ДОДАТОК А

Код програми

Серверний додаток для основного функціонального навантаження мовою TypeScript:

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from '@modules/app.module';
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  await app.listen(process.env.PORT);
}
bootstrap();

export const t0 = 273;
export const alpha = 0.05;

export const controllerParams = {
  vOut: 72,
};
export const inverterParams = {
  p: 1500,
};
export const chargerParams = {
  u: 12,
};
export const panelsParams = {
  count: 2,
  pOne: 400,
};

import { alpha, t0 } from '@constants/ physics'
export const getTCelsius = (tKelvin: number) => tKelvin - t0;
export const getPTerm = (pNom: number, tCelsius: number) =>
  pNom * (1 - alpha * (tCelsius - 20));
```

```
export const getPEf = (pTerm: number, cCloud: number) =>
  pTerm * (1 - cCloud / 100);
export const getIChange = (pEf: number, vOut: number) => pEf / vOut;
export const getCForecast = (iChangeArr: number[], deltaTArr: number[]) =>
  iChangeArr.reduce(
    (sum, item, currentIndex) => sum + item * deltaTArr[currentIndex],
    0,
  );
export const getIConsumer = (pCurrentInverter: number, uBattery: number) =>
  pCurrentInverter / uBattery;
export const getTWork = (c: number, iConsumer: number) => c / iConsumer;

export const getDateAsTimeString = (date: Date): string => {
  const day = String(date.getDate()).padStart(2, '0');
  const month = String(date.getMonth() + 1).padStart(2, '0');
  const year = date.getFullYear();
  const hours = String(date.getHours()).padStart(2, '0');
  const minutes = String(date.getMinutes()).padStart(2, '0');
  const seconds = String(date.getSeconds()).padStart(2, '0');
  return `${year}-${month}-${day} ${hours}:${minutes}:${seconds}`;
};
export const getCurrentDateAsTimeString = (): string => {
  const now = new Date();
  return getDateAsTimeString(now);
};
export const delay = (ms: number) =>
  new Promise((resolve) => setTimeout(resolve, ms));
const minutesInHour = 60;
const secondsInMinute = 60;
export const convertMillisecondsToHours = (milliseconds: number) =>
  milliseconds / minutesInHour / secondsInMinute;
export const twentyFourHoursInMs = 86400;

export const chatRoutes = {
  start: '/start',
```

```
  help: '/help',
  history: '/history',
  historyYesterday: '/history-y',
  historyClear: '/history-clear',
  on: '/on',
  off: '/off',
  forecastTomorrow: '/forecast-tomorrow',
} as const;
export const startText =
  'Доброго дня,\n' +
  'Це система контролю електроенергії. \n' +
  'Введіть "/help" для отримання даних про функціонал додатку';
export const helpText =
  '*/on* - це маршрут для включення електроенергії без графіка відключення; \n' +
  '*/off* - це маршрут для виключення електроенергії без графіка відключення; \n' +
  '*/history* - це маршрут для отримання історії подій; \n' +
  '*/history-y* - це маршрут для отримання історії подій за вчорашній день; \n' +
  '*/history-clear* - це маршрут для видалення даних історії подій; \n' +
  '*/forecast-tomorrow* - це маршрут для отримання прогнозного значення отримуваної електроенергетики та максимального часу роботи; \n' +
  '*/[00]-[00]* - це шаблонний маршрут для змини графіку відключень електроенергії.Кожне число позначає кількість годин. Перше число відповідає за кількість годин, в які є електрика,а друге - на період часу, коли вона буде відключеною. Час встановлюється на добу, тому сума чисел повинна дорівнювати 24 і жодне з них не повинно дорівнювати 0.Приклади правильного маршруту: /12-12, /9-15, /16-8.';

import { forwardRef, Module } from '@nestjs/common';
import { ChatService } from './chat.service';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { TelegramModule } from '@modules/telegram/telegram.module';
import { MqttServerModule } from '@modules/mqttServer/mqttServer.module';
import { ChatEventModule } from '@modules/chatEvent/chatEvent.module';
import { SequelizeModule } from '@nestjs/sequelize';
import { ChatEvent } from '@modules/chatEvent/chatEvent.entity';
import { ChatEventService } from '@modules/chatEvent/chatEvent.service';
```

```
import { WhatsappModule } from '@modules/whatsapp/whatsapp.module';
import { ElectricityForecastModule } from '@modules/electricityForecast/electricityForecast.module';
import { ElectricityForecastService } from '@modules/electricityForecast/electricityForecast.service';
@Module({
 imports: [
  SequelizeModule.forFeature([ChatEvent]),
  forwardRef(() => TelegramModule),
  forwardRef(() => WhatsappModule),
  forwardRef(() => MqttServerModule),
  forwardRef(() => ElectricityForecastModule),
  ChatEventModule,
  ConfigModule,
 ],
 providers: [ConfigService, ChatService, ChatEventService],
 exports: [ChatService],
})
export class ChatModule {}

import { forwardRef, Inject, Injectable } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { TelegramService } from '@modules/telegram/telegram.service';
import { MqttServerService } from '@modules/mqttServer/mqttServer.service';
import { ChatEventService } from '@modules/chatEvent/chatEvent.service';
import { WhatsappService } from '@modules/whatsapp/whatsapp.service';
import { chatRoutes, helpText, startText } from '@modules/chat/chat.constants';
import { ElectricityForecastService } from '@modules/electricityForecast/electricityForecast.service';
@Injectable()
export class ChatService {
 public readonly messengers: string[] = [];
 constructor(
  private readonly configService: ConfigService,
  @Inject(ChatEventService)
```

```
  private readonly chatEventService: ChatEventService,
  @Inject(forwardRef(() => TelegramService))
  private readonly telegramChatService: TelegramService,
  @Inject(forwardRef(() => WhatsappService))
  private readonly whatsappChatService: WhatsappService,
  @Inject(forwardRef(() => MqttServerService))
  private readonly mqttServerService: MqttServerService,
  @Inject(forwardRef(() => ElectricityForecastService))
  private readonly electricityForecastService: ElectricityForecastService,
) {
  this.messengers = this.configService.get<string>('MESSENGERS').split('|');
}
private readonly messageSenders = {
  telegram: this.telegramChatService.sendMessageToTelegram.bind(
    this.telegramChatService,
  ),
  whatsapp: this.whatsappChatService.sendMessageToWhatsapp.bind(
    this.whatsappChatService,
  ),
};
private readonly senderIDs = {
  telegram: this.telegramChatService.getTelegramChatID(),
  whatsapp: this.whatsappChatService.getWhatsappChatID(),
};
getCurrentActiveChatID(chatID: string, botID: string) {
  let currentChatID = '';
  if (!chatID && botID) currentChatID = botID;
  else if (chatID && !botID) currentChatID = chatID;
  else if (chatID && botID) currentChatID = chatID;
  return currentChatID;
}
public sendMsgToSomeMessengers = (messengers: string[], data: string) => {
  messengers.forEach((messenger) => {
    this.messageSenders[messenger](data, this.senderIDs[messenger]);
  });
```

```
};
public sendMsgToCurrentMessenger = (messenger: string, data: string) => {
  this.messageSenders[messenger](data, this.senderIDs[messenger]);
};
async router(command: string, senderID: string, messenger: string) {
  const regex = /^\/(\d{1,2})-(\d{1,2})$/;
  const match = command.match(regex);
  if (command === chatRoutes.start) {
    this.sendMsgToCurrentMessenger(messenger, startText);
  } else if (command === chatRoutes.help)
    this.sendMsgToCurrentMessenger(messenger, helpText);
  else if (command === chatRoutes.history) {
    const history = await this.chatEventService.getParsedHistoryStr();
    this.sendMsgToCurrentMessenger(messenger, history);
  } else if (command === chatRoutes.historyYesterday) {
    const history =
      await this.chatEventService.getParsedYesterdayHistoryStr();
    this.sendMsgToCurrentMessenger(messenger, history);
  } else if (command === chatRoutes.historyClear) {
    await this.chatEventService.clearChatEvents();
    this.sendMsgToSomeMessengers(
      this.messengers,
      'Історію подій було очищено',
    );
  } else if (command === chatRoutes.on) {
    this.mqttServerService.stopWorkSchedule();
    this.mqttServerService.chModeMqttServer('on', this.senderIDs[messenger]);
  } else if (command === chatRoutes.off) {
    this.mqttServerService.stopWorkSchedule();
    this.mqttServerService.chModeMqttServer('off', this.senderIDs[messenger]);
  } else if (command === chatRoutes.forecastTomorrow) {
    const electricityData =
      await this.electricityForecastService.getTomorrowForecastAsStr();
    this.sendMsgToCurrentMessenger(messenger, electricityData);
  } else if (match) {
```

```
      const hoursOn = parseInt(match[1], 10);
      const hoursOff = parseInt(match[2], 10);
      if (
        hoursOn > 0 &&
        hoursOn < 24 &&
        hoursOff > 0 &&
        hoursOff < 24 &&
        hoursOn + hoursOff === 24
      ) {
        this.mqttServerService.startWorkSchedule(hoursOn, hoursOff, senderID);
      }
    }
  }
}

import {
  AutoIncrement,
  Column,
  Model,
  PrimaryKey,
  Table,
} from 'sequelize-typescript';
import { DataTypes } from 'sequelize';
@Table({ timestamps: false })
export class ChatEvent extends Model {
  @PrimaryKey
  @AutoIncrement
  @Column({
    type: DataTypes.INTEGER,
  })
  id: number;
  @Column
  action: string;
  @Column({
    type: DataTypes.DATE,
```

```
    allowNull: false,
    defaultValue: DataTypes.NOW,
  })
  eventDate: Date;
}

import { Module } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { SequelizeModule } from '@nestjs/sequelize';
import { ChatEvent } from '@modules/chatEvent/chatEvent.entity';
import { ChatEventService } from '@modules/chatEvent/chatEvent.service';
@Module({
  imports: [SequelizeModule.forFeature([ChatEvent])],
  providers: [ConfigService, ChatEventService],
  exports: [ChatEventService],
})
export class ChatEventModule {}

import { Injectable } from '@nestjs/common';
import { InjectModel } from '@nestjs/sequelize';
import { Op } from 'sequelize';
import { Repository } from 'sequelize-typescript';
import { ChatEvent } from '@modules/chatEvent/chatEvent.entity';
import { getDateAsTimeString } from '@helpers/time';
@Injectable()
export class ChatEventService {
  constructor(
    @InjectModel(ChatEvent)
    private readonly chatEventRepository: Repository<ChatEvent>,
  ) {}

  async getYesterdayEvents() {
    const today = new Date();
    const startOfYesterday = new Date(today);
    const endOfYesterday = new Date(today);
```

```
  startOfYesterday.setDate(today.getDate() - 1);
  startOfYesterday.setHours(0, 0, 0, 0);
  endOfYesterday.setDate(today.getDate() - 1);
  endOfYesterday.setHours(23, 59, 59, 999);
  const events: ChatEvent[] = await this.chatEventRepository.findAll({
    where: {
      eventDate: {
        [Op.between]: [startOfYesterday, endOfYesterday],
      },
    },
  });
  return events || [];
}
parseEventsToString(events: ChatEvent[]) {
  let historyStr: string = '';
  events.forEach(
    (history) =>
      (historyStr += `${history.action} o ${history.eventDate}\n\n`),
  );
  return historyStr || 'Подій не було';
}
async getParsedHistoryStr() {
  const events: ChatEvent[] = await this.chatEventRepository.findAll();
  const modifyEvents = events.map((event) => ({
    ...event.dataValues,
    eventDate: getDateAsTimeString(event.eventDate),
  }));
  return this.parseEventsToString(modifyEvents);
}
async getParsedYesterdayHistoryStr() {
  const events: ChatEvent[] = await this.getYesterdayEvents();
  return this.parseEventsToString(events);
}
async addChatEvent(action: string) {
  await this.chatEventRepository.create({
```

```
      action,
      eventDate: new Date(),
    });
  }
  async clearChatEvents() {
    await this.chatEventRepository.destroy({
      truncate: true,
    });
  }
}

import { Module } from '@nestjs/common';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { SequelizeModule as NestSequelizeModule } from '@nestjs/sequelize';
@Module({
  imports: [
    NestSequelizeModule.forRootAsync({
      imports: [ConfigModule],
      inject: [ConfigService],
      useFactory: async (configService: ConfigService) => ({
        dialect: 'mysql',
        host: configService.get<string>('MYSQL_HOST'),
        port: Number(configService.get<string>('MYSQL_PORT')),
        username: configService.get<string>('MYSQL_USER'),
        password: configService.get<string>('MYSQL_PASSWORD'),
        database: configService.get<string>('MYSQL_DATABASE'),
        autoLoadModels: true,
        // models: ['dist/modules/**/*.entity.js'],
        synchronize: true,
        // migrations: [ 'dist/db/migrations/**/*.js' ],
        // cli: { migrationsDir: 'src/db/migrations' },
      }),
    }),
  ],
})
```

```
export class SequelizeModule {}

import { Module } from '@nestjs/common';
import { ElectricityForecastService } from './electricityForecast.service';
import { WeatherModule } from '@modules/weather/weather.module';
import { WeatherService } from '@modules/weather/weather.service';
import { ConfigService } from '@nestjs/config';
@Module({
  imports: [WeatherModule],
  providers: [ElectricityForecastService, WeatherService, ConfigService],
  exports: [ElectricityForecastService],
})
export class ElectricityForecastModule {}

import { Injectable } from '@nestjs/common';
import { WeatherService } from '@modules/weather/weather.service';
import {
  getCForecast,
  getIChange,
  getIConsumer,
  getPEf,
  getPTerm,
  getTCelsius,
  getTWork,
} from '@helpers/formuls';
import {
  chargerParams,
  controllerParams,
  inverterParams,
  panelsParams,
} from '@constants/electricityDevicesParams';
import { IWeatherForecastItem } from '@modules/weather/weather.types';
import { convertMillisecondsToHours } from '@helpers/time';
@Injectable()
export class ElectricityForecastService {
```

```
    private iConsumer = getIConsumer(inverterParams.p, chargerParams.u);
    constructor(private readonly weatherService: WeatherService) {}
    async getDeltaTsInHours(forecastList: IWeatherForecastItem[]) {
      const { sunrise, sunset } =
        await this.weatherService.getApproximatelyTomorrowSys();
      return forecastList.map(({ dt }, index) => {
        let periodInMilliseconds: number;
        if (index === 0) periodInMilliseconds = dt - sunrise;
        else if (index === forecastList.length - 1)
          periodInMilliseconds = sunset - dt;
        else periodInMilliseconds = dt - forecastList[index - 1].dt;
        return convertMillisecondsToHours(periodInMilliseconds);
      });
    }
    async getTomorrowForecastAsStr() {
      const tomorrowForecastList =
        await this.weatherService.getWeatherTomorrowForecastLightList();
      const tomorrowForecastData = tomorrowForecastList.map((item) => {
        const maxT = getTCelsius(item.main.temp_max);
        const cloudsPercent = item.clouds.all;
        const totalPanelsPower = panelsParams.pOne * panelsParams.count;
        const termPower =
          maxT > 20 ? getPTerm(totalPanelsPower, maxT) : totalPanelsPower;
        const effectivePower = getPEf(termPower, cloudsPercent);
        return getIChange(effectivePower, controllerParams.vOut);
      });
      const deltaTArr = await this.getDeltaTsInHours(tomorrowForecastList);
      const cForecast = getCForecast(tomorrowForecastData, deltaTArr);
      const tWork = getTWork(cForecast, this.iConsumer);
      const cForecastForShow = cForecast.toFixed(2);
      const regulator = 10;
      const tWorkForShow = (tWork * regulator).toFixed(2);
      return `Приблизне значення\nЕмності: ${cForecastForShow}Ah;\nЧасу роботи:
${tWorkForShow}год.`;
    }
```

```
}

export const mqttServerRoutes = {
  connect: 'connect',
  message: 'message',
  error: 'error',
  serverResFirstOn: 'server/res/first-on',
  serverResOn: 'server/res/on',
  serverResOff: 'server/res/off',
  serverReq: 'server/req/',
  serverReqSchedule: 'server/req/schedule',
  serverResTest: 'server/res/test',
} as const;

import { forwardRef, Module } from '@nestjs/common';
import { MqttServerService } from './mqttServer.service';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { TelegramModule } from '@modules/telegram/telegram.module';
import { TelegramService } from '@modules/telegram/telegram.service';
import { ChatEventModule } from '@modules/chatEvent/chatEvent.module';
import { SequelizeModule } from '@nestjs/sequelize';
import { ChatEvent } from '@modules/chatEvent/chatEvent.entity';
import { ChatEventService } from '@modules/chatEvent/chatEvent.service';
import { ChatModule } from '@modules/chat/chat.module';
@Module({
  imports: [
    SequelizeModule.forFeature([ChatEvent]),
    ConfigModule,
    forwardRef(() => TelegramModule),
    forwardRef(() => ChatModule),
    ChatEventModule,
  ],
  providers: [
    ConfigService,
    MqttServerService,
```

```
    TelegramService,
    ChatEventService,
  ],
  exports: [MqttServerService],
})
export class MqttServerModule {}

import { forwardRef, Inject, Injectable } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import mqtt, { MqttClient } from 'mqtt';
import { TelegramService } from '@modules/telegram/telegram.service';
import { mqttServerRoutes } from '@modules/mqttServer/mqttServer.constants';
import { delay, getCurrentDateAsTimeString } from '@helpers/time';
import { TMqttServerMode } from '@modules/mqttServer/mqttServer.types';
import { ChatEventService } from '@modules/chatEvent/chatEvent.service';
import { ChatService } from '@modules/chat/chat.service';
@Injectable()
export class MqttServerService {
  private readonly mqttServer: string;
  private readonly mqttPort: number;
  private readonly mqttUser: string;
  private readonly mqttPass: string;
  private mqttClient: MqttClient;
  private isSchedulingActive: boolean = false;
  private scheduleInterval: number | undefined | NodeJS.Timeout | any;
  private mqttServerTestInterval: number | undefined | NodeJS.Timeout | any;
  private isFirstMqttServerTestRes = true;
  private previousIDOfTestMqttServerRes = 0;
  private IDOfTestMqttServerRes = 0;
  private countOfTestMqttServerExtraRes = 0;

  constructor(
    @Inject(forwardRef(() => TelegramService))
    private readonly telegramChatService: TelegramService,
    @Inject(forwardRef(() => ChatService))
```

```
    private readonly chatService: ChatService,
    @Inject(ChatEventService)
    private readonly chatEventService: ChatEventService,
    private readonly configService: ConfigService,
  ) {
    console.log('MqttServerService initialized');
    this.mqttServer = this.configService.get<string>('MQTT_SERVER');
    this.mqttPort = Number(this.configService.get<string>('MQTT_PORT'));
    this.mqttUser = this.configService.get<string>('MQTT_USER');
    this.mqttPass = this.configService.get<string>('MQTT_PASS');
  }
  async onModuleInit() {
    const options = {
      port: this.mqttPort,
      username: this.mqttUser,
      password: this.mqttPass,
    };
    this.mqttClient = mqtt.connect(this.mqttServer, options);
    this.mqttClient.on(mqttServerRoutes.connect, () => {
      console.log('Connected to MQTT broker');
      this.mqttSubscriber();
    });
    this.mqttClient.on(mqttServerRoutes.message, async (topic, message) => {
      await this.router(topic, message.toString());
    });
    this.mqttClient.on(mqttServerRoutes.error, (err) => {
      console.error('MQTT connection error:', err);
    });
  }
  onModuleDestroy() {
    this.stopTestMqttServerInterval();
  }
  mqttSubscriber() {
    this.mqttClient.subscribe(mqttServerRoutes.serverResOn, (err) => {
      if (!err) console.log('Subscribed to server/on');
```

```
  });
  this.mqttClient.subscribe(mqttServerRoutes.serverResOff, (err) => {
    if (!err) console.log('Subscribed to server/off');
  });
  this.mqttClient.subscribe(mqttServerRoutes.serverResFirstOn, (err) => {
    if (!err) console.log('Subscribed to server/res/first-on');
  });
  this.mqttClient.subscribe(mqttServerRoutes.serverResTest, (err) => {
    if (!err) console.log('Subscribed to server/res/test');
  });
}
stopTestMqttServerInterval() {
  clearInterval(this.mqttServerTestInterval);
  this.isFirstMqttServerTestRes = true;
  this.previousIDOfTestMqttServerRes = 0;
  this.IDOfTestMqttServerRes = 0;
  this.countOfTestMqttServerExtraRes = 0;
}
async testMqttServerIntervalCallback() {
  // console.log(
  //   'callback',
  //   this.previousIDOfTestMqttServerRes,
  //   this.IDOfTestMqttServerRes,
  // );
  if (
    this.previousIDOfTestMqttServerRes !== this.IDOfTestMqttServerRes &&
    this.countOfTestMqttServerExtraRes <= 2
  ) {
    this.previousIDOfTestMqttServerRes = this.IDOfTestMqttServerRes;
    this.countOfTestMqttServerExtraRes = 0;
  } else if (
    this.previousIDOfTestMqttServerRes === this.IDOfTestMqttServerRes &&
    this.countOfTestMqttServerExtraRes > 2
  ) {
    this.chatService.sendMsgToSomeMessengers(
```

```
      this.chatService.messengers,
      `Електрика була вмикнута о ${getCurrentDateAsTimeString()}`,
    );
    await this.chatEventService.addChatEvent('Електрика була вмикнута');
    this.stopTestMqttServerInterval();
  } else if (
    this.previousIDOfTestMqttServerRes === this.IDOfTestMqttServerRes &&
    this.countOfTestMqttServerExtraRes <= 2
  ) {
    this.countOfTestMqttServerExtraRes++;
    this.chatService.sendMsgToSomeMessengers(
      this.chatService.messengers,
      `Сервер не відповів ${this.countOfTestMqttServerExtraRes} раз`,
    );
  }
}
mqttServerTestResHandler(testResID: number) {
  this.IDOfTestMqttServerRes = testResID;
  if (!this.mqttServerTestInterval && this.isFirstMqttServerTestRes) {
    this.mqttServerTestInterval = setInterval(
      this.testMqttServerIntervalCallback.bind(this),
      10000,
    );
    this.isFirstMqttServerTestRes = false;
  }
}
async router(topic: string, resData: string) {
  if (topic === mqttServerRoutes.serverResFirstOn) {
    this.chatService.sendMsgToSomeMessengers(
      this.chatService.messengers,
      `Електрику було вимкнено о ${getCurrentDateAsTimeString()}`,
    );
    await this.chatEventService.addChatEvent('Електрику вимкнено');
  } else if (topic === mqttServerRoutes.serverResOn) {
    this.chatService.sendMsgToSomeMessengers(
```

```
      this.chatService.messengers,
      'Електрика була вмикнута',
    );
    await this.chatEventService.addChatEvent('Електрика була вмикнута');
  } else if (topic === mqttServerRoutes.serverResOff) {
    this.chatService.sendMsgToSomeMessengers(
      this.chatService.messengers,
      'Електрику було вимкнено',
    );
    await this.chatEventService.addChatEvent('Електрика була вмикнута');
  } else if (topic === mqttServerRoutes.serverResTest)
    this.mqttServerTestResHandler(Number(resData));
}
chModeMqttServer(mode: TMqttServerMode, senderID: string) {
  this.mqttClient.publish(`${mqttServerRoutes.serverReq}${mode}`, senderID);
}
startWorkSchedule(hoursOn: number, hoursOff: number, senderID: string) {
  this.stopWorkSchedule();
  if (this.isSchedulingActive) return;
  this.isSchedulingActive = true;
  const schedule = async () => {
    while (this.isSchedulingActive) {
      this.chModeMqttServer('on', senderID);
      await delay(hoursOn * 1000);
      this.chModeMqttServer('off', senderID);
      await delay(hoursOff * 1000);
    }
  };
  this.scheduleInterval = setTimeout(schedule, 0);
}
stopWorkSchedule() {
  if (!this.isSchedulingActive) return;
  this.isSchedulingActive = false;
  if (this.scheduleInterval) clearTimeout(this.scheduleInterval);
}
```

```
}

export type TMqttServerMode = 'on' | 'off';

export const weatherRoutes = {
  forecast: '/forecast',
  weather: '/weather',
} as const;

import { Module } from '@nestjs/common';
import { WeatherService } from './weather.service';
import { ConfigModule, ConfigService } from '@nestjs/config';
@Module({
  imports: [ConfigModule],
  providers: [ConfigService, WeatherService],
  exports: [WeatherService, WeatherService],
})
export class WeatherModule {}

import axios from 'axios';
export const weatherApiFactory = (baseURL: string, appid: string) =>
  axios.create({
    baseURL,
    params: {
      appid,
    },
  });

import { Injectable } from '@nestjs/common';
import { weatherApiFactory } from '@modules/weather/weather.queries';
import { weatherRoutes } from '@modules/weather/weather.constants';
import { ConfigService } from '@nestjs/config';
import { AxiosInstance } from 'axios';
import {
  ITodayWeather,
```

```
  IWeatherForecast,
} from '@modules/weather/weather.types';
import { twentyFourHoursInMs } from '@helpers/time';
@Injectable()
export class WeatherService {
  private weatherApi: AxiosInstance;
  constructor(private readonly configService: ConfigService) {
    this.weatherApi = weatherApiFactory(
      configService.get<string>('WEATHER_URL'),
      configService.get<string>('WEATHER_APP_ID'),
    );
  }
  async fetchWeatherData<T extends IWeatherForecast | ITodayWeather>(
    url: string,
  ): Promise<T> {
    const { data: weatherData } = await this.weatherApi.get<T>(url, {
      params: {
        q: 'Kharkiv,UA',
      },
    });
    return weatherData;
  }
  async getWeatherTomorrowForecastList() {
    const today = new Date();
    const tomorrow = new Date(today);
    tomorrow.setDate(today.getDate() + 1);
    const tomorrowDateStr = tomorrow.toISOString().split('T')[0];
    const weatherData = (
      await this.fetchWeatherData<IWeatherForecast>(weatherRoutes.forecast)
    ).list;

    return weatherData.filter((item) =>
      item.dt_txt.startsWith(tomorrowDateStr),
    );
  }
```

```
  async getTodayWeatherSys() {
    return (await this.fetchWeatherData<ITodayWeather>(weatherRoutes.weather))
      .sys;
  }
  async getApproximatelyTomorrowSys() {
    const todayWeatherSys = await this.getTodayWeatherSys();
    return {
      sunrise: (todayWeatherSys.sunrise += twentyFourHoursInMs),
      sunset: (todayWeatherSys.sunset += twentyFourHoursInMs),
    };
  }
  async getWeatherTomorrowForecastLightList() {
    const tomorrowForecastList = await this.getWeatherTomorrowForecastList();
    const { sunrise, sunset } = await this.getApproximatelyTomorrowSys();
    return tomorrowForecastList.filter(
      (listItem) => listItem.dt >= sunrise && listItem.dt < sunset,
    );
  }
}

interface IWeatherMain {
  temp: number;
  feels_like: number;
  temp_min: number;
  temp_max: number;
  pressure: number;
  sea_level: number;
  grnd_level: number;
  humidity: number;
  temp_kf: number;
}
interface IWeather {
  id: number;
  main: string;
  description: string;
```

```
  icon: string;
}
interface IWeatherClouds {
  all: number;
}
interface IWeatherWind {
  speed: number;
  deg: number;
  gust: number;
}
interface IWeatherForecastSys {
  pod: string;
}
export interface IWeatherForecastItem {
  dt: number;
  main: IWeatherMain;
  weather: IWeather[];
  clouds: IWeatherClouds;
  wind: IWeatherWind;
  visibility: number;
  pop: number;
  sys: IWeatherForecastSys;
  dt_txt: string;
}
export interface IWeatherForecast {
  cod: number;
  message: number | string;
  cnt: number;
  list: IWeatherForecastItem[];
}
interface IWeatherCoordinates {
  lon: number;
  lat: number;
}
export interface ICurrentWeatherSys {
```

```
  country: string;
  sunrise: number;
  sunset: number;
 }
 export interface ITodayWeather {
  coord: IWeatherCoordinates;
  weather: IWeather[];
  base: string;
  main: IWeatherMain;
  visibility: number;
  wind: IWeatherWind;
  clouds: IWeatherClouds;
  dt: number;
  sys: ICurrentWeatherSys;
  timezone: number;
  id: number;
  name: string;
  cod: number;
 }

 export const whatsappRoutes = {
  message: 'message',
  qr: 'qr',
  ready: 'ready',
 } as const;

 import { forwardRef, Module } from '@nestjs/common';
 import { WhatsappService } from './whatsapp.service';
 import { ConfigModule, ConfigService } from '@nestjs/config';
 import { MqttServerModule } from '@modules/mqttServer/mqttServer.module';
 import { ChatModule } from '@modules/chat/chat.module';
 import { TelegramModule } from '@modules/telegram/telegram.module';
 import { ElectricityForecastModule } from
'@modules/electricityForecast/electricityForecast.module';
 @Module({
```

```
  imports: [
    ConfigModule,
    forwardRef(() => ChatModule),
    forwardRef(() => MqttServerModule),
    forwardRef(() => TelegramModule),
    forwardRef(() => ElectricityForecastModule),
  ],
  providers: [ConfigService, WhatsappService],
  exports: [WhatsappService],
})
export class WhatsappModule {}

import { forwardRef, Inject, Injectable } from '@nestjs/common';
import { join } from 'path';
import { Client, LocalAuth } from 'whatsapp-web.js';
import * as QRCode from 'qrcode';
import { ConfigService } from '@nestjs/config';
import { whatsappRoutes } from '@modules/whatsapp/whatsapp.constants';
import { ChatService } from '@modules/chat/chat.service';
@Injectable()
export class WhatsappService {
  private readonly whatsappChatName: string;
  private readonly whatsappBotID: string;
  private whatsappChatID: string;
  private chat: any;
  public readonly isWhatsappInUse: boolean;
  constructor(
    private readonly configService: ConfigService,
    @Inject(forwardRef(() => ChatService))
    private readonly chatService: ChatService,
  ) {
    this.whatsappChatName =
      this.configService.get<string>('WHATSAPP_CHAT_NAME');
    this.whatsappBotID = this.configService.get<string>('WHATSAPP_BOT_ID');
    this.isWhatsappInUse = this.configService
```

```
    .get<string>('MESSENGERS')
    .split('|')
    .includes('whatsapp');
}
async onModuleInit() {
  const client: any = new Client({
    authStrategy: new LocalAuth({
      dataPath: join('.', 'messengersData', 'whatsapp', '.wwebjs_auth'),
    }),
    webVersionCache: {
      type: 'local',
      path: join('.', 'messengersData', 'whatsapp', '.wwebjs_cache'),
    },
  });
  client.on(whatsappRoutes.ready, async () => {
    console.log('Client is ready!');
    const chats = await client.getChats();
    this.whatsappChatID = this.getCurrentGroupID(chats);
    this.chat = await client.getChatById(this.whatsappChatID);
  });
  client.on(whatsappRoutes.qr, (qr) => {
    QRCode.toString(
      qr,
      {
        type: 'terminal',
        small: true,
      },
      (err, url) => {
        if (err) throw err;
        if (!this.isWhatsappInUse) return;
        console.log(url);
      },
    );
  });
  client.on(whatsappRoutes.message, async (msg) => {
```

```
      await this.chatService.router(msg.body, this.whatsappChatID, 'whatsapp');
    });
    await client.initialize();
  }
  sendMessageToWhatsapp(message: string, senderID: string) {
    this.chat.sendMessage(message);
  }
  getWhatsappChatID() {
    return this.chatService.getCurrentActiveChatID(
      this.whatsappChatID,
      this.whatsappBotID,
    );
  }
  getCurrentGroupID(chats: any[]) {
    const chat = chats.find(
      (chat) =>
        chat.name === this.configService.get<string>('WHATSAPP_CHAT_NAME'),
    );
    return chat.id._serialized;
  }
}

import { Module } from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';
import { TelegramModule } from '@modules/telegram/telegram.module';
import { MqttServerModule } from '@modules/mqttServer/mqttServer.module';
import { SequelizeModule } from '@db/sequelize.module';
import { ChatEventModule } from '@modules/chatEvent/chatEvent.module';
import { WhatsappModule } from '@modules/whatsapp/whatsapp.module';
import { ChatModule } from '@modules/chat/chat.module';
import { ElectricityForecastModule } from
'@modules/electricityForecast/electricityForecast.module';
import { WeatherModule } from '@modules/weather/weather.module';
@Module({
  imports: [
```

```
    ConfigModule.forRoot({
      envFilePath: ['.env', '.env.local', '.env.production'],
    }),
    SequelizeModule,
    TelegramModule,
    MqttServerModule,
    WhatsappModule,
    ChatModule,
    ChatEventModule,
    WeatherModule,
    ElectricityForecastModule,
  ],
  controllers: [],
  providers: [],
})
export class AppModule {}
```

ДОДАТОК Б

“Опубліковані результати”

Міністерство освіти і науки України

NURE

Харківський національний університет радіоелектроніки

ЗБІРНИК

студентських наукових статей

«Автоматизація та приладобудування»

«Automation and Development of Electronic Devices»

ADED-2025

(Випуск 1)

[електронне видання]

http://nure.ua/department/kafedra-komp-yuterno-integrovanih-tehnologiy-avtomatizatsiyi-ta-mehatroniki-kitam

http://itez.zntu.edu.ua/

http://kafea.kdu.edu.ua

Харків 2025

ЗМІСТ

УДК 658.5:338.3

РОЗРОБЛЕННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ ВІДДАЛЕНОГО КЕРУВАННЯ АВАРІЙНИМ ЕЛЕКТРОПОСТАЧАННЯМ НА ВИРОБНИЧОМУ ПІДПРИЄМСТВІ

Александрович Д.П.
Харківський національний університет
радіоелектроніки Україна, 61166, Харків, пр.
Науки, 14
E-mail: daniel.aleksandrovych@nure.ua

Анотація: У статті розглянуто використання сучасних технології і пристроїв в системах управління аварійним електропостачанням в промислових умовах. Розглядаються основні переваги таких систем, зокрема віддалене регулювання аварійним електропостачанням, інтеграція в сучасні месенджери та можливість автоматичного керування електрикою за графіком.

Ключові слова: Серверний додаток, управління електрикою, промисловість, мікроконтролер, автоматизація, оптимізація управління, технологія обміну повідомленнями.

DEVELOPMENT OF AUTOMATED REMOTE CONTROL SYSTEM EMERGENCY ELECTRICITY SUPPLY AT THE MANUFACTURING ENTERPRISE

D.Aleksandrovych.
Kharkiv National University of Radio
Electronics Ukraine, 61166, Kharkiv, pr.
Nauki, 14
E-mail: daniel.aleksandrovych@nure.ua

Abstract: The article examines the use of modern technology and devices in emergency power supply control systems in industrial conditions. The main advantages of such systems are considered, in particular, remote regulation of emergency power supply, integration into modern messengers, and the possibility of automatic control of electricity according to a schedule.

Keywords: Server application, electricity management, industry, microcontroller, automation, control optimization, messeging technology.

ВСТУП. У сучасному світі, де промисловість постійно розвивається, стабільність електропостачання є ключовим аспектом забезпечення конкурентоспроможності підприємств. Управління аварійним електропостачанням на виробництвах є важливим у цьому контексті, оскільки споживання електроенергії є головним елементом працездатності виробництва. Традиційні методи управління аварійним електропостачанням можуть бути недостатньо ефективними в змінних виробничих умовах та строгих потребах електричної стабільності.

У сучасних системах автоматизації електропостачання на виробництвах найчастіше можна здійснити не лише автоматичний перехід на системи безперебійного живлення, тобто використання акумуляторних батарей. Розвиток технологій програмування, використання інтернету речей та мікроконтролерної техніки дозволяє здійснити перехід до можливостей віддаленого керування електрикою на виробничих підприємствах.

У цьому контексті використання технологій обміну повідомленнями (ТОП) та мікроконтролерів (МК) у системах управління аварійним електропостачанням на виробництвах стає перспективним рішенням. Мікроконтролери дозволяють створювати системи, які дозволяють здійснювати віддалене керування, передавати та отримувати дані через мережу і здійснювати взаємодію з апаратними пристроями, що робить їх ефективними інструментами для покращення аварійного електропостачання та надання комфортних робочих умов для персоналу.

ТОП – це програми для обміну миттєвими повідомленнями, які забезпечують комунікацію у реальному часі через текст, аудіо, відео та мультимедіа. Такі програми допомагають інтегрувати системи керування у процес спілкування керівництва та надавати можливість швидкої та гнучкої можливості керування.

Серверні додатки (СД) можуть надати підтримку у прийнятті рішень на основі аналізу даних, прогнози. Це допомагає керівництву підприємства приймати обґрунтовані та стратегічні рішення, що забезпечують успішність їхньої діяльності.

ЗБІР ТА АНАЛІЗ ДАНИХ. Системи управління аварійним електропостачанням, які використовують мікроконтролери для віддаленого керування, починають свою роботу із повідомлення користувача про зникнення електрики в мережі. Для цього використовується протокол передачі даних та сервер-обробник, який передає дані безпосередньо користувачу. Цей сервер є обробником всіх подій, як зі сторони мікроконтролера, так і користувача.

Для простоти роботи, уникнення необхідності віддаленого розгортання серверу контролера та необхідності отримання доменного імені гарним рішенням є асиметричний протокол MQTT[1].

Використання мережевих протоколів накладає певні вимоги до мікроконтролера. Однією з них є підтримка доступу до мережі за допомогою wi-fi. Одним з простих рішень із доступом до мережі є відладочна плата[2] ESP32-CAM.

Після передачі даних з плати необхідним є вияснити куди саме інформація передається. Одним з варіантів передачі інформації є відправка її до брокера, тобто передавача даних між клієнтами. Одним із сервісів, які дозволяють безкоштовно розгорнути брокера є CloudAMQP[3].

Брокер відправляє дані до клієнтів. Отже необхідно розуміти, що можливо обрати у якості клієнту. Одним із багатофункціональних рішень є Nest [4] сервер, який дозволяє інтегрувати отримання даних з брокера асиметричного протоколу та надалі передати їх до засобів передачі повідомлень.

Програмні забезпечення для обміну повідомленнями, в контексті даної роботи, є рішенням, яке дозволить оперативно повідомляти керівництво про важливі події та надавати можливості здійснення сумісного керівництва. В світі існує багато таких програм. Самою розповсюдженою є Whatsapp. В проекти Nest вона інтегрується завдяки бібліотеці whatsapp-web.js [5].

Часто в серверних проектах використовують бази даних для збереження, швидкої взаємодії з даними та їх подальшого аналізу. Для системи аварійного електропостачання важливими даними є інформація про події вимкнення та включення електроенергії, а також графіки регулювання аварійним електропостачанням. Існує багато різних баз даних, які дозволяють створити подібне сховище даних[6-7]. Крім цього треба розуміти що безпосередня взаємодія з базою даних із використанням мови запитів є досить небезпечною. Тому використовують інструменти для програмної взаємодії з базою, такі як Sequelize[8].

Отримання безкоштовної енергії для забезпечення підтримки заряду системи аварійного електропостачання можуть дозволити модулі сонячних панелей[9].

Для прогнозування стану виробничої системи є декілька шляхів. Їх прикладами є використання штучного інтелекту або власні розрахунки. Так для прогнозування можливої ємності отримуваної електрики на виробничому підприємстві на майбутнє необхідним є знання погодних умов, які можуть суттєво впливати на роботу системи [10]. Одним із рішень у вільному доступі для отримання стану погоди є ресурс OpenWeather [11].

ВПЛИВ ВИКОРИСТАННЯ МІКРОКОНТРОЛЕРІВ В ПОЄДНАННІ З ТЕХНОЛОГІЯМИ ОБМІНУ ПОВІДОМЛЕННЯМИ НА ЕФЕКТИВНІСТЬ КЕРУВАННЯ. Мікроконтролери в поєднанні з технологіями обміну повідомленнями відіграють все більш значну роль у сучасних промислових виробництвах, надаючи функціонал для ефективного керування бізнес-процесами. У цій статті розглядається вплив МК та ТОП на ефективність керування процесами на підприємствах, виявляючи основні аспекти, які сприяють зростанню продуктивності та конкурентоспроможності.

Мікроконтролери та ТОП дозволяють автоматизувати та спростити процес керування. Це сприяє підвищенню ефективності роботи підприємства завдяки виключення ситуацій довгого простою обладнання та виконання працівниками недоцільних операцій.

МК дозволяють керувати електрикою безпосередньо. В їх можливості може входити включення та виключення електрики та повідомлення про стан роботи системи. Останнє є перевагою у порівнянні зі звичайними пристроями безперебійного живлення.

ТОП дозволяють інтегрувати системи керування в чат спілкування, що в свою чергу надає можливості отримати знання про стан виробництва більше, ніж одному співробітнику. Також процес керування можуть здійснювати декілька працівників. Це може зробити процес керування більш гнучким надаючи можливості для прийняття спільних рішень і швидкої передачі критично важливої інформації.

ТОП дозволяють підвищити оперативність доставлення інформації про стан електрики на виробництві відповідальним особам без виконання додаткових дій людини. Прикладом такого функціоналу є група, в яку додані всі відповідальні за електрику працівники. При надходженні повідомлення до групи його бачать одразу всі учасники. Крім цього можливе швидке керування через відповідні повідомлення-реакції органів керівництва на критично важливі події.

АВТОМАТИЗОВАНА СИСТЕМА ВІДДАЛЕНОГО КЕРУВАННЯ АВАРІЙНИМ ЕЛЕКТРОПОСТАЧАННЯМ ЯК ІНСТРУМЕНТ АВТОМАТИЗАЦІЇ ЕЛЕКТРОПОСТАЧАННЯ НА ВИРОБНИЦТВАХ. Робота систем електропостачання може базуватися на постійному отриманні сонячної енергії, тобто автоматичному заряді джерел живлення. Крім цього можливе також додання елементів віддаленого керування, а саме віддаленого керування з будь-якої точки світу. І ще можлива інтеграція засобів прогнозування можливого об’єму отримуваної електроенергії для подальшого аналізу і планування графіку роботи виробничого підприємства за рахунок використання ресурсу, що дозволяє отримати дані про погодні умови (подібних до OpenWeather[11]) та розрахунків.

Особливістю переходу на віддалене керування є те, що система зберігає всі властивості, подібні до системи безперебійного живлення (тобто може розпочати автоматичне передачу електрики одразу після зникнення мережевого струму), але крім цього набуває нові можливості, такі як отримання сонячної енергії для підтримання енергозабезпечення,

можливість ручного налаштування стану електрики, або графіків включення та відключення електроенергії, отримання історії подій зміни стану електроенергії та розрахунки для визначення прогнозного значення щодо отримуваної електроенергії, наприклад, на завтрашній день (для аналізу та проектування графіку включення та виключення електроенергії).

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ОБМІНУ ПОВІДОМЛЕННЯМИ, ЯК ІНСТРУМЕНТ КЕРУВАННЯ. Найпопулярніші програми з ТОП, такі як WhatsApp, Telegram, Viber, Signal та Facebook Messenger, відрізняються функціоналом, рівнем безпеки та можливостями інтеграції. Ключовою особливістю сучасних месенджерів є підтримка наскрізного шифрування, що забезпечує конфіденційність даних. Наприклад, Signal і WhatsApp використовують цю технологію для захисту повідомлень від несанкціонованого доступу. Окрім текстових повідомлень, месенджери пропонують функції відеодзвінків, групових чатів, обміну файлами, що робить корпоративне спілкування більш різноманітним і зручним.

Інтеграція з іншими сервісами, такими як боти, також розширює можливості таких програм. Вони використовуються не лише для особистого спілкування, але і в бізнесі, освіті та обслуговуванні клієнтів, що робить їх універсальним інструментом комунікації.

В контексті керування месенджери надають гнучкий функціонал для керування автоматизованими системами як через чат ботів (якщо для керування потрібен лише один працівник), так і можливості для цілої групи керування (бот в групі або каналі).

АРХІТЕКТУРА СИСТЕМИ. Архітектура системи керування, що розроблюється, є основою, яка дозволяє здійснювати віддалене керування електроенергією. В її основі лежать ТОП, база даних, дві серверні платформи, джерела живлення та невелика сонячна станція. Перша серверна платформа зберігається безпосередньо на мікроконтролері керування електроенергією. Її функціонал дуже простий – вона обробляє включення та відключення електроенергії, а також повідомляє про те, що електроенергія до сих пір не з’явилась. Другий додаток є більш навантаженим. Від відповідає за передачу та отримання повідомлень з месенджерів, взаємодію з базою даних, підключення до стороннього ресурсу та розрахунків прогнозування. Важливо зауважити, що запропонавана архітектура системи побудована таким чином, що дозволяє використання сучасних протоколів та методів передачі даних і є гнучким рішенням подібних завдань.

Також важливою частиною архітектури є блок забезпечення електроенергією. Він складається з сонячних панелей, контролера заряду та акумулятора. Контролер заряду дозволяє акумулятору отримати лише необхідний заряд (тобто захист від перезарядки).

Крім цього для передачі електроенергії в мережу необхідним є використання інвертора, який дозволяє перетворити струм з напругою акумулятора в такий, який можуть споживати звичайні мережеві пристрої.

Схематичне зображення архітектури системи автоматизованого керування аварійним електропостачанням на виробничому підприємстві приведено на рисунку 1.

Як видно з рисунку – сонячна енергія потрапляє через контролер заряду на акумулятор. При зникненні мережевої електроенергії спрацьовує реле, і від акумуляторного живлення вмикається плата (в даному випадку – ESP32-CAM) та інтернет. Після цього плата вмикає електроенергію, а також повідомляє вантажний сервер про відключення електрики. Сервер в свою чергу повідомляє працівника або працівників через месенджер і далі працівник (працівники) взаємодіють з системою. При відправки працівником запиту на зміну стану електроенергії змінюється стан електрики, що надходить з ESP32-CAM до модулю керування

напругою 5В. Якщо напруга подавалась – то відбувається відключення і навпаки. Модулю керування напругою 5В реагує на дану подію і відповідно замикає, або розмикає ключ низького струму на інверторі.

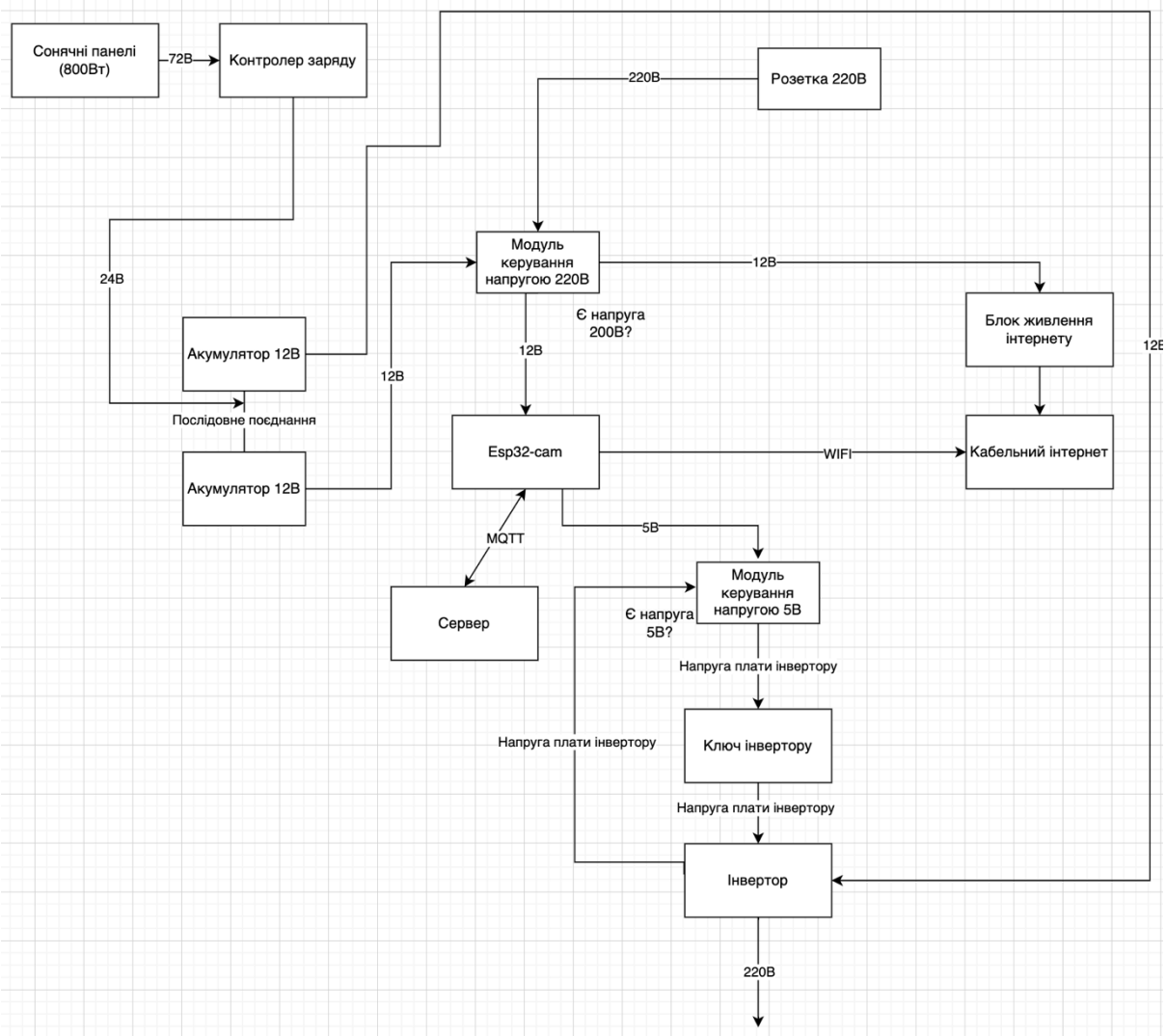


Рисунок 1 – Схематичне зображення автоматизованої системи віддаленого керування аварійним електропостачанням на виробничому підприємств

На рисунку 1 вказано модуль сервера у композиційному позначенні. Схему декомпозованого модуля сервера приведено на рисунку 2. Він складається з самого серверного додатку, написаного із використанням Nest.js, бази даних (в даному випадку MySQL), боту і групи або каналу з месенджера та підключення до стороннього API OpenWeather.

Зі схеми 2 видно, що сервер може бути налаштований як для керування однією особою, так і для керування командою працівників. Для одиничного керування достатнім є відправка повідомлень безпосередньо до бота.

Також перевагою даної системи є можливість використання декількох програм з ТОП одночасно або легкий перехід з однієї на іншу.

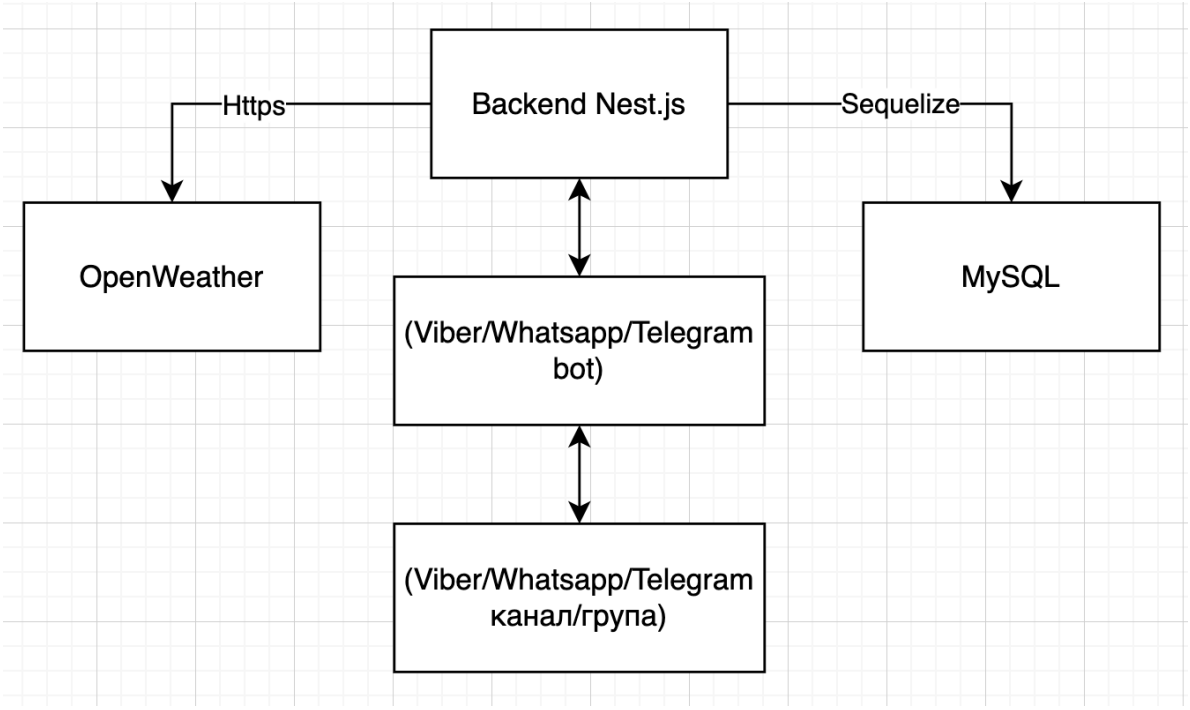


Рисунок 2 – Схематичне зображення серверного модулю розроблюваної системи

ВИСНОВКИ. Підсумовуючи, впровадження мікроконтролерів, програмних технологій, систем отримання сонячної енергії та програм, які дозволяють використання технологій обміну повідомленнями у системи керування аварійним електропостачанням на виробничих підприємствах є важливим кроком до підвищення ефективності та надійності виробничих процесів. Поєднання цих технологій дозволяє автоматизувати контроль і регулювання електропостачання, забезпечуючи швидке реагування на аварійні ситуації та мінімізуючи час простою обладнання.

Мікроконтролери забезпечують гнучке та надійне віддалене керування енергосистемами, що дозволяє оптимізувати споживання енергії та підвищити загальну продуктивність підприємства. Використання серверних платформ для обробки даних може бути використано для прогнозування майбутнього стану системи.

Інтеграція програм з ТОП у систему управління надає можливість ефективної комунікації між працівниками, спрощуючи спільне керування та передачу критично важливої інформації. Завдяки цьому підвищується оперативність реагування на аварійні ситуації, а процеси управління стають більш прозорими та доступними для кількох відповідальних осіб.

Таким чином, впровадження автоматизованих систем керування аварійним електропостачанням на виробничому підприємстві дозволяє підвищити якість керування, зменшити ризики простоїв та забезпечити стабільну роботу виробничих підприємств, що є важливим фактором їх конкурентоспроможності на сучасному ринку.

ЛІТЕРАТУРА

1. Комплекс навчально-методичного забезпечення навчальної дисципліни "Технологія промислового інтернету речей" підготовки магістра спеціальності 151 - Автоматизація та комп'ютерно-інтегровані технології [Електронний ресурс] : спеціалізація "Комп`ютерно-

інтегровані технологічні процеси і виробництва" / ХНУРЕ; розроб.: С. П. Новоселов, О. В. Сичова. – Харків, 2020. – 129 с.

2. Распіновка ESP32-cam AI-Thinker. URL: https://arduino-tex.ru/news/34/raspinovka-esp32-cam-ai-thinker-naznachenie-gpio.html (дата звернення: 25.11.2024).
3. Que starts here. URL: https://www.cloudamqp.com/ (дата звернення: 12.10.2024).
4. Introduction. URL: https://docs.nestjs.com/ (дата звернення: 22.09.2024).
5. Whatsapp-web.js. URL: https://wwebjs.dev/ (дата звернення: 08.11.2024).
6. Комплекс навчально-методичного забезпечення навчальної дисципліни "Системи управління базами даних" підготовки рівня бакалавра спеціальності 151 - Автоматизація та комп'ютерно-інтегровані технології [Електронний ресурс] : освітня програма "Системна інженерія" / ХНУРЕ; розроб. С. В. Хрустальова. – Харків, 2022. – 287 с.
7. Комплекс навчально-методичного забезпечення навчальної дисципліни "Технологія організації баз даних і знань" підготовки магістра спеціальності 151 - Автоматизація та комп'ютерно-інтегровані технології [Електронний ресурс] : освітня програма "Комп`ютерно-інтегровані технологічні процеси і виробництва" / ХНУРЕ; розроб. С. С. Максимова. – Харків, 2022. – 128 с.
8. Sequelize. URL: https://sequelize.org/ (дата звернення: 18.10.2024).
9. Сонячна станція для економії на підприємстві. URL: https://onlysolar.in.ua/ru/katalog/solnechnaya-stanciya-dlya-ekonomii-na-predpriyatii/ (дата звернення: 07.10.2024).
10. Як високі температури впливають на сонячні панелі. URL: https://leworld.org/tpost/9vbk629o61-kak-visokie-temperaturi-vliyayut-na-soln#:~:text=%D0%A1%D0%BE%D0%B3%D0%BB%D0%B0%D1%81%D0%BD%D0%BE%20%D0%BA%D0%BE%D0%BC%D0%BF%D0%B0%D0%BD%D0%B8%D0%B8%20CED%20Greentech%2C%20%D1%81%D0%BB%D0%B8%D1%88%D0%BA%D0%BE%D0%BC,%D0%BF%D0%B0%D0%B4%D0%B0%D0%B5%D1%82%20%D0%BD%D0%B0%200%2C5%25. (дата звернення: 18.11.2024).
11. OpenWeather. URL: https://openweathermap.org/ (дата звернення: 14.11.2024).

Науковий керівник: *Новоселов Сергій Павлович, к.т.н., професор кафедри КІТАР Харківського національного університету радіоелектроніки*

ДОДАТОК В

Демонстраційний матеріал

Харківський національний університет радіоелектроники

Кафедра КІТАР

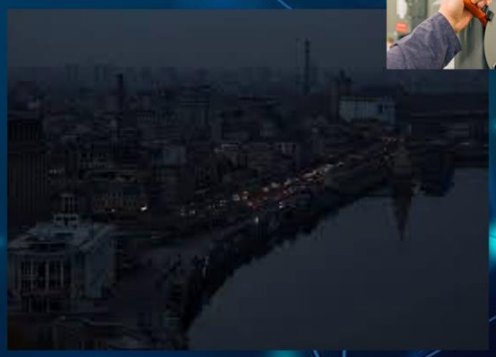
Кваліфікаційна робота на тему:

Розроблення автоматизованої системи віддаленого керування аварійним електропостачанням на виробничому підприємстві

Виконав:
Студент групи КІТПВм-24-1
Александрович Даніель Павлович

Керівник:
Професор кафедри КІТАР
Новоселов Сергій Павлович

Актуальність роботи



Причинами необхідності покращення електропостачання на підприємствах є:

- руйнування та ускладнення роботи енергетики через бойові дії;
- очевидно недостатній захист від ситуацій частого зникання електроенергії;
- відсутність можливості зменшення вартості екологічних систем автономного електрозабезпечення;
- відсутність можливості віддаленого керування електропостачанням на підприємствах, де це було б можливо і потрібно.

Постановка задачі

Мета роботи
Покращення керування електропостачанням на виробничих підприємствах і підвищення рівня автоматизації та безпеки роботи електричних складових на виробничих підприємствах.

Об'єкт розробки
Процес керування аварійним електропостачанням на виробничому підприємстві.

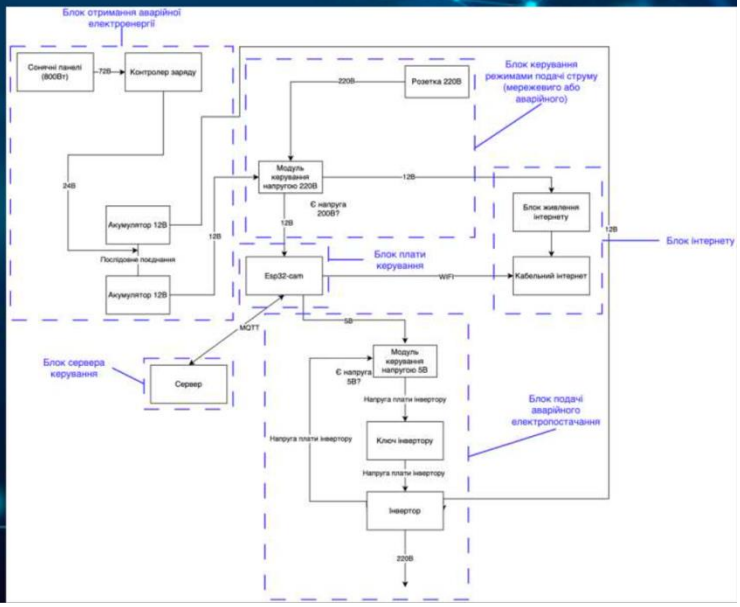
Предмет розробки
Макет автоматизованої системи віддаленого керування аварійним електропостачанням на виробничому підприємстві.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- розробити структурну схему макету та алгоритм роботи системи;
- описати методи та технології взаємодії компонентів системи;
- провести підбір елементної бази та технологій реалізації компонентів макету системи;
- виконати реалізацію складових макету та їх поєднання;
- провести тестування розробленого макету.

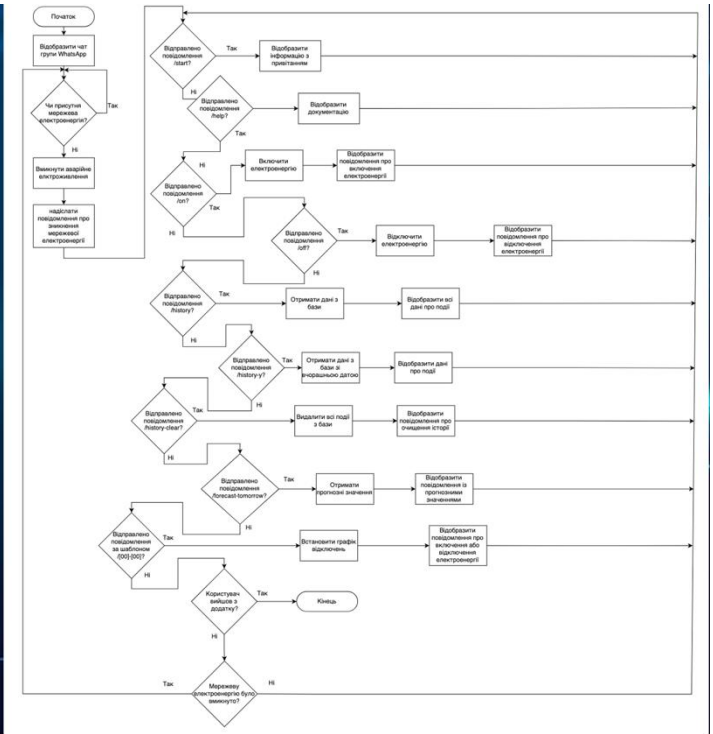
Структурна схема макету та алгоритм роботи системи

1. Блок отримання аварійної електроенергії (сонячні панелі, контролер заряду та АКБ)
2. Блок керування режимами подачі струма, до складу якого входить модуль керування напругою 220В
3. Блок інтернету (забезпечення інтернету)
4. Блок плати керування
5. Сервер керування
6. Блок подачі аварійного електропостачання (інвертор і модуль керування напругою ключа інвертору)



Алгоритм роботи

Суть алгоритму: при зниканні мережевої електроенергії починає роботу система; надалі відбувається очікування різних команд, а після появи мережевої електроенергії система припиняє свою дію

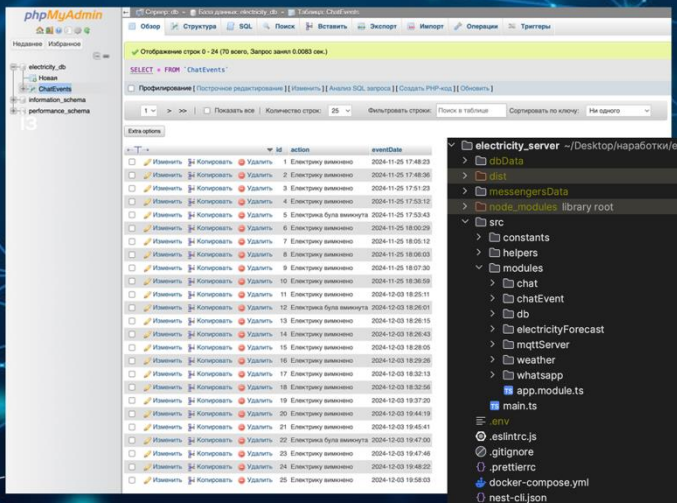


Технології реалізації компонентів макету системи

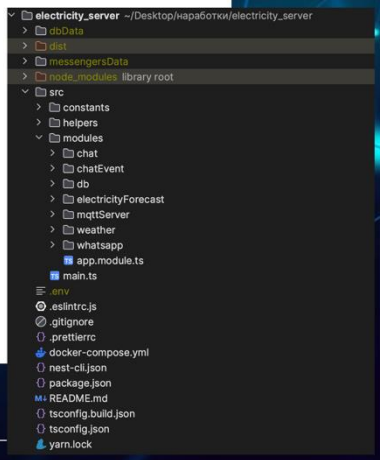
Серверу керування реалізовано використанням наступних технологій:

- TypeScript, як мова програмування;
- Фреймворк Nest.js;
- бібліотека Sequalize для легкої взаємодії з базою даних MySQL.

В якості бази даних використано MySQL.



База даних



Структура проекту

До складу макету системи входять кілька плат, реле та модулів:

Esp32-cam має wifi-модуль, слот для камери OV2640. Даний модуль є дешевим та багатофункціональним рішенням для безлічі завдань.

FT232RL є гнучким пристроєм для живлення пристроїв з напругою живлення 3,3В або 5В

Перетворювач, що понижує напругу з 12В до 5В.

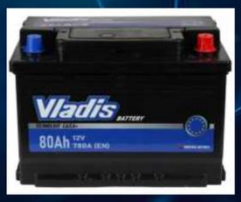
Реле HH54P (220В)

Реле 3,3В

Інвертор UKC 1500W (220В)

Risen (36В/400Вт)

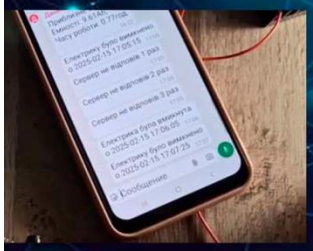
Контролер заряду PowMr

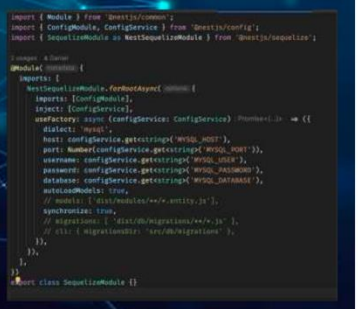


АКБ фірми Vadis

Засоби та технології взаємодії компонентів системи

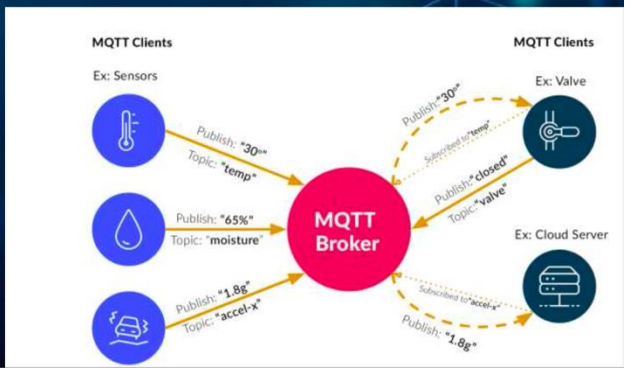
Взаємодія між сервером плати esp32-cam та сервером керування відбувається за допомогою протоколу MQTT.
MQTT - це протокол обміну повідомленнями. За даним протоколом система працює таким чином, що є видавник, який надсилає за певною темою повідомлення до брокера. Брокер в свою чергу надсилає це повідомлення всім підписникам цієї теми.
Для взаємодії з користувачем системи використовується WhatsApp бот.
Взаємодія між сервером керування та базою відбувається завдяки бібліотеки sequalize:



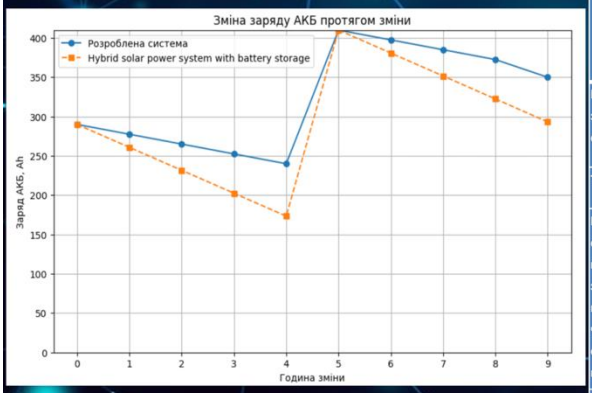


```
import { Module } from '@nestjs/common';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { SequelizeModule as NestSequelizeModule } from '@nestjs/sequelize';

@Module({
  imports: [
    NestSequelizeModule.forRootAsync({
      imports: [ConfigModule],
      inject: [ConfigService],
      useFactory: async (configService: ConfigService) => ({
        dialect: 'mysql',
        host: configService.get<string>('MYSQL_HOST'),
        port: Number(configService.get<string>('MYSQL_PORT')),
        username: configService.get<string>('MYSQL_USER'),
        password: configService.get<string>('MYSQL_PASSWORD'),
        database: configService.get<string>('MYSQL_DATABASE'),
        autoLoadModels: true,
        // models: ['dist/modules/**/*.entity.js'],
        synchronize: true,
        // migrations: ['dist/db/migrations/**/*.js'],
        // cli: { migrationsDir: 'src/db/migrations' },
      }),
    }),
  ],
})
export class SequelizeModule {}
```



Порівняння параметрів системи із Hybrid Solar Power System with Battery Storage



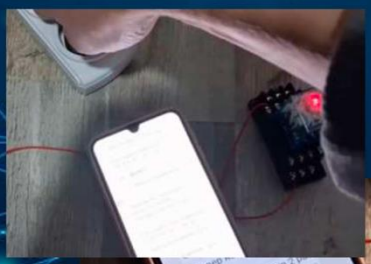
Параметр	Значення
Напруга	24 В
Максимальний заряд акумулятора	410 А*Год
Початкова потужність сонячної станції	3 кВт
Споживання	6,3 КВт/Год
Потужність сонячної панелі	400 Вт

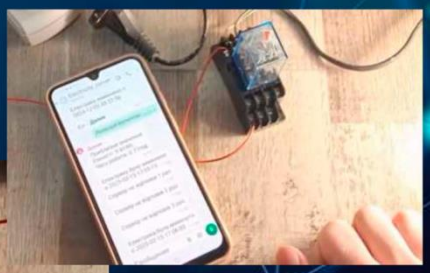
Параметр	Подібна до розробленої система	Система з ПБЖ та живленням від акумулятора	Система з ПБЖ, живленням від акумулятора та зарядженням від сонячних панелей
Максимальний час роботи за початкової потужності сонячної станції	2 години 42 хвилини	1 година 28 хвилин	2 години 42 хвилини
Тип керування	Автоматичний або автоматизований	Автоматичний	Автоматичний та ручний
Мінімальна Кількість сонячних панелей, необхідних для забезпечення роботи протягом робочої зміни (в час обіду споживання електроенергії в 3 рази менше ніж в робочу)	14	-	15
Спожита електроенергія протягом зміни	59.68 кВт	Не може працювати повну робочу зміну	62.347 кВт

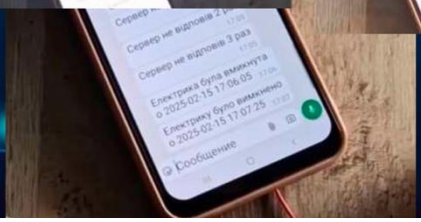
Тестування основних функцій системи

Основними функціями системи є обробка події аварійного відключення електроенергії, операції віддаленого керування, перевірка історії і прогнозу. Також сюди входить реакція на подію включення електроенергії.

Після зникання мережевої електроенергії повинне надійти повідомлення до whatsapp, а також вмикнутися аварійне електроживлення

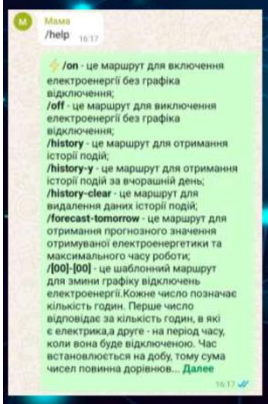


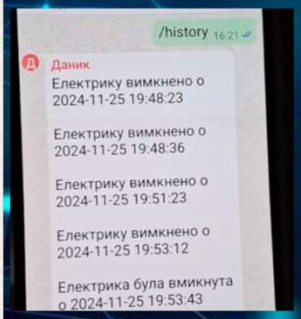


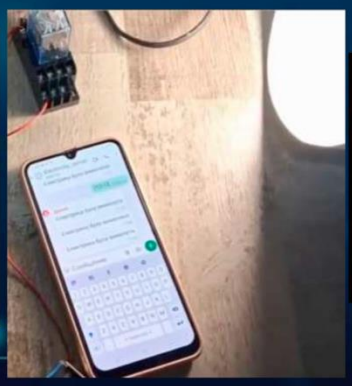


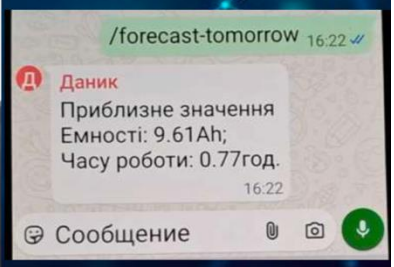
Тестування основних функцій системи

Під час роботи аварійного електроживлення є можливість виконання декількох команд: вмикання/вимкнення, встановлення графіку, робота з історією, отримання прогнозу.



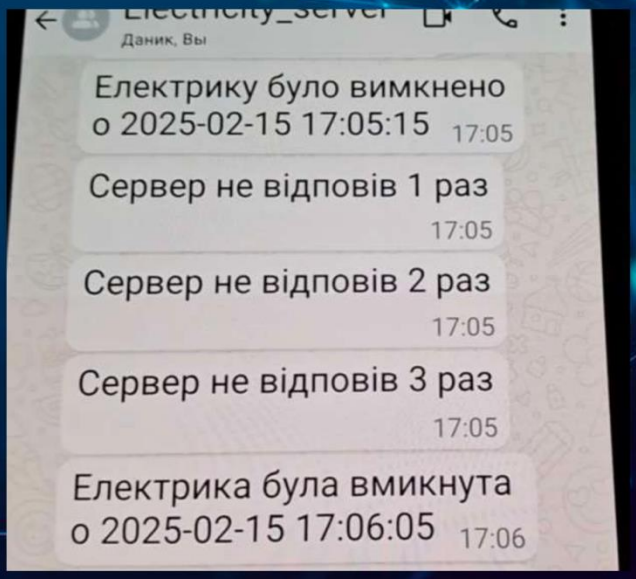






Тестування основних функцій системи

Після появи мережевої електроенергії повинна відбутися перевірка на її роботу (запити до MQTT брокера) і після - відправка повідомлення про її роботу.



Висновки

Під час виконання роботи було виконано аналіз сучасних систем аварійного електроживлення, автономного електрозабезпечення та систем автоматизованого керування.

Крім того, було поставлено та виконано ряд наступних завдань:

- розроблено структурну схему макету та алгоритм роботи системи;
- описано засоби та технології взаємодії компонентів системи;
- проведено підбір елементної бази та технологій реалізації макету і програмної складової системи;
- виконано реалізацію складових макету та їх поєднання;
- проведено тестування розробленого макету.

Розроблено програмний серверний додаток з функціоналом для керування через whatsapp бот. Також розроблено структуру і було виконано поєднання з базою даних.

Виконано тестування основних операцій системи.

	Позначення	Найменування	Дод. Відомості
		Текстові документи	
1	ГЮІК.421455.001 ПЗ	Пояснювальна записка	129 с.
		Графічні документи	
2		Демонстраційний матеріал	13 с.

Змін.	Арк.	Номер докум.	Підп.	Дата	ГЮІК.421455.001ВД			
Розроб.		Александрович Д.П.			Розроблення автоматизованої системи віддаленого управління роботизованим сховищем на виробничому підприємстві		Аркуш	Аркушів
Перевір.		Новоселов С.П.						1
Н.контр.		Стародубцев М.Г.				ХНУРЕ Кафедра КІТАМ		
Затв.		Невлюдов І.Ш.						