SMART RIVER MONITORING REPORT

The smart river monitoring system has been set up as depicted in Figure 1. Details on the components of the two circuits can be found in the respective schematics files.

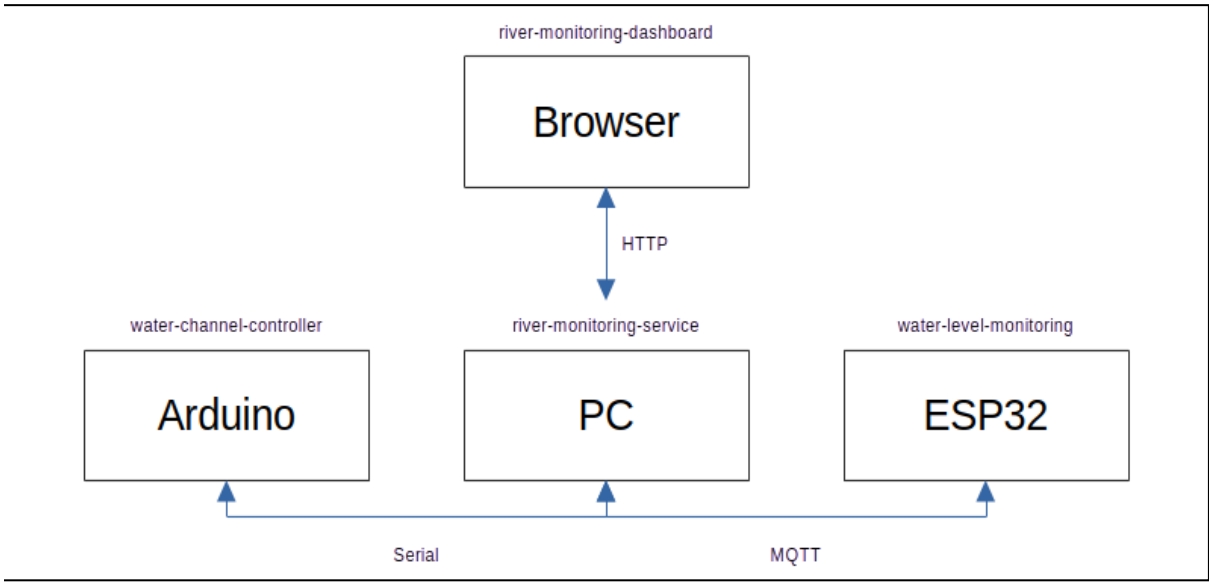

Figure 1: System overview.

Water Level Monitoring

The ESP serves to monitor the water level using a sonar sensor. Internally, it follows a simple super loop. In each iteration, it first checks if it is connected to the Internet. Next, it measures the current distance to the water (or any other surface in front of the sensor) and reports it back to the River Monitoring Service. To do so, it is connected with an MQTT broker[1]. Water level updates are sent to the topic “esiot-2023-smart-river-monitoring-45983” as a simple String containing the measured distance.
Because the system state is maintained in the River Monitoring Service, there had to be a communication channel to the ESP that can remotely update the measurement frequency if necessary. We used a second topic (“esiot-2023-smart-river-monitoring-58329”) to semantically differentiate between the messages. Frequency updates (either “F1” or “F2”) are handled by the ESP inside a callback method that is triggered on any loop iteration, presuming a message has previously been sent to the broker.
Lastly, the ESP displays its current connection status using two LEDs.

1 For this assignment, we chose to use the MQTT broker instance broker.mqtt-dashboard.com. Due to the fact that we did not transmit any sensitive data, this was a simple solution. Alternatively, we could have deployed a MQTT server using Verticle or a Docker container. In this case, we would have had to expose our local MQTT port via ngrok.

Table 1: API endpoints of the HTTPServer.

Method	Path	Purpose
GET	/api/data	Retrieve the current internal state (measurements, state, valve opening level).
POST	/api/data	Update the valve opening level (DASHBOARD mode).
POST	/api/mode	Toggle DASHBOARD mode (only in AUTOMATIC/DASHBOARD mode).

River Monitoring Service

The River Monitoring Service is the central element of our architecture. It maintains connections to all three subcomponents and keeps an internal state. Data is stored in a Singleton class “DataStore” that synchronizes all accesses. This is important because the message agents are executed in three parallel threads and could lead to situations where two parts of the system want to update the same data.
The three parts of the River Monitoring Service act as follows:

- The **MQTTAgent** is connected to the MQTT broker and, thus, communicates with the ESP board. It was implemented as a Verticle and executes a callback function once a message is available. Incoming messages contain water level values that have to be evaluated according to the different thresholds (W1–W4) to adjust valve opening levels, frequencies, and alarm states accordingly. The new measurement frequency is sent to the ESP whereas the other data is persisted in the DataStore. All of this is only executed when the system is in AUTOMATIC mode.
- The **HTTPServer** provides a simplistic REST API on http://localhost:8080. Three routes allow to retrieve the current state, to update data from the dashboard, and to toggle the DASHBOARD mode (see Table 1). This API is used by the River Monitoring Dashboard. If the dashboard requests to activate DASHBOARD mode, a flag is set in the DataStore. In each cycle of the SerialAgent, this flag is read and, if it is set, the information about entering the DASHBOARD state is once propagated to the Water Channel Controller. This can be seen as a simplistic BlockingQueue.
- The **SerialAgent** serves to communicate with the Water Channel Controller via the Serial Line (baud rate 9600). Its protocol is explained in Table 2. To implement the polling, the classes from the lecture have been reused with one alteration. In the method CommChannel#receiveMsg(), we replaced BlockingQueue#take() by the non-blocking method BlockingQueue#poll(). Thus, if no message arrives after 10 milliseconds, the SerialAgent’s main loop continues and other messages from the ESP or the dashboard can be handled, restoring the system’s responsitivity.

Table 2: Messages sent and received by the SerialAgent.

Direction	Message	Purpose
received	ANGLE <angle>	Update the valve opening level (MANUAL mode).
received	MANUAL	Switch to MANUAL mode (AUTOMATIC mode).
received	AUTOMATIC	Switch to AUTOMATIC mode (MANUAL mode).
sent	NEWANGLE <angle>	Update/confirm valve opening level.
sent	DASHBOARD	Inform about DASHBOARD mode.

Internally, the River Monitoring Service operates based on a simple state machine (see Figure 2). In AUTOMATIC mode, all operations are carried out as described in the assignment document. The service computes valve opening levels and measurement frequencies according to pre-defined limits and instructs the Water Channel Controller accordingly. From AUTOMATIC mode, it is possible to either enter MANUAL mode (by pressing the button in the Water Channel Controller) or DASHBOARD mode (by clicking on a button in the dashboard). The same action has to be carried out to return to AUTOMATIC mode.

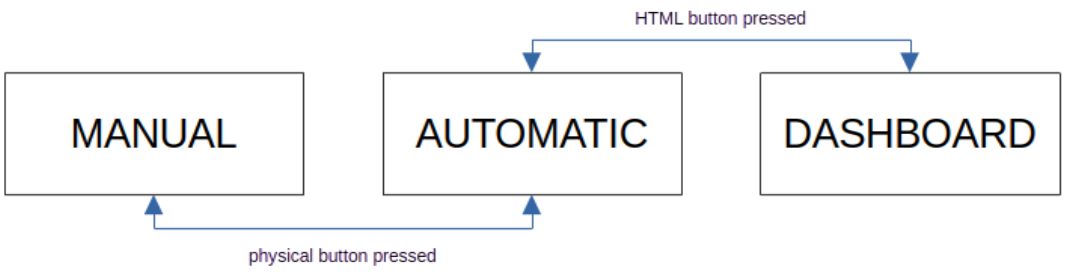

Figure 2: Internal state machine of the River Monitoring Service.

River Monitoring Dashboard

The River Monitoring Dashboard is a Node application using the Svelte framework. It fetches new data from the River Monitoring Service every 500 ms and displays it (see Figure 3). Data points are visualized in a SVG-based diagram. When in AUTOMATIC mode, the dashboard allows to change to the DASHBOARD mode and to control the valve opening level using the slider in the top right corner.

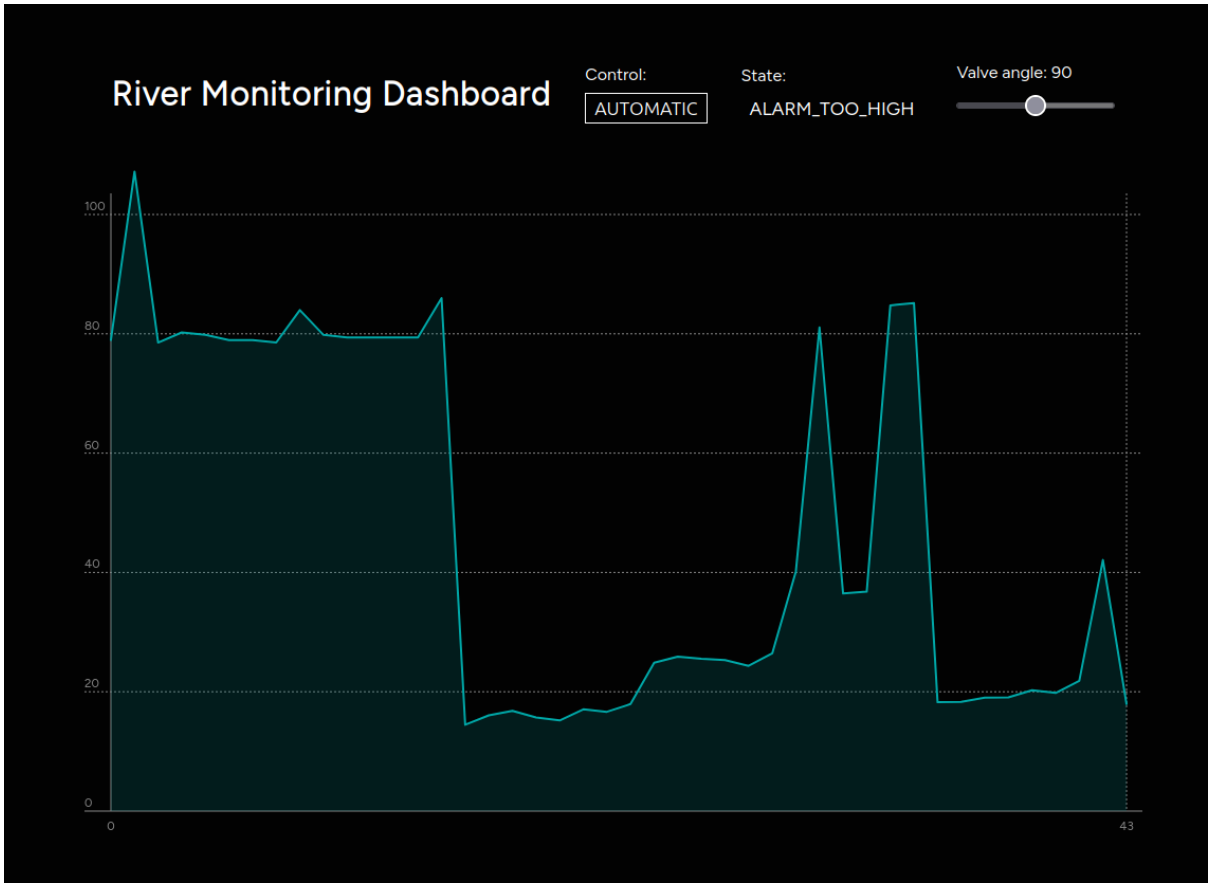

Figure 3: The River Monitoring Dashboard.

Water Channel Controller

The water channel controller uses a simple super loop. It maintains the same inner state as the River Monitoring Service. Depending on the current state, either the AUTOMATIC task is executed (receiving a new angle, instructing the servo motor, and displaying it on the display, or switching to DASHBOARD mode), or the MANUAL task (reading the potentiometer value and sending it to the River Monitoring Service), or the DASHBOARD task (analogously to the AUTOMATIC task, but switching back to AUTOMATIC mode if instructed). Regardless of the current state, we listen for button presses to switch between AUTOMATIC and MANUAL mode.

VIDEO LINK:
https://cloud.florianknoch.de/s/3DWF9CMPW7kne8P