ДОДАТКИ

Додаток А

(Обов’язковий)

ПРОГРАМНИЙ КОД ОБРОБНИКА ЧЕРГИ

Лістинг 1. main.go

```
package telemetry

import (
	"context"
	"log"
	"encoding/json"
	"time"

	"cloud.google.com/go/bigquery"
)

var client *bigquery.Client

//Record represents database struct
type Record struct {
	DeviceId  string
	Timestamp string
	Data      string
}

//Save describes how struct should be saved to database
func (r *Record) Save() (map[string]bigquery.Value, string, error) {
	return map[string]bigquery.Value{
		"DeviceId":  r.DeviceId,
		"Timestamp": r.Timestamp,
		"Data":      r.Data,
	}, "", nil
}

//PubSubMessage represents Pub/Sub message struc
type PubSubMessage struct {
	Data       []byte `json:"data"`
	Attributes map[string]string
}

//init initialize BigQuery client
func init() {
	var err error
	client, err = bigquery.NewClient(context.Background(), "project-id")
	if err != nil {
		log.Fatalf("bigquery.NewClient: %v", err)
	}
}

//HandlTelemetry handle message from Pub/Sub
func HandleTelemetry(ctx context.Context, m PubSubMessage) error {
	var d string
```

```
	err := json.Unmarshal(m.Data, &d)
	if err != nil {
		log.Fatal(err)
	}
	log.Println(string(m.Data))
	inserter := client.Dataset("telemetry").Table("metrics").Inserter()
	r := []*Record{
		{
			DeviceId:  m.Attributes["deviceId"],
			Timestamp: time.Now().String(),
			Data:      d,
		},
	}

	if err := inserter.Put(context.Background(), r); err != nil {
		if multiError, ok := err.(bigquery.PutMultiError); ok {
			for _, err1 := range multiError {
				for _, err2 := range err1.Errors {
					log.Println("customerr")
					log.Println(err2)
				}
			}
		} else {
			log.Println("Here")
			log.Println(err)
		}
		return err
	}

	return nil
}
```

Лістинг 2. go.mod

```
module telemetry_handler

go 1.13

require cloud.google.com/go/bigquery v1.3.0
```

Додаток Б

(Обов’язковий)

ПРОГРАМНИЙ КОД МЕТЕОСТАНЦІЇ

Лістинг 1. meteostation.ino

```
#include "DHT.h"
#include <ArduinoJson.h>
#include <Http.h>

#define DHTTYPE DHT11

const int DHTPIN = A0;
const int RX_PIN = 8;
const int TX_PIN = 9;
const int RST_PIN = 10;

const char deviceId[30] = "deviceId";

HTTP http(9600, RX_PIN, TX_PIN, RST_PIN, true);
DHT dht(DHTPIN, DHTTYPE);
 //setup contains setup command and runs only ones
void setup() {
  delay(5000);
  dht.begin();
  initStation();
}
//loop is standard Arduino method which running in endless loop
void loop() {
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();
  sendMetrics(temperature, humidity);
  delay(500);
}
// sendMetrics sends metric to API endpoint
 void sendMetrics(float temperature,float humidity) {
  char address[90];
  char body[90];
  char response[90];
  Result result;
  http.configureBearer("internet");
  result = http.connect();
  sprintf(body, "{\"temperature\":&v, \"humidity\":&v}", temperature, humidity);
  sprintf(address, "your.api/metrics/&s", deviceId);
  http.post(address, body, response);
}
```

Додаток В

(Обов’язковий)

ПРОГРАМНИЙ КОД ВЕБ-СЕРВЕРУ

Лістинг 1. main.go

```
package main

import (
	"cloud.google.com/go/bigquery"
	"context"
	"encoding/json"
	"fmt"
	"github.com/go-chi/chi"
	"google.golang.org/api/iterator"
	"log"
	"net/http"
	"os"
)

//Metric represent metric table structure in BigQuery
type Metric struct {
	DeviceID  string `json:"deviceId"`
	Timestamp string `json:"timestamp"`
	Data      string `json:"data"`
}
var client *bigquery.Client

func main() {
	var err error
	client, err = bigquery.NewClient(context.Background(), "project-id")
	if err != nil {
		log.Fatal("Can't load bigquery client", err)
	}
	r := chi.NewRouter()
	r.Get("/device/{deviceID}/metrics", func(w http.ResponseWriter, r *http.Request) {
		deviceID := chi.URLParam(r, "deviceID")
```

```
                size := r.URL.Query()["size"]

                var metrics []Metric
                if len(size) > 0 {
                        metrics, err = findMetricsWithLimit(r.Context(), deviceID, size[0])
                } else {
                        metrics, err = findMetrics(r.Context(), deviceID)
                }
                if err != nil {
                        RespondWithError(w, 500, "Can't find metrics for device")
                        return
                }
                RespondWithJSON(w, 200, metrics)
        })
        port := os.Getenv("PORT")
        if port == "" {
                port = "8080"
        }
        log.Fatal(http.ListenAndServe(fmt.Sprintf(":%s", port), r))
}

//RespondWithJSON send response in JSON format
func RespondWithJSON(w http.ResponseWriter, code int, payload interface{}) {
        w.Header().Set("Content-Type", "application/json")
        response, err := json.Marshal(payload)
        if err != nil {
                w.WriteHeader(http.StatusInternalServerError)
                w.Write([]byte("Can't marshal response"))
        }
        w.WriteHeader(code)
        w.Write(response)
}

//RespondWithError send HTTP response with specifiend HTTP code and payload
//in JSON format
```

```
func RespondWithError(w http.ResponseWriter, code int, msg string) {
	RespondWithJSON(w, code, map[string]string{"message": msg})
}
// findMetricsWithLimit searches records for specified device and limit it
func findMetricsWithLimit(ctx context.Context, deviceID, limit string) ([]Metric, error) {
	query := client.Query(`SELECT * FROM telemetry.metrics WHERE deviceId = '` + deviceID + `' ORDER BY timestamp DESC LIMIT ` + limit + `;`)
	itr, err := query.Read(ctx)
	if err != nil {
		return nil, err
	}
	var metrics []Metric
	for {
		var metric Metric
		err := itr.Next(&metric)
		if err == iterator.Done {
			return metrics, nil
		}
		if err != nil {
			return nil, err
		}
		metrics = append(metrics, metric)
	}
}
// findMetrics searches records for specified device
func findMetrics(ctx context.Context, deviceID string) ([]Metric, error) {
	query := client.Query(`SELECT * FROM telemetry.metrics WHERE deviceId = '` + deviceID + `' ORDER BY timestamp DESC;`)
	itr, err := query.Read(ctx)
	if err != nil {
		return nil, err
	}
	var metrics []Metric
	for {
		var metric Metric
```

```
                err := itr.Next(&metric)
                if err == iterator.Done {
                        return metrics, nil
                }
                if err != nil {
                        return nil, err
                }
                metrics = append(metrics, metric)
        }
}
```

Лістинг 1.2 go.mod

```
module telemetry_api

go 1.13
require (
        cloud.google.com/go v0.49.0 // indirect
        cloud.google.com/go/bigquery v1.3.0
        github.com/go-chi/chi v4.0.2+incompatible
        github.com/golang/groupcache v0.0.0-20191027212112-611e8accdfc9 // indirect
        github.com/hashicorp/golang-lru v0.5.3 // indirect
        github.com/jstemmer/go-junit-report v0.9.1 // indirect
        go.opencensus.io v0.22.2 // indirect
        golang.org/x/exp v0.0.0-20191129062945-2f5052295587 // indirect
        golang.org/x/lint v0.0.0-20191125180803-fdd1cda4f05f // indirect
        golang.org/x/net v0.0.0-20191209160850-c0dbc17a3553 // indirect
        golang.org/x/oauth2 v0.0.0-20191202225959-858c2ad4c8b6 // indirect
        golang.org/x/sys v0.0.0-20191210023423-ac6580df4449 // indirect
        golang.org/x/tools v0.0.0-20191213221258-04c2e8eff935 // indirect
        google.golang.org/api v0.14.0
        google.golang.org/appengine v1.6.5 // indirect
        google.golang.org/genproto v0.0.0-20191206224255-0243a4be9c8f // indirect
        google.golang.org/grpc v1.25.1 // indirect
)
```

Лістинг 3. Dockerfile

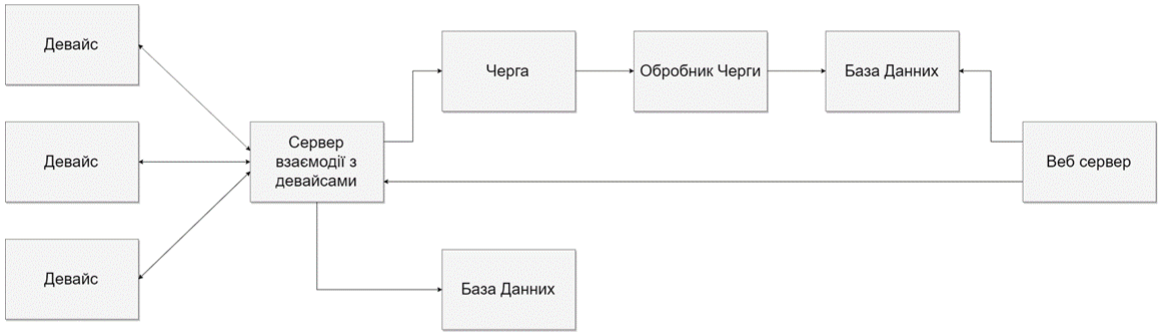
```
FROM golang:1.13 as builder
# Create and change to the app directory.
WORKDIR /app
# Retrieve application dependencies.
# This allows the container build to reuse cached dependencies.
COPY go.* ./
RUN go mod download
# Copy local code to the container image.
COPY . ./
# Build the binary.
RUN CGO_ENABLED=0 GOOS=linux go build -mod=readonly -v -o server
# Use the official Alpine image for a lean production container.
FROM alpine:3
RUN apk add --no-cache ca-certificates
# Copy the binary to the production image from the builder stage.
COPY --from=builder /app/server /server
# Run the web service on container startup.
CMD ["/server"]
```

Додаток Г

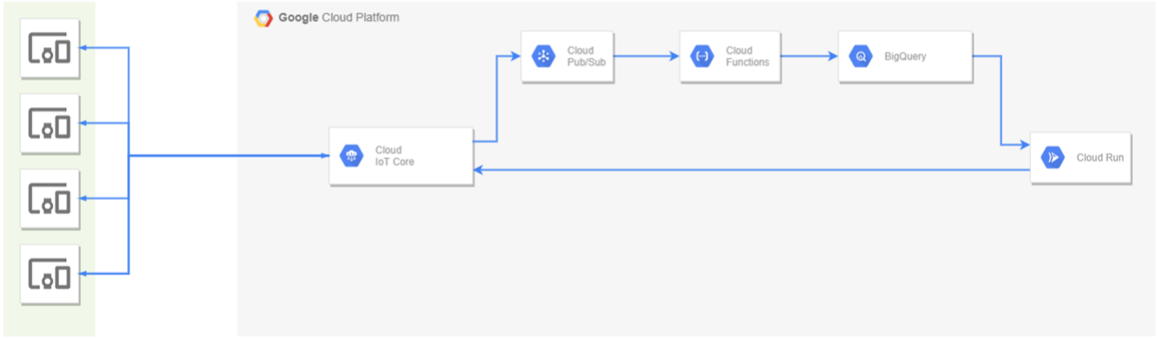
(Обов’язковий)

СЛАЙДИ ПРЕЗЕНТАЦІЇ

Структурная схема платформи



Структурная схема платформи на базі GCP(Google Cloud Platform)



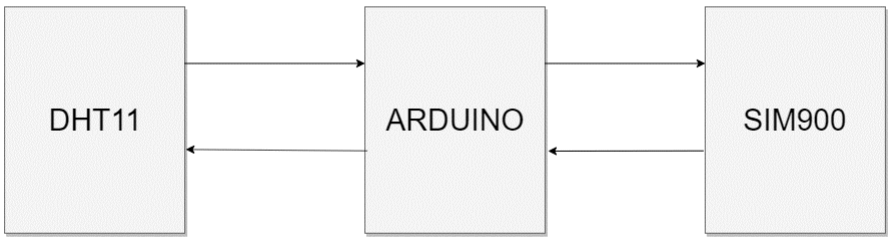
Головні компоненти метеостанції

Arduino Nano

SIM900

DHT11

Структурная схема метеостанції



Додаток Ґ

(Обов’язковий)

ВІДОМІСТЬ АТЕСТАЦІЙНОГО ПРОЕКТУ

№ п/п	Позначення	Найменування	Додаткові відомості
		Текстові документи	
1	ГЮІК. 467750.004 ПЗ	Пояснювальна записка	120 с.

Зм.	Арк.	№ докум.	Підп.	Дата	ГЮІК.809017.000Д1		
Розроб.		Болдиш М.І.			Розробка платформи дистанційного контролю параметрів віддалених об’єктів на прикладі метеостанції. Відомість атестаційної роботи	Літера	Аркуш	Аркушів
Перевір.		Бітченко О.М.				У	1	1
Т.контр.						ХНУРЕ Кафедра РТІКС		
Н.контр.		Зарудний О.А						
Затв.		Цопа О.І.						