

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РАДІОЕЛЕКТРОНІКИ

ЗВІТ

з лабораторної роботи №3

з дисципліни

«Аналіз та рефакторинг коду»

Виконала

ст. гр. ПЗПІ-22-5

Черевко Марина Романівна

Перевірив:

Дашенков Д. С.

Харків 2024

3 РОЗРОБКА БІЗНЕС-ЛОГІКИ ТА ФУНКЦІЙ АДМІНІСТРУВАННЯ СЕРВЕРНОЇ ЧАСТИНИ ПРОГРАМНОЇ СИСТЕМИ

3.1. Мета роботи

Розробити бізнес-логіку та функції адміністрування серверної частини програмної системи.

3.2. Порядок виконання роботи

3.2.1. Розробка бізнес-логіки серверної частини програмної системи

Розробка бізнес-логіки серверної частини передбачає створення функцій, які забезпечують коректне виконання основних процесів програмної системи. Основною метою бізнес-логіки є забезпечення коректного опрацювання даних, виконання умов і автоматизація рутинних процесів.

Основні функції бізнес-логіки

- **Перевірка автентифікації користувача:**
 - перевірка логіна і пароля при вході в акаунт користувача;
 - хешування пароля при створенні або оновленні користувача;
 - відповідь при невірному логіні або паролі (помилка 401 Unauthorized).

- **Додавання датчика до акаунта користувача:**
 - прив'язка нового датчика до конкретного користувача;
 - перевірка, чи існує датчик у системі, перед додаванням.

- **Автоматичне створення повідомлень при виході даних за межі:**
 - аналіз показників датчика (температури, вологості);
 - перевірка меж, вказаних у таблиці NotificationSettings;
 - формування тексту повідомлення;
 - визначення типу повідомлення.

- Систематичне збереження показників датчиків:
 - система кожні кілька секунд або хвилин отримує дані з датчиків.;
 - поточне значення записується у таблицю Sensors (актуальні дані);
 - старе значення переноситься до таблиці SensorData як історичне.
- Взаємодія з історією показників (SensorData):
 - надання історичних даних за обраний період часу;
 - генерація графіків або таблиць для візуалізації даних.
- Обробка помилок: централізоване управління помилками, що забезпечує зрозумілу відповідь користувачам:
 - 400 Bad Request: передано некоректні дані;
 - 404 Not Found: елемент не знайдено;
 - 500 Internal Server Error: внутрішні збої серверу.

3.2.2. Програмна реалізація бізнес-логіки та функцій адміністрування

3.2.2.1. Перевірка автентифікації користувача

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

    const user = await User.findOne({ where: { username } });
    if (!user)
      return res.status( code: 401 ).json( body: { error: "User not found" } );

    const password_hash = Func.fnv1aHash(password);
    if (password_hash.toString() !== user.password_hash)
      return res.status( code: 401 ).json( body: { error: "Wrong password" } );

    res.json(user)
  } catch (error) {
    console.log(`Error: ${error}`);
  }
})
```

3.2.2.2. Додавання датчика до акаунта користувача

```
router.post( path: "/addSensorToUser", handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Respon

try {
  const { user_id, sensor_id } = req.body;
  if (!user_id || !sensor_id) {
    return res.status( code: 400 ).json( body: { error: "All fields are required" } );
  }

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

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

  // const newUserSensor = await UserSensor.create({ user_id, sensor_id });
  const [newUserSensor : unknown[] ] = await sequelize.query(
    sql: `
      INSERT INTO UserSensor (user_id, sensor_id)
      OUTPUT INSERTED.*
      VALUES (:user_id, :sensor_id)
    `,
    options: {
      replacements: {
        user_id,
        sensor_id,
      },
    },
  );

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

3.2.2.3. Автоматичне створення повідомлень при виході даних за межі

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

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

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

    let tempAlertMessage : null = null;
    let tempAlertType : string = "Temperature Alert";
    let humAlertMessage : null = null;
    let humAlertType : string = "Humidity Alert";
    let resolved : boolean = false;
    let temperatureAlert : { alert: null, message: string } = { message: "Sensor value is normal", alert: null };
    let humidityAlert : { alert: null, message: string } = { message: "Sensor value is normal", alert: null };

    const temp_min = notification.temperature_min;
    const temp_max = notification.temperature_max;
    const hum_min = notification.humidity_min;
    const hum_max = notification.humidity_max;

    if ((temp_min && temperature < temp_min) || (temp_max && temperature > temp_max)) {
      tempAlertMessage = `Temperature out of range: ${temperature}°C (Min: ${notification.temperature_min}, Max: ${notification.temperature_max})`;
    }
    if ((hum_min && humidity < hum_min) || (hum_max && humidity > hum_max)) {
      humAlertMessage = `Humidity out of range: ${humidity}% (Min: ${notification.humidity_min}, Max: ${notification.humidity_max})`;
    }

    if (tempAlertMessage) {
      temperatureAlert.message = "Alert created successfully";
      temperatureAlert.alert = await Alert.create( values: {
        alert_type: tempAlertType,
        alert_message: tempAlertMessage,
        user_id,
        notification_id,
        resolved
      });
    }
    if (humAlertType) {
      humidityAlert.message = "Alert created successfully";
      humidityAlert.alert = await Alert.create( values: {
        alert_type: humAlertType,
        alert_message: humAlertMessage,
        user_id,
        notification_id,
        resolved
      });
    }

    res.status( code: 201 ).json( body: { temperature_alert: temperatureAlert, humidity_alert: humidityAlert });
  } catch (error) {
    console.log(`Error: ${error}`);
  }
})
```

3.2.2.4. Систематичне збереження показників датчиків

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

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

    const sensorData : Model<any, TModelAttributes> = await SensorData.create( values: {
      sensor_id: sensor_id,
      record_time: new Date(),
      temperature: sensor.temperature,
      humidity: sensor.humidity
    });

    await sensor.update( keys: {
      temperature: temperature || sensor.temperature,
      humidity: humidity || sensor.humidity,
    });

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

3.2.2.5. Взаємодія з історією показників (SensorData)

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

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

    const sensorData : Model<any, TModelAttributes> = await SensorData.create( values: {
      sensor_id: sensor_id,
      record_time: new Date(),
      temperature: sensor.temperature,
      humidity: sensor.humidity
    });

    await sensor.update( keys: {
      temperature: temperature || sensor.temperature,
      humidity: humidity || sensor.humidity,
    });

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

```
router.get( path: "/averageValuesFromSensor/:sensorid", handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> ) : Promise<...> => {
  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" });
    }

    const { duration } = req.body;
    let timeCondition : string = "";
    if (duration === "day"){
      timeCondition = `record_time >= DATEADD(DAY, -1, GETDATE())`;
    } else if (duration === "week"){
      timeCondition = `record_time >= DATEADD(WEEK, -1, GETDATE())`;
    } else if (duration === "month"){
      timeCondition = `record_time >= DATEADD(MONTH, -1, GETDATE())`;
    } else {
      return res.status( code: 404).json( body: { error: "Wrong duration value" });
    }
  }
```

```

let result: {...} = {
  sensor_id: sensor_id,
  sensor_name: sensor.label,
  duration: duration,
  avg_temperature: null,
  avg_humidity: null,
}

const [avgData: unknown[]] = await sequelize.query(
  sql: `SELECT
    AVG(temperature) AS avg_temperature,
    AVG(humidity) AS avg_humidity
  FROM SensorData
  WHERE sensor_id = :sensor_id AND ${timeCondition} `,
  options: {
    replacements: { sensor_id },
  }
);

if (avgData.length === 0 || avgData[0].avg_temperature === null) {
  return res.status(404).json({ body: { error: "No data available for the specified period" } });
}

result.avg_temperature = avgData[0].avg_temperature;
result.avg_humidity = avgData[0].avg_humidity;

res.json(result)
} catch (error) {
  console.log(`Error: ${error}`);
}
}
}

```

3.2.3. Перевірка роботи серверної частини системи

3.2.3.1. Перевірка автентифікації користувача

The screenshot displays a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:3000/api/users/login
- Body (raw JSON):**

```

{
  "username": "username",
  "password": "password"
}

```
- Status:** 200 OK
- Response Time:** 30 ms
- Response Size:** 330 B
- Response Body (Pretty JSON):**

```

{
  "user_id": 1,
  "username": "username",
  "password_hash": "910909208",
  "email": "testemail@example.com"
}

```

POST ▼ | http://localhost:3000/api/users/login Send ▼

Params Auth Headers (8) **Body** ● Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

```
1 {
2   "username": "username",
3   "password": "passwor"
4 }
```

Body ▼ 🕒 **401 Unauthorized** · 30 ms · 271 B · 🌐 | ⋮

Pretty Raw Preview Visualize JSON ▼ ≡ 🔗 📄 🔍

```
1 {
2   "error": "wrong password"
3 }
```

POST ▼ | http://localhost:3000/api/users/login Send ▼

Params Auth Headers (8) **Body** ● Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

```
1 {
2   "username": "useinam",
3   "password": "password"
4 }
```

Body ▼ 🕒 **401 Unauthorized** · 34 ms · 271 B · 🌐 | ⋮

Pretty Raw Preview Visualize JSON ▼ ≡ 🔗 📄 🔍

```
1 {
2   "error": "User not found"
3 }
```


3.2.3.2. Додавання датчика до акаунта користувача

POST ▼ `http://localhost:3000/api/sensors/addSensorToUser` Send ▼

Params Auth Headers (8) **Body** ● Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

```
1 {
2   "user_id": 1,
3   "sensor_id": 1
4 }
```

Body ▼ ↺ **201 Created** · 60 ms · 324 B · 🌐 | ⋮

Pretty Raw Preview Visualize JSON ▼ ≡ 🔗 📄 🔍

```
1 {
2   "message": "Sensor created successfully",
3   "userSensor": [
4     {
5       "user_id": 1,
6       "sensor_id": 1
7     }
8   ]
9 }
```

POST ▼ `http://localhost:3000/api/sensors/addSensorToUser` Send ▼

Params Auth Headers (8) **Body** ● Scripts Settings Cookies

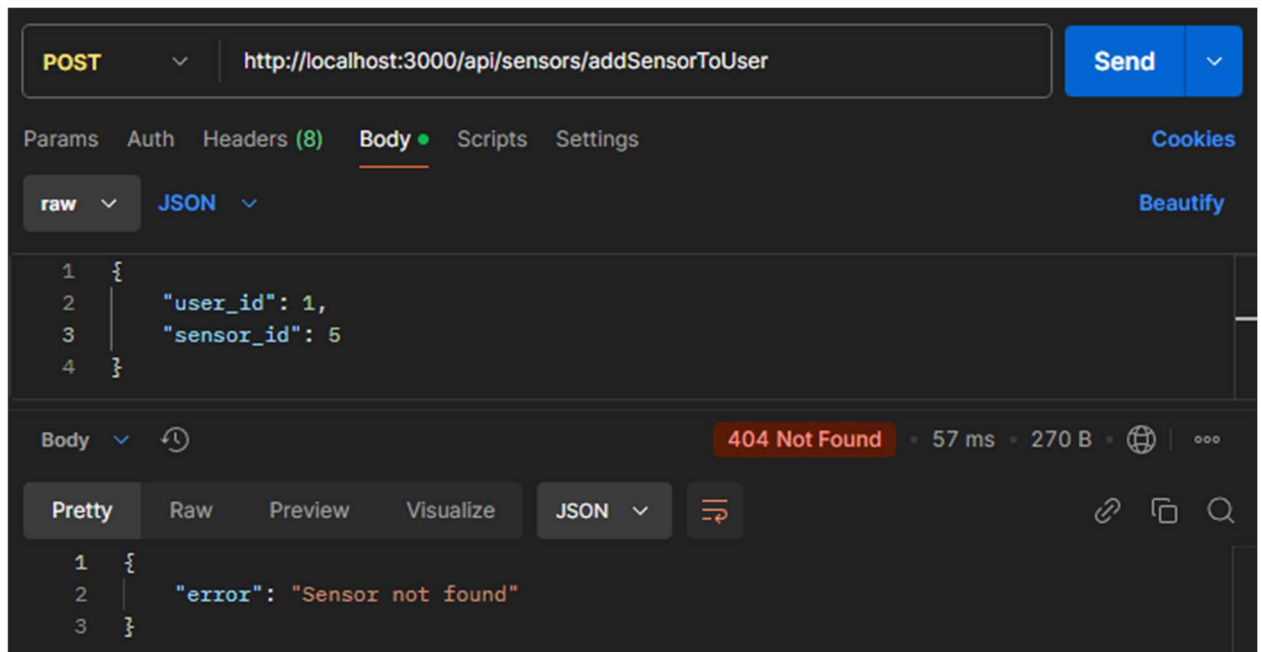
raw ▼ JSON ▼ Beautify

```
1 {
2   "user_id": 4,
3   "sensor_id": 1
4 }
```

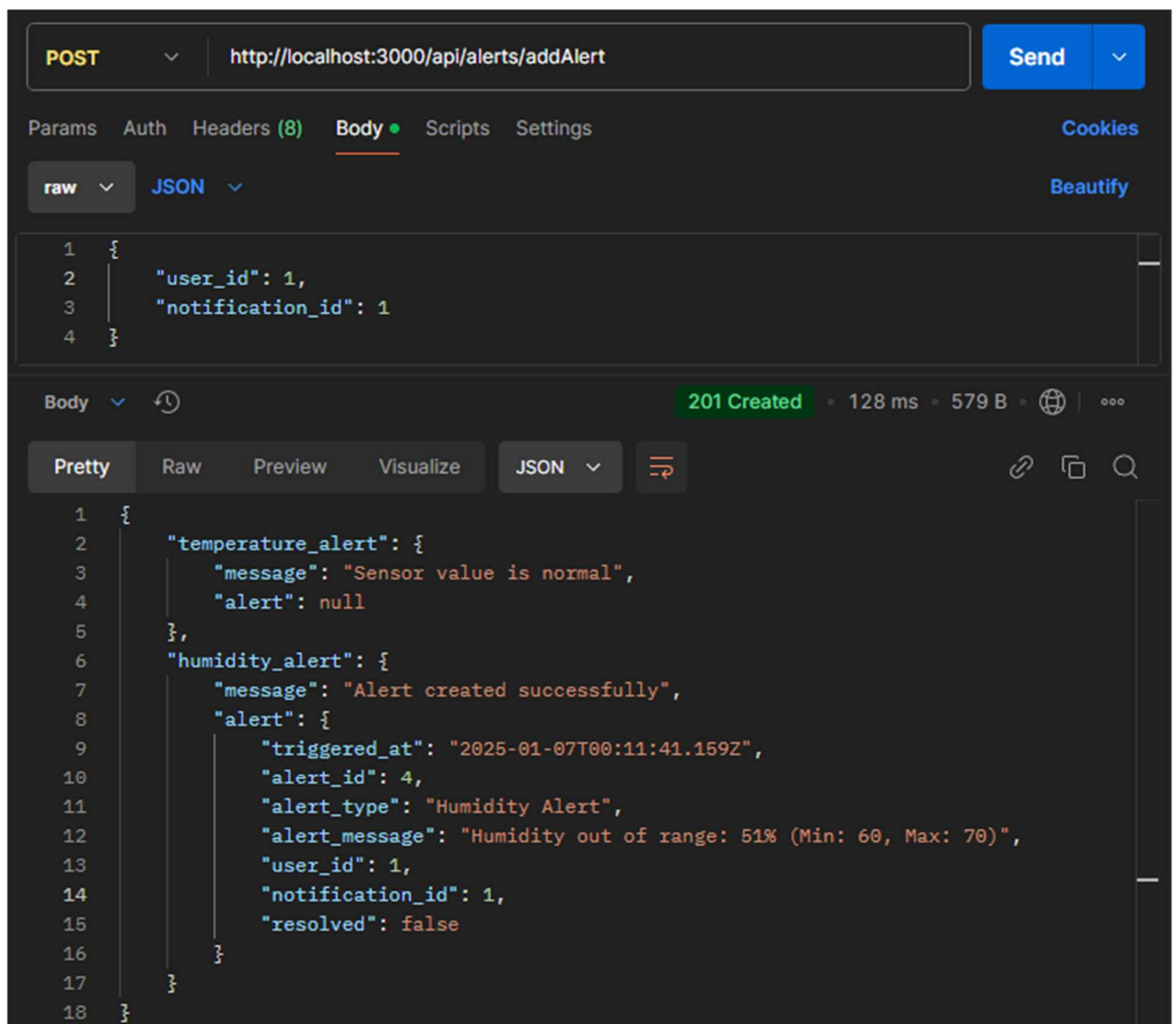
Body ▼ ↺ **404 Not Found** · 30 ms · 268 B · 🌐 | ⋮

Pretty Raw Preview Visualize JSON ▼ ≡ 🔗 📄 🔍

```
1 {
2   "error": "User not found"
3 }
```



3.2.3.3. Автоматичне створення повідомлень при виході даних за межі



POST ▼ http://localhost:3000/api/alerts/addAlert Send ▼

Params Auth Headers (8) **Body** • Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

```
1 {
2   "user_id": 3,
3   "notification_id": 1
4 }
```

Body ▼ 🔄 404 Not Found • 72 ms • 268 B • 🌐 | ⋮

Pretty Raw Preview Visualize JSON ▼ ≡ 🔗 📄 🔍

```
1 {
2   "error": "User not found"
3 }
```

POST ▼ http://localhost:3000/api/alerts/addAlert Send ▼

Params Auth Headers (8) **Body** • Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

```
1 {
2   "user_id": 1,
3   "notification_id": 10
4 }
```

Body ▼ 🔄 404 Not Found • 32 ms • 276 B • 🌐 | ⋮

Pretty Raw Preview Visualize JSON ▼ ≡ 🔗 📄 🔍

```
1 {
2   "error": "Notification not found"
3 }
```

3.2.3.4. Систематичне збереження показників датчиків

The screenshot displays a REST client interface with a PUT request to `http://localhost:3000/api/sensors/updateSensorValues/1`. The request body is a JSON object with `temperature: 8` and `humidity: 62`. The response is a 200 OK status with a response time of 113 ms and a body size of 523 B. The response body is shown in a 'Pretty' JSON format.

Request:

```
PUT http://localhost:3000/api/sensors/updateSensorValues/1
```

Request Body (JSON):

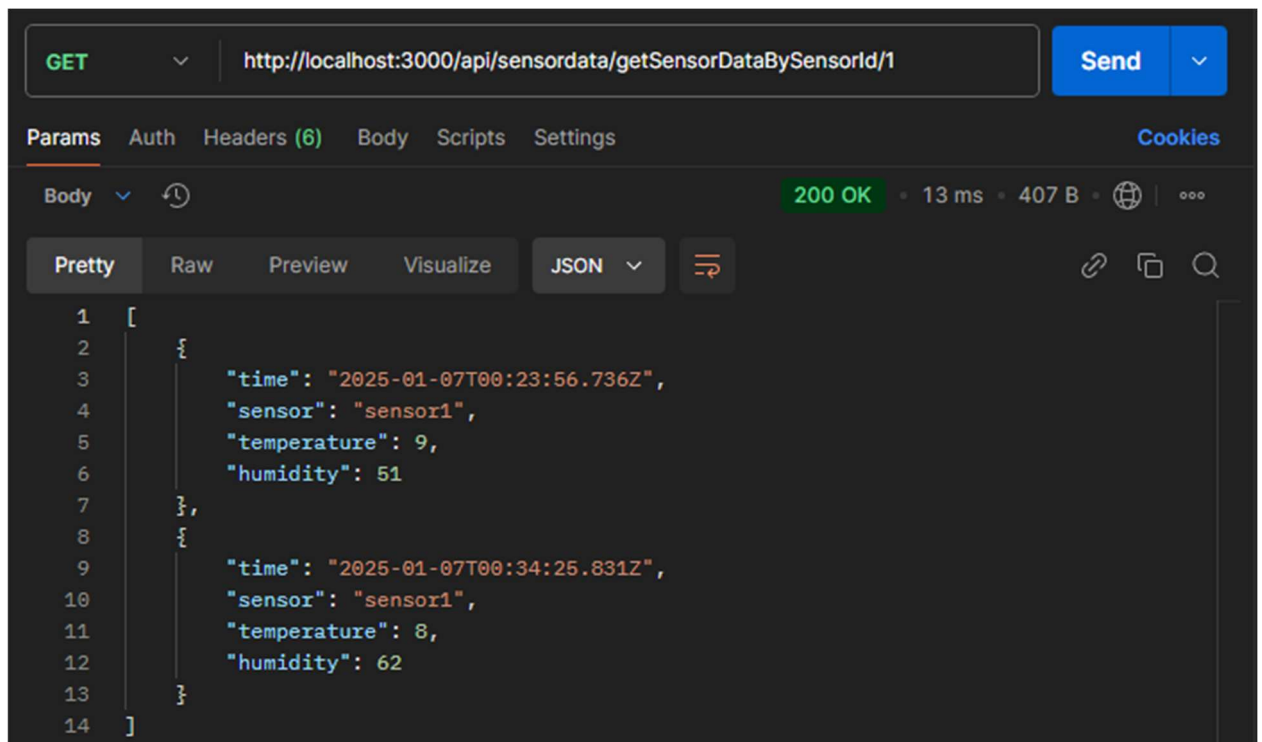
```
{
  "temperature": 8,
  "humidity": 62
}
```

Response: 200 OK • 113 ms • 523 B

Response Body (Pretty JSON):

```
{
  "message": "Sensor values updated successfully",
  "sensor_data": {
    "data_id": 1,
    "sensor_id": 1,
    "record_time": "2025-01-07T00:23:56.736Z",
    "temperature": 9,
    "humidity": 51
  },
  "sensor": {
    "sensor_id": 1,
    "label": "sensor1",
    "device_type": "Датчик температури",
    "temperature": 8,
    "humidity": 62
  }
}
```

3.2.3.5. Взаємодія з історією показників (SensorData)



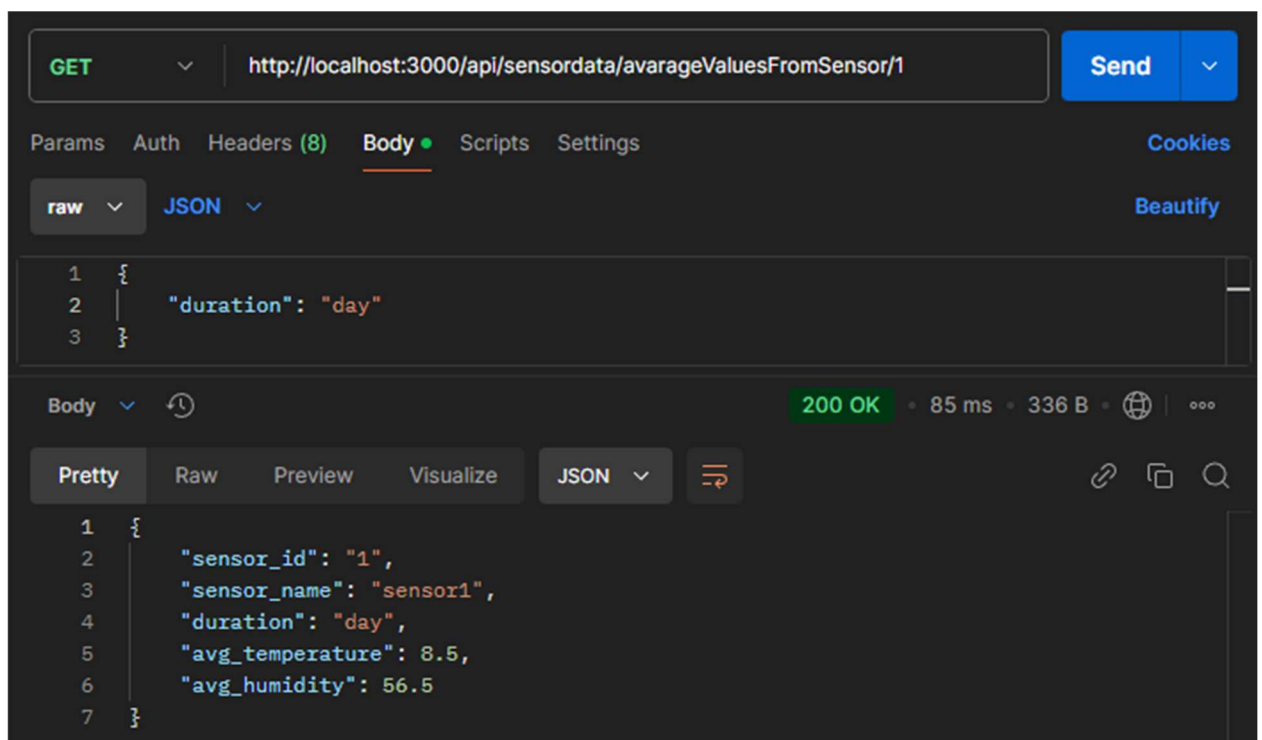
GET <http://localhost:3000/api/sensordata/getSensorDataBySensorId/1> Send

Params Auth Headers (6) Body Scripts Settings Cookies

Body 200 OK · 13 ms · 407 B

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "time": "2025-01-07T00:23:56.736Z",
4     "sensor": "sensor1",
5     "temperature": 9,
6     "humidity": 51
7   },
8   {
9     "time": "2025-01-07T00:34:25.831Z",
10    "sensor": "sensor1",
11    "temperature": 8,
12    "humidity": 62
13  }
14 ]
```



GET <http://localhost:3000/api/sensordata/avarageValuesFromSensor/1> Send

Params Auth Headers (8) Body Scripts Settings Cookies

raw JSON Beautify

```
1 {
2   "duration": "day"
3 }
```

Body 200 OK · 85 ms · 336 B

Pretty Raw Preview Visualize JSON

```
1 {
2   "sensor_id": "1",
3   "sensor_name": "sensor1",
4   "duration": "day",
5   "avg_temperature": 8.5,
6   "avg_humidity": 56.5
7 }
```