

### 3. DEVELOPMENT OF A GREEN ENERGY MONITORING SYSTEM FOR EFFICIENT BALANCING OF THE POWER SYSTEM

#### 3.1 Description of the green energy monitoring system for efficient balancing of the power system

The Green Energy Monitoring System for Efficient Power System Balancing is designed to collect parameters from green energy plants, analyze and visualize them to make it easier to monitor and manage the power system as a whole.

The main tasks for the future green energy monitoring system for efficient balancing of the power system:

- 1) Collect voltage parameters or other data from sensors, sensors, or nodes of green energy power plants (solar power plants, wind power generators, hydroelectric power plants, geothermal power plants, bioenergy plants, etc.);
- 2) Process and route the received data further through the monitoring system infrastructure;
- 3) Collect and store the obtained parameters and metrics;
- 4) Create graphs, schedules, charts, and models based on the obtained parameters;
- 5) Have a modern interface that should not be oversaturated with details and settings, easy to understand and easy to use;
- 6) To have reliable protection against hacker attacks and unauthorized access to the interface and data based on the principles of authorization and authentication.

Figure 3.1 shows the general scheme of the green energy monitoring system.

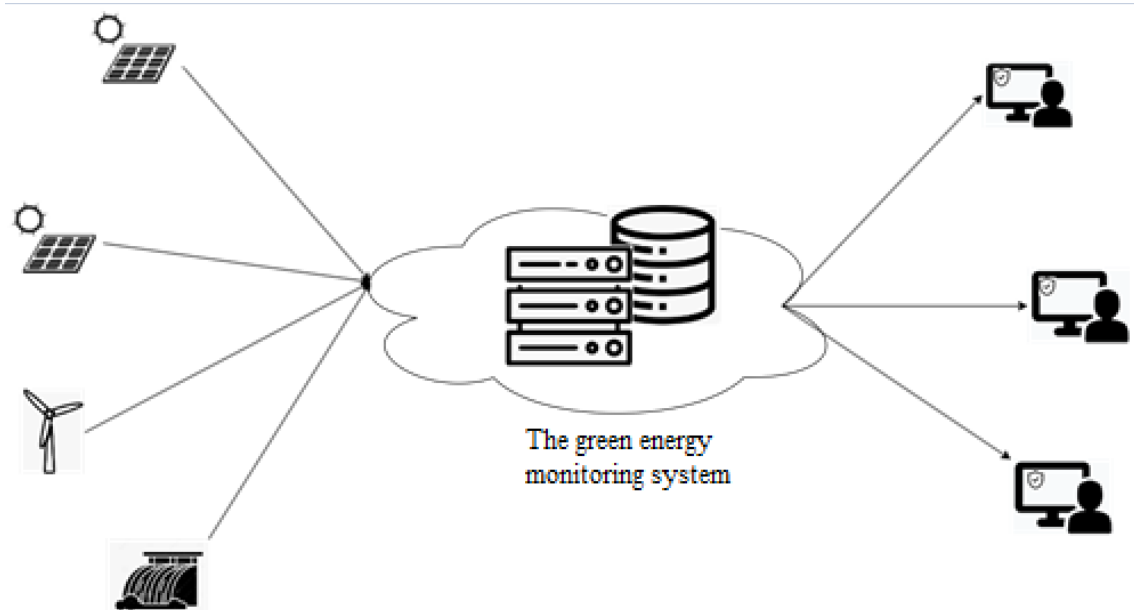


Figure 3.1 - General scheme of the green energy monitoring system

As you can see, the green energy monitoring system collects data from the sensors of green energy power plants. Then it transmits it to remote servers, where it stores it, processes it, and creates visualizations. Users can remotely use the system, monitor parameters, and analyze data.

Figure 3.2 shows a more detailed diagram of the green energy monitoring system using the technologies from Section 2.

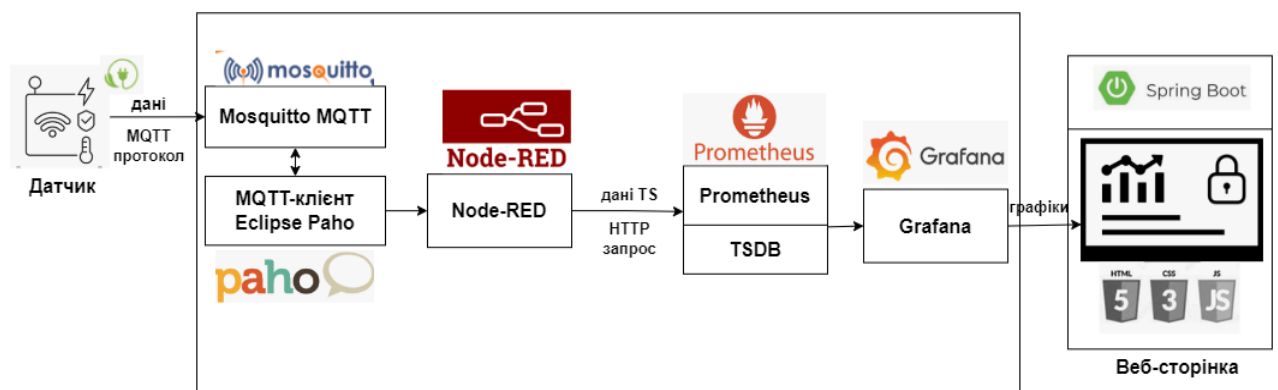
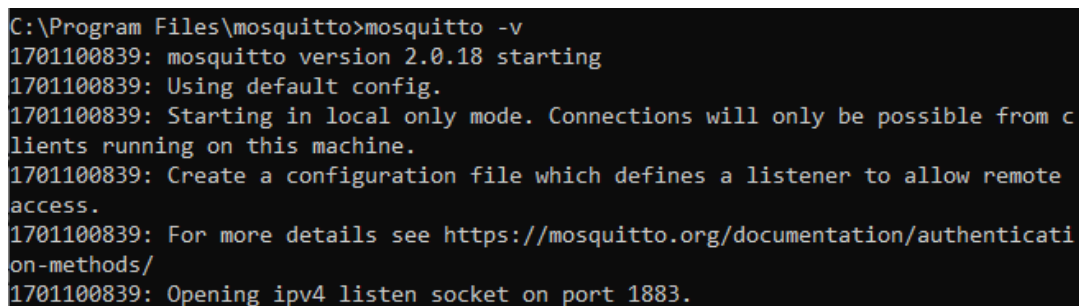


Figure 3.2 - Scheme of the green energy monitoring system with a technology stack

The data from the sensor is processed by the Eclipse Mosquito broker via the MQTT protocol. The Eclipse Paho MQTT client interacts with the MQTT broker and emulates requests from the sensors. Node-RED creates a data stream and transmits an HTTP request to Prometheus, which collects metrics and acts as a TSDB. Grafana visualizes the data and creates graphs. Spring Boot runs the web server and its libraries help protect access through authorization. The web page displays real-time graphs and an interactive user interface using HTML, CSS, and JavaScript programming languages.

### 3.2 Installing and configuring Mosquito

The Eclipse Mosquitto installer for Windows should be downloaded from the official website at <https://mosquitto.org/>. The MQTT broker is launched through the command "mosquitto -v", which must be entered into the Windows command line. Figure 3.3 shows the result of running Mosquitto in the command line.

A screenshot of a Windows command prompt window. The title bar shows the path "C:\Program Files\mosquitto". The command prompt shows the command "mosquitto -v" being executed. The output consists of several lines of text: "1701100839: mosquitto version 2.0.18 starting", "1701100839: Using default config.", "1701100839: Starting in local only mode. Connections will only be possible from clients running on this machine.", "1701100839: Create a configuration file which defines a listener to allow remote access.", "1701100839: For more details see https://mosquitto.org/documentation/authentication-methods/", and "1701100839: Opening ipv4 listen socket on port 1883.".

```
C:\Program Files\mosquitto>mosquitto -v
1701100839: mosquitto version 2.0.18 starting
1701100839: Using default config.
1701100839: Starting in local only mode. Connections will only be possible from c
lients running on this machine.
1701100839: Create a configuration file which defines a listener to allow remote
access.
1701100839: For more details see https://mosquitto.org/documentation/authenticati
on-methods/
1701100839: Opening ipv4 listen socket on port 1883.
```

Figure 3.2 - Running Mosquito in the Windows command prompt

The figure shows that Mosquito is running and listening on port 1883.

Next, you need to check the status of the Mosquitto service. To do this, enter the command "sc query mosquitto" at the command prompt. Figure 3.3 shows the operation of this command.

```

C:\Program Files\mosquitto>sc query mosquitto

SERVICE_NAME: mosquitto
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE                : 4   RUNNING
                                (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE       : 0   (0x0)
        SERVICE_EXIT_CODE   : 0   (0x0)
        CHECKPOINT           : 0x0
        WAIT_HINT            : 0x0

```

Figure 3.3 - Checking the status of the Mosquito service

The status of the service is shown as "RUNNING". This means that the Mosquito service is active.

To check port 1883, run the command "netstat -an | find "1883"". Figure 3.4 shows how the command works.

```

C:\Program Files\mosquitto>netstat -an | find "1883"
TCP    127.0.0.1:1883      0.0.0.0:0          LISTENING
TCP    [::1]:1883         [::]:0             LISTENING
TCP    [::1]:1883         [::1]:65409        ESTABLISHED
TCP    [::1]:65409        [::1]:1883         ESTABLISHED

```

Figure 3.4 - Checking port 1883

Opposite "TCP 127.0.0.1:1883" you can see the inscription "LISTENING". This means that the Mosquito broker is listening to port 1883, so it can receive data from the detectors.

### 3.3 Setting up the Eclipse Paho Java Client and emulating sensor operation

To configure the Eclipse Paho MQTT client, you need to create a project in the Eclipse development environment. Figure 3.5 shows how to create a project.

New Maven Project

Specify Archetype parameters

Group Id: energy

Artifact Id: Energy\_System

Version: 0.0.1-SNAPSHOT

Package: energy.Energy\_System

☒ run archetype generation interactively

Properties available from archetype:

Name	Value
------	-------

Advanced

< Back Next >

Figure 3.5 - Creating an Eclipse Paho project

The name of the project is "Energy\_System". The project is built and compiled using the Maven tool.

After creating the project, you need to add the dependency "org.eclipse.paho" to the "pom.xml" file to connect the Eclipse Paho Java Client. Figure 3.6 shows the created dependency in the "pom.xml" file.

```
20
21
22 <dependencies>
23   <dependency>
24     <groupId>org.eclipse.paho</groupId>
25     <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
26     <version>1.2.5</version>
27   </dependency>
28 </dependencies>
```

Figure 3.6 - Adding a dependency to the "pom.xml" file

The dependency is placed between two "<dependencies>" scripts and contains the group name, version, and name corresponding to the package location of the Eclipse Paho library. Maven automatically downloads the library and updates the project.

Next, you need to create a Java class to connect to the MQTT broker and create

visibility of the sensors. In the created class, you first need to designate the port that the MQTT broker listens to and the names of the sensors. Figure 3.7 shows the port designation and sensor names.

```
public class App
{
    public static void main(String[] args) {
        String broker = "tcp://localhost:1883";
        String clientId = "SensorEmulator";
        String topic = "sensor/data";

        //позначаємо датчики станцій зеленої енергетики
        String hydroStationTopic = "hydro_station_sensor/data";
        String solarStationTopic = "solar_station_sensor/data";
        String windStationTopic = "wind_station_sensor/data";
        String geothermalStationTopic = "geothermal_station_sensor/data";
        String bioStationTopic = "bio_station_sensor/data";
    }
}
```

Figure 3.7 - MQTT port designation of the broker and sensors

The names of the sensors are responsible for hydro, solar, wind, geothermal, and bio stations. They are needed to form the topic of the message.

Next, you need to connect to the MQTT broker using the "MqttClient" class, where the designated port is passed. The "MqttClient" class is responsible for connecting to the broker, delivering messages over the network, and restarting the client.

Next, you need to generate a message with the readings from each station and send it through the set port. The message is sent using the "client.publish()" method. Figure 3.8 shows an example of connecting to an MQTT broker and generating a message from a hydroelectric station.

```
//емулюємо запити від датчиків
try {
    System.out.println("Connecting to broker: " + broker);
    MqttClient client = new MqttClient(broker, clientId);
    client.connect();

    while (true) {
        String message = " ";

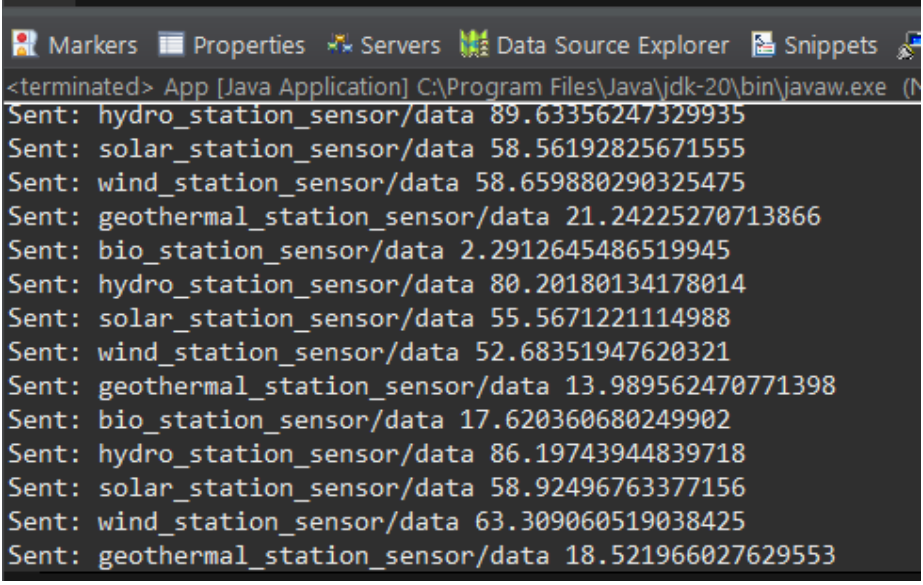
        //HydroStation
        message = sensorData(80, 100);
        MqttMessage mqttMessageHydro = new MqttMessage(message.getBytes());
        client.publish(hydroStationTopic, mqttMessageHydro);
        System.out.println("Sent: " + hydroStationTopic + " " + message);

        //SolarStation
    }
}
```

Figure 3.7 - Connecting to the MQTT broker and generating a message

After generating a message with readings from each station, you need to add "Thread.sleep(5000)" to emulate a 5-second wait before the next data link. The full program code is shown in Appendix A.

After launching the program, the console will start displaying the generated messages from the stations. Figure 3.8 shows the operation of the console.



```
<terminated> App [Java Application] C:\Program Files\Java\jdk-20\bin\javaw.exe (N
Sent: hydro_station_sensor/data 89.63356247329935
Sent: solar_station_sensor/data 58.56192825671555
Sent: wind_station_sensor/data 58.659880290325475
Sent: geothermal_station_sensor/data 21.24225270713866
Sent: bio_station_sensor/data 2.2912645486519945
Sent: hydro_station_sensor/data 80.20180134178014
Sent: solar_station_sensor/data 55.5671221114988
Sent: wind_station_sensor/data 52.68351947620321
Sent: geothermal_station_sensor/data 13.989562470771398
Sent: bio_station_sensor/data 17.620360680249902
Sent: hydro_station_sensor/data 86.19743944839718
Sent: solar_station_sensor/data 58.92496763377156
Sent: wind_station_sensor/data 63.309060519038425
Sent: geothermal_station_sensor/data 18.521966027629553
```

Figure 3.8 - Operation of the console of the Eclipse Paho Java Client program

Notifications from detectors are generated every 5 seconds.

### 3.4 Installing Node-RED and creating a stream

To convert data from MQTT via the HTTP port to Prometheus, you need to use a special tool called Node-RED. To do this, you need to install Node.js and download Node-RED using the command "npm install -g --unsafe-perm node-red".

To start Node-RED, enter the command "node-red" at the Windows command prompt. Figure 3.9 shows the operation of the "node-red" command.

```
C:\Users\ddenc>node-red
11 Dec 17:56:36 - [info]

Welcome to Node-RED
=====

11 Dec 17:56:36 - [info] Node-RED version: v3.1.0
11 Dec 17:56:36 - [info] Node.js version: v18.18.0
11 Dec 17:56:36 - [info] Windows_NT 10.0.19045 x64 LE
11 Dec 17:56:38 - [info] Loading palette nodes
11 Dec 17:56:42 - [info] Settings file : C:\Users\ddenc\.node-red\settings.js
11 Dec 17:56:42 - [info] Context store : 'default' [module=memory]
11 Dec 17:56:42 - [info] User directory : \Users\ddenc\.node-red
11 Dec 17:56:42 - [warn] Projects disabled : editorTheme.projects.enabled=false
11 Dec 17:56:42 - [info] Flows file : \Users\ddenc\.node-red\flows.json
11 Dec 17:56:42 - [warn]

-----
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
-----

11 Dec 17:56:42 - [info] Server now running at http://127.0.0.1:1880/
11 Dec 17:56:42 - [info] Starting flows
```

Figure 3.9 - Running Node-RED in the Windows command line

After that, the Node-RED interface was accessed at "http://localhost:1880".

The Node-RED interface consists of a palette of nodes on the left, a workspace in the middle, and information panels on the right. Figure 3.10 shows the Node-RED interface running at "http://localhost:1880" in an Internet browser.



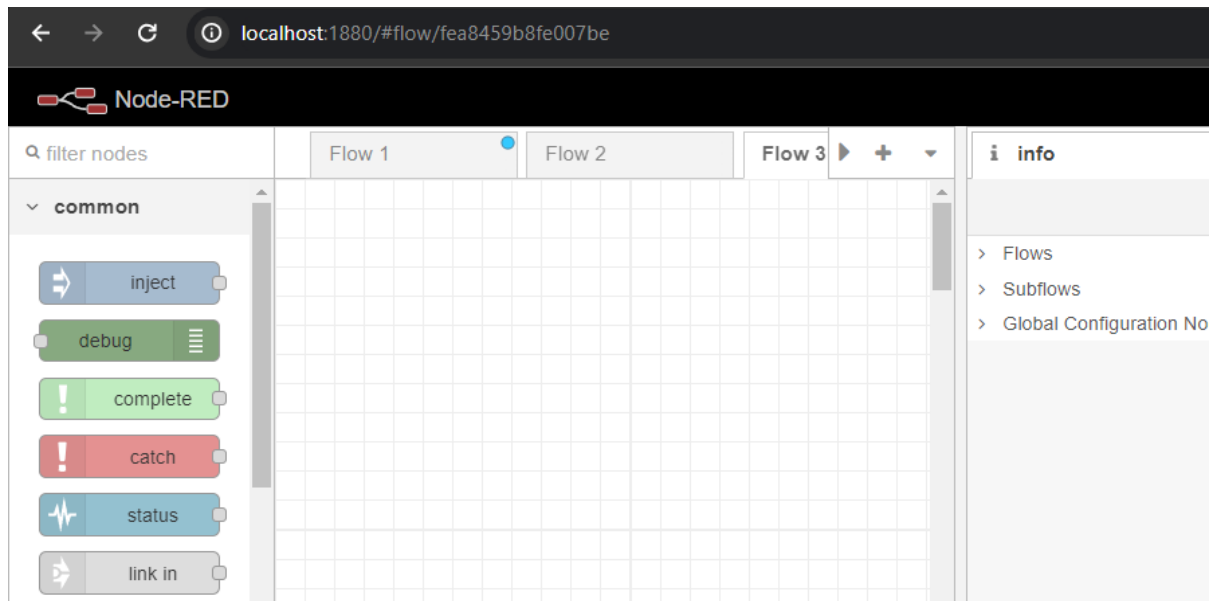


Figure 3.10 - The initial Node-RED interface in an Internet browser

To create a data stream, you need to install the "node-red-contrib-mqtt-broker" and "node-red-contrib-prometheus-exporter" libraries in the "Nodes" panel. They will give you access to the nodes that you need to configure the message flow. Nodes should be moved from the node palette to the workspace and connected to each other.

The "mqtt in" node connects to the MQTT broker and subscribes to messages from the specified topic (thread). It indicates the MQTT port, which for our system corresponds to the server "http://localhost:1883", and the name of the message topic from the detector.

The "function" node acts on messages that arrive at the node using a JavaScript function. It helps to attach a label and form the correct presentation of the message.

The "prometheus out" node sends a message to Prometheus. It contains the name of the metric that corresponds to the name of the sensor.

The "debug" node displays selected message properties in the sidebar tab and the execution log. It is used for tracking thread performance and testing.

To form a message flow, you need to connect the specified nodes together and configure them. Figure 3.11 shows an example of the generated flow for a

hydroelectric power plant.

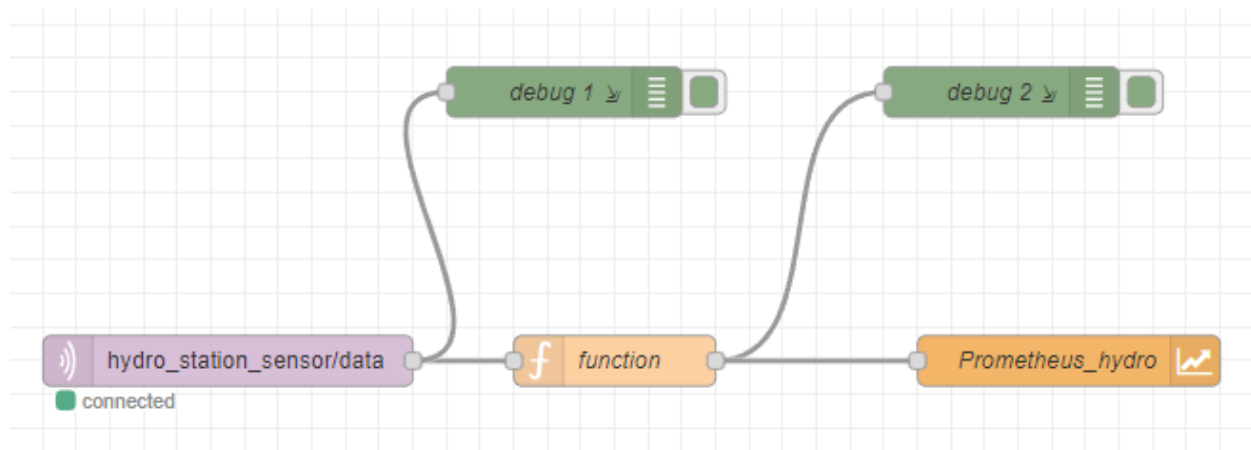


Figure 3.11 - Created message flow in Node-RED for a hydroelectric power plant

The "connected" indicator means that the "mqtt in" node is connected to the MQTT port and can read messages from it.

It is necessary to create and configure message flows for each green energy station, specifying the necessary topical areas and metrics.

To launch the threads, click on the red "Deploy" button located on the left side of the page's navigation bar. It will refresh the interface and launch the message threads. If the launch and connection to external services were successful, a block with the inscription "Successfully deployed" will be displayed.

Figure 3.12 shows the launch of message flows for all green energy stations.

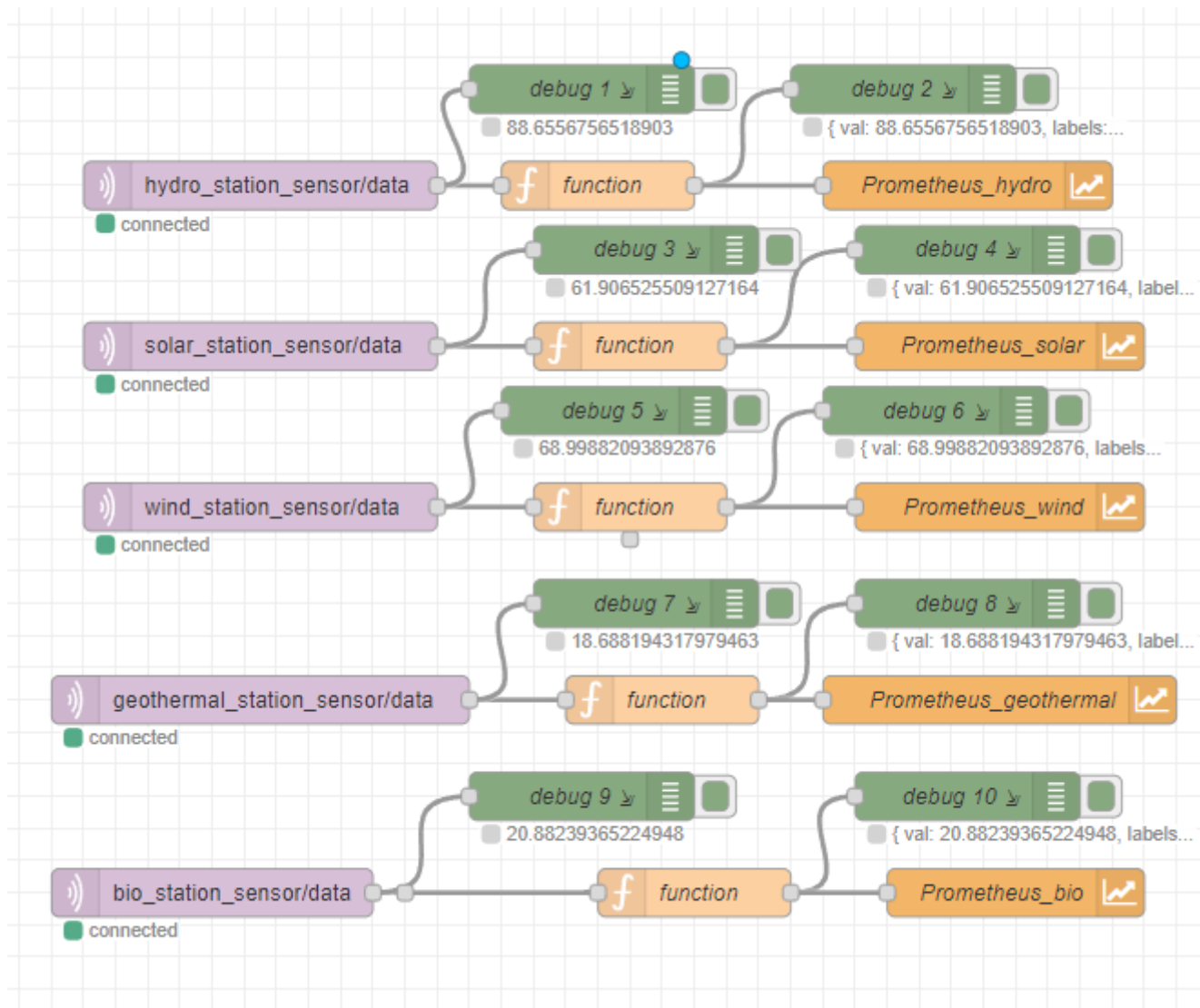


Figure 3.12 - Operation of message flows for green energy stations in Node-RED

Under the "debug" node, indicators for each station are displayed.

### 3.5 Installing and configuring Prometheus

To install Prometheus, go to the official Prometheus website at "<https://prometheus.io/>" and download the latest version for Windows. Unzip the downloaded archive. In the main directory, you need to change the contents of the configuration file "prometheus.yml", specifying the connection to Node-RED. To

do this, add the Node-RED connection port "[localhost:1880]" to the "static\_configs: - targets:" tag. Figure 3.13 shows the changes to the configuration file "prometheus.yml".

```
# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  - job_name: 'node-red'
    static_configs:
      - targets: ['localhost:1880']
```

Figure 3.13 - Changes to the configuration file "prometheus.yml"

To start Prometheus, you need to go to the main directory and enter the command "prometheus.exe --config.file=prometheus.yml" in the Windows command prompt. Figure 3.14 shows Prometheus running at the Windows command prompt.

```
C:\Diplom\prometheus\prometheus-2.48.0.windows-amd64>prometheus.exe --config.file=prometheus.yml
ts=2023-12-11T18:20:19.981Z caller=main.go:539 level=info msg="No time or size retention was set so using the default time retention" duration=15d
ts=2023-12-11T18:20:19.983Z caller=main.go:583 level=info msg="Starting Prometheus Server" mode=server version="(version=2.48.0, branch=HEAD, revision=6d80b30990bc297d95b5c844e118c4011fad8054)"
ts=2023-12-11T18:20:19.983Z caller=main.go:588 level=info build_context="(go=go1.21.4, platform=windows/amd64, user=root@755dcbe41edf, date=20231116-04:38:48, tags=builtinassets,stringlabels)"
ts=2023-12-11T18:20:19.983Z caller=main.go:589 level=info host_details="(windows)"
ts=2023-12-11T18:20:19.983Z caller=main.go:590 level=info fd_limits=N/A
ts=2023-12-11T18:20:19.983Z caller=main.go:591 level=info vm_limits=N/A
ts=2023-12-11T18:20:20.006Z caller=web.go:566 level=info component=web msg="Start listening for connections" address=0.0.0.0:9090
ts=2023-12-11T18:20:20.012Z caller=main.go:1024 level=info msg="Starting TSDB ..."
ts=2023-12-11T18:20:20.016Z caller=tsdb.go:274 level=info component=web msg="Listening on" address=[::]:9090
ts=2023-12-11T18:20:20.018Z caller=tsdb.go:277 level=info component=web msg="TLS is disabled." http2=false address=[::]:9090
ts=2023-12-11T18:20:20.035Z caller=repair.go:56 level=info component=tsdb msg="Found healthy block" mint=1701103848745 maxt=1701122400000 ulid=01HGB74AA0WWR9Q5X6Y2JMHM4
ts=2023-12-11T18:20:20.049Z caller=repair.go:56 level=info component=tsdb msg="Found healthy block" mint=1701167264481 maxt=1701194400000 ulid=01HGC2K3H40MJJN2CG7YE1BTYD
```

Figure 3.14 - Running Prometheus from the Windows command prompt

Prometheus is available at <http://localhost:9090>. To open the Prometheus web interface, you need to go to a web browser and enter this address.

In Figure 3.15, you can see the Prometheus web interface running in a web browser.

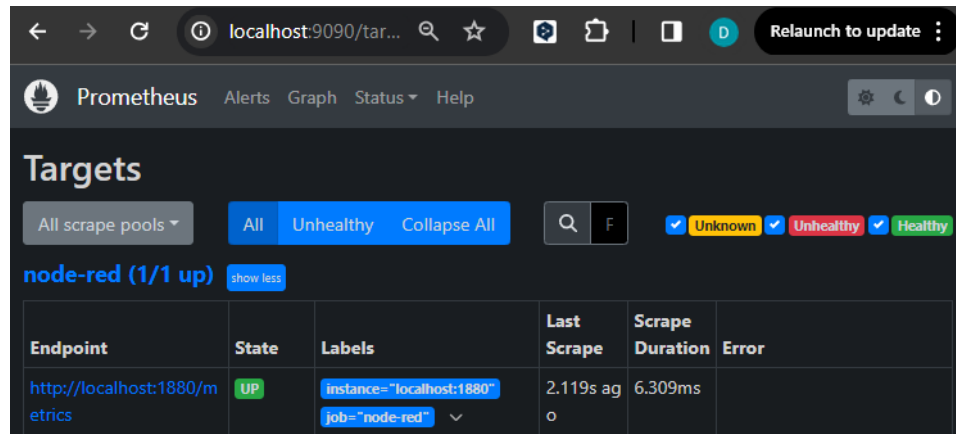


Figure 3.15 - Prometheus web interface

In the "Targets" tab, the server connected to the Node-RED stream has the status "UP". Thus, Prometheus is up and running and can collect and store metrics from green energy stations. Metrics are stored in the form of time series, i.e. they contain data and time of creation.

### 3.6 Installing and configuring Grafana

First, you need to install the Grafana visualization environment on Windows via the official Grafana Installation website. The Grafana interface can be accessed at "http://localhost:3000/". Figure 3.16 shows the Grafana interface in a web browser.

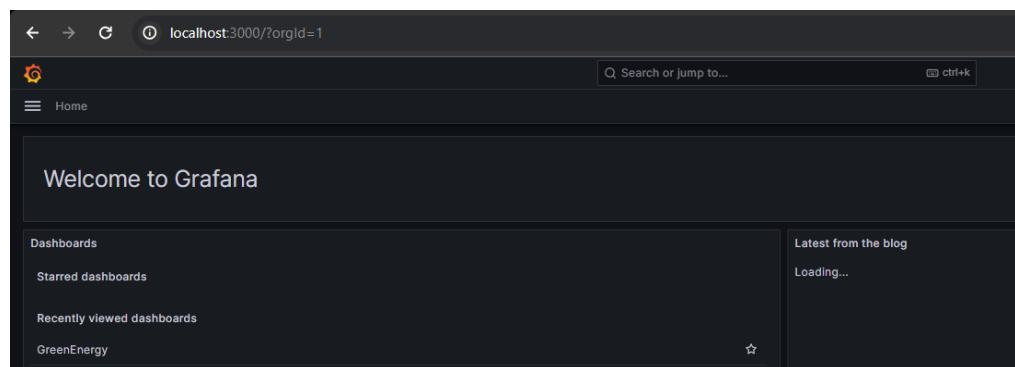


Figure 3.16 - Grafana interface in a web browser

To set up a connection between Grafana and Prometheus, you need to add a Prometheus data source in Grafana. To do this, go to "Configuration", "Data Sources", then click "Add your first data source", select "Prometheus" among the data sources and set the URL "http://localhost:9090/" to Prometheus.

To create graphs, go to the Create tab, Dashboard, add a new graph panel, select Prometheus, and set the Prometheus query in the Metric section.

For the hydroelectric station, you need to select the metric "hydro\_station\_sensor\_data". Figure 3.17 shows the graph creation panel for the hydroelectric station.

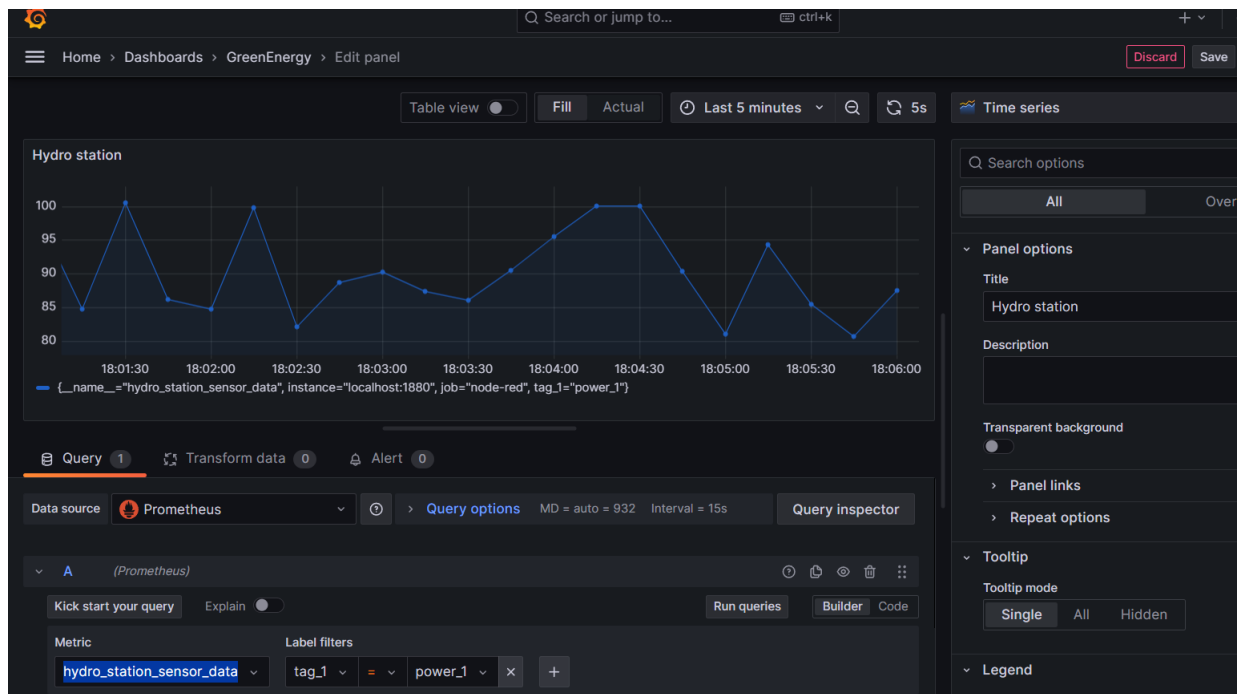


Figure 3.17 - Graph for a hydroelectric power plant in the Grafana environment

You can change the style, width, fill, color, and other customization of the graph. The Grafana environment is distinguished by its ability to improve the visualization of the data obtained. Each metric contains parameters and a date.

To create a shared graph with all green energy stations, you need to create a new graph and add additional metrics from Prometheus in the Metric section. Figure 3.18 shows a shared graph with all green energy stations in the graph creation panel.

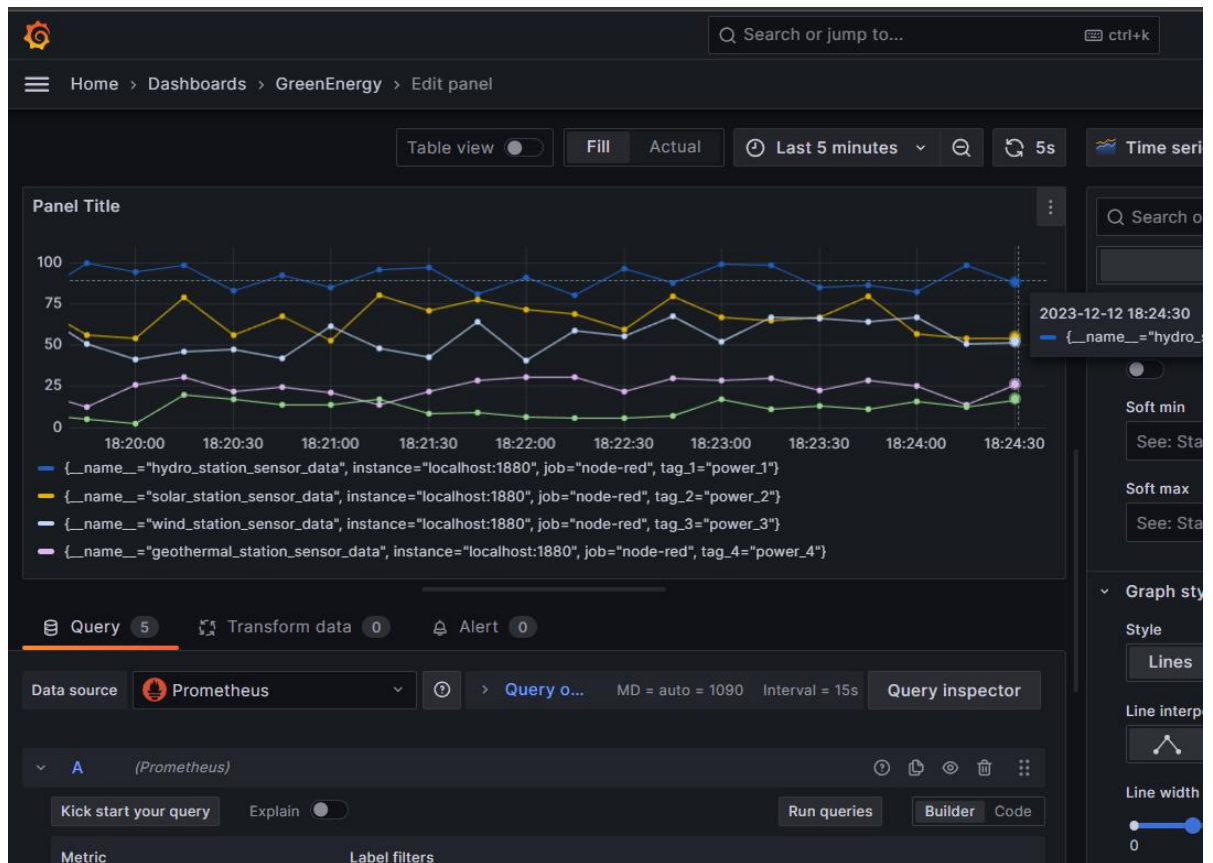


Figure 3.18 - Shared schedule for stations in the Grafana environment

Each station has its own color. Hovering over a metric displays time, data, and other characteristics.

In Grafana, you can specify the time period to be displayed by specifying a start and end time. You can also specify the period of data tracking (5 seconds, 30 seconds, 1 minute, 15 minutes, etc.) and the refresh time.

You need to create graphs for each green energy station by changing the color and indicating the name of the station.

Figure 3.19 shows the graphs for each green energy station and the overall graph in the Dashboard.

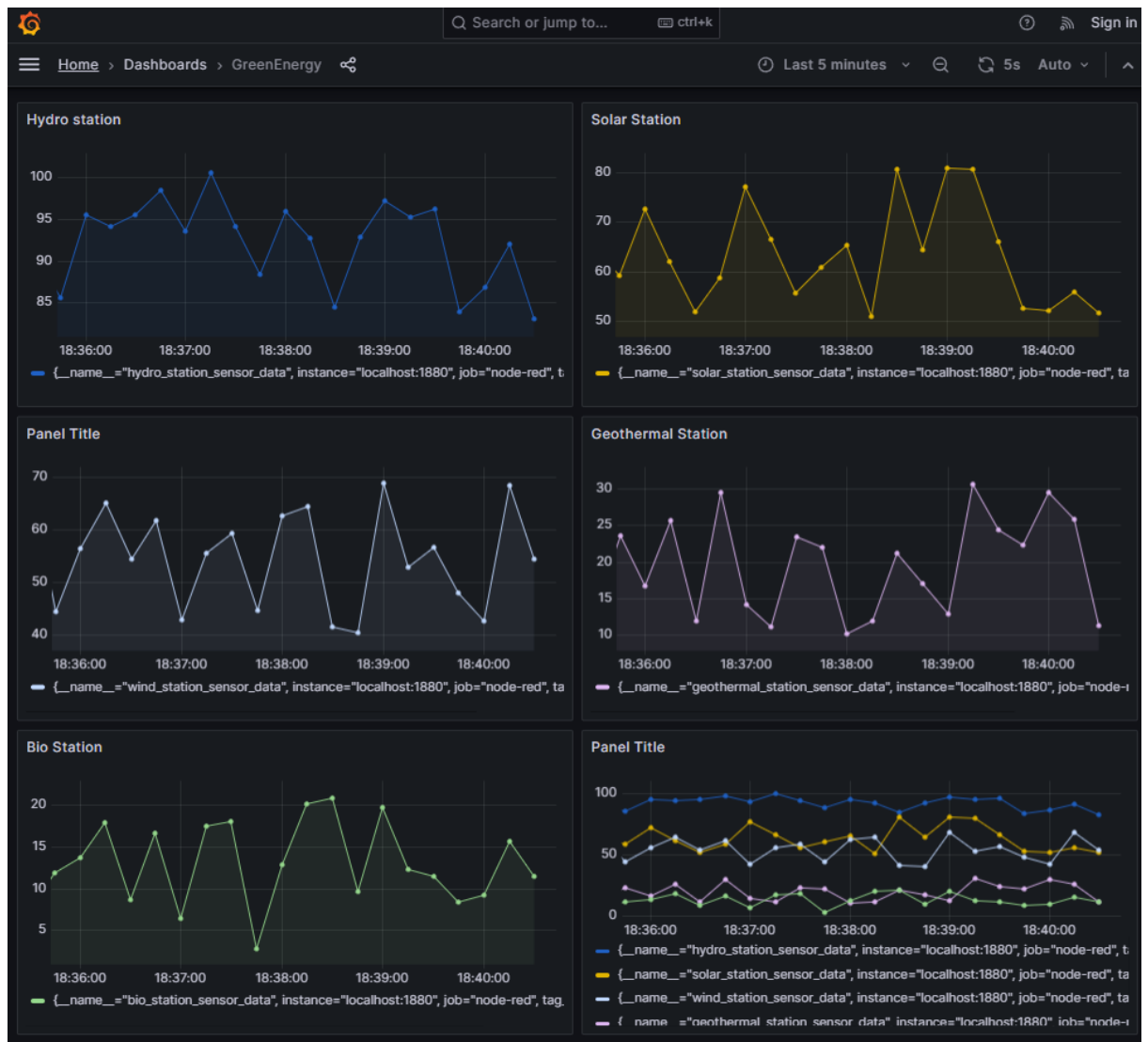


Figure 3.19 - Created graphs for each station in the Grafana environment

The graphs are updated in real time every 5 seconds. Each graph has its own color and contains data from green energy stations.

### 3.7 Setting up Spring Boot and web interface security

To implement the server side of the web interface of the green energy monitoring system, Spring Boot and libraries containing the necessary functions are used. First, you need to create a Spring project "GreenEnergy" based on the Maven project builder and add the dependencies to the "pom.xml" file so that Maven



automatically loads them into the project. Figure 3.20 shows the dependencies in the "pom.xml" file.

```
19      <dependencies>
20      <dependency>
21          <groupId>org.springframework.boot</groupId>
22          <artifactId>spring-boot-starter-thymeleaf</artifactId>
23      </dependency>
24      <dependency>
25          <groupId>org.springframework.boot</groupId>
26          <artifactId>spring-boot-starter-web</artifactId>
27      </dependency>
28      <dependency>
29          <groupId>org.springframework.boot</groupId>
30          <artifactId>spring-boot-starter-security</artifactId>
31      </dependency>
32  </dependencies>
```

Figure 3.20 - Project dependencies

The "spring-boot-starter-thymeleaf" library is used to serve HTML in web applications to reduce code writing. The library

The "spring-boot-starter-web" library is used to add all the libraries needed to create a web application to the project. The "spring-boot-starter-security" library to add security and authorization and authentication capabilities.

The "GreenEnergyApplication" class is the main class and it is where the program starts. Figure 3.21 shows the GreenEnergyApplication class.

```
@SpringBootApplication
public class GreenEnergyApplication {

    public static void main(String[] args) { SpringApplication.run(GreenEnergyApplication.class, args); }

}
```

Figure 3.21 - Class "GreenEnergyApplication"

The "@SpringBootApplication" annotation tells Spring Boot which file to start the application from.

To create error handling, you need to create files with text that will appear

when errors occur. Figure 3.22 shows the code for the "error.html" file, which is written in HTML.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Error</title>
6  </head>
7  <body>
8      <h1>Error, something went wrong!</h1>
9  </body>
10 </html>
```

Figure 3.22 - Code of the file "error.html"

The text "Error, something went wrong!" should appear in case of errors.

Figure 3.23 shows the HTML code of the "404.html" file.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Not found</title>
6  </head>
7  <body>
8      <h1>Page not found!</h1>
9  </body>
10 </html>
```

Figure 3.23 - The code of the file "404.html"

The text "Page not found!" will appear if the user enters the name of a web page incorrectly in the address bar of the web browser.

To make the browser aware of the application's error handling, you need to enter the commands "server.error.whitelabel.enabled=false" and

"server.error.path=/error" in the "application.properties" file, which is responsible for project configuration. These commands will allow you to transfer the responsibility for error handling to Spring Boot.

Next, you need to create a special controller that will regulate error handling. The controller is designed to process requests from the client and return the results. Figure 3.24 shows the "WebErrorController" controller, which is responsible for handling errors.

```
@Controller
public class WebErrorController implements ErrorController {
    @RequestMapping("/error")
    public String handleError(HttpServletRequest request){
        Object status = request.getAttribute(RequestDispatcher.ERROR_STATUS_CODE);
        //Перевірка на статус 404
        if(status != null &&
            Integer.valueOf(status.toString()) == HttpStatus.NOT_FOUND.value()){
            return "404";
        }
        return "error";
    }
}
```

Figure 3.24 - The "WebErrorController" controller

The "@Controller" annotation tells Spring Boot that this class is a controller. The annotation "@RequestMapping("/error")" shows that this controller works with an error request. If the error is from the server or the user, then the browser will load the "error.html" file. And if the error is when the user enters the name of the web page incorrectly, then the browser will display the "404.html" file.

Protecting information and providing secure access to the web interface is an important part of application development. To do this, you need to create a separate class that will be responsible for protecting the application, authorizing and authenticating the system. The class is called "WebSecurityConfig", where the "UserDetailsService" interface implements secure access to the web page and redirection to the authorization page. Figure 3.25 shows part of the code of the

"WebSecurityConfig" class.

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig {

    @Bean
    public UserDetailsService userDetailsService(){
        return new InMemoryUserDetailsManager(
            User.builder()
                .username("admin")
                .password(passwordEncoder()
                    .encode(rawPassword: "admin"))
                .roles("ADMIN")
                .build()
        );
    }
}
```

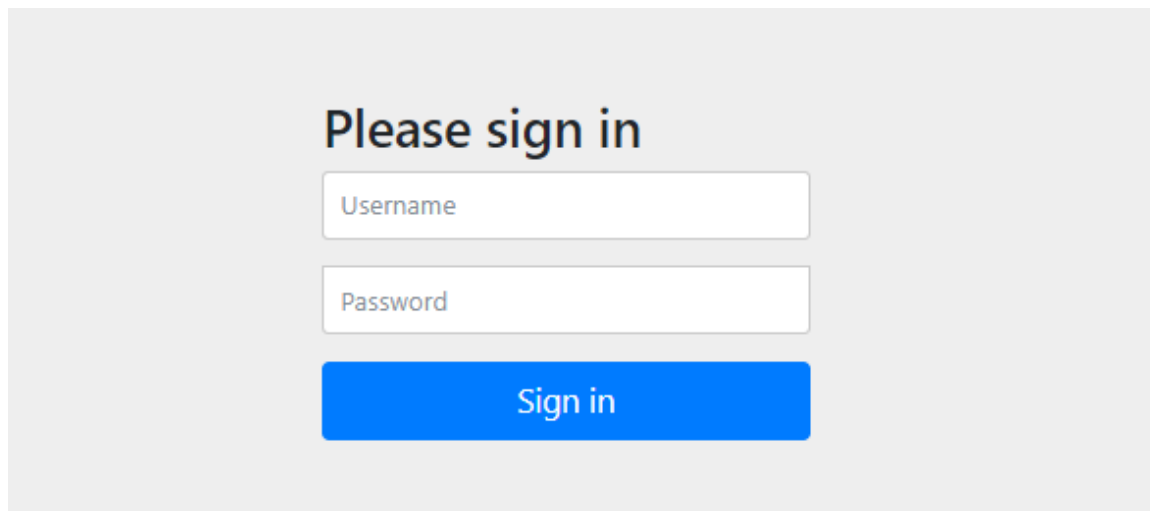
Figure 3.25 - Part of the code of the "WebSecurityConfig" class

The "@Configuration" annotation shows Spring Boot that this is the project configuration, i.e. its settings. The "@EnableWebSecurity" annotation indicates that this configuration is responsible for protecting information. The "@Bean" annotation is used to make an object visible to the entire program.

For priority access, the "userDetailsService()" method describes the "admin" user with administrator access and the same login and password. To add new users, you can connect the database. The password is encrypted in the "passwordEncoder()" method using the "BCryptPasswordEncoder" encoding, which is based on data hashing. The full code of the "WebSecurityConfig" class is in Appendix B.

When a user wants to go to a web page, the web application will redirect them to the authorization page where they will need to enter a login and password. If the entered login or password does not match, then access to the web interface will be blocked.

Figure 3.26 shows the project authorization form created by the Spring Boot security library.



The image shows a simple web form for user authentication. At the top, the text 'Please sign in' is displayed in a large, bold, black font. Below this, there are two input fields. The first field is labeled 'Username' and the second is labeled 'Password'. Both fields have a light gray border and a small blue icon on the right side. Below the input fields is a blue button with the text 'Sign in' in white. The entire form is centered on a light gray background.

Figure 3.26 - Authorization form

In the "Username" line, enter the username (login), and in the "Password" line, enter the password. If everything is entered correctly, access will be allowed and the web application will redirect the user to the web interface.

To configure the request from the user and form the address bar of the web interface, you need to create the "GreenEnergyController" controller. Figure 3.27 shows the implementation of the "GreenEnergyController" controller.

```
8      @Controller
9      @RequestMapping("/greenEnergy")
10     public class GreenEnergyController {
11         @GetMapping
12         public String getIndex(){
13             return "index";
14         }
15     }
```

Figure 3.27 - "GreenEnergyController" controller

The "@RequestMapping("/greenEnergy")" annotation processes the

"/greenEnergy" address bar where the web interface will be deployed. The "@GetMapping" annotation works on the result link during the request. The "GreenEnergyController" controller redirects the request to the "index.html" file, where the web interface is implemented.

Spring Boot allows you to run a web application on a local server. Figure 3.28 shows how to run an application using Spring Boot.

```
INFO 21848 --- [main] c.d.greenEnergy.GreenEnergyApplication : Starting GreenEnergyApplication using Java 20.0.1 with PID 21848 (D
INFO 21848 --- [main] c.d.greenEnergy.GreenEnergyApplication : No active profile set, falling back to 1 default profile: "default"
INFO 21848 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
INFO 21848 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
INFO 21848 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.16]
INFO 21848 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
INFO 21848 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1195 ms
INFO 21848 --- [main] o.s.b.a.w.s.WelcomePageHandlerMapping : Adding welcome page template: index
```

Figure 3.28 - Running the application using Spring Boot

The web interface is now available on port 8080 at the address "http://localhost:8080/".

### 3.8 Development of a user web interface for the system

The user interface of the system should be simple and interactive enough. To implement it, you will need HTML, CSS, and JavaScript programming languages.

First, you need to use HTML to mark up the page, divide it into sections, and add a navigation bar. The navigation bar is created using the <nav> tag for separation, the <img> tag for the logo, and the <a> tags for links to parts of the page or other resources. Sections are indicated by "<div>" tags. Two main sections need to be implemented: the first for separate graphs for each green energy station, the second for the general graph. Adding graphs from the Grafana tool is done using the "<iframe>" tag, where you need to specify a link to a specific graph. The full code of the program can be found in Appendix B.

To add colors, styles, positioning, and sizes, you need to use CSS. Styles are described in the "<style>" tag of the HTML page. A description of the styles for each section, object, and element is provided in Appendix C.

To make your web interface interactive, you need to use the JavaScript programming language. It has many possibilities for building modern and interactive web pages. For example, you can add the ability to show a graph when you hover the cursor over a particular station icon. This is done in order to reduce the oversaturation of the interface with various details and make it easy to use. To do this, use the "'mouseover'" command. Figure 3.29 shows the implementation of the mouseover event.

```
130 // Додаємо обробник події при наведенні курсору
131 stationElement.addEventListener('mouseover', () => {
132     // Створюємо контейнер для графіка
133     const graphContainer = document.createElement('div');
134     graphContainer.className = 'graph-container';
135
136     // Створюємо iframe із графіком Grafana
137     const iframe = document.createElement('iframe');
138     iframe.src = station.grafanaUrl;
139     iframe.width = "450";
140     iframe.height = "200";
```

Figure 3.29 - Implementation of the event on cursor hover

When the mouse cursor is over the green energy station icon, the "'mouseover'" script is triggered, which creates an "iframe" element that displays the graph from Grafana.

The JavaScript script also provides a link to the Grafana graphs for each station, their sizes, and icons. All figures and icons are located in the "images" folder of the project directory. You need a link to the images to use them. The JavaScript code is located between the "<script>" tags in the HTML markup. The full script code is provided in Appendix B.

Figure 3.30 shows the implementation of the web interface of the green energy monitoring system for efficient balancing of the power system.



Figure 3.30 - Implementation of the web interface

If you hover the mouse cursor over the station icon, a graph from Grafana with real-time sensor readings will appear next to it. This works with every station. In the figure, you hovered over the solar station icon, so the graph of the solar station's sensor is displayed on the right.

At the top, there is a navigation bar with the GreenEnergy logo and links to the main section with all the stations, the section with the general schedule, and various services and resources.

To display a general graph showing the operation of all sensors simultaneously to visualize the entire power system, the "Total" section was created. Figure 3.31 shows the general graph and the "Total" section.



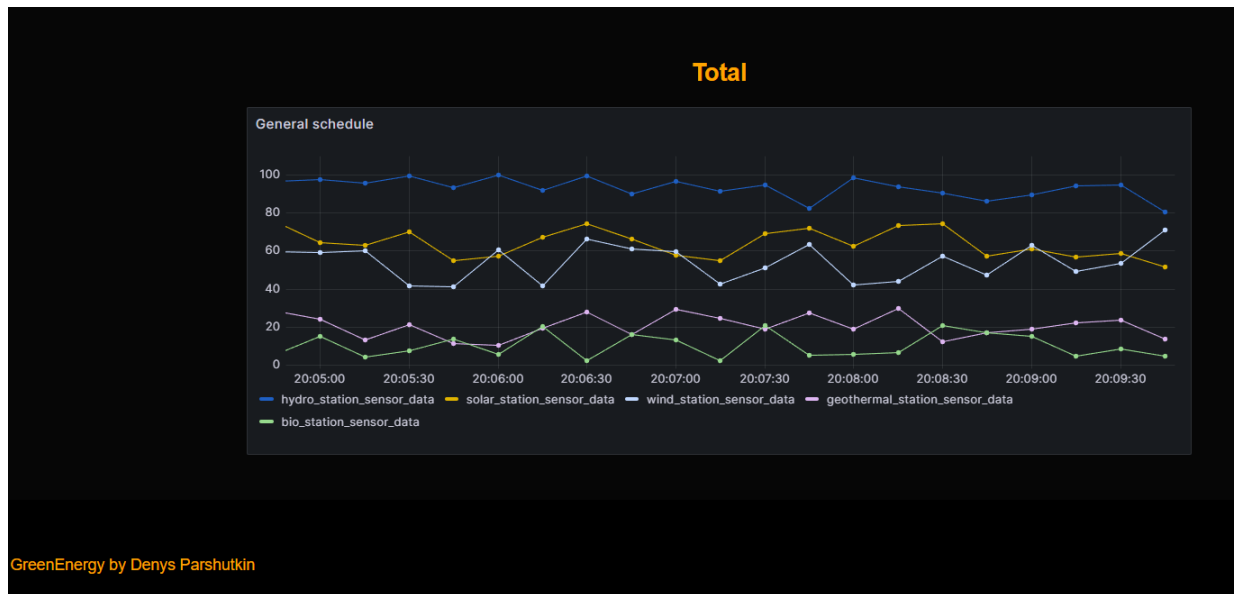


Figure 3.31 - General graph in the "Total" section

The graph shows the real-time readings of the sensors of green energy stations.

Thus, the user interface of the green energy monitoring system meets the criteria of simple interactivity.