OPIS APLIKACJI
STWORZONYCH W RAMACH PRACY DYPLOMOWEJ
WIELOSTREFOWEGO STEROWNIKA NAWADNIANIA

OPTI-SERV

Z BEZPRZEWODOWYMI CZUJNIKAMI ŚRODOWISKA

Wersja dokumentu: 1.0

1. Oprogramowanie w językach kompilowanych

Poniżej przedstawione parametry konfiguracyjne i informacje dotyczą oprogramowania:

- Sterownika nawadniania (1.1),
- Czujnika wilgotności gleby (1.2),
- Układu pomiarowego do badań czujników wilgotności gleby (1.3).

Platforma docelowa: ESP32-WROOM-32E

Język programowania: C++11

Zintegrowane środowisko programistyczne (IDE): PlatformIO 5.2.3 (dodatek do Visual Studio Code wersja 1.63.2)

Tabela 1: Parametry środowiska programistycznego i platformy sprzętowej

Lp	Nazwa	Wartość
1	Platform	espressif32
2	Board	esp32doit-devkit-v1
3	Framework	arduino
4	Hardware	ESP32 240MHz, 320KB RAM, 4MB Flash
5	Packages	- framework-arduinoespressif32 @ 3.20004.220825 (2.0.4) - tool-esptoolpy @ 1.30300.0 (3.3.0) - toolchain-xtensa-esp32 @ 8.4.0+2021r2-patch3
6	Library Dependency Finder Modes	Finder ~ chain, Compatibility ~ soft
7	Debug	cmsis-dap

Komenda zastosowana do zbudowania programu:

```
platformio run --environment esp32doit-devkit-v1
```

Komenda zastosowana do przesłania produktów kompilacji do platformy ESP32:

```
platformio run --target upload --environment esp32dev
```

Tabela 2: Schemat partycji pamięci flash zastosowany w ESP32

Nazwa	Typ	Podtyp	Offset	Rozmiar[hex]	Rozmiar[dec]	Flags
nvs	data	nvs	0x9000	0x5000	20.48KB	
otadata	data	ota	0xe000	0x2000	8.192KB	
app0	app	ota_0	0x10000	0x140000	1.31MB	
app1	app	ota_1	0x150000	0x140000	1.31MB	
spiffs	data	spiffs	0x290000	0x170000	1.51MB	

```
Dependency Graph
|-- AceButton @ 1.9.2
|-- DallasTemperature @ 3.11.0
|   |-- OneWire @ 2.3.7
|-- LoRa @ 0.8.0
|   |-- SPI @ 2.0.0
|-- Adafruit BME280 Library @ 2.2.2
|   |-- Adafruit BusIO @ 1.13.2
|   |   |-- Wire @ 2.0.0
|   |   |-- SPI @ 2.0.0
|   |-- Wire @ 2.0.0
|   |-- SPI @ 2.0.0
|   |-- Adafruit Unified Sensor @ 1.1.6
|-- ArduinoJson @ 6.19.4
|-- FS @ 2.0.0
|-- SD @ 2.0.0
|   |-- FS @ 2.0.0
|   |-- SPI @ 2.0.0
|-- mbedtls @ 2.23.0
|-- Adafruit Unified Sensor @ 1.1.6
|-- OneWire @ 2.3.7
|-- SPI @ 2.0.0
|-- SPIFFS @ 2.0.0
|   |-- FS @ 2.0.0
|-- Wire @ 2.0.0
```

Rysunek 1: Graf zależności bibliotek programistycznych czujnika wilgotności

```
Dependency Graph
|-- LiquidCrystal_I2C @ 1.1.4
|   |-- Wire @ 2.0.0
|-- HX711 @ 0.7.5
|-- HX711_ADC @ 1.2.12
|-- OneWire @ 2.3.7
|-- DallasTemperature @ 3.10.0
|   |-- OneWire @ 2.3.7
|-- RTClib @ 2.0.3
|   |-- Adafruit BusIO @ 1.9.6
|   |   |-- Wire @ 2.0.0
|   |   |-- SPI @ 2.0.0
|   |-- Wire @ 2.0.0
|-- FS @ 2.0.0
|-- SD @ 2.0.0
|   |-- FS @ 2.0.0
|   |-- SPI @ 2.0.0
|-- SPI @ 2.0.0
|-- HTTPClient @ 2.0.0
|   |-- WiFi @ 2.0.0
|   |-- WiFiClientSecure @ 2.0.0
|   |   |-- WiFi @ 2.0.0
|-- WiFi @ 2.0.0
|-- Wire @ 2.0.0
```

Rysunek 2: Graf zależności bibliotek programistycznych układu pomiarowego do badania czujników wilgotności gleby

```
Dependency Graph
|-- LiquidCrystal_I2C @ 1.1.4
|   |-- Wire @ 2.0.0
|-- Adafruit AHTX0 @ 2.0.3
|   |-- Adafruit BusIO @ 1.11.6
|   |   |-- Wire @ 2.0.0
|   |   |-- SPI @ 2.0.0
|   |-- Wire @ 2.0.0
|   |-- SPI @ 2.0.0
|   |-- Adafruit Unified Sensor @ 1.1.5
|-- MCP23017 @ 0.2.6+sha.5fcdf39
|   |-- Wire @ 2.0.0
|-- LoRa @ 0.8.0
|   |-- SPI @ 2.0.0
|-- ArduinoJson @ 6.19.4
|-- ConfigManager @ 2.1.1
|   |-- ArduinoJson @ 6.19.4
|   |-- DNSServer @ 2.0.0
|   |   |-- WiFi @ 2.0.0
|   |-- EEPROM @ 2.0.0
|   |-- FS @ 2.0.0
|   |-- SPIFFS @ 2.0.0
|   |   |-- FS @ 2.0.0
|   |-- WebServer @ 2.0.0
|   |   |-- WiFi @ 2.0.0
|   |   |-- FS @ 2.0.0
|   |-- WiFi @ 2.0.0
|-- RTC @ 2.3.5
|-- RotaryEncoder @ 1.5.3
|-- AceButton @ 1.9.2
|-- ESPAsyncWebServer-esphome @ 2.1.0
|   |-- AsyncTCP-esphome @ 1.2.2
|   |-- ArduinoJson @ 6.19.4
|   |-- FS @ 2.0.0
|   |-- WiFi @ 2.0.0
|-- Time @ 1.6.1
|-- NTPClient @ 3.2.1
|-- AceSorting @ 1.0.0
|-- MQTT @ 2.5.0
|-- Wire @ 2.0.0
|-- WiFi @ 2.0.0
|-- WiFiClientSecure @ 2.0.0
|   |-- WiFi @ 2.0.0
|-- FS @ 2.0.0
|-- SD @ 2.0.0
|   |-- FS @ 2.0.0
|   |-- SPI @ 2.0.0
|-- SPI @ 2.0.0
|-- SPIFFS @ 2.0.0
|   |-- FS @ 2.0.0
|-- EEPROM @ 2.0.0
```

Rysunek 3: Graf zależności bibliotek programistycznych sterownika nawadniania

1.1 Sterownik nawadniania

Opis aplikacji: Oprogramowanie układowe sterownika ma za zadanie realizację algorytmu Optiserv oraz umożliwić manualne sterowanie nawadnianiem, obsługę interfejsu bezprzewodowego do wykorzystania przez aplikację mobilną, możliwość trwałej zmiany konfiguracji w tym dodawaniem zaufanych czujników bezprzewodowych, monitorowanie przepływu w instalacji nawodnieniowej, wykrywanie ruchu w pomieszczeniu po zazbrojeniu alarmu, wysyłkę informacji o zdarzeniach niepożądanych przez protokół MQTT.

Wymagania funkcjonalne do oprogramowania sterownika nawadniania zostały przestawione w podrozdziale 3.2 pt *„Zbieranie, analiza i specyfikacja wymagań projektu”* i oznaczone literą B.

Struktura oprogramowania, algorytmy. Praca dyplomowa: diagram przypadków użycia (rys. 38), diagram sekwencji wymiany komunikatów sterownika i czujnika (rys. 39), odpowiedź na raport środowiskowy (procedura nr 3 i 4), organizacja struktur danych w pamięci sterownika (rys. 41), definicja klasy na przykładzie `WeatherForecast` (procedura 5), struktura kodu źródłowego (tabela nr 36).

Interfejsy: Strukturę węzłów końcowych interfejsu bezprzewodowego dostępnego przez protokół HTTP przedstawiono w tabeli nr 16 pracy dyplomowej. Drugi interfejs bezprzewodowy jest udostępniony przy wykorzystaniu protokołu Lora i obejmuje wymianę komunikatów o strukturze z rys. 40 z pracy dyplomowej.

Możliwości rozwoju oprogramowania sterownika zostały przedstawione w załączniku B do pracy dyplomowej pt. „*Wskazówki dla projektanta i do dalszego rozwoju systemu sterownika nawaniania*”.

Elementy graficzne: projekt własny, ikony 5x8 pixeli

1.2 Czujnik wilgotności gleby

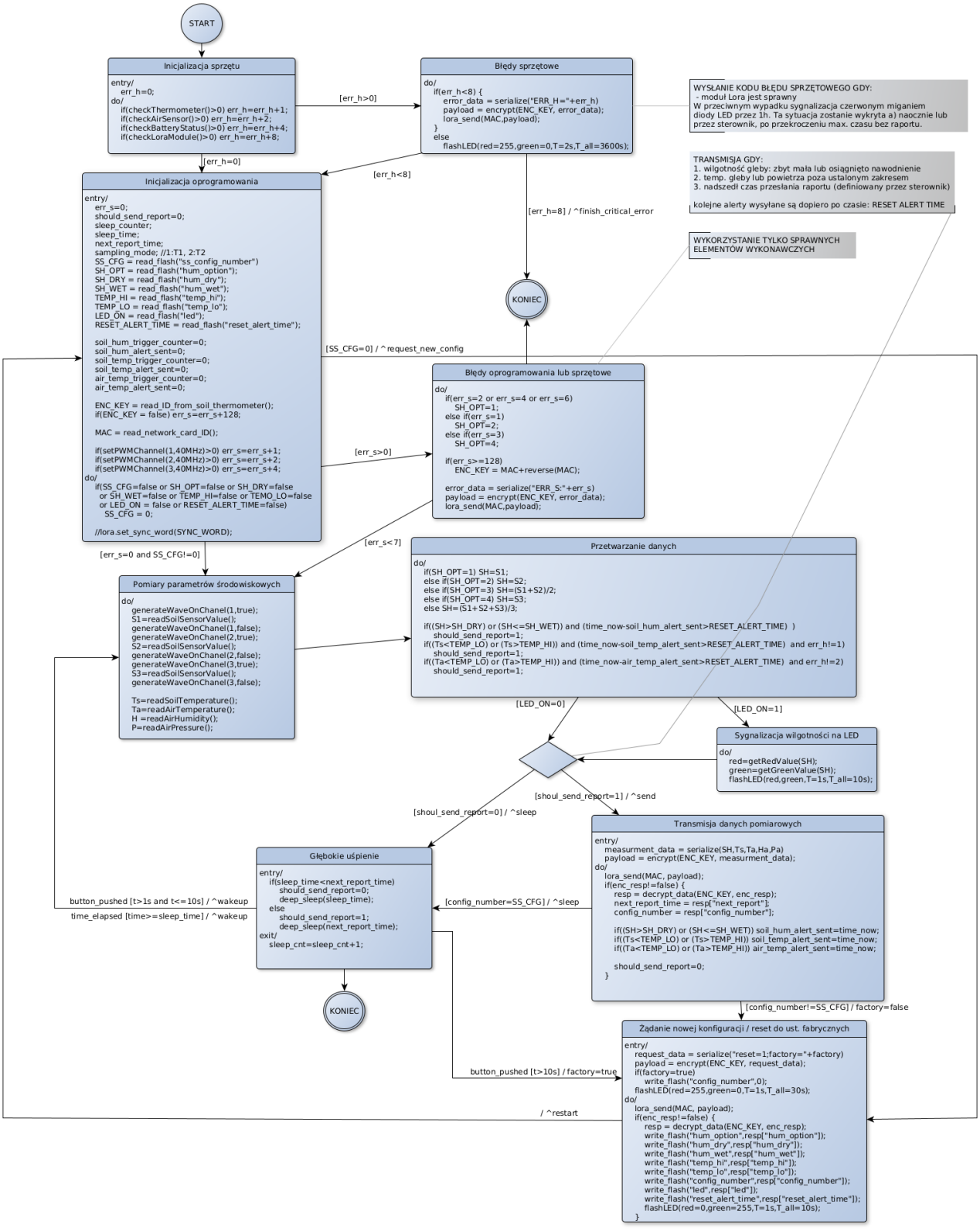
Opis aplikacji: Oprogramowanie układowe czujnika ma za zadanie przeprowadzać pomiary wilgotności i temperatury gleby i powietrza co minutę, a przy zdefiniowanych warunkach wysyłać je do sterownika przez protokół Lora. Sygnalizacja aktualnej wilgotności ma się odbywać przy wykorzystaniu diod LED.

Wymagania funkcjonalne do oprogramowania sterownika nawadniania zostały przestawione w podrozdziale 3.2 pt *„Zbieranie, analiza i specyfikacja wymagań projektu”* i oznaczone literą C.

Struktura oprogramowania, algorytmy. Diagram maszyny stanów przedstawiono na poniższym rysunku nr 4. Praca dyplomowa: diagram przypadków użycia (rys. 37), organizacja struktur danych w pamięci czujnika (rys. 42), struktura kodu źródłowego (tabela nr 37).

Interfejsy: Interfejs bezprzewodowy jest udostępniony przy wykorzystaniu protokołu Lora i obejmuje wymianę komunikatów o strukturze z rys. 40 z pracy dyplomowej.

Możliwości rozwoju systemu zostały przedstawione w załączniku B do pracy dyplomowej pt. „Wskazówki dla projektanta i do dalszego rozwoju systemu sterownika nawaniania”.



Rysunek 4: Diagram maszyny stanowej czujnika wilgotności

1.3 Układ pomiarowy do badań referencyjnych czujników i elementów wykonawczych projektowanego czujnika wilgotności gleby

Opis aplikacji: Przeprowadzanie badań czujników wilgotności gleby poprzez jej pomiary 4 czujnikami i zestawieniem z ubytkiem masy wody w wyniku parowania, zapis wyników na kartę pamięci i do zewnętrznej bazy danych

Wymagania funkcjonalne:

1. Możliwość obsługi do 4 czujników gleby, których wynik wystawiany jest jako sygnał napięciowy <0;3.3V>,
2. Zapis wyników pomiarów, daty i godziny, masy układu i temperatury gleby:
   - do pliku tekstowego na karcie pamięci,
   - do zewnętrznej bazy danych wykorzystując bezprzewodową sieć komputerową w standardzie WiFi,
3. Wyświetlanie godziny, masy układu oraz wyników pomiarów lokalnie na ekranie LCD i ich aktualizacja po każdej serii pomiarowej,
4. Zachowywanie daty i czasu w przypadku utraty zasilania,
5. Możliwość pomiaru masy układu pomiarowego do 5kg.

Struktura oprogramowania, algorytmy. Diagram maszyny stanów: przedstawiono na poniższym rysunku nr 4. Strukturę danych zapisywanych na karcie pamięci wraz z objaśnieniem przestawiono w pracy dyplomowej w tabeli nr 25.

Możliwości rozwoju systemu obejmują w zależności od potrzeb:

- zwiększenie ilości wykorzystanych czujników wilgotności do 8szt,
- wielopunktowy pomiar temperatury, do 8 termometrów cyfrowych,
- sygnalizację błędów za pomocą dodatkowego przetwornika piezoelektrycznego,
- sygnalizację błędów poprzez wysyłkę wiadomości e-mail na określony adres e-mail,
- stworzenie trybu konfiguracyjnego do trwałego zapisu danych do logowania do sieci bezprzewodowej i innych, parametrów np. okresu pomiaru (w wersji użytej w badaniach zmiany parametrów wymagały ponownego zbudowania aplikacji i wgrania na platformę sprzętową ESP32)

1.4 Aplikacja mobilna systemu Android

Platforma docelowa: Urządzenia mobilne z systemem Android w wersji 5 lub wyższej

Język programowania: Java, Javascript

Zintegrowane środowisko programistyczne (IDE): Android Studio 2021.2.1

Opis aplikacji: Bezprzewodowy klient sterownika nawadniania umożliwiający manualne sterowanie nawadnianiem w strefach i podgląd stanu, zmianę parametrów konfiguracyjnych sterownika i stref, dodawanie i edycję czujników wilgotności, podgląd historii zdarzeń, włącznie/wyłączanie alarmu, podgląd stanu instalacji nawodnieniowej

Wymagania funkcjonalne do oprogramowania sterownika nawadniania zostały przestawione w podrozdziale 3.2 pt *„Zbieranie, analiza i specyfikacja wymagań projektu”* i oznaczone literą D.

Opis modułów programowych: Aplikacja mobilna jest oparta o komponent WebView, do którego wczytywana jest aplikacja internetowa (szablon HTML z kodem Javasript i arkuszami stylów CSS) przechowywana i udostępniania przez sterownik nawadniania. Na właściwą aplikację mobilną zaimplementowaną w Java składają się 4 pliki realizujące funkcjonalności opisane w poniższej tabeli nr 3. Natomiast strukturę plików aplikacji internetowej uwidoczniono na rysunku nr 5.

Rysunek 5: Struktura plików aplikacji internetowej wyświetlanej osadzonej w komponencie WebView pakietu Android

Tabela 3: Struktura kodu aplikacji mobilnej

Nazwa	Funkcjonalność
MainActivity.java	Obsługa cyklu życia aplikacji, inicjalizacja aktywności, obsługa metod `onPause()`, `OnResume()` i reakcji na zmianę położenia ekranu
DBHelper.java	Klasa pomocnicza do obsługi bazy danych SQLite przechowujących informacje o nazwie ogrodu, adresie IP sterownika i haśle
Garden.java	Wyświetlanie i walidacja formularza z nazwą ogrodu, adresie IP i haśle do sterownika
WebActivity.java	Wyświetlanie komponentu WebView i ładowanie strony www z aplikacją internetową ze sterownika. Obsługa: błędów sieci, kodów odpowiedzi HTTP, żądania autoryzacji oraz realizacja zadań po pomyślnym wczytaniu strony. Udostępnienie interfejsu Javacript/Java do wywoływania natywnych komunikatów Android z poziomu aplikacji internetowej.

Struktura oprogramowania została opisana w pracy dyplomowej w 6,7, i 8 akapicie podrozdziału 5.2.2. pt. „*Implementacja aplikacji mobilnej do zarządzania sterownikiem*”

Możliwości rozwoju systemu:

- obsługa wielu sterowników, przy wykorzystaniu już zaprojektowanej bazy danych,
- stworzenie aplikacji na urządzenia Apple

Elementy graficzne: źródłem ikon w aplikacji jest strona https://freeicons.io/

2. Oprogramowanie w językach interpretowanych

2.1 Generator symulacji Opti-Serv z przeglądarką symulacji

Architektura sprzętowa: 64bit

Język programowania: JavaScript (*client-side* i *server-side*), PHP

Środowisko programistyczne: Edytor kodu *Sublime*

Opis aplikacji: aplikacja do przeprowadzania symulacji systemu nawadniania sterowanego zgodnie z algorytmem Optiserv, przeprowadzająca po 10 symulacji dla każdej z 8 grup, w której określone są inne parametry nawadniania. Przebieg symulacji wyświetlany jest w dedykowanej przeglądarce.

Wymagania funkcjonalne:

- generowanie symulacji sytuacji wodnej gleby od pierwszego dnia wiosny wiosny do ostatniego dnia lata,
- obliczanie punktów strefowych dotyczących kolejności nawadniania zgodnie z algorytmem Optiserv,
- uwzględnianie zmiennej długości dnia,
- obliczanie statystyk dla każdej strefy dotyczących okresu nawadniania,
- wizualizacja wyników w postaci harmonogramu nawadniania oraz zmiany punktów strefowych w czasie, umożliwiająca analizę kolejności uruchamianych stref,
- wizualizacja wyświetlana w przeglądarce internetowej,
- możliwość wyboru numeru symulacji (1-80), numeru dnia (1-180), zakresu dni (1-15), pojedynczej strefy bądź wszystkich w wizualizacji harmonogramu
- prezentacja statystyk dla stref dla symulacji w obrębie grup symulacyjnych,
- agregacja danych w obrębie grup symulacyjnych i generowanie podsumowania symulacji w odniesieniu do zadanych parametrów strefowych, zawierające ich porównanie procentowe

Opis modułów programowych:

Tabela 4: Struktura kodu aplikacji mobilnej

Nazwa modułu	**Funkcjonalność**
def.js	definicja 6 stref nawadniania w oparciu klasę Zone (rys. 31, praca dyplomowa)
app.js	moduł generujący wyniki symulacji, z każdej symulacji powstaje plik .sql z kwerendami do bazy MySQL
stat.js	obliczanie statystyk (min., max., średnia, mediana, odchylenie standardowe) m. in. okresu i długości nawadniania dla każdej stref przy wykorzystaniu funkcji statystycznych MySQL
graph.php	prezentacja graficznej historii nawadniania dla wybranej symulacji
symulacje.php	definicja symboli, opis symulacji, kryteria oceny rozwiązań oraz podsumowanie wyników w odniesieniu do grupy symulacji (załącznik F. do pracy dyplomowej)

Komenda zastosowana do wywołania generatora symulacji

```
node app.js
```

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	NULL	auto_increment
epoch	int(11)	NO		NULL	
zone_nr	int(11)	NO		NULL	
T	float	NO		NULL	
ZP	float	NO		NULL	
procT	float	NO		NULL	
procSH	float	NO		NULL	
last_irr	int(11)	NO		NULL	
SH	int(6)	NO		NULL	

*Rysunek 6: Struktura tabeli **sensor_data***

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
epoch	int(11)	NO		NULL	
D	int(11)	NO		NULL	
sunrise	float	NO		NULL	
sunset	float	NO		NULL	
sunset_T	float	NO		NULL	

*Rysunek 7: Struktura tabeli **sun***

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
epoch	int(11)	NO		NULL	
zone_nr	int(11)	NO		NULL	
T	float	NO		NULL	
PERIOD	float	NO		NULL	
is_sensorless	varchar(1)	NO		NULL	
PERIOD_H	int(11)	NO		NULL	
N_PARALLEL_ZONES	tinyint(4)	NO		NULL	
SUNSET_OFFSET	float	NO		NULL	
SH0	smallint(6)	NO		NULL	
avg	float	NO		NULL	
median	float	NO		NULL	
std	float	NO		NULL	
min	float	NO		NULL	
max	float	NO		NULL	
proc_above	float	NO		NULL	
sum_above	float	NO		NULL	
sum_proc	float	NO		NULL	

*Rysunek 8: Struktura tabeli **stat***

Na rysunkach 6, 7 oraz 8 przedstawiono struktury tabel wykorzystanych do przechowywania wyników symulacji (*sensor_data*), wschodów i zachodów słońca (*sun*) oraz do obliczonych statystyk (*stat*).

Możliwości rozwoju systemu:

- automatyczne wywoływanie sekwencji zadań (aktualnie po zdefiniowaniu stref w grupie symulacyjnej, następuje ręczny import do bazy danych, a następnie ręczne wywołanie modułu stat.js – zgodnie z rys. 32 z pracy dyplomowej)

Spis treści