
# POLITECHNIKA LUBELSKA
# WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI

KIERUNEK STUDIÓW
INFORMATYKA

Przedmiot: Programowanie aplikacji w chmurze obliczeniowej

## Sprawozdanie – Zadanie 1

Autor:
Wiktoria Matacz
Gr.6.9

## CZĘŚĆ OBOWIĄZKOWA

1. Aplikacja została wykonana w języku Go. Po jej uruchomieniu w wierszu polecenia pokazują się logi z informacjami o dacie uruchomienia aplikacji, imieniu i nazwisku autora oraz numeru portu na której aplikacja działa. Aplikacja pokazuję aktualną pogodę na podstawie wybranego z listy kraju oraz miasta. Po kliknięciu ,,Sprawdź pogodę" użytkownik jest przenoszony na stronę, gdzie pojawiają się informację pogodowe wybranego miasta, a pod spodem znajduję się przycisk powrotu do formularza.

Zawartość app.go:

app.go X Dockerfile

Zadanie1 > app.go > obslugaPogody

```go
package main

import (
    "encoding/json"
    "fmt"
    "io"
    "log"
    "net/http"
    "net/url"
    "os"
    "time"
)

// Dane autora i port aplikacji
const autor = "Wiktoria Matacz"
const port = "8080"

// Mapa: pełna nazwa kraju => kod ISO
var kraje = map[string]string{
    "Polska": "PL",
    "Niemcy": "DE",
    "USA":    "US",
}

// Miasta dla każdego kraju (wg pełnej nazwy kraju)
var lokalizacje = map[string][]string{
    "Polska": {"Warszawa", "Kraków", "Gdańsk"},
    "Niemcy": {"Berlin", "Hamburg"},
    "USA":    {"Chicago", "San Francisco", "Los Angeles"},
}
```

```go
// Klucz API pobrany z zmiennej środowiskowej
var kluczAPI = os.Getenv("OPENWEATHER_API_KEY")

// Struktura danych pogodowych odbieranych z API
type WynikPogody struct {
    Main struct {
        Temperatura float64 `json:"temp"`
        Wilgotnosc  int     `json:"humidity"`
    } `json:"main"`
    Pogoda []struct {
        Stan  string `json:"main"`
        Opis  string `json:"description"`
        Ikona string `json:"icon"`
    } `json:"weather"`
}

// Funkcja główna
func main() {
    if kluczAPI == "" {
        log.Fatal("Brak klucza API! Ustaw zmienną środowiskową OPENWEATHER_API_KEY.")
    }

    log.Printf("Data uruchomienia: %s", time.Now().Format(time.RFC3339))
    log.Printf("Autor: %s", autor)
    log.Printf("Numer portu: %s", port)

    http.HandleFunc("/", obslugaStartowa)
    http.HandleFunc("/pogoda", obslugaPogody)

    log.Fatal(http.ListenAndServe(":"+port, nil))
}
```

```go
// Strona główna z formularzem wyboru kraju i miasta
func obslugaStartowa(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Content-Type", "text/html; charset=utf-8")

    // Serializacja mapy lokalizacji do JSON
    lokalizacjeJSON, err := json.Marshal(lokalizacje)
    if err != nil {
        http.Error(w, "Błąd danych", http.StatusInternalServerError)
        return
    }

    fmt.Fprint(w, `
        <!DOCTYPE html>
        <html lang="pl">
        <head>
            <meta charset="UTF-8">
            <title>Aplikacja Pogodowa</title>
            <style>
                body {
                    font-family: Arial, sans-serif;
                    background-color: #e0f2ff;
                    color: #003f5c;
                    text-align: center;
                }
                form {
                    margin-top: 30px;
                }
                .kafelek {
                    margin: 30px auto;
                    background: #cceeff;
                    padding: 20px;
                    border-radius: 10px;
                    width: 250px;
                    box-shadow: 2px 2px 6px rgba(0,0,0,0.2);
                }
                .kafelek img {
                    width: 100px;
                    height: 100px;
                }
            </style>
        </head>
        <body>
            <h1>Aplikacja Pogodowa</h1>
            <form action="/pogoda" method="get">
                Kraj:
                <select name="kraj" id="kraj" onchange="aktualizujMiasta()">
    `)

    // Generowanie opcji krajów
    for kraj := range lokalizacje {
        fmt.Fprintf(w, "<option value='%s'>%s</option>", kraj, kraj)
    }
```

```
    fmt.Fprint(w, `
                </select><br>
                Miasto:
                <select name="miasto" id="miasto"></select><br>
                <input type="submit" value="Sprawdź pogodę">
            </form>

            <script>
                const lokalizacje = `+string(lokalizacjeJSON)+`;

                function aktualizujMiasta() {
                    const kraj = document.getElementById("kraj").value;
                    const miastaSelect = document.getElementById("miasto");
                    miastaSelect.innerHTML = "";

                    if (lokalizacje[kraj]) {
                        lokalizacje[kraj].forEach(miasto => {
                            const option = document.createElement("option");
                            option.value = miasto;
                            option.text = miasto;
                            miastaSelect.appendChild(option);
                        });
                    }
                }

                //Wywołane przy załadowaniu strony, by ustawić miasta dla domyślnego kraju
                window.onload = aktualizujMiasta;
            </script>
        </body>
        </html>
    `)
}
```

```
// Obsługa zapytań pogodowych
func obslugaPogody(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Content-Type", "text/html; charset=utf-8")

    krajPelnaNazwa := r.URL.Query().Get("kraj")
    miasto := r.URL.Query().Get("miasto")

    if krajPelnaNazwa == "" || miasto == "" {
        http.Error(w, "Brakuje kraju lub miasta w zapytaniu", http.StatusBadRequest)
        return
    }

    kodKraju, ok := kraje[krajPelnaNazwa]
    if !ok {
        http.Error(w, "Nieznany kraj", http.StatusBadRequest)
        return
    }
```

```go
// Budowanie URL zapytania do API pogodowego
q := fmt.Sprintf("%s,%s", miasto, kodKraju)
url := fmt.Sprintf("https://api.openweathermap.org/data/2.5/weather?q=%s&appid=%s&units=metric&lang=pl",
    url.QueryEscape(q), kluczAPI)
resp, err := http.Get(url)
if err != nil {
    http.Error(w, fmt.Sprintf("Błąd połączenia z API: %v", err), http.StatusInternalServerError)
    return
}
defer resp.Body.Close()

if resp.StatusCode != 200 {
    body, _ := io.ReadAll(resp.Body)
    http.Error(w, fmt.Sprintf("API zwróciło błąd: %s", body), http.StatusBadRequest)
    return
}
defer resp.Body.Close()

// Dekodowanie odpowiedzi JSON
var wynik WynikPogody
if err := json.NewDecoder(resp.Body).Decode(&wynik); err != nil {
    http.Error(w, "Błąd dekodowania danych", http.StatusInternalServerError)
    return
}
```

```go
// Generowanie HTML wyniku
fmt.Fprint(w, `
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="UTF-8">
    <title>Wynik pogody</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #e0f2ff;
            color: #003f5c;
            text-align: center;
        }
        .kafelek {
            margin: 30px auto;
            background: #cceeff;
            padding: 20px;
            border-radius: 10px;
            width: 250px;
            box-shadow: 2px 2px 6px rgba(0,0,0,0.2);
        }
        .kafelek img {
            width: 100px;
            height: 100px;
        }
        a {
            display: block;
            margin-top: 20px;
        }
    </style>
</head>
```

```
    <body>
    `)

    fmt.Fprintf(w, `<div class="kafelek">
        <h2>%s, %s</h2>
        <img src="https://openweathermap.org/img/wn/%s@2x.png" alt="ikona pogodowa">
        <p><strong>Temperatura:</strong> %.1f°C</p>
        <p><strong>Wilgotność:</strong> %d%%</p>
        <p><strong>Stan:</strong> %s (%s)</p>
    </div>`,
        miasto, krajPelnaNazwa,
        wynik.Pogoda[0].Ikona,
        wynik.Main.Temperatura,
        wynik.Main.Wilgotnosc,
        wynik.Pogoda[0].Stan,
        wynik.Pogoda[0].Opis,
    )

    fmt.Fprint(w, `<a href="/">Wróć</a></body></html>`)
}
```

2. Poniżej pokazany jest Dockerfile z odpowiednimi komentarzami, który po uruchomieniu pozwala zbudować kontener z przedstawionej w punkcie 1 aplikacji.

Zawartość Dockerfile:

Zadanie1 > Dockerfile

```
# Etap 1: Budowanie

# Warstwa 1: Bazowy obraz buildowy – zawiera kompilator Go oraz Alpine Linux
FROM golang:1.21-alpine AS builder

# Warstwa 2: Ustawienia środowiska
# CGO_ENABLED=0 - wyłącza zależności C
# GOOS=linux, GOARCH=amd64 - targetujemy Linuksa 64-bit
ENV CGO_ENABLED=0 GOOS=linux GOARCH=amd64

# Warstwa 3: Informacja o autorze obrazu (standard OCI)
LABEL org.opencontainers.image.authors="Wiktoria Matacz"

# Warstwa 4: Katalog roboczy w kontenerze
WORKDIR /app

# Warstwa 5: Kopiujemy pliki zależności, by cache był lepiej wykorzystywany
COPY go.mod ./
RUN go mod download

# Warstwa 6: Kopiujemy cały projekt do kontenera
COPY . .

# Warstwa 7: Kompilacja aplikacji do pliku binarnego
RUN go build -o app_pogoda .
```

```
#Etap 2: Obraz

# Warstwa 8: Scratch – minimalny obraz.
FROM scratch

# Warstwa 9: Informacja o autorze (dla końcowego obrazu)
LABEL org.opencontainers.image.authors="Wiktoria Matacz"

# Warstwa 10: Kopiujemy skompilowaną binarkę z etapu `builder`
COPY --from=builder /app/app_pogoda /app_pogoda

# Warstwa 11: Dodajemy certyfikaty SSL (wymagane przez http.Get dla HTTPS)
COPY --from=builder /etc/ssl/certs/ca-certificates.crt /etc/ssl/certs/

# Warstwa 12: Otwieramy port aplikacji
EXPOSE 8080

# Warstwa 13: HEALTHCHECK sprawdza, czy aplikacja działa (ping na localhost:8080 co 30 sekund)
HEALTHCHECK --interval=30s --timeout=5s --start-period=5s --retries=3 \
  CMD wget --spider -q http://localhost:8080/ || exit 1

# Warstwa 14: ENTRYPOINT ustawia aplikację jako główny proces kontenera
ENTRYPOINT ["/app_pogoda"]
```

3. Wykorzystane polecenia:

a. zbudowania opracowanego obrazu kontenera,

```
PS C:\Users\Dell\Desktop\Semestr6\ProgAplChmuraObl\Zadanie1> docker build -t app_pogoda .
[+] Building 22.4s (14/14) FINISHED                                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                0.1s
 => => transferring dockerfile: 1.66kB                                                                              0.0s
 => [internal] load metadata for docker.io/library/golang:1.23.8-alpine                                             2.8s
 => [auth] library/golang:pull token for registry-1.docker.io                                                       0.0s
 => [internal] load .dockerignore                                                                                   0.0s
 => => transferring context: 2B                                                                                     0.0s
 => [builder 1/6] FROM docker.io/library/golang:1.23.8-alpine@sha256:b7486658b87d34ecf95125e5b97e8dfe86c21f712aa36fc0c702e5d  9.8s
 => => resolve docker.io/library/golang:1.23.8-alpine@sha256:b7486658b87d34ecf95125e5b97e8dfe86c21f712aa36fc0c702e5dc41dc63e  0.0s
 => => sha256:849d4d456ba942c2e931336145571e5763d6ad35ced95ecfd3c27f3c5c1b059e 126B / 126B                          0.4s
 => => sha256:70eabb1d9476e2a74ca18f982fa2cd1a722047e0ac01f746221f76c65893fe80 74.06MB / 74.06MB                    6.9s
 => => sha256:7a6ff827c8892e8204aa182819ee79f83a82a20872414e9165e425084cd97020 294.89kB / 294.89kB                  1.0s
 => => extracting sha256:7a6ff827c8892e8204aa182819ee79f83a82a20872414e9165e425084cd97020                            0.1s
 => => extracting sha256:70eabb1d9476e2a74ca18f982fa2cd1a722047e0ac01f746221f76c65893fe80                            2.8s
 => => extracting sha256:849d4d456ba942c2e931336145571e5763d6ad35ced95ecfd3c27f3c5c1b059e                            0.0s
 => => extracting sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1                            0.0s
 => [internal] load build context                                                                                   0.1s
 => => transferring context: 1.75kB                                                                                 0.0s
 => [builder 2/6] WORKDIR /app                                                                                      0.3s
 => [builder 3/6] COPY go.mod ./                                                                                    0.0s
 => [builder 4/6] RUN go mod download                                                                               0.4s
 => [builder 5/6] COPY . .                                                                                          0.1s
 => [builder 6/6] RUN go build -o app_pogoda .                                                                      8.0s
 => [stage-1 1/2] COPY --from=builder /app/app_pogoda /app_pogoda                                                   0.0s
 => [stage-1 2/2] COPY --from=builder /etc/ssl/certs/ca-certificates.crt /etc/ssl/certs/                           0.0s
 => exporting to image                                                                                              0.6s
 => => exporting layers                                                                                             0.5s
 => => exporting manifest sha256:98651ebd51e1da9e1e6595c25dfccb8a3ace3d56f17296166b4f57e5927f133b                   0.0s
 => => exporting config sha256:92f28d7715ce650b07475598e31320d142cf1b15bba641bd3dba9f4ba05d9996                     0.0s
 => => exporting attestation manifest sha256:bfaa4f1a7ad0b74ea74dd651e43d2707291f46cf7cbbb4a8caabbfb3625cdb14       0.0s
 => => exporting manifest list sha256:4ca2d7be700e05bab45431dda63dc16a969814ca83e0360c318290e416780687              0.0s
 => => naming to docker.io/library/app_pogoda:latest                                                                0.0s
 => => unpacking to docker.io/library/app_pogoda:latest                                                             0.1s
```

b. uruchomienia kontenera na podstawie zbudowanego obrazu,

```
PS C:\Users\Dell\Desktop\Semestr6\ProgAplChmuraObl\Zadanie1> docker run -d --name pogoda -p 8080:8080 -e OPENWEATHER_API_KEY=18a148
[redacted] app_pogoda
3ec45ab44bf35335810eb49d367b9ee7260f64942460e36328bfa6a8b846e2c4
```

API_KEY został częściowo zasłonięty ze względu bezpieczeństwa.

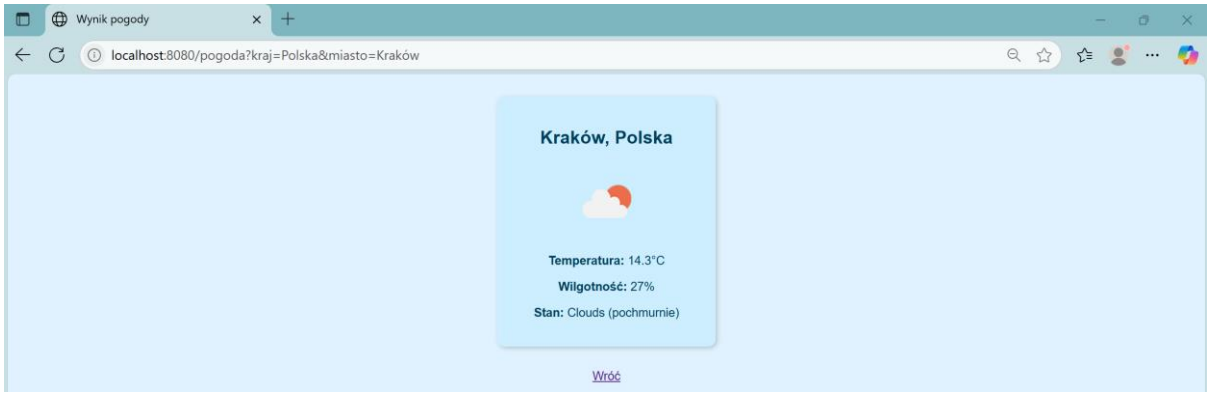
c. sposobu uzyskania informacji z logów, które wygenerowała opracowana aplikacja podczas uruchamiana kontenera (patrz: punkt 1a),

```
PS C:\Users\Dell\Desktop\Semestr6\ProgAplChmuraObl\Zadanie1> docker logs pogoda
2025/05/13 10:51:13 Data uruchomienia: 2025-05-13T10:51:13Z
2025/05/13 10:51:13 Autor: Wiktoria Matacz
2025/05/13 10:51:13 Numer portu: 8080
```

d. sprawdzenia, ile warstw posiada zbudowany obraz oraz jaki jest rozmiar obrazu.

```
PS C:\Users\Dell\Desktop\Semestr6\ProgAplChmuraObl\Zadanie1> docker image inspect app_pogoda --format='{{len .RootFS.Layers}} layer
s'
2 layers
PS C:\Users\Dell\Desktop\Semestr6\ProgAplChmuraObl\Zadanie1> docker image inspect app_pogoda --format='{{.Size}} bytes'
5027803 bytes
```

Widok aplikacji w przeglądarce:

## CZĘŚĆ NIEOBOWIĄZKOWA (DODATKOWA)

Obraz nie posiada zagrożeń na poziomie CRITICAL i HIGH:

```
PS C:\Users\Dell\Desktop\Semestr6\ProgAplChmuraObl\Zadanie1> docker scout quickview app_pogoda
    i New version 1.18.0 available (installed version is 1.16.1) at https://github.com/docker/scout-cli
    v SBOM of image already cached, 3 packages indexed

    i Base image was auto-detected. To get more accurate results, build images with max-mode provenance attestations.
      Review docs.docker.com ↗ for more information.

  Target      | app_pogoda:latest | 0C | 0H | 0M | 0L
    digest    | 4ca2d7be700e      |
```

```
PS C:\Users\Dell\Desktop\Semestr6\ProgAplChmuraObl\Zadanie1> docker scout cves app_pogoda
    i New version 1.18.0 available (installed version is 1.16.1) at https://github.com/docker/scout-cli
    v Image stored for indexing
    v Indexed 3 packages
    v No vulnerable package detected

## Overview

                    |        Analyzed Image
--------------------|---------------------
  Target            |  app_pogoda:latest
    digest          |  4ca2d7be700e
    platform        |  linux/amd64
    vulnerabilities |  0C  0H  0M  0L
    size            |  5.0 MB
    packages        |  3

## Packages and Vulnerabilities

  No vulnerable packages detected

PS C:\Users\Dell\Desktop\Semestr6\ProgAplChmuraObl\Zadanie1>
```

3.

Zawartość Dockerfile_dod:

Zadanie1 > Dockerfile_dod

```dockerfile
# Etap 1 - Budowanie aplikacji
# Wykorzystanie golang:alpine jako obrazu bazowego dla kompilacji
FROM --platform=$BUILDPLATFORM golang:1.23.8-alpine AS builder

LABEL org.opencontainers.image.authors="Wiktoria Matacz"
LABEL org.opencontainers.image.source="https://github.com/WikMat02/ProgApChZAD1"

WORKDIR /src

# Instalacja zależności - git oraz wymagane certyfikaty
RUN apk add --no-cache git ca-certificates && update-ca-certificates

# Wczytanie kodu bezpośrednio z GitHub przy użyciu mount=ssh i mount=secret
RUN --mount=type=ssh \
    --mount=type=secret,id=github_token \
    git clone https://github.com/WikMat02/ProgApChZAD1 . && \
    go mod download && \
    go build -o /out/app_pogoda .

# Etap 2 - Zbudowanie minimalnego obrazu
FROM scratch

# Kopiowanie skompilowanej aplikacji oraz certyfikatów SSL
COPY --from=builder /out/app_pogoda /app_pogoda
COPY --from=builder /etc/ssl/certs/ca-certificates.crt /etc/ssl/certs/

EXPOSE 8080

# Healthcheck do monitorowania stanu kontenera
HEALTHCHECK --interval=30s --timeout=5s --start-period=5s --retries=3 \
  CMD wget --spider -q http://localhost:8080/ || exit 1

ENTRYPOINT ["/app_pogoda"]
```

Potwierdzenie działania linux/arm64 oraz linux/amd64: