Important rules and remarks:

- You are allowed to use any source of information; **communication with third parties is forbidden**
- You must be able to explain your code and you must be able to perform modifications if asked; otherwise the exam is not passed
- If there is anything unclear, **ask** us
- You must use the technologies from the lab/lecture
- There are NO PARTIAL points, only the ones specified in the problem statement
- **The requirements MUST be solved in the order they appear in the problem statement; the points for any feature are given only if all previous features are completely correct**
- **after solving each GROUP of features notify the teachers for evaluation; you must present each GROUP of feature before proceeding to the next one; the GROUPS of features are the following:**
    - **F1, F2**
    - **F3**
    - **F4, F5, F6**
    - **F7, F8, F9**
    - **F10**
    - **F11**

Factory Equipment Monitor

In a factory, several sensors are attached to different pieces of equipment and take various measurements regarding the way they work. All sensors are continuously sending data (sensor info, measurement info etc) to a server using tcp sockets.
The tcp-socket-server processes the received data and saves the processed data to a database.
A factory employee may see the data from each sensor in a web interface and may take appropriate actions.
Write an application that simulates this process. The project will be a modular one and will be composed of the following modules: sensor-app, server-app, monitor-core, monitor-web. There will also be a *webapp* (Angular) that may or may not be part of monitor-web.

The **sensor-app** reads the following data from the console: *sensor-name* (string, assumed unique), *sensor-id* (int, assumed unique, *should* be greater than 1023), *lower-bound*, *upper-bound*. Then the sensor-app will continuously generate numbers in the interval [*lower-bound, upper-bound*) representing measurements of a certain piece of equipment and, after each measurement, the following information is sent to the server-app using tcp-sockets: *sensor-name, sensor-id, measurement.*
The **server-app** concurrently processes information from sensors (for demonstration at least two sensors will be used, *but the entire application should work without changes for any number of sensors, i.e., no hardcoding regarding the number of sensors*).

The server prints in the console the data received from each sensor: *sensor-name, sensor-id, measurement.* Before sending the data to the server, the sensor simulates a delay: Thread.sleep(new Random().nextInt(...)). In the same fashion, the server simulates a delay in processing each client. **[F1 25p]**
For each received measurement, the **server-app** saves only the following information in a database table called *sensor*: *sensor-name, measurement, time*. The *sensor* table has an id which is automatically incremented. The *sensor-id* is not saved in the database. The *time* represents the system current time in milliseconds. **[F2 30p]**

At the address http://localhost:4200/all all information from the *sensor* table will be displayed, one item per line, using a rest-api exposed by the **monitor-web** module, which uses the **monitor-core** module in order to get the sensor data from the database using SpringDataJpa repositories (architecture from the same). **[F3 30p]**

When visiting the address http://localhost:4200/sensors the following elements are shown **[F4 30p]**:

- A list of sensor names (one *sensor-name* per line); each name appears only once
- A “Show” button, below the list

Only the *sensor-names* (each name, once) are fetched from the backend **[F5 30p]**.
In the backend, there will be no custom implementation, i.e., no processing in the service (only delegate to repository); at the repository level, only interfaces will be used **[F6 30p]**.

When pressing the “Show” button, at the same address, the following elements are shown for each sensor:

- A label “Sensor name: ”, followed by the <sensor-name>, followed by a “Stop” button (same line)
- A list with data corresponding to <sensor-name> (id, sensor-name, measurement, time); the list is sorted in descending order by time; only 4 elements are shown **[F7 35p]**

The elements from above are dynamically created according to the number of sensors **[F8 35p]**.
For fetching the data, only the following rest-api endpoint in used: /sensors/{name} and there is no custom code for data processing in the backend, i.e., no processing in the service (only delegate to repository) and at the repository level, only interfaces; the backend returns the needed 4 items, as specified **[F9 35p]**.
The sensor data (from each list) is updated automatically at regular intervals as new data is produced by the sensor-app (page reload not allowed) **[F10 35p]**.

When pressing the “Stop” button from any sensor, the **server-app** is notified (a message is printed in the **server-app** console) and the corresponding **sensor-app** finishes execution (being notified by the **server-app**) **[F11 45p]**.