МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
РАДІОЕЛЕКТРОНІКИ

ЗВІТ

з лабораторної роботи №2

з дисципліни

«Аналіз та рефакторинг коду»

Виконала
ст. гр. ПЗПІ-22-5
Черевко Марина Романівна

Перевірив:
Дашенков Д. С.

Харків 2024

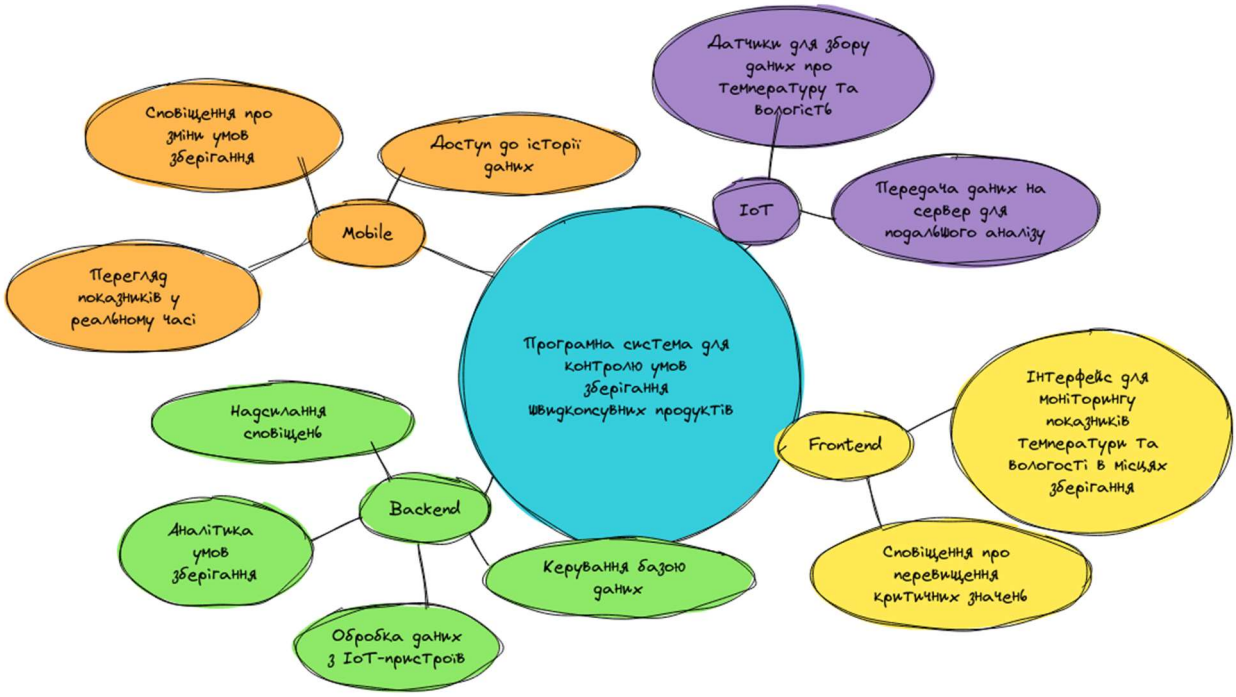
2 РОЗРОБКА БАЗИ ДАНИХ ДЛЯ СЕРВЕРНОЇ ЧАСТИНИ ПРОГРАМНОЇ СИСТЕМИ ТА ПРИКЛАДНОГО ПРОГРАМНОГО ІНТЕРФЕЙСУ (API)

2.1. Мета роботи

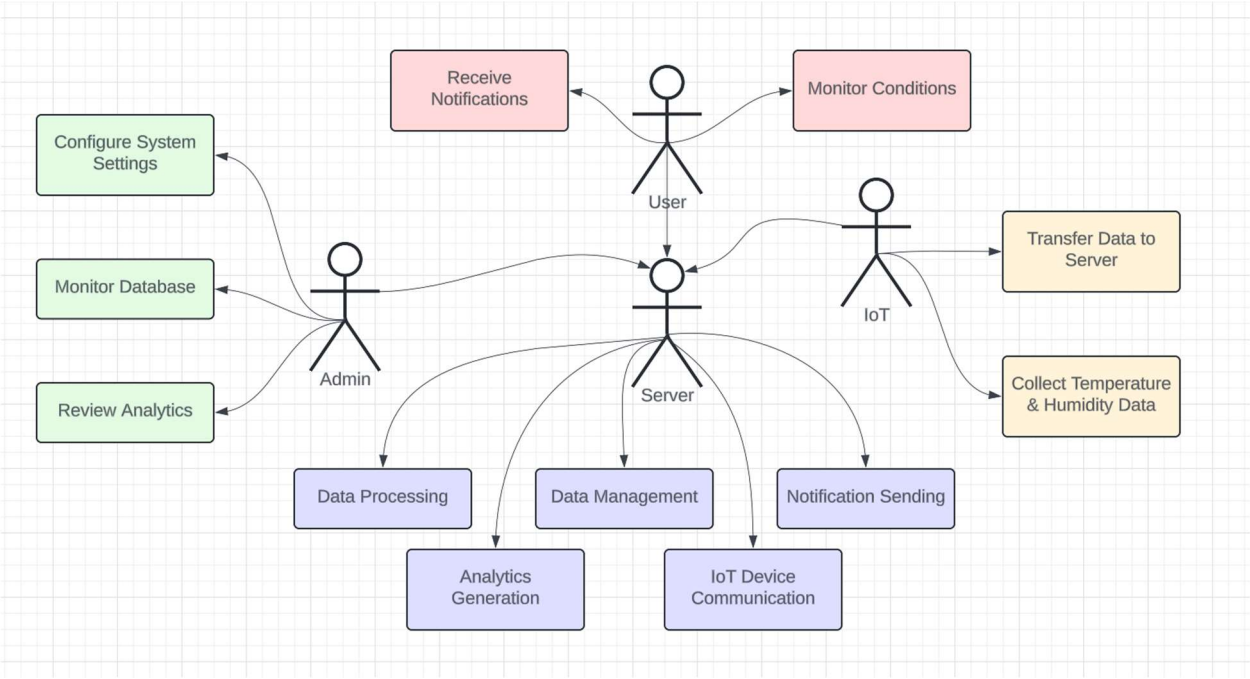
Розробити базу даних для серверної частини програмної системи та прикладного програмного інтерфейсу.

2.2. Порядок виконання роботи

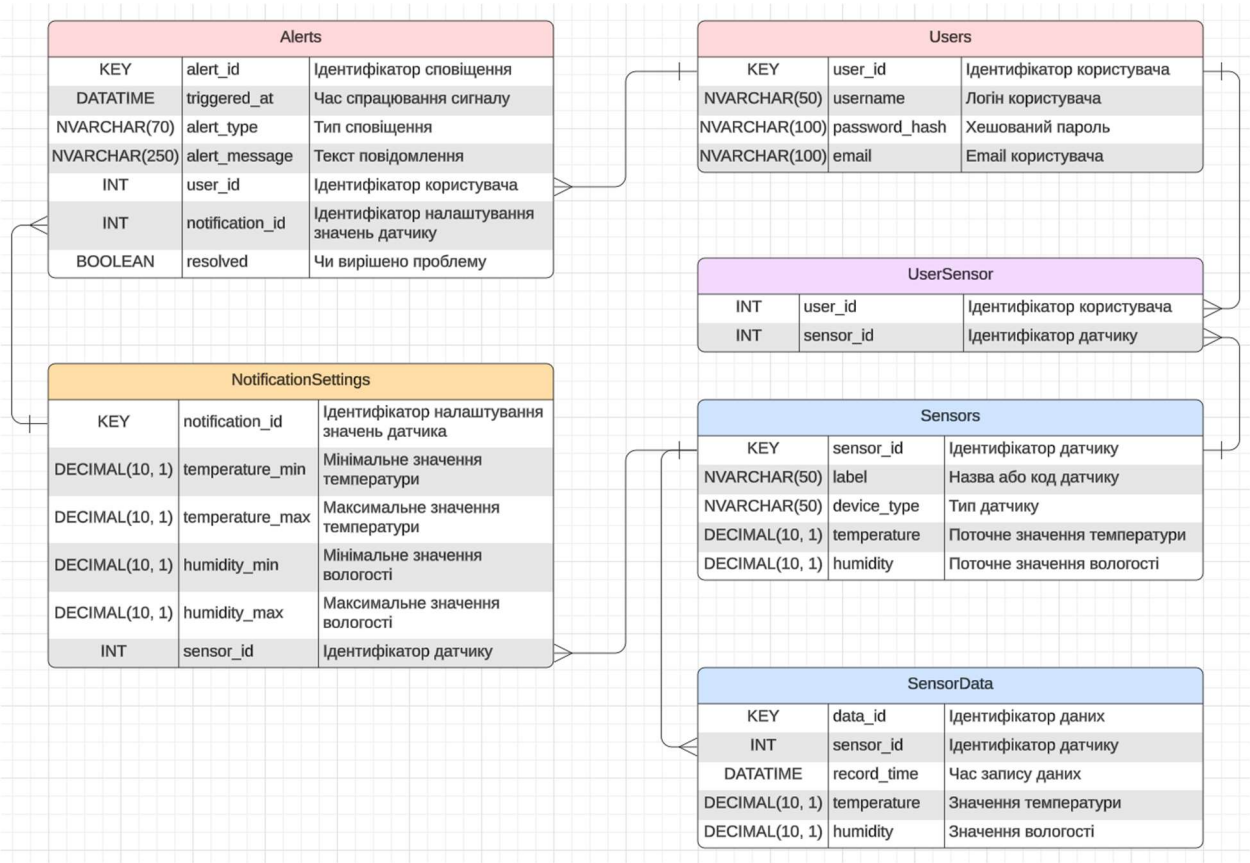
2.2.1. Будова програмної системи



2.2.2. UML діаграма прецедентів для серверної частини системи



2.2.3. ER діаграма бази даних



Alerts

KEY	alert_id	Ідентифікатор сповіщення
DATATIME	triggered_at	Час спрацювання сигналу
NVARCHAR(70)	alert_type	Тип сповіщення
NVARCHAR(250)	alert_message	Текст повідомлення
INT	user_id	Ідентифікатор користувача
INT	notification_id	Ідентифікатор налаштування значень датчику
BOOLEAN	resolved	Чи вирішено проблему

Users

KEY	user_id	Ідентифікатор користувача
NVARCHAR(50)	username	Логін користувача
NVARCHAR(100)	password_hash	Хешований пароль
NVARCHAR(100)	email	Email користувача

UserSensor

INT	user_id	Ідентифікатор користувача
INT	sensor_id	Ідентифікатор датчику

NotificationSettings

KEY	notification_id	Ідентифікатор налаштування значень датчика
DECIMAL(10, 1)	temperature_min	Мінімальне значення температури
DECIMAL(10, 1)	temperature_max	Максимальне значення температури
DECIMAL(10, 1)	humidity_min	Мінімальне значення вологості
DECIMAL(10, 1)	humidity_max	Максимальне значення вологості
INT	sensor_id	Ідентифікатор датчику

Sensors

KEY	sensor_id	Ідентифікатор датчику
NVARCHAR(50)	label	Назва або код датчику
NVARCHAR(50)	device_type	Тип датчику
DECIMAL(10, 1)	temperature	Поточне значення температури
DECIMAL(10, 1)	humidity	Поточне значення вологості

SensorData

KEY	data_id	Ідентифікатор даних
INT	sensor_id	Ідентифікатор датчику
DATATIME	record_time	Час запису даних
DECIMAL(10, 1)	temperature	Значення температури
DECIMAL(10, 1)	humidity	Значення вологості

2.2.4. Розробка бази даних програмної системи

2.2.4.1. Скрипт створення бази даних та таблиць

```
CREATE DATABASE DHT22DB;
GO

USE DHT22DB;
GO

CREATE TABLE Users (
    user_id INT IDENTITY PRIMARY KEY,
    username NVARCHAR(50) NOT NULL,
    password_hash NVARCHAR(100) NOT NULL,
    email NVARCHAR(100) NOT NULL UNIQUE
);

CREATE TABLE Sensors (
    sensor_id INT IDENTITY PRIMARY KEY,
    label NVARCHAR(50) NOT NULL,
    device_type NVARCHAR(50) NOT NULL,
    temperature DECIMAL(10, 1) NOT NULL,
    humidity DECIMAL(10, 1) NOT NULL,
);

CREATE TABLE SensorData (
    data_id INT IDENTITY PRIMARY KEY,
    sensor_id INT NOT NULL,
    record_time DATETIMEOFFSET NOT NULL,
    temperature DECIMAL(10, 1) NOT NULL,
    humidity DECIMAL(10, 1) NOT NULL,
    FOREIGN KEY (sensor_id) REFERENCES Sensors(sensor_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE UserSensor (
    user_id INT NOT NULL,
    sensor_id INT NOT NULL,
    PRIMARY KEY (user_id, sensor_id),
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (sensor_id) REFERENCES Sensors(sensor_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

```sql
CREATE TABLE NotificationSettings (
    notification_id INT IDENTITY PRIMARY KEY,
    temperature_min DECIMAL(10, 1),
    temperature_max DECIMAL(10, 1),
    humidity_min DECIMAL(10, 1),
    humidity_max DECIMAL(10, 1),
    sensor_id INT NOT NULL,
    FOREIGN KEY (sensor_id) REFERENCES Sensors(sensor_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE Alerts (
    alert_id INT IDENTITY PRIMARY KEY,
    triggered_at DATETIMEOFFSET NOT NULL,
    alert_type NVARCHAR(70) NOT NULL,
    alert_message NVARCHAR(250) NOT NULL,
    user_id INT NOT NULL,
    notification_id INT NOT NULL,
    resolved BIT NOT NULL DEFAULT 0,
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (notification_id) REFERENCES NotificationSettings(notification_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

2.2.4.2. Список сутностей

Users (Користувачі)

Таблиця зберігає інформацію про користувачів системи. Користувачі можуть мати доступ до сенсорів, переглядати дані та налаштовувати сповіщення.

Поля:

– user_id: унікальний ідентифікатор користувача;

– username: логін користувача;

– password_hash: хешований пароль для забезпечення безпеки;

– email: унікальна електронна адреса користувача.

Sensors (Датчики)

Таблиця містить інформацію про всі підключені IoT-пристрої, які зчитують температуру та вологість.

Поля:

– sensor_id: унікальний ідентифікатор сенсора;

– label: назва або код сенсора;

– device_type: тип пристрою (наприклад, температурний чи вологості);

– temperature: поточна температура, зчитана сенсором;

– humidity: поточна вологість, зчитана сенсором.

SensorData (Дані сенсорів)

Таблиця зберігає історичні дані, отримані від сенсорів. Це дозволяє аналізувати умови зберігання з плином часу.

Поля:

– data_id: унікальний ідентифікатор запису даних;

– sensor_id: ідентифікатор сенсора, який надіслав дані;

– record_time: час запису показників;

– temperature: температура, зчитана сенсором на момент запису;

– humidity: вологість, зчитана сенсором на момент запису;

UserSensor (Користувачі та сенсори)

Таблиця забезпечує зв’язок «багато до багатьох» між користувачами та сенсорами. Один користувач може мати доступ до кількох сенсорів, і один сенсор може бути пов’язаний з кількома користувачами.

Поля:

– user_id: ідентифікатор користувача;

– sensor_id: ідентифікатор сенсора.

NotificationSettings (Налаштування сповіщень)

Таблиця містить порогові значення температури та вологості, які користувачі можуть налаштовувати для кожного сенсора. Якщо показники перевищують ці значення, система генерує сповіщення.

Поля:

– notification_id: унікальний ідентифікатор налаштувань сповіщення;

– temperature_min: мінімально допустима температура;

– temperature_max: максимально допустима температура;

– humidity_min: мінімально допустима вологість;

– humidity_max: максимально допустима вологість;

– sensor_id: ідентифікатор сенсора, для якого застосовуються ці налаштування.

Alerts (Сповіщення)

Таблиця зберігає записи про критичні ситуації, коли умови зберігання виходять за допустимі межі.

Поля:

– alert_id: унікальний ідентифікатор сповіщення;

– triggered_at: час, коли було зафіксовано проблему;

– alert_type: тип сповіщення (наприклад, перевищення температури);

– alert_message: опис проблеми та рекомендація для її вирішення;

– user_id: ідентифікатор користувача, який отримав сповіщення;

– notification_id: ідентифікатор налаштувань, які були порушені.

Роль сутностей в системі:

– Users забезпечують доступ до системи;

– Sensors реєструють параметри умов зберігання;

– SensorData дозволяють аналізувати історичні дані;

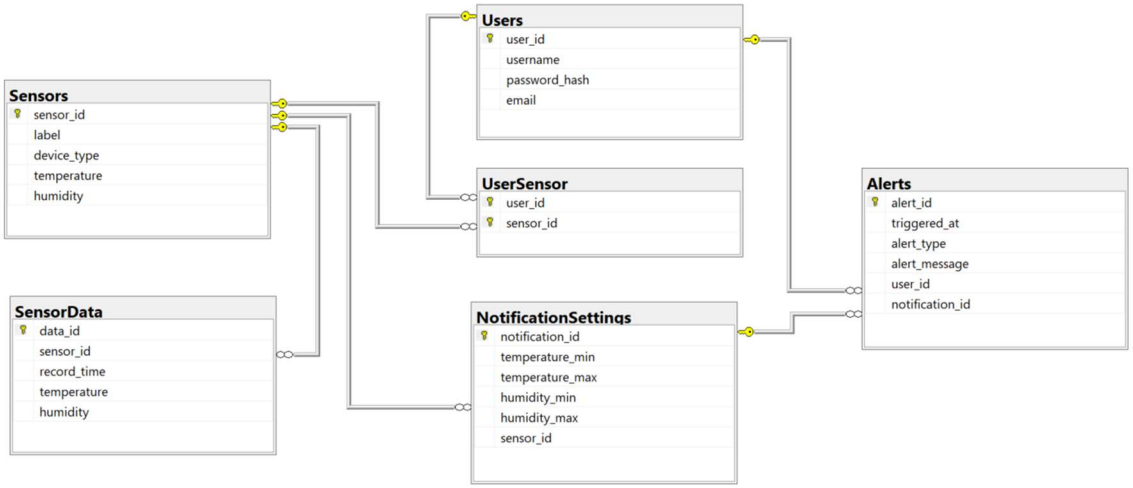
– UserSensor визначають, які сенсори доступні певним користувачам;

– NotificationSettings налаштовують порогові значення умов зберігання;

– Alerts фіксують проблеми та сповіщають користувача про них.

Ця структура забезпечує всі необхідні зв’язки для роботи системи.

2.2.5. Діаграма структури БД



2.2.6. Розробка API

За налаштування серверу відповідають наступні файли:

– .env: параметри підключення до бази даних

```
.env
1  DB_SERVER=localhost
2  DB_PORT=1433
3  DB_DATABASE=DHT22DB
4  DB_USER=cherm
5  DB_PASSWORD=************
```

– server.js: налаштування порту підключення до БД

```
server.js
1  const express : e | () => core.Express  = require('express');
2  const app : any | Express  = express();
3
4  const PORT : string | number  = process.env.PORT || 3000;
5
6  app.get('/', (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> ) : void  => {
7      res.send( body: 'Connection Successful');
8  })
9
10 app.listen(PORT,  hostname: () : void  => {
11     console.log(`Server started on port ${PORT}`);
12 })
```

– app.js: запуск сервера та налаштування маршрутів

app.js

```
const express : e | () => core.Express = require('express');
const {connectDB} = require("./config/database");
const path : PlatformPath | path = require("path");

require('dotenv').config();

const app : any | Express = express();
const PORT : number = 3000;

connectDB();

app.use(express.json());
app.use(express.static(path.join(__dirname, 'public')));

// маршрути
app.use('/api/users', require('./routes/user'));
app.use('/api/alerts', require('./routes/alert'));
app.use('/api/sensors', require('./routes/sensor'));
app.use('/api/sensordata', require('./routes/sensorData'));
app.use('/api/notifications', require('./routes/notificationSettings'));

app.get('/', (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> ) : void => {
    res.sendFile(path.join(__dirname, 'public', 'index.html'));
})

app.listen(PORT, hostname: () : void => {
    console.log(`Server started on port http://localhost:${PORT}`);
})
```

2.2.7. Створення специфікації розробленого API

Стандартом для документування REST API було обрано Swagger. Для опису специфікації необхідно створити файл swagger.yaml:

```
openapi: 3.0.0
info:
  title: DHT22 API
  version: 1.0.0
  description: API для моніторингу умов зберігання швидкопсувних продуктів
servers:
  - url: http://localhost:3000/api
    description: Local server
paths:
  /users/getAllUsers:
    get:
      tags:
        - Users
      summary: Отримати список усіх користувачів
      responses:
        '200':
          description: Список користувачів
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/User'
  /users/getUserById/{userid}:
    get:
      tags:
        - Users
      summary: Отримати користувача за ідентифікатором
      parameters:
        - name: userid
          in: path
          required: true
          description: ID користувача
          schema:
            type: string
      responses:
        '200':
          description: Інформація про користувача
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/User'
```

Swagger
Supported by SMARTBEAR

DHT22 API 1.0.0 OAS 3.0

Users

- GET /api/users/getAllUsers Get all users
- GET /api/users/getUserById/{userid} Get user by ID
- POST /api/users/addUser Add a new user
- PUT /api/users/updateUser/{userid} Update a user by ID
- DELETE /api/users/delUserById/{userid} Delete a user by ID

Alerts

- GET /api/alerts/getAllAlerts Get all alerts
- GET /api/alerts/getAlertsByUserId/{userid} Get alerts by user ID
- GET /api/alerts/getAlertById/{alertid} Get alert by ID
- POST /api/alerts/addAlert Add a new alert

Notifications

- GET /api/notifications/getAllNotifications Get all notification settings
- GET /api/notifications/getNotificationById/{notificationid} Get notification settings by ID
- POST /api/notifications/addNotificationSettings Add new notification settings

Sensors

- GET /api/sensors/getAllSensors Get all sensors
- GET /api/sensors/getSensorById/{sensorid} Get sensor by ID
- POST /api/sensors/addSensor Add a new sensor
- PUT /api/sensors/updateSensorParameters/{sensorid} Update sensor parameters

SensorData

- GET /api/sensordata/getSensorDataById/{dataid} Get sensor data by data ID
- GET /api/sensordata/getSensorDataBySensorId/{sensorid} Get sensor data by sensor ID

2.2.8. Створення програмної реалізації API та функцій роботи з БД

2.2.8.1. Отримати всіх користувачів

```
router.get( path: "/getAllUsers", handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> ,
    try {
        const [users : unknown[] ] = await sequelize.query(
            sql: `SELECT user_id, username, email FROM Users`
        );
        res.json(users)
    } catch (error) {
        console.log(`Error: ${error}`);
    }
})
```

2.2.8.2. Отримати певного користувача за ідентифікатором

```
router.get( path: "/getUserById/:userid", handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> ,
    try {
        const user_id = req.params.userid;
        const user : Model<any, TModelAttributes> = await User.findByPk(user_id);

        if (!user) {
            return res.status( code: 404).json( body: {error: "User not found"});
        }
        res.json(user)
    } catch (error) {
        console.log(`Error: ${error}`);
    }
})
```

2.2.8.3. Додати нового користувача

```
router.post( path: "/addUser", handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsOb
    try {
        const { username, password, email } = req.body;

        if (!username || !password || !email) {
            return res.status( code: 400).json( body: { error: "All fields are required" });
        }

        let password_hash = Func.fnv1aHash(password);
        const newUser : Model<any, TModelAttributes> = await User.create( values: { username, password_hash, email });

        res.status( code: 201).json( body: { message: "User created successfully", user: newUser });
    } catch (error) {
        console.log(`Error: ${error}`);
    }
})
```

2.2.8.4. Оновити дані користувача

```
router.put( path: "/updateUser/:userid",  handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> ) : Promise<...>  => {
    try {
        const user_id = req.params.userid;
        const { username, password, email } = req.body;

        const user : Model<any, TModelAttributes>  = await User.findByPk(user_id);
        if (!user) {
            return res.status( code: 404).json( body: { error: "User not found" });
        }

        if (username !== undefined){
            await user.update( keys: {
                username: username || user.username,
            });
        }
        if (password !== undefined){
            let password_hash = Func.fnv1aHash(password);
            await user.update( keys: {
                password_hash: password_hash || user.password_hash,
            });
        }
        if (email !== undefined){
            await user.update( keys: {
                email: email || user.email,
            });
        }

        res.status( code: 200).json( body: { message: "User updated successfully", user: user });
    } catch (error) {
        console.log(`Error: ${error}`);
    }
})
```

2.2.8.5. Видалити користувача за ідентифікатором

```
router.delete( path: "/delUserById/:userid",  handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<R
    try {
        const user_id = req.params.userid;
        const user : Model<any, TModelAttributes>  = await User.findByPk(user_id);
        if (!user) {
            return res.status( code: 404).json( body: { error: "User not found" });
        }

        await user.destroy();

        res.status( code: 200).json( body: { message: `User with ID ${user_id} deleted successfully` });
    } catch (error) {
        console.log(`Error: ${error}`);
    }
})
```

2.2.8.6. Отримати всі датчики

```
router.get( path: "/getAllSensors",  handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res
    try {
        const sensors : Promise<...>  = Func.getAllSensors();
        res.json(sensors)
    } catch (error) {
        console.log(`Error: ${error}`);
    }
})
```

2.2.8.7. Отримати інформацію про датчик за ідентифікатором

```
router.get( path: "/getSensorById/:sensorid", handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response
    try {
        const sensor_id = req.params.sensorid;
        const sensor : Model<any, TModelAttributes> = await Sensor.findByPk(sensor_id);

        if (!sensor) {
            return res.status( code: 404).json( body: {error: "Sensor not found"});
        }
        res.json(sensor)
    } catch (error) {
        console.log(`Error: ${error}`);
    }
})
```

2.2.8.8. Додати новий датчик

```
router.post( path: "/addSensor", handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> )
    try {
        const { label, device_type } = req.body;

        if (!label || !device_type) {
            return res.status( code: 400).json( body: { error: "All fields are required" });
        }

        const newSensor : Model<any, TModelAttributes> = await Sensor.create( values: { label, device_type });

        res.status( code: 201).json( body: { message: "Sensor created successfully", sensor: newSensor });
    } catch (error) {
        console.log(`Error: ${error}`);
    }
})
```

2.2.8.9. Оновити параметри датчику

```
router.put( path: "/updateSensorParameters/:sensorid", handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> )
    try {
        const sensor_id = req.params.sensorid;
        const { label, device_type } = req.body;

        const sensor : Model<any, TModelAttributes> = await Sensor.findByPk(sensor_id);
        if (!sensor) {
            return res.status( code: 404).json( body: { error: "User not found" });
        }

        if (label !== undefined){
            await sensor.update( keys: {
                label: label || sensor.label,
            });
        }
        if (device_type !== undefined){
            await sensor.update( keys: {
                device_type: device_type || sensor.device_type,
            });
        }

        res.status( code: 200).json( body: { message: "Sensor updated successfully", sensor: sensor });
    } catch (error) {
        console.log(`Error: ${error}`);
    }
})
```

2.2.8.10. Отримати всі налаштування повідомлень

```
router.get( path: "/getAllNotifications",  handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : R
    try {
        const notifications : (Model<...>)[]  = await NotificationSettings.findAll();
        res.json(notifications)
    } catch (error) {
        console.log(`Error: ${error}`);
    }
})
```

2.2.8.11. Отримати налаштування повідомлення по ідентифікатору

```
router.get( path: "/getNotificationById/:notificationid",  handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> ,
    try {
        const notification_id = req.params.notificationid;
        const notification : Model<any, TModelAttributes>  = await NotificationSettings.findByPk(notification_id);

        if (!notification) {
            return res.status( code: 404).json( body: {error: "Notification not found"});
        }
        res.json(notification)
    } catch (error) {
        console.log(`Error: ${error}`);
    }
})
```

2.2.8.12. Додати нові налаштування повідомлень

```
router.post( path: "/addNotificationSettings",  handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> ) : Promise<...> => {
    try {
        const { temperature_min, temperature_max, humidity_min, humidity_max, sensor_id } = req.body;

        if (!sensor_id) {
            return res.status( code: 400).json( body: { error: "Sensor is required" });
        }

        const sensor : Model<any, TModelAttributes>  = await Sensor.findByPk(sensor_id);
        if (!sensor) {
            return res.status( code: 400).json( body: { error: "Sensor not found" });
        }

        const newNotificationSettings : Model<any, TModelAttributes>  = await NotificationSettings.create( values: { temperature_min, temperature_max, humidity_min, humidity_max, sensor_id });

        res.status( code: 201).json( body: { message: "Notification created successfully", notification: newNotificationSettings });
    } catch (error) {
        console.log(`Error: ${error}`);
    }
})
```

2.2.8.13. Отримати всі сповіщення

```
router.get( path: "/getAllAlerts",  handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> ,
    try {
        const alerts : (Model<...>)[]  = await Alert.findAll();
        res.json(alerts)
    } catch (error) {
        console.log(`Error: ${error}`);
    }
})
```

2.2.8.14. Отримати всі сповіщення певного користувача

```
router.get( path: "/getAlertsByUserId/:userid",  handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> )
    try {
        const user_id = req.params.userid;

        const [alerts : unknown[] ] = await sequelize.query(
            sql: `SELECT * FROM Alerts WHERE user_id = :user_id`,
            options: {
                replacements: { user_id },
            }
        );

        if (!alerts) {
            return res.status( code: 404).json( body: {error: "User not found"});
        }
        res.json(alerts)
    } catch (error) {
        console.log(`Error: ${error}`);
    }
})
```

2.2.8.15. Отримати сповіщення за ідентифікатором

```
router.get( path: "/getAlertById/:alertid",  handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Resp
    try {
        const alert_id = req.params.alertid;
        const alert : Model<any, TModelAttributes>  = await Alert.findByPk(alert_id);

        if (!alert) {
            return res.status( code: 404).json( body: {error: "Alert not found"});
        }
        res.json(alert)
    } catch (error) {
        console.log(`Error: ${error}`);
    }
})
```

2.2.8.16. Додати нове сповіщення

```
router.post( path: "/addAlert",  handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> ) : Promise<...>  => {
    try {
        const { alert_type, message, user_id, notification_id } = req.body;
        if (!alert_type || !message || !notification_id) {
            return res.status( code: 400).json( body: { error: "All fields are required" });
        }

        const notification : Model<any, TModelAttributes>  = await NotificationSettings.findByPk(notification_id);
        if (!notification) {
            return res.status( code: 400).json( body: { error: "Notification not found" });
        }

        const sensor : Model<any, TModelAttributes>  = await Sensor.findByPk(notification.sensor_id);

        let alert_message : string  = `${message}\nПоточна темрепатура:${sensor.temperature}\nПоточна вологість:${sensor.humidity}`
        let resolved : boolean  = false;

        const newNotification : Model<any, TModelAttributes>  = await Alert.create( values: { alert_type, alert_message, user_id, notification_id, resolved });

        res.status( code: 201).json( body: { message: "Notification created successfully", notification: newNotification });
    } catch (error) {
        console.log(`Error: ${error}`);
    }
})
```

2.2.8.17. Отримати всі данні датчиків

```
router.get( path: "/getAllSensorData", handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> ,
    try {
        const sensorData : Promise<...> = SensorData.findAll();
        res.json(sensorData)
    } catch (error) {
        console.log(`Error: ${error}`);
    }
})
```

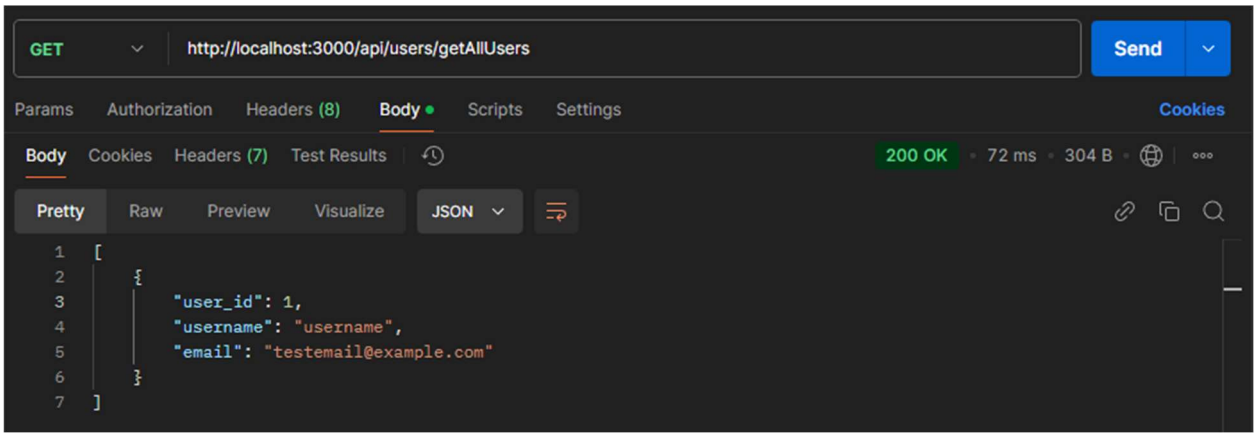
2.2.8.18. Отримати всі данні певного датчика

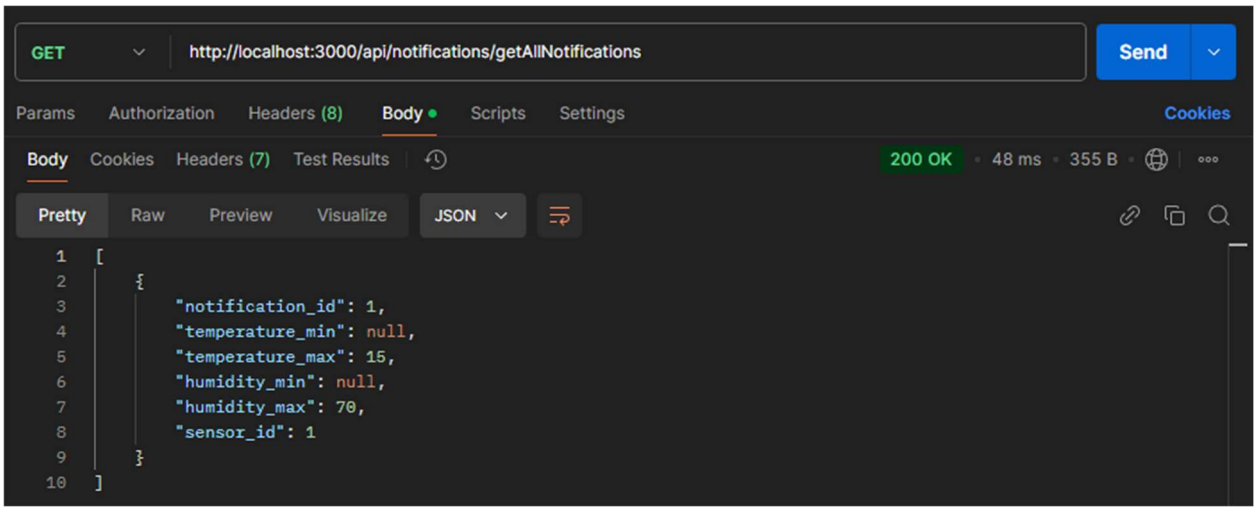
```
router.get( path: "/getSensorDataById/dataid", handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Respon
    try {
        const data_id = req.params.sensordataid;
        const sensorData : Promise<...> = SensorData.findByPk(data_id);
        res.json(sensorData)
    } catch (error) {
        console.log(`Error: ${error}`);
    }
})
```

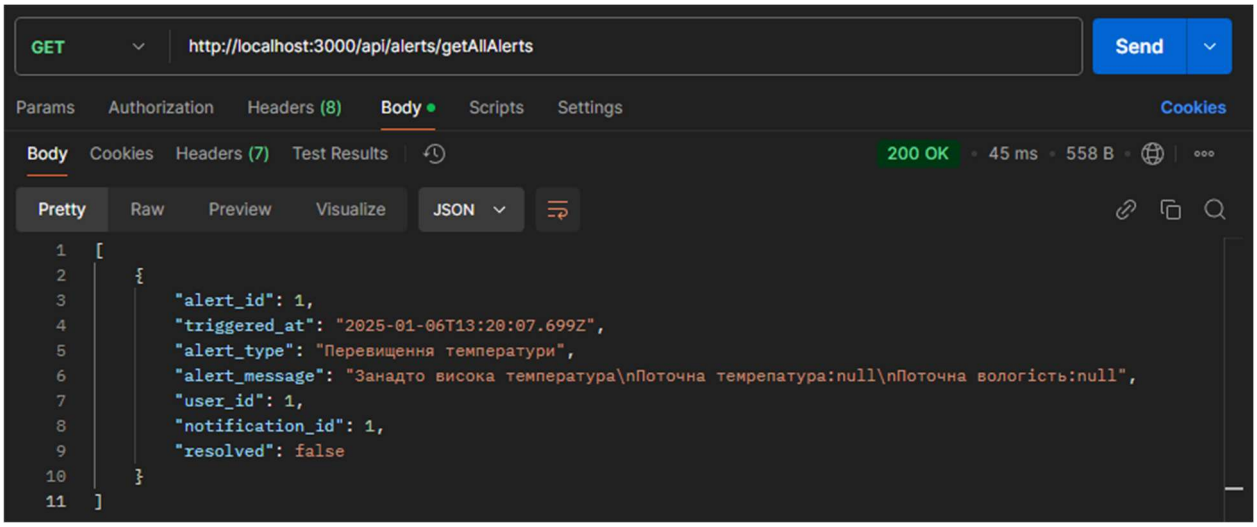
2.2.8.19. Отримати певні дані за ідентифікатором

```
router.get( path: "/getSensorDataBySensorId/:sensorid", handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res
    try {
        const sensor_id = req.params.sensorid;
        const [sensorData : unknown[] ] = await sequelize.query(
            sql: `SELECT * FROM SensorData WHERE sensor_id = :sensor_id`,
            options: {
                replacements: { sensor_id },
            }
        );
        res.json(sensorData)
    } catch (error) {
        console.log(`Error: ${error}`);
    }
})
```

2.2.9. Перевірка роботи програмного коду серверної частини системи




GET
http://localhost:3000/api/notifications/getAllNotifications
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
Body
Cookies
Headers (7)
Test Results
200 OK
48 ms
355 B
Pretty
Raw
Preview
Visualize
JSON
[
{
"notification_id": 1,
"temperature_min": null,
"temperature_max": 15,
"humidity_min": null,
"humidity_max": 70,
"sensor_id": 1
}
]


GET
http://localhost:3000/api/alerts/getAllAlerts
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
Body
Cookies
Headers (7)
Test Results
200 OK
45 ms
558 B
Pretty
Raw
Preview
Visualize
JSON
[
{
"alert_id": 1,
"triggered_at": "2025-01-06T13:20:07.699Z",
"alert_type": "Перевищення температури",
"alert_message": "Занадто висока температура\nПоточна темрепатура:null\nПоточна вологість:null",
"user_id": 1,
"notification_id": 1,
"resolved": false
}
]