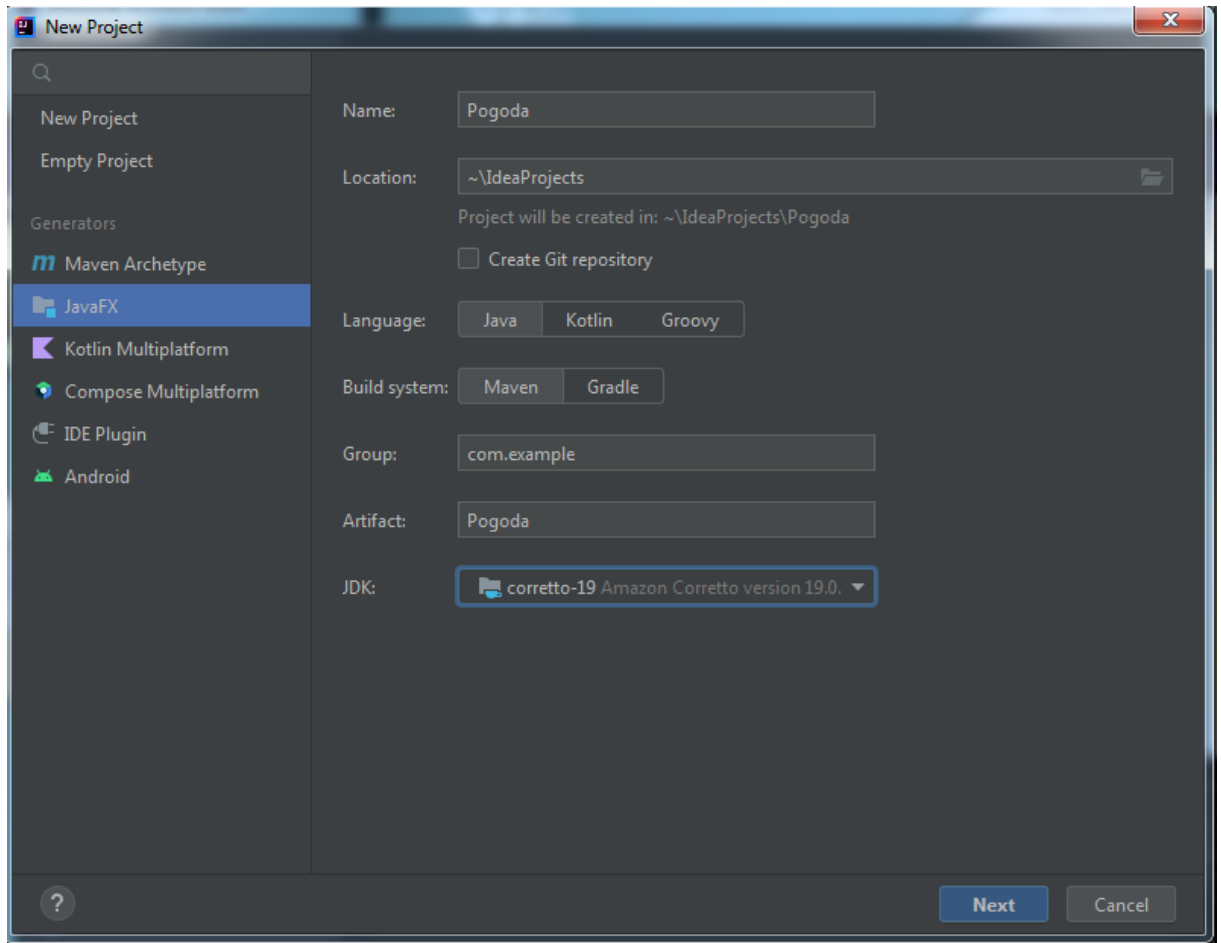
Build system: Maven

Należy również pobrać JDK które nie jest domyślnie dostarczane wraz ze środowiskiem.

W celu pobrania JDK należy w sekcji JDK wybrać opcję Download JDK... i z wyskakującej listy wybrać 19 wersję Javy „Amazon Corretto” i pobrać ją korzystając z przycisku Download jak pokazano na zrzucie ekranu poniżej.



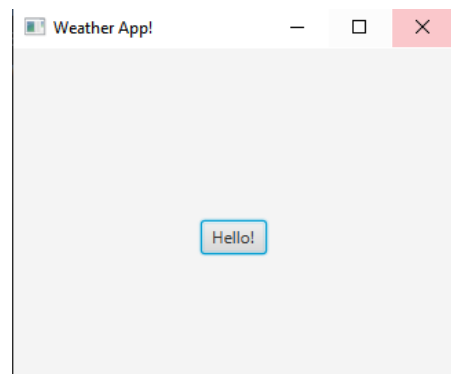
Po pomyślnej instalacji JDK pełen ekran tworzenia nowego projektu powinien wyglądać tak jak na obrazku poniżej



Następnie należy przejść do następnego okna klikając Next w którym należy z listy wybrać dodatkową bibliotekę „FormsFX”.

4. Po uruchomieniu i zaindeksowaniu się projektu należy zbudować i uruchomić aplikację (plik HelloApplication.java) w celu sprawdzenia poprawności procesu tworzenia aplikacji. Ekran który powinien pojawić się po uruchomieniu aplikacji przedstawiony jest na zrzucie ekranu obok.

Po kliknięciu w przycisk powinien pojawić się nad nim komunikat „Welcome to JavaFX Application!”



5. Należy dokonać modyfikacji automatycznie wygenerowanego kodu zmieniając rozmiar okna, nazwę aplikacji oraz tło aplikacji. Zmiany dokonywane są w obrębie pliku `com/example/pogoda/HelloApplication.java`

- a. Nazwa/tytuł okna aplikacji

```
stage.setTitle("Weather App!");
```

- b. Rozmiar okna aplikacji

```
Scene scene = new Scene(fxmlLoader.load(), 320, 640);
```

- c. Pozwolenie użytkownikowi na zmianę rozmiaru okna

```
stage.setResizable(false);
```

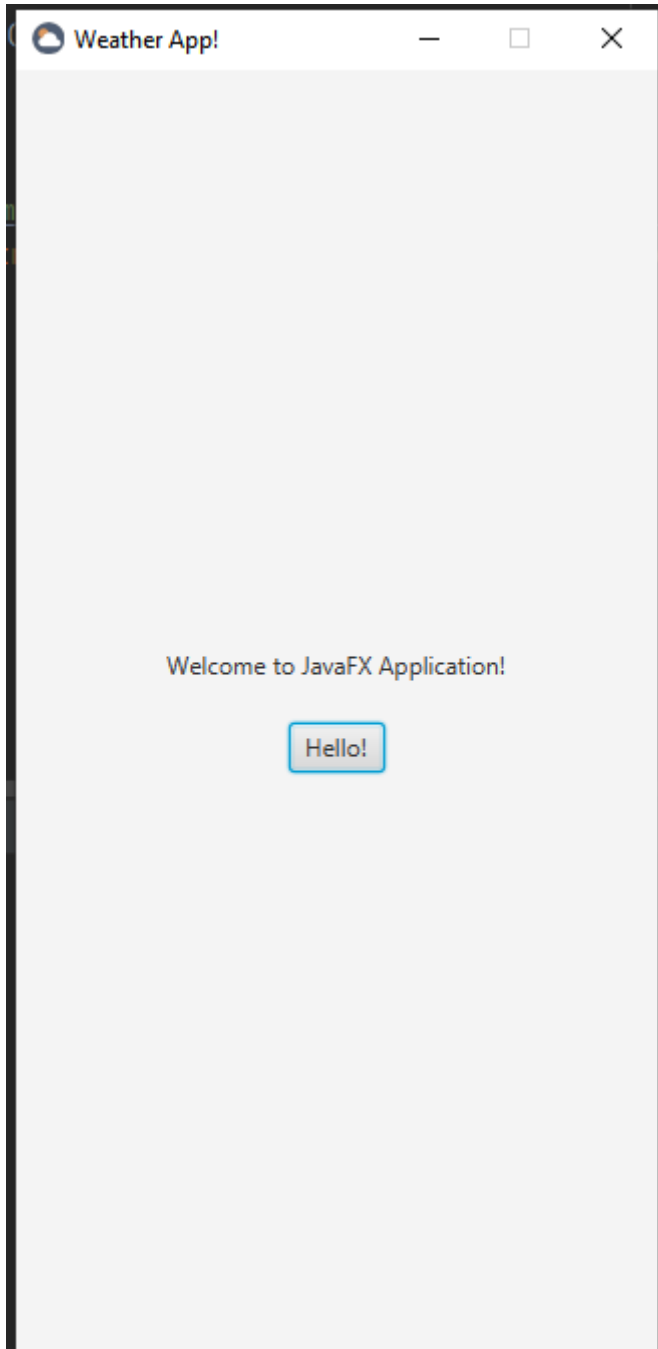
- d. Ustawienie ikony. W przypadku tego projektu ikona pobierana jest z zewnętrznego serwera (Wikipedia)

```
stage.getIcons().add(new
javafx.scene.image.Image("https://upload.wikimedia.org/wikipedia/commons/th
umb/b/bf/Circle-icons-weather.svg/2048px-Circle-icons-weather.svg.png"));
```

- e. Ustawienie wypełnienia/tła ekranu. W tym celu skorzystano z funkcji `setFill` jako argument podając wartości Gradientu które mają się wyświetlić

```
scene.setFill(new RadialGradient(0, 0, 0.5, 0.5, 1, true, null, new Stop(0,
Color.web("#ff99ff")), new Stop(1, Color.web("#32ff12"))));
```

Po edycji pliku `HelloApplication.java` należy uruchomić aplikację aby sprawdzić czy kompiluje się ona bez problemu. Obraz aplikacji powinien wyglądać jak na zdjęciu poniżej. Warto zwrócić uwagę na zmieniony tytuł aplikacji oraz ikonę.



Pełny kod pliku:

```

package com.example.pogoda;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.paint.RadialGradient;
import javafx.scene.paint.Stop;
import javafx.stage.Stage;

import java.io.IOException;

public class HelloApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(HelloApplication.class.getResource("hello-view.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 320, 640);
        stage.setTitle("Weather App!");
        stage.setResizable(false);
        stage.getIcons().add(new
javafx.scene.image.Image("https://upload.wikimedia.org/wikipedia/commons/th
umb/b/bf/Circle-icons-weather.svg/2048px-Circle-icons-weather.svg.png"));
        scene.setFill(new RadialGradient(0, 0, 0.5, 0.5, 1, true, null, new
Stop(0, Color.web("#ff99ff")), new Stop(1, Color.web("#32ff12"))));
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch();
    }
}

```

6. Następnie należy przejść do pliku hello-view.fxml w którym zdefiniujemy obiekty na ekranie takie jak przyciski, miejsca na tekst oraz pola wprowadzania danych.
  - a. Pierwszą rzeczą którą należy zmienić jest dopisanie informacji o tle aplikacji w tagu VBox za pomocą tagu style

```

<VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/fxml"
fx:controller="com.example.pogoda.HelloController"
style="-fx-background-color: #FEC8D8">

```

- b. Należy zmienić także odstęp od górnej części ekranu tak aby aplikacja wyglądała ładnie. Atrybut top w obiekcie „Insets” należy zmienić z domyślnych 20.0 na np. 5.0
  - c. Kolejną najważniejszą rzeczą jest dodanie w podanej kolejności Labeled, miejsc na zdjęcia i przycisku będących miejscem wyświetlania informacji.

Potrzebne będą następujące obiekty:

- i. Label o id meow\_fact z pustym tekstem w której znajdzie się losowy cytat o kotach

```

<Label fx:id="meow_fact" text=""/>

```

- ii. ImageView o id weatherIcon w którym wyświetlone zostanie ikona aktualnej pogody. Warto tutaj sprecyzować rozmiar takiej ikony korzystając z atrybutów fitHeight oraz fitWidth. Ustawiono oba tagi na 128.0

```

<ImageView fx:id="weatherIcon" fitHeight="128.0" fitWidth="128.0"/>

```

iii. Label o id welcomeText bez tekstu

```
<Label fx:id="welcomeText"/>
```

iv. Label bez id z tekstem proszącym o wpisanie nazwy miejscowości dla której ma zostać sprawdzona pogoda

```
<Label text="Enter the city name"/>
```

v. Pole tekstowe TextField o id cityNameTextField do którego użytkownik będzie wpisywał nazwę miejscowości

```
<TextField fx:id="cityNameTextField"/>
```

vi. Przycisk z tekstem „Show weather!” ze zdefiniowanym atrybutem onAction o nazwie #onShowWeatherButton. Po zdefiniowaniu nazwy środowisko powinno zapytać czy utworzyć metodę o takiej nazwie w kontrolerze. Należy wyrazić zgodę.

```
<Button text="Show weather!" onAction="#onShowWeatherButton"/>
```

vii. 3 Labelki o id „googleMaps”, „google”, „wikipedia” bez tekstu które przechowywać będą linki do wspomnianych portali.

```
<Label fx:id="googleMaps" text=""/>
<Label fx:id="google" text=""/>
<Label fx:id="wikipedia" text=""/>
```

d. Po edycji pliku należy ponownie skompilować cały projekt i go uruchomić w celu sprawdzenia błędów. Aplikacja powinna wyglądać jak na zrzucie poniżej.



Przycisk jak i pole tekstowe na tym etapie nie „działają” – nic się nie dzieje po ich wciśnięciu.

Po pomyślnym utworzeniu projektu aplikacji pod względem graficznym należy przejść do pliku `HelloController.java` w którym utworzona zostanie logika aplikacji.

7. Inicjalizacja pól dodanych w pliku `fxml` w Kontrolerze. Pola zostały zainicjalizowane na początku klasy kontrolera.

```
public TextField cityNameTextField;
public ImageView weatherIcon;
public Label googleMaps;
public Label google;
public Label wikipedia;
public Label meow_fact;
@FXML
private Label welcomeText;
```

8. Funkcjonalność pobierania losowego cytatu o kotach

Do tego celu wykorzystano darmowe API nie wymagające rejestracji i klucza API zwracające taki cytat. Utworzono funkcję `getMeow` pobierającą z API tekst i zwracającą tekst jako `String`.

- a. Utworzenie na początku programu, zaraz po inicjalizacji pól wywołania funkcji zwracającej cytat o kocie. Funkcja przypisuje tekst do pola `meow_fact` pobrany z funkcji `getMeow` która została omówiona poniżej.

```
@FXML
public void initialize() throws IOException {
    meow_fact.setText(getMeow());
}
```

- b. Utworzenie na spodzie klasy kontrolera funkcji `String` o nazwie `GetMeow`. W kodzie funkcji sprecyzowano url API, wybrano metodę oraz właściwości żądania `http`.

```
public String getMeow() throws IOException {
    URL url = new URL("https://meowfacts.herokuapp.com/");
    HttpURLConnection con = (HttpURLConnection)
url.openConnection();
    con.setRequestMethod("GET");
    con.setRequestProperty("Content-Type", "application/json; utf-8");
}
```

- c. Następnie ustanowiono połączenie sprawdzające czy połączenie przebiegło pomyślnie. Jeżeli status połączenia był inny niż 200 (HTTP OK) funkcja zwracała błąd a w innym przypadku otrzymane dane zostały przetworzone i zwrócone.

```
int status = con.getResponseCode();
if (status!=200) {
    System.out.println("Error");
    return "I can't get the meow facts right now";
}
else {
```

- d. W celu zczytania danych wykorzystany został obieg `Scanner` oraz `StringBuilder`. Z wykorzystaniem obu obiektów zostało odczytane całe żądanie i zapisane do zmiennej `inline`. Po odczytaniu zamknięty obiekt `Scanner` został zamknięty.

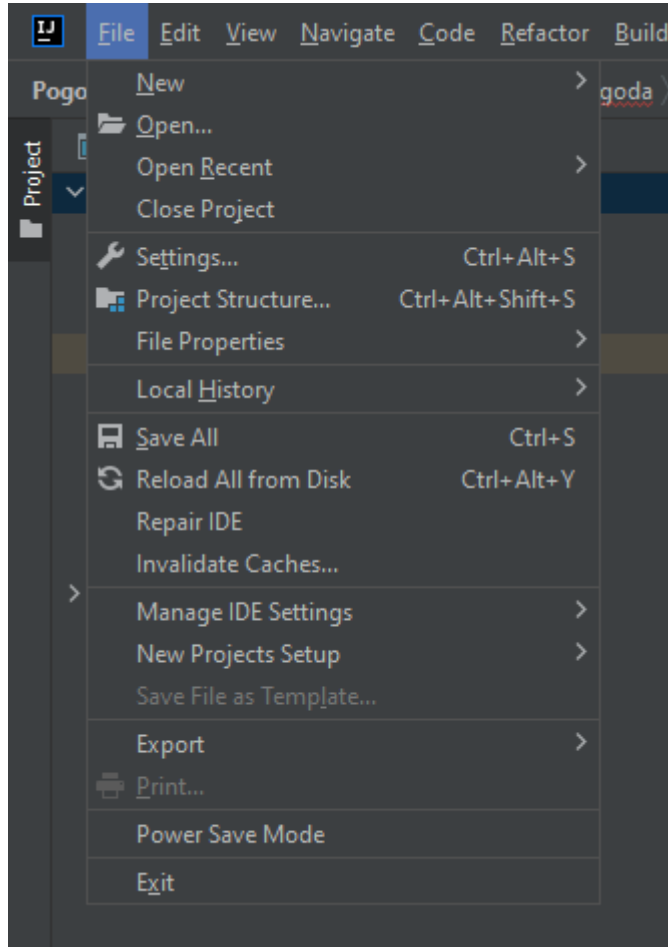


```

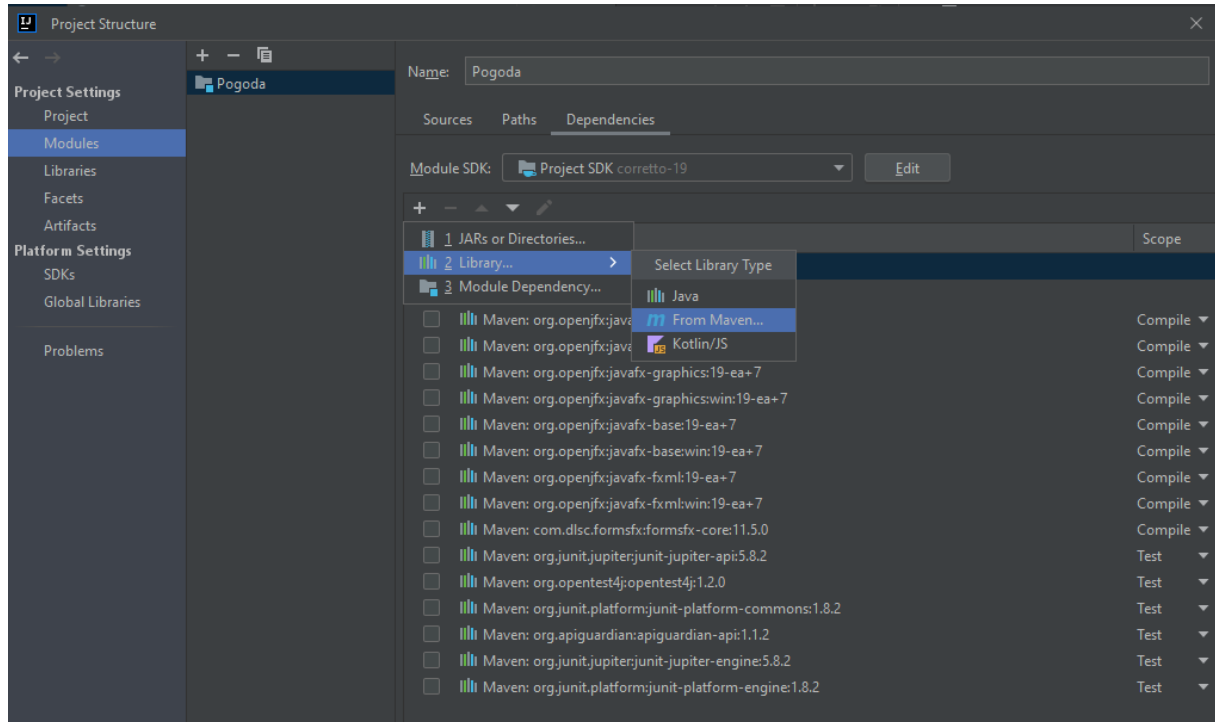
else {
    Scanner sc = new Scanner(url.openStream());
    StringBuilder inline = new StringBuilder();
    while (sc.hasNext()) {
        inline.append(sc.nextLine());
    }
    sc.close();
}

```

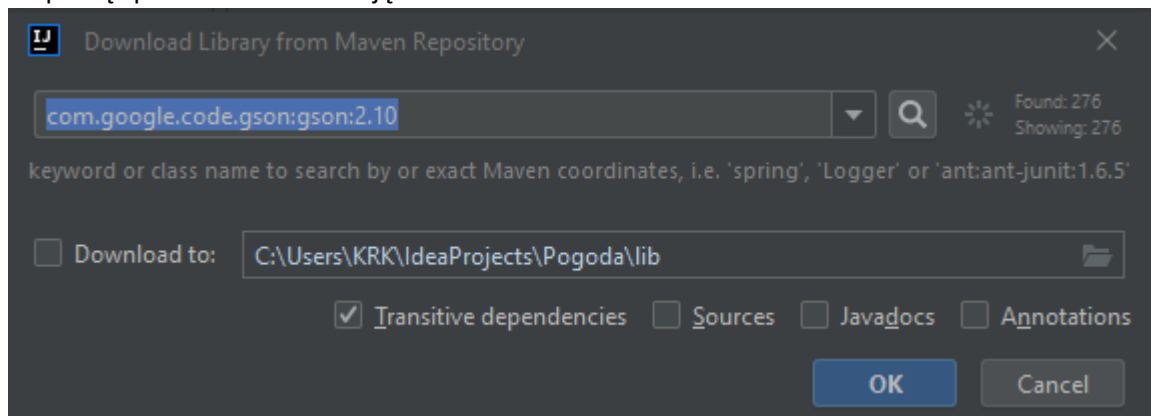
- e. Następnie z wykorzystaniem biblioteki Gson operującej na zmiennej `inline` pobrano tylko tekst cytatu (pomijając np. informacje o połączeniu) i został on zwrócony. Biblioteka Gson nie jest biblioteką domyślnie zainstalowaną w IntelliJ dlatego należy ją ręcznie doinstalować. W tym celu należy przejść do ustawień projektu (Project Structure). Skrót klawiaturowy `Ctrl+Alt+Shift+S`.



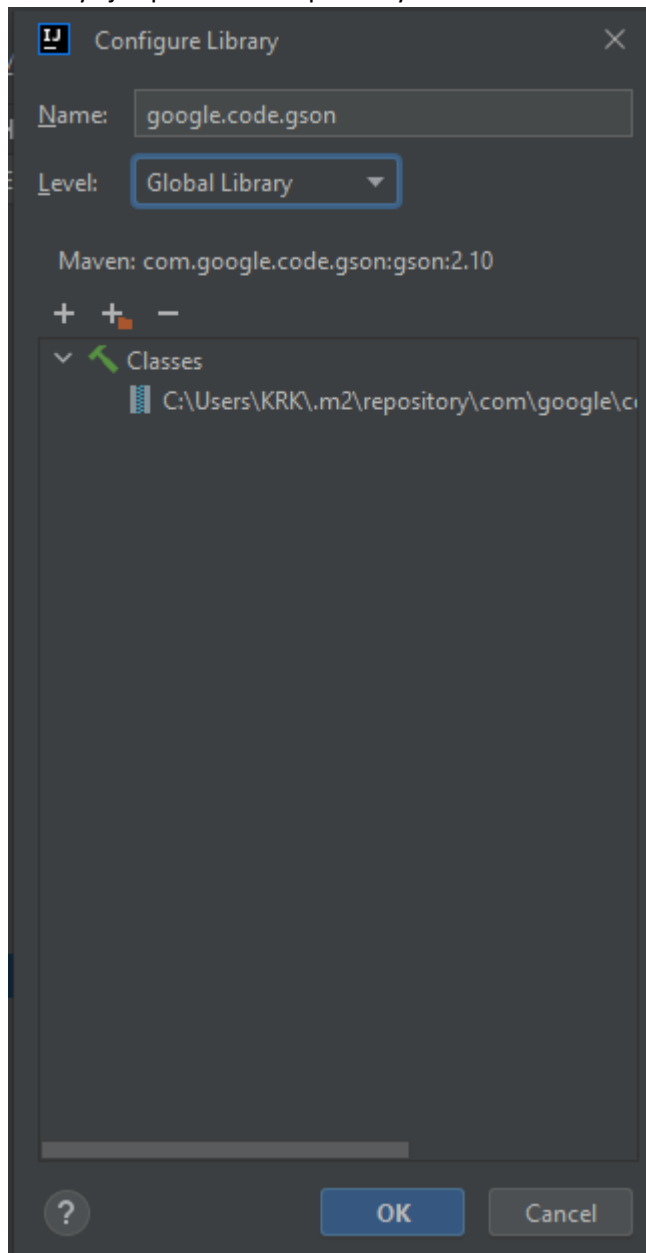
Następnie w oknie które wyskoczyło należy przejść do zakładki modules, kliknąć na plusa i wybrać opcję from Maven



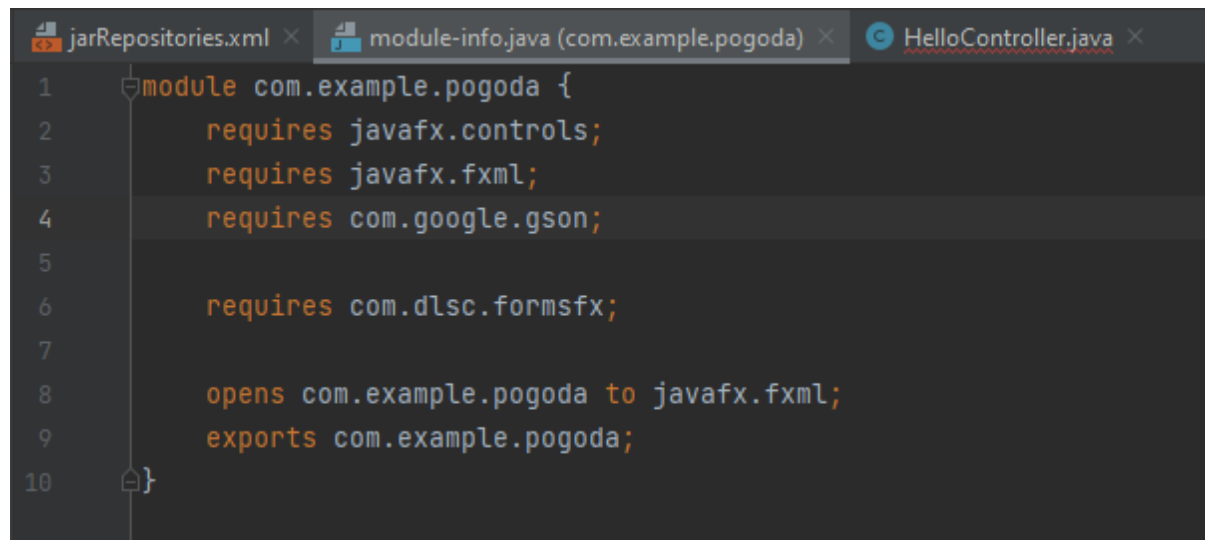
W polu wyszukiwania należy wpisać gson i wybrać „com.google.code.gson:gson:2.10”. Po wybraniu należy kliknąć OK aby rozpocząć pobieranie i instalację.



Po zainstalowaniu należy skonfigurować bibliotekę zmieniając jej poziom na „Global Library” jak pokazano na poniższym zrzucie



- f. Po instalacji biblioteki należy przejść jeszcze do pliku *src/main/java/module-info.java* i w nim dopisać linijkę „requires com.google.gson” jak zaprezentowano na zrzucie poniżej.



```
1 module com.example.pogoda {
2     requires javafx.controls;
3     requires javafx.fxml;
4     requires com.google.gson;
5
6     requires com.dlsc.formsfx;
7
8     opens com.example.pogoda to javafx.fxml;
9     exports com.example.pogoda;
10 }
```

Teraz po instalacji biblioteki można dokończyć funkcję i finalnie zwrócić dane. W tym celu ze Stringa inline zwracana jest tylko wartość „data”.

```
JsonObject jsonObject = new Gson().fromJson(inline.toString(),
JsonObject.class);
return jsonObject.get("data").getAsString();
```

- g. Po uruchomieniu aplikacji w górnej części ekranu powinien znaleźć się pobrany cytat który zresetuje się po ponownym uruchomieniu aplikacji.



Dotychczas utworzony kod aplikacji znajduje się poniżej

```
package com.example.pogoda;

import com.google.gson.Gson;
import com.google.gson.JsonObject;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.image.ImageView;

import javax.net.ssl.HttpURLConnection;
```

```

import java.io.IOException;
import java.net.URL;
import java.util.Scanner;

public class HelloController{
    public TextField cityNameTextField;
    public ImageView weatherIcon;
    public Label googleMaps;
    public Label google;
    public Label wikipedia;
    public Label meow_fact;
    @FXML
    private Label welcomeText;
    @FXML
    public void initialize() throws IOException {
        meow_fact.setText(getMeow());
    }

    @FXML
    protected void onHelloButtonClick() {
        welcomeText.setText("Welcome to JavaFX Application!");
    }

    public void onShowWeatherButton(ActionEvent actionEvent) {
    }

    public String getMeow() throws IOException {
        URL url = new URL("https://meowfacts.herokuapp.com/");
        HttpURLConnection con = (HttpURLConnection) url.openConnection();
        con.setRequestMethod("GET");
        con.setRequestProperty("Content-Type", "application/json; utf-8");

        int status = con.getResponseCode();
        if (status!=200) {
            System.out.println("Error");
            return "I can't get the meow facts right now";
        }
        else {
            Scanner sc = new Scanner(url.openStream());
            StringBuilder inline = new StringBuilder();
            while (sc.hasNext()) {
                inline.append(sc.nextLine());
            }
            sc.close();

            JsonObject jsonObject = new Gson().fromJson(inline.toString(),
JsonObject.class);
            return jsonObject.get("data").getAsString();
        }
    }
}

```

9. Kolejnym krokiem w celu utworzenia aplikacji jest część „pogodowa” aplikacji.
  - a. Pierwszym krokiem jest rejestracja w serwisie weatherapi.com w celu wygenerowania klucza API potrzebnego do generowania żądań. WeatherAPI oferuje darmowy plan, oraz wersję testową z których też należy skorzystać podczas pracy na projektem. Jeżeli prace nad projektem są wykonywane przed 16 grudnia 20222 roku

można skorzystać z mojego klucza API **75645df15c7c497d82194624220212** i pominąć krok rejestracji w serwisie.

- b. Po wygenerowaniu klucza należy przejść do IntelliJ i rozpocząć pisanie aplikacji. Kod będzie pisany wewnątrz automatycznie wygenerowanej funkcji `onShowWeatherButton`.
- c. Pierwszym krokiem będzie pobranie tekstu z pola wprowadzania tekstu aplikacji oraz sprawdzenie czy nie jest pusty i czy nie jest nullem. Jeżeli tak się zdarzy należy poprosić użytkownika o wprowadzenie nazwy miasta ustawiając tekst zmiennej `welcomeText` na „Please enter a city name”.

```
public void onShowWeatherButton(ActionEvent actionEvent) {
    if (!(cityNameTextField != null && cityNameTextField.getText() !=
null && cityNameTextField.getText().length() > 0)) {
        welcomeText.setText("Please enter a city name");
    } else {
```

- d. Gdy jednak nazwa miejscowości została wpisana poprawnie należy przejść do utworzenia żądania API dla miejscowości podanej przez użytkownika. W tym celu utworzono obiekt URL zawierający żądanie. Został wykorzystany wzór podany w dokumentacji właściciela API (<https://www.weatherapi.com/docs/>). W URL wpisano wcześniej wygenerowany klucz API, oraz nazwę miejscowości podaną przez użytkownika. Następnie ustanowiono połączenie. Jeżeli połączenie się powiodło(status http OK 200) program rozpoczyna „przechwytywanie odpowiedzi” w celu zapisania jej do zmiennej `inLine`. Wykorzystano tak jak w przypadku faktów o kotach Obiekty Scanner i String Builder.

```
welcomeText.setText("Showing forecast for " +
cityNameTextField.getText());
URL url = new
URL("https://api.weatherapi.com/v1/current.json?key=75645df15c7c497d8
2194624220212&q=" + cityNameTextField.getText() + "&aqi=no");
HttpsURLConnection con = (HttpsURLConnection) url.openConnection();
con.setRequestMethod("GET");
con.setRequestProperty("Content-Type", "application/json; utf-8");

int status = con.getResponseCode();

if (!(status == 200)) {
    welcomeText.setText("Error: " + status);
} else {
    StringBuilder inLine = new StringBuilder();
    Scanner scanner = new Scanner(url.openStream());

    while (scanner.hasNext()) {
        inLine.append(scanner.nextLine());
    }
    System.out.println(inLine);
```

- e. Po tym etapie odpowiedź z API znajduje się w zmiennej `inLine` skąd zostaną pobrane interesujące nas parametry takie jak temperatura, wiat, ciśnienie, opady, wilgotność oraz ikona obecnej pogody. W tym celu skorzystano z wcześniej doinstalowanej biblioteki Gson. Przetworzone dane zostały ustawione jako tekst zmiennej `welcomeText`.

```
JsonObject convertObj = new Gson().fromJson(inLine.toString(),
JsonObject.class);
```

```
String city_name =
convertObj.get("location").getAsJsonObject().get("name").getString(
);
welcomeText.setText("Weather in: " + city_name +
": Temperature: " +
convertObj.get("current").getAsJsonObject().get("temp_c").getString(
) +
"C\n Wind: " +
convertObj.get("current").getAsJsonObject().get("wind_kph").getAsString() +
"kph\nPressure: " +
convertObj.get("current").getAsJsonObject().get("pressure_mb").getAsString() +
"mb\nPrecipitation: " +
convertObj.get("current").getAsJsonObject().get("precip_mm").getAsString() +
"mm\nHumidity: " +
convertObj.get("current").getAsJsonObject().get("humidity").getAsString() +
"%\nCondition: " +
convertObj.get("current").getAsJsonObject().get("condition").getAsJsonObject().get("text").getString());
```

f. Następnie zmieniono font tekstu oraz jego kolor. Ustawiono także zawijanie tekstu

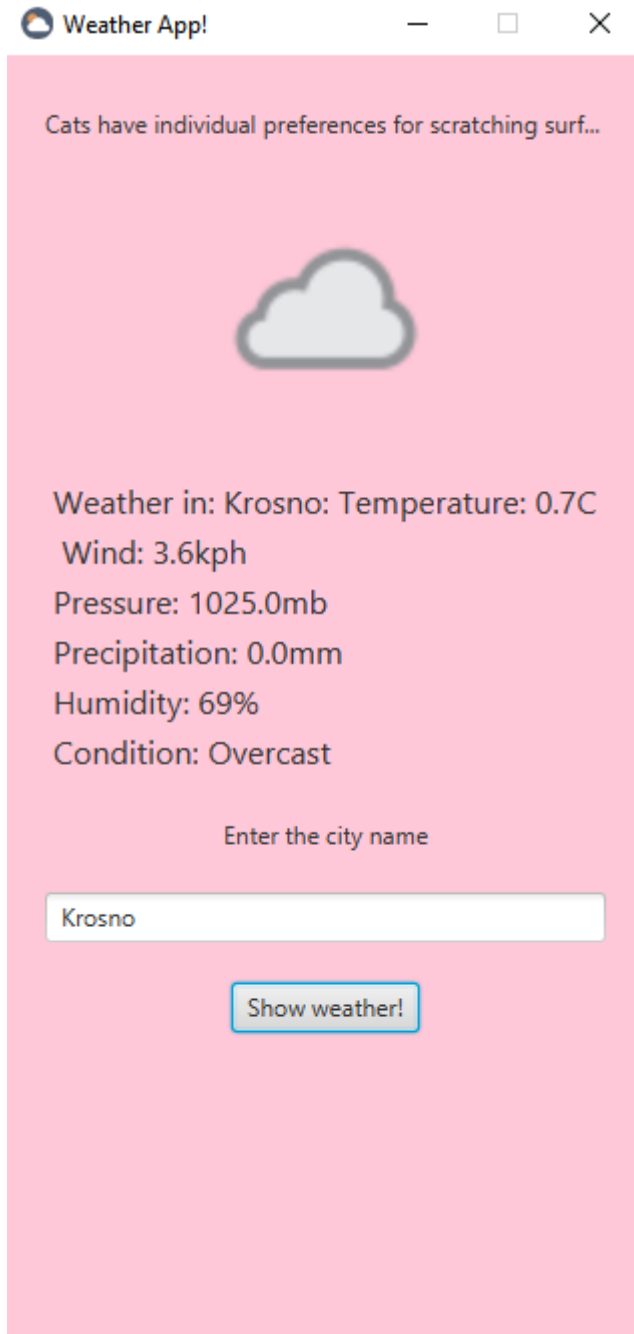
```
welcomeText.setStyle("-fx-text-fill: blue;");
welcomeText.setStyle("-fx-font-size: 16px;");
welcomeText.setWrapText(true);
```

g. Ostatnią kwestią z pogodą jest ustawienie zdjęcia – ikony.

```
Image image = new Image("https:" +
convertObj.get("current").getAsJsonObject().get("condition").getAsJsonObject().get("icon").getString());
weatherIcon.setImage(image);
```



- h. Można teraz przetestować aplikację kompilując ją i wyświetlając pogodę.



Jak widać na załączonym zrzucie aplikacja działa i wyświetla pogodę.

- i. Na zrzucie powyżej widać niektóre problemy aplikacji które należy poprawić, są to:
- i. Tekst faktów o kotach nie „zawija się”, w celu wyeliminowania błędu w funkcji initialize dopisano liniijkę

```
meow_fact.setTextWrapText(true);
```

- ii. Brak znaku nowej linii po nazwie miejscowości – wyświetla się tekst „Weather in Krosno: Temperature...” co nie wygląda atrakcyjnie. Tekst powinien wyglądać następująco: „Weather in Krosno: Tempearture:...”. W celu wyeliminowania błędu dodano znak nowej linii w

funkcji ustawienia tekstu użytkownikowi.

```
welcomeText.setText("Weather in: " + city_name +
    "\n Temperature: " + convertObj.get("current").getAsJsonObject().get("temp_c").getString() +
    "C\n Wind: " + convertObj.get("current").getAsJsonObject().get("wind_kph").getString() +
```

j. Kod pliku HelloController po wykonaniu kroków powyżej

```
package com.example.pogoda;

import com.google.gson.Gson;
import com.google.gson.JsonObject;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;

import javax.net.ssl.HttpURLConnection;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.ProtocolException;
import java.net.URL;
import java.util.Scanner;

public class HelloController{
    public TextField cityNameTextField;
    public ImageView weatherIcon;
    public Label googleMaps;
    public Label google;
    public Label wikipedia;
    public Label meow_fact;
    @FXML
    private Label welcomeText;
    @FXML
    public void initialize() throws IOException {
        meow_fact.setText(getMeow());
        meow_fact.setWrapText(true);
    }

    public void onShowWeatherButton(ActionEvent actionEvent) throws
    IOException {
        if (!(cityNameTextField != null &&
cityNameTextField.getText() != null &&
cityNameTextField.getText().length() > 0)) {
            welcomeText.setText("Please enter a city name");
        } else {
            welcomeText.setText("Showing forecast for " +
cityNameTextField.getText());
            URL url = new
URL("https://api.weatherapi.com/v1/current.json?key=75645df15c7c497d8
2194624220212&q=" + cityNameTextField.getText() + "&aqi=no");
            HttpURLConnection con = (HttpURLConnection)
url.openConnection();
            con.setRequestMethod("GET");
            con.setRequestProperty("Content-Type", "application/json;
utf-8");

            int status = con.getResponseCode();

            if (!(status == 200)) {
                welcomeText.setText("Error: " + status);
            }
        }
    }
}
```

```

    } else {
        StringBuilder inLine = new StringBuilder();
        Scanner scanner = new Scanner(url.openStream());

        while (scanner.hasNext()) {
            inLine.append(scanner.nextLine());
        }
        System.out.println(inLine);

        JsonObject convertObj = new
Gson().fromJson(inLine.toString(), JsonObject.class);

        String city_name =
convertObj.get("location").getAsJsonObject().get("name").getString(
);
        welcomeText.setText("Weather in: " + city_name +
            "\n: Temperature: " +
convertObj.get("current").getAsJsonObject().get("temp_c").getString(
) +
            "C\n Wind: " +
convertObj.get("current").getAsJsonObject().get("wind_kph").getAsString(
) +
            "kph\nPressure: " +
convertObj.get("current").getAsJsonObject().get("pressure_mb").getAsString(
) +
            "mb\nPrecipitation: " +
convertObj.get("current").getAsJsonObject().get("precip_mm").getAsString(
) +
            "mm\nHumidity: " +
convertObj.get("current").getAsJsonObject().get("humidity").getAsString(
) +
            "%\nCondition: " +
convertObj.get("current").getAsJsonObject().get("condition").getAsJsonObject().get("text").getString());

        welcomeText.setStyle("-fx-text-fill: blue;");
        welcomeText.setStyle("-fx-font-size: 16px;");
        welcomeText.setWrapText(true);
        Image image = new Image("https:" +
convertObj.get("current").getAsJsonObject().get("condition").getAsJsonObject().get("icon").getString());
        weatherIcon.setImage(image);
    }
}

public String getMeow() throws IOException {
    URL url = new URL("https://meowfacts.herokuapp.com/");
    HttpsURLConnection con = (HttpsURLConnection)
url.openConnection();
    con.setRequestMethod("GET");
    con.setRequestProperty("Content-Type", "application/json;
utf-8");

    int status = con.getResponseCode();
    if (status!=200) {
        System.out.println("Error");
        return "I can't get the meow facts right now";
    }
}

```

```

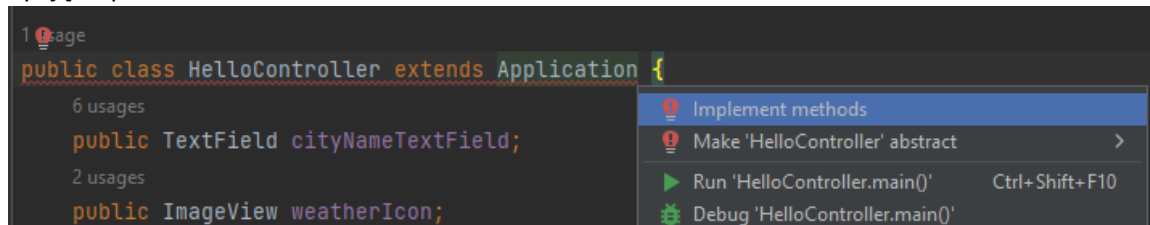
    }
    else {
        Scanner sc = new Scanner(url.openStream());
        StringBuilder inline = new StringBuilder();
        while (sc.hasNext()) {
            inline.append(sc.nextLine());
        }
        sc.close();

        JsonObject jsonObject = new
Gson().fromJson(inline.toString(), JsonObject.class);
        return jsonObject.get("data").getAsString();
    }
}
}

```

10. Dodatkową funkcjonalnością programu są linki do danego miasta w serwisie Google Maps, Google oraz Wikipedia.

- a. W celu dodania takiego linku należy zmienić definicję klasy dopisując „extends” Application jak pokazano na zrzucie oraz najechać na nową implementację i wybrać opcję Implement methods.



Po zaimplementowaniu metod kod nie powinien mieć żadnych błędów.

- b. Kolejnym krokiem jest dopisanie, zaraz pod kodem ustawiającym ikonę pogody część programu odpowiedzialną za tworzenie linków. Należy także zainicjalizować obiekt HostServices odpowiedzialny za „klikalne linki”. Odnośniki zostaną wyświetlone we wcześniej utworzonych w pliku .fxml Labelkach.

```

//Open in google maps feature
String googleMapsUrl =
"https://www.google.com/maps/search/?api=1&query=" + city_name;
googleMaps.setText("Open " + city_name + " in Google Maps");
googleMaps.setOnMouseClicked(event ->
hostServices.showDocument(googleMapsUrl));

//Open in Google
String googleUrl = "https://www.google.com/search?q=" + city_name;
google.setText("Search " + city_name + " in Google");
google.setOnMouseClicked(event ->
hostServices.showDocument(googleUrl));

//Open in wikipedia
String wikipediaUrl = "https://en.wikipedia.org/wiki/" + city_name;
wikipedia.setText("Search " + city_name + " in Wikipedia");
wikipedia.setOnMouseClicked(event ->
hostServices.showDocument(wikipediaUrl));

```

- c. Po uruchomieniu aplikacji pojawiają się te linki w „klikalnej” formie.



- d. Finalny kod pliku HelloController.java po wykonaniu poprzednich kroków

```
11. package com.example.pogoda;

import com.google.gson.Gson;
import com.google.gson.JsonObject;
import javafx.application.Application;
import javafx.application.HostServices;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;
```

```

import javax.net.ssl.HttpsURLConnection;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.ProtocolException;
import java.net.URL;
import java.util.Scanner;

public class HelloController extends Application {
    public TextField cityNameTextField;
    public ImageView weatherIcon;
    public Label googleMaps;
    public Label google;
    public Label wikipedia;
    public Label meow_fact;
    @FXML
    private Label welcomeText;
    @FXML
    public void initialize() throws IOException {
        meow_fact.setText(getMeow());
        meow_fact.setWrapText(true);
    }

    public void onShowWeatherButton(ActionEvent actionEvent) throws
    IOException {
        if (!(cityNameTextField != null &&
cityNameTextField.getText() != null &&
cityNameTextField.getText().length() > 0)) {
            welcomeText.setText("Please enter a city name");
        } else {
            welcomeText.setText("Showing forecast for " +
cityNameTextField.getText());
            URL url = new
URL("https://api.weatherapi.com/v1/current.json?key=75645df15c7c497d8
2194624220212&q=" + cityNameTextField.getText() + "&aqi=no");
            HttpsURLConnection con = (HttpsURLConnection)
url.openConnection();
            con.setRequestMethod("GET");
            con.setRequestProperty("Content-Type", "application/json;
utf-8");

            int status = con.getResponseCode();

            if (!(status == 200)) {
                welcomeText.setText("Error: " + status);
            } else {
                StringBuilder inLine = new StringBuilder();
                Scanner scanner = new Scanner(url.openStream());

                while (scanner.hasNext()) {
                    inLine.append(scanner.nextLine());
                }
                System.out.println(inLine);

                JsonObject convertObj = new
Gson().fromJson(inLine.toString(), JsonObject.class);

                String city_name =
convertObj.get("location").getAsJsonObject().get("name").getString(
);

```

```

        welcomeText.setText("Weather in: " + city_name +
            "\n: Temperature: " +
convertObj.get("current").getAsJsonObject().get("temp_c").getString() +
            "C\n Wind: " +
convertObj.get("current").getAsJsonObject().get("wind_kph").getString() +
            "kph\nPressure: " +
convertObj.get("current").getAsJsonObject().get("pressure_mb").getString() +
            "mb\nPrecipitation: " +
convertObj.get("current").getAsJsonObject().get("precip_mm").getString() +
            "mm\nHumidity: " +
convertObj.get("current").getAsJsonObject().get("humidity").getString() +
            "%\nCondition: " +
convertObj.get("current").getAsJsonObject().get("condition").getAsJsonObject().get("text").getString());

        welcomeText.setStyle("-fx-text-fill: blue;");
        welcomeText.setStyle("-fx-font-size: 16px;");
        welcomeText.setWrapText(true);
        Image image = new Image("https:" +
convertObj.get("current").getAsJsonObject().get("condition").getAsJsonObject().get("icon").getString());
        weatherIcon.setImage(image);

        HostServices hostServices = getHostServices();

        //Open in google maps feature
        String googleMapsUrl =
"https://www.google.com/maps/search/?api=1&query=" + city_name;
        googleMaps.setText("Open " + city_name + " in Google
Maps");
        googleMaps.setOnMouseClicked(event ->
hostServices.showDocument(googleMapsUrl));

        //Open in Google
        String googleUrl = "https://www.google.com/search?q="
+ city_name;
        google.setText("Search " + city_name + " in Google");
        google.setOnMouseClicked(event ->
hostServices.showDocument(googleUrl));

        //Open in wikipedia
        String wikipediaUrl =
"https://en.wikipedia.org/wiki/" + city_name;
        wikipedia.setText("Search " + city_name + " in
Wikipedia");
        wikipedia.setOnMouseClicked(event ->
hostServices.showDocument(wikipediaUrl));

    }
}

public String getMeow() throws IOException {
    URL url = new URL("https://meowfacts.herokuapp.com/");
    HttpURLConnection con = (HttpURLConnection)

```

```

url.openConnection();
    con.setRequestMethod("GET");
    con.setRequestProperty("Content-Type", "application/json;
utf-8");

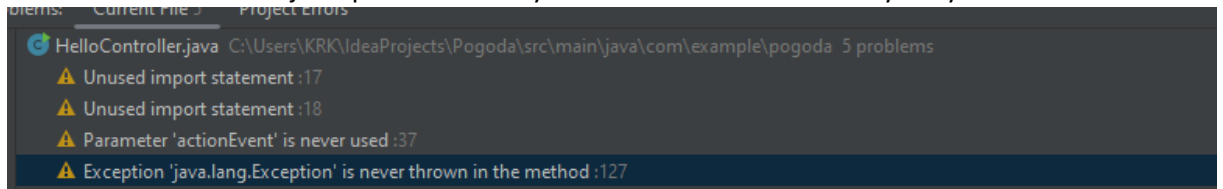
    int status = con.getResponseCode();
    if (status!=200) {
        System.out.println("Error");
        return "I can't get the meow facts right now";
    }
    else {
        Scanner sc = new Scanner(url.openStream());
        StringBuilder inline = new StringBuilder();
        while (sc.hasNext()) {
            inline.append(sc.nextLine());
        }
        sc.close();

        JsonObject jsonObject = new
Gson().fromJson(inline.toString(), JsonObject.class);
        return jsonObject.get("data").getAsString();
    }
}

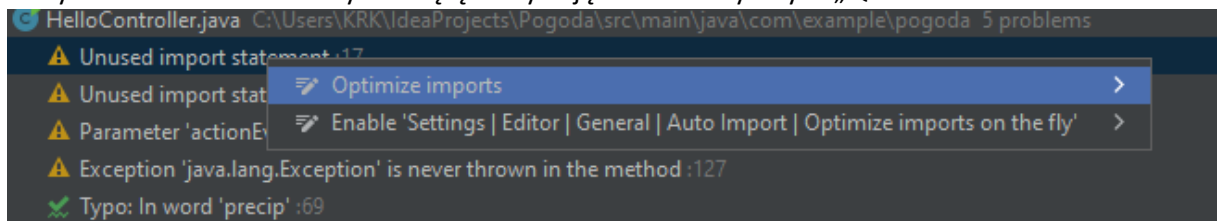
@Override
public void start(Stage stage) throws Exception {
}
}

```

12. Przedostatnim krokiem jest sprawdzenie i wyeliminowanie ostrzeżeń których było 4.



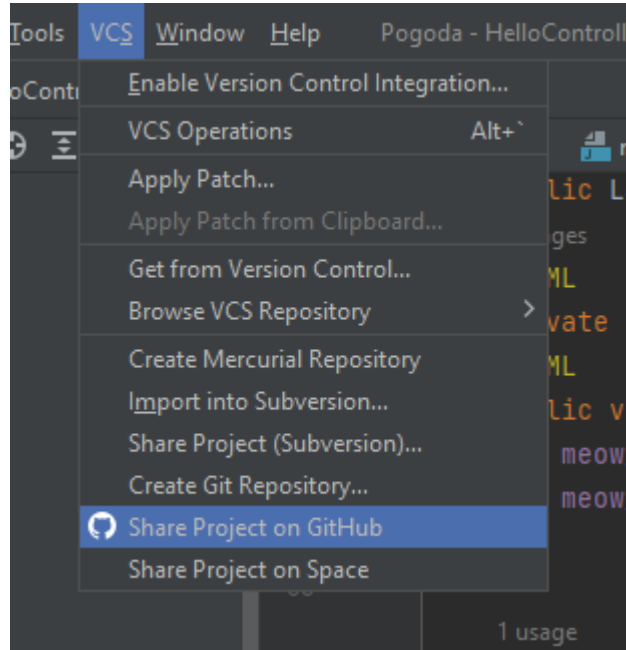
Wszystkie ostrzeżenia zostały usunięte korzystając z automatycznych „Qucik Fixes”.



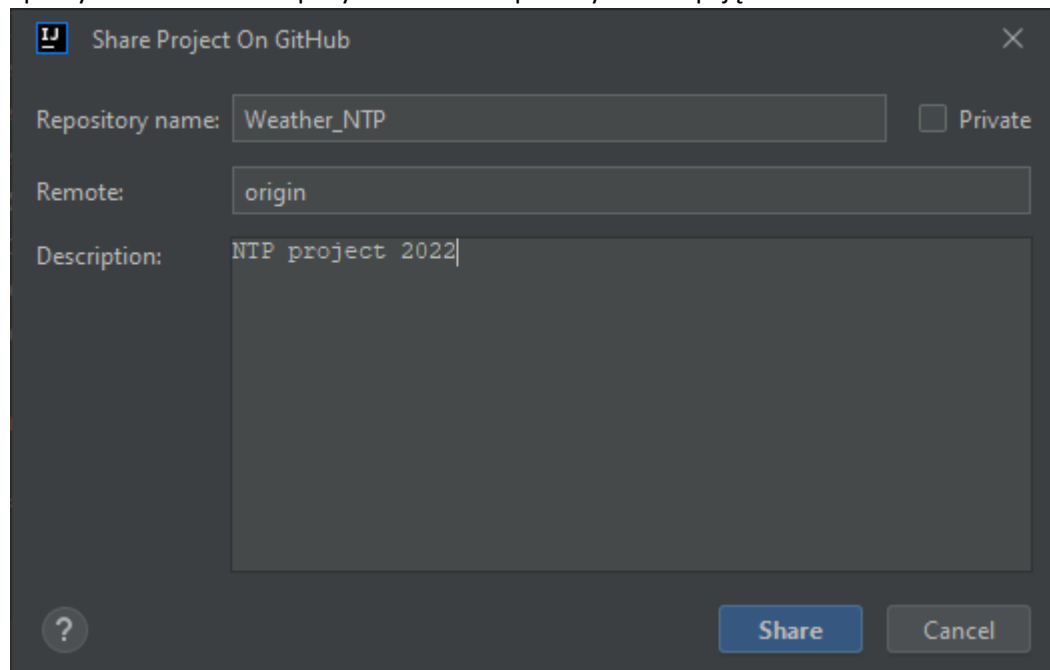
13. Ostatnim krokiem jest zapisanie projektu w systemie kontroli wersji.



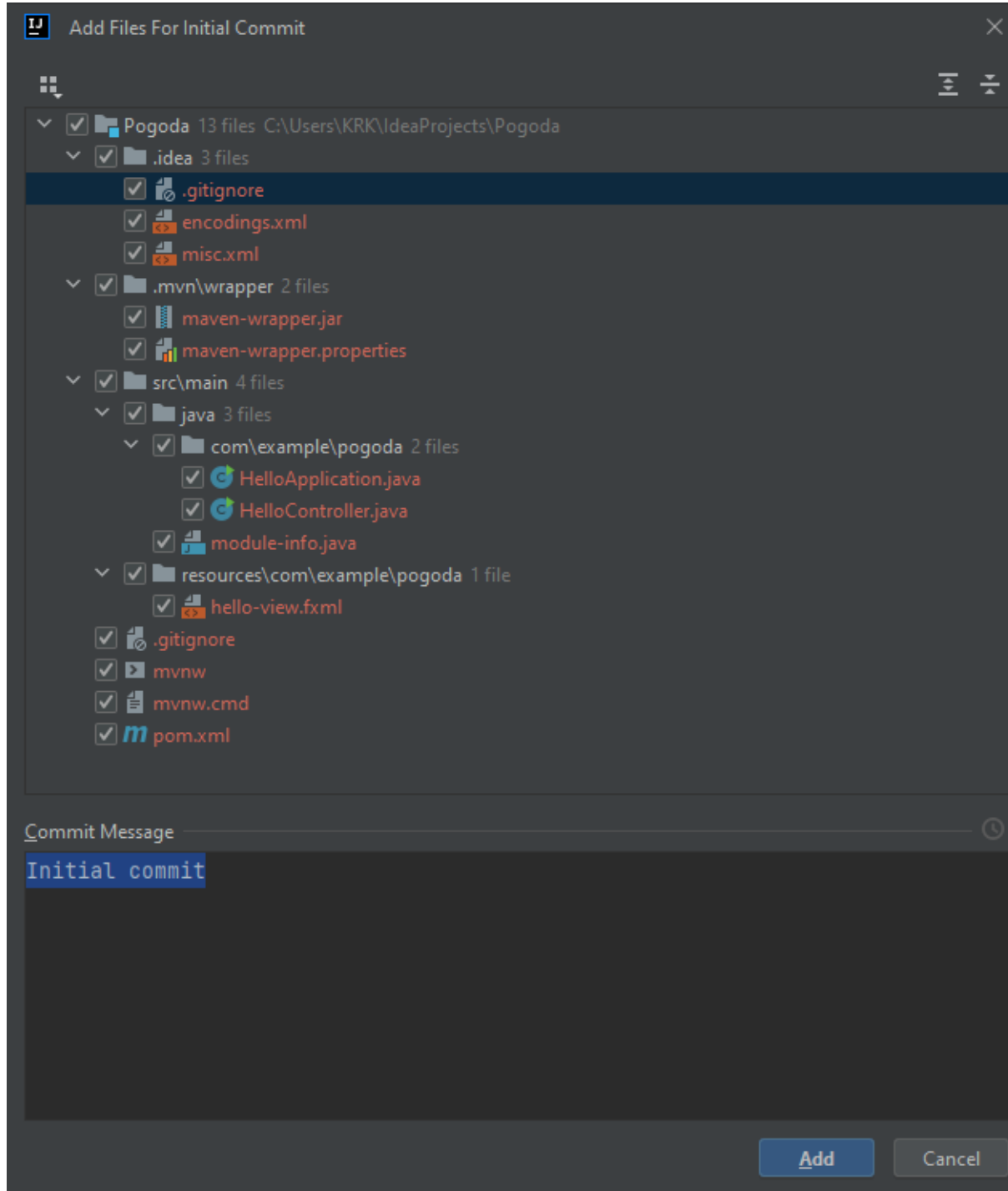
- a. Wybranie opcji Share Project on GitHub



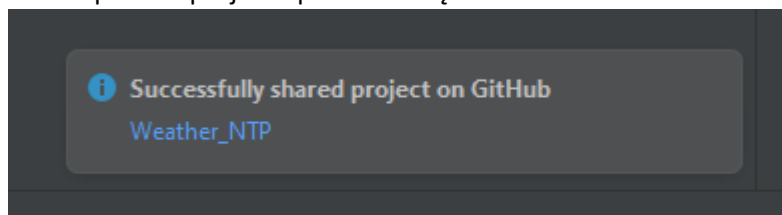
- b. Sprecyzowano nazwę repozytorium oraz opis i wybrano opcję Share



- c. Utworzono pierwszy Commit dodając wszystkie wytworzone pliki



- d. Udostępnienie projektu powiodło się



Link do kodu źródłowego: [https://github.com/KrystianGraba/Weather\\_NTP](https://github.com/KrystianGraba/Weather_NTP)