1 VERSION HISTORY

v2.0 Revised version, 26-09-2024

Carefully **read** this document. Please talk to the teacher about any unclear or ambiguity you find so that this document can be revised beforehand.

2 INTRODUCTION

The project is part of the course IDATA2304 Computer Communication and Network Programming examination. It is a group-activity. This project intends to learn network programming in practice. Students are creating a custom solution for a realistic situation.

3 THE SCENARIO

You work at a company called "SMG" (Student-Made Gadgets) which manufactures smart systems for farmers. From the user's perspective, the systems consist of sensor/actuator nodes and control-panel node(s), see Figure 1.

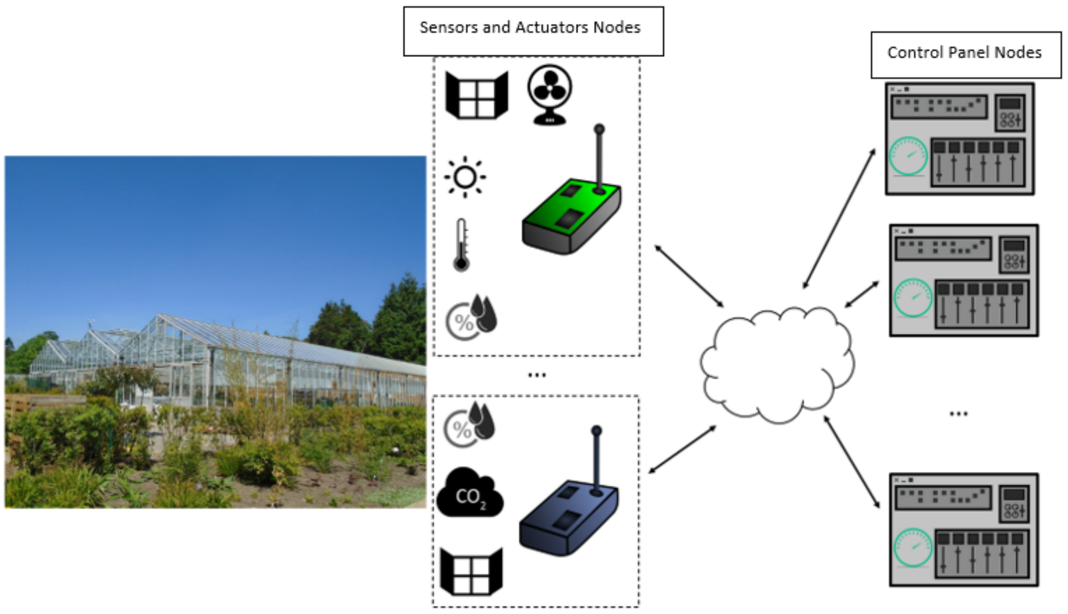


Figure 1: The overall architecture of the system, from the user perspective: Sensor and actuator nodes located on a greenhouse, with remote control nodes communicating over the Internet

A farmer would typically buy and install several sensor/actuator nodes, place them in their environment (such as greenhouse). They will then connect them to electricity and the Internet. They would also make necessary connections to their infrastructure: connect the necessary actuator node pins to switches opening windows, turning on fans, open the valve of shower etc. Each sensor/actuator node can have different sensors attached to it: temperature, light, humidity, fertilizer (Nitrogen), PH, wind speed etc. Farmers choose the sensors when they purchase the nodes. In addition, each node also provides control capabilities through actuators, such as turning on a heater, opening a window, turning on a fan, etc.

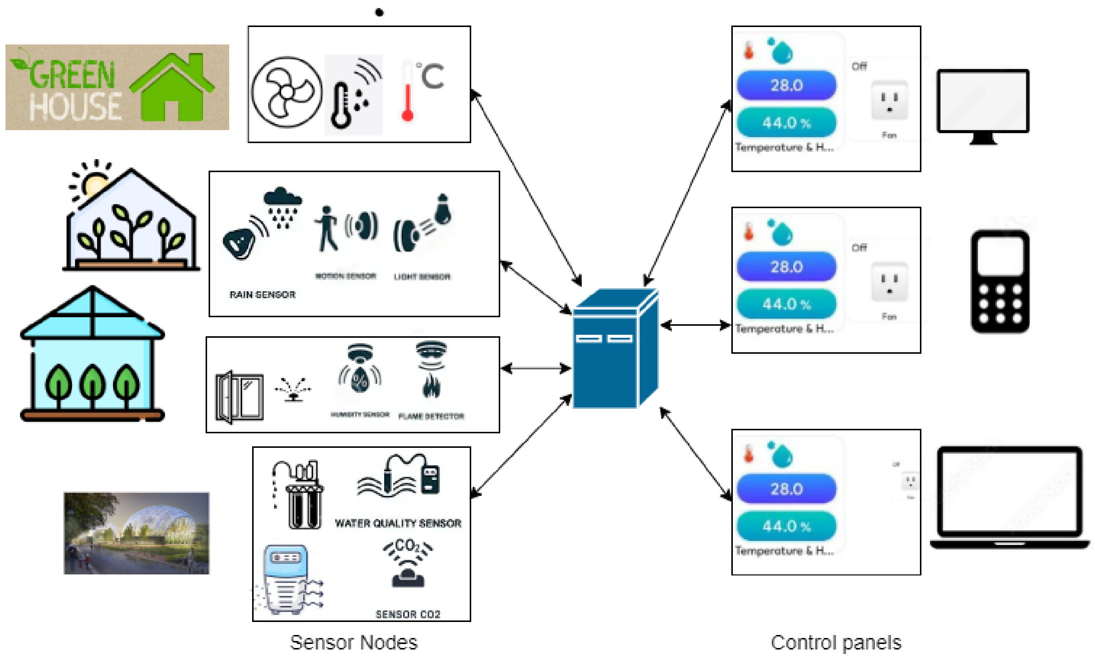


Figure 2: Sensor nodes and control panel nodes

There are also control-panel nodes, owned either by the farmers or other users who need insight into the farms. The control nodes have an overview of sensor data and capabilities to control the actuator nodes.

3.1 HARDWARE SIMULATION

You don't need to worry about the hardware in this project. No wires, soldering, no analog sensor bugs, etc. You need to work with simulated sensor nodes and control-panel nodes.

4 THE TASK OF THE PROJECT

Your task in this project is **to implement a meaningful application** and its components for a complete smart farming solution. The solution includes communication with sockets. You need to **design your application-layer communication protocol** and **implement necessary communication** between the nodes so that the control-panel nodes can visualize sensor data and control the actuators on the sensor nodes.

You get a template code of a sensor/actuator node which generates simulated sensor data (temperature, humidity) and has a simple interface for controlling the actuators (opening a greenhouse window, turning on a heater). The provided template is a JavaFX application. You also get a template of a control-panel node, which has a user interface for sensor data visualization and controls. The visualizations and controls are not wired to any real sensors or actuators. You can download the template code from here.

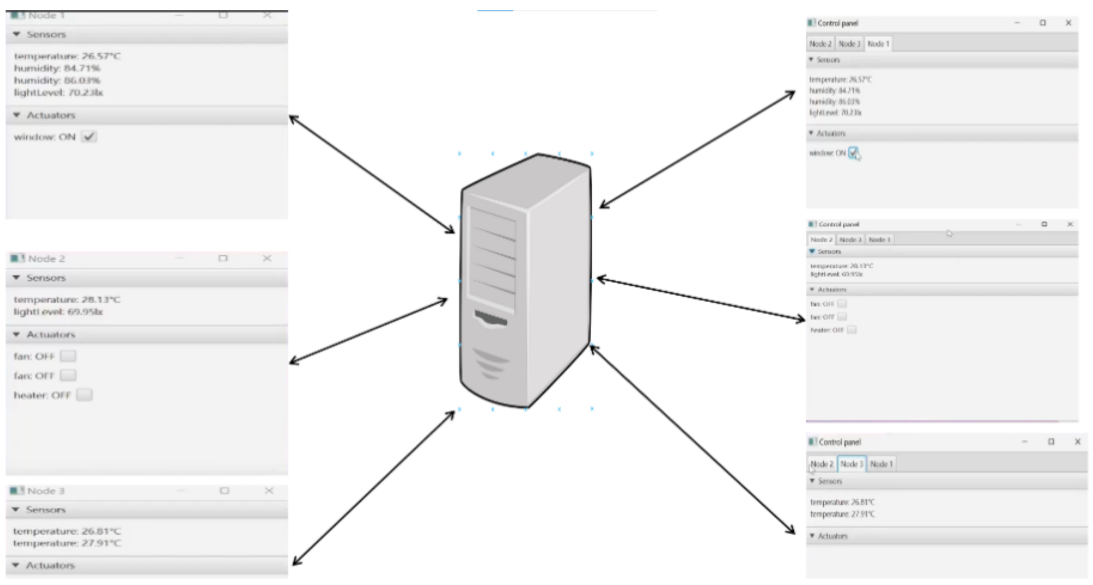


Figure 3: An example of user interface-GUI

Here you have an opportunity to refresh and showcase your GUI development skills learned in IDATT2003: Programming-2 by making a simple GUI. GUI may display sensor nodes and control-panel nodes.

See more detailed requirements below.

5 REQUIREMENT OVERVIEW

Your solution consists of several parts, all of which must be delivered by your group:

1. Design an application-layer communication protocol for data exchange between the sensor/actuator nodes, and control panel nodes.
2. Describe your protocol. You need to write a documentation of your protocol (Report), where you describe the protocol as clearly and unambiguously as possible.
3. Implement the protocol (program it) for the sensor nodes, control-panel nodes, and the necessary server side.
4. Ensure quality for the source code you write.
5. Document the work process throughout the semester.
6. Create a presentation video.

5.1 PROTOCOL REQUIREMENTS

The following fundamental requirements must be followed by the application-layer communication protocol you design:

1. You are NOT allowed to use any higher-level protocols (HTTP, MQTT). You need to implement your application-layer protocol directly on top of TCP or UDP sockets. The reason behind this is learning. You learn significantly less about network protocols when using pre-made libraries, reference implementations etc.

2. You need to decide - will you use TCP or UDP as a means of transport?
3. You need to support multiple sensor nodes and multiple control-panel nodes in the network. I.e., a solution where only one sensor node or only one control-panel node will function will not be considered a complete solution.
4. Different sensor types must be supported. The different types are not fixed. I.e., your protocol should allow adding new sensor types when needed.
5. Different commands must be supported. Commands are not predefined.
6. Commands are always sent from a specific control-panel node to a specific sensor node. That is, control-panel node x can't control anything for another control-panel node y. As a minimum, a command to a specific actuator on a single sensor node must be supported (for example, turn off fan number 3 on sensor node 42). It is allowed to address the same command to multiple sensor nodes at a time or multiple actuators on the same sensor node.

   Examples: turn off all actuators (heaters, fans, window openers) at sensor node 7; or turn on all fans at sensor nodes 7, 12, and 19.
7. Decide the form of information flow. Will the control-panel nodes request (pull) the information every time? Or will the sensor nodes actively report the information without requesting (push)? Is this a publish-subscribe type of information system?

Both the protocol and the implementation must ensure that the expected features of both the sensor-nodes and control-panel nodes are provided. See the expectations of the nodes below.

5.1.1 Sensor node requirements

Each sensor-node can do the following:

1. Have a unique identifier (address). To simplify things, you can assign the identifier to the sensor nodes manually when starting them.
2. Have multiple sensors attached to each node. As a minimum, one sensor of each type must be supported.
3. Support different sensors. For example, one node may report humidity and light, while the other node reports only temperature.
4. Act as an actuator node as well. That is, each sensor node is a "sensor and actuator node", which can have several actuators attached.
5. Support different actuators. For example, fan, heater, window opener, door lock, shower opener.

Hint: if your protocol will support only one instance of each sensor type on a node (only one temperature sensor per node, one humidity sensor, etc.), it is probably enough to address the sensors by their type. If you want to support multiple instances of the same sensor type per node, you need to introduce the addressing of the sensors (and actuators). For example, temperature sensors 1 and 2 on the sensor node 7, humidity sensors 1, 2 and 3 on sensor node 12, etc.

5.1.2 Control-panel node requirements

Each control-panel node can do the following:

1. See a list of all sensor nodes.

2. Receive sensor data from any sensor-node.
3. Visualize received sensor data. No need to visualize charts, textual visualization is enough. (You can create charts, if you wish, but that is not the main focus of the project).
4. Receive actuator status data from any sensor node. For example, is a window open or closed, is the fan on or off?
5. Visualize actuator status for each sensor node. Simple, textual visualization is enough.
6. Send control commands to any sensor node. For example, turn off the fan on sensor-node number 5.

5.2 PROTOCOL DESCRIPTION REQUIREMENTS

You need to describe your protocol in a markdown **(https://www.markdownguide.org/cheat-sheet/)** document named **protocol.md** inside your repository. There you need to describe the following aspects of your protocol:

1. A short introduction: "This document describes _ _"
2. Terminology: a list of special terms you use
3. The underlying transport you use (TCP or UDP)
4. The used port number
5. The overall architecture:
   - Who are the actors (nodes) in your solution?
   - Who are the clients, who is/are the server(s)?
6. The flow of information: when and how the messages are sent?
7. The type of your protocol:
   - Is your protocol connection-oriented or connection-less?
   - Is the protocol state-full or state-less?
8. The different types and special values (constants) used
9. The message format:
   - The allowed message types (sensor messages, command messages)
   - The type of marshalling used (fixed size, separators, TLV?)
   - Which messages are sent by which node? For example, are some messages only sent by the control-panel node?
10. The different errors that can occur and how each node should react on the errors. For example, what if a message in an unexpected format is received? Is it ignored, or does the recipient send a reply with an error code?
11. Describe a realistic scenario: what would happen from user perspective and what messages would be sent over the network?
12. The reliability mechanisms in your protocol (handling of network errors), if you have any

13. The security mechanisms in your protocol, if you have any

For each of the design choices provide a short justification: why did you choose to design it the way you did?

5.3 PROGRAMMING REQUIREMENTS

Your group needs to program all the necessary parts for the whole application:

- Sensor/actuator nodes
- Control-panel nodes
- An intermediate server, if necessary

You need to program the necessary business logic and communication interface, including:

- Establishing a connection between the nodes. This should require minimal configuration from the user.
- Receiving data from the sensors, sending sensor data to the control-panel nodes.
- Visualizing received sensor data on the control-panel nodes.
- Forwarding a command received from the user interface in a control-panel node to the desired sensor node(s).
- Forwarding the commands to the desired actuator(s) on the sensor node.
- Handling connection errors and messaging errors properly.

5.4 CODE QUALITY REQUIREMENTS

Throughout the whole study program, you are learning good code development principles. You started those in Programming 1 and 2 and continue in this and other courses. Therefore, we expect the same quality standards as are expected in all other courses involving programming. The grade of the project may be decreased if your code looks messy and unprofessional.

1. Follow clean-code design principles. You should think about coupling and cohesion. In short: think about the right place for the code, responsibilities, dependencies, and repetition.
2. For a more detailed list of professional practices, see the **Code like a pro** book on GitHub.
3. If you copy part of code from any external sources and examples (StackOverflow, and others), add a reference to the original in a comment. For example, if you copy a function, add JavaDoc for it, and mention **Code adapted from <linkToOriginal>**. Things are more difficult when you use ChatGPT, GitHub CoPilot or similar tools. To avoid misunderstandings, it is suggested to add a comment **Code generated by an AI** to every class and method which is generated. Note: it is totally OK to copy code to some limit and combine different pieces of code. All software developers do that. What is not OK: copying code without referring to the source, especially if that source has specific license requirements.
4. **It is not allowed to copy code from other students and any other source!** All the groups get the same task, and each group must solve it in its own way. It is NOT OK to copy code from another group! NTNU takes a serious view of cheating.

5. We expect that you use Git for storing your code throughout the whole lifecycle of its development. I.e., it is a bad idea to store code in the git one day before the project deadline. It is a good idea to create Git repository as soon as possible and store all the code changes there. Even if you are just experimenting and creating prototype code which you may later throw away, store it in your Git to document the activity.

6. The folder structure" should be clear. I.e., it is a bad idea to have a Git repository with folders such as "projectNew", "projectFinal", "tempProject", "project2", "Copy of Project" etc. Suggested way: have the project in the root folder of the repository, avoid sub-folders with different project versions. It must be clear for the examiners which folders contain what. In the final hand-in delete all the temporary code (or move it to a separate folder, name it in a clear way (such as "experiments")). In case examiners will have hard time finding the right folder with the "final code", they may evaluate something that was an intermediary version, and you may get a reduced grade for it.

5.5 WORK PROCESS DOCUMENTATION REQUIREMENTS

We don't define any strict rules on which tools you must use (except for Git), but we expect that you provide some form of documentation of the following:

1. How the work was planned in terms of iterations (sprints)

2. How the tasks were distributed - who was responsible for what. Note - it is NOT OK to have someone in the group responsible only for creating videos. This course is primarily about network protocols and network programming. Every group member must demonstrate expertise within these fields. Perhaps one student works more on the protocol, another on the server programming, and the third on client programming, that is fine. But it is not fine if one student does all the programming, one just writes documentation (sprint reports) and the third creates a video. How are the second and third students showing their computer network expertise?

3. How was the work broken down into issues? Remember the principles you learned about issue definition in the System development course. Issues should be small, have descriptive names, clear "definition of done", etc. For example, an issue like "improve GUI" is a bad issue. What does it mean? When is it complete?

4. How the work was progressing - what did you accomplish in each sprint? Which issues were planned, and which ones were completed?

5. A short retrospective after each sprint - what went well, and how can you improve?

It is suggested that you keep the sprint documentation in Git as well, but you are allowed to use Wiki pages, Jira or other tools if you want.

5.6 VIDEO-PRESENTATION REQUIREMENTS

The group must record a 10-15 minute video-presentation of the project (English), where the following elements are included:

1. Introduction of the application: what problem does your solution solve? (1min)

2. The information, materials, approach you used, and the research you did. (1min)

3. The work process: how did you organize work throughout the semester? Role of each team member? How did you work with the sprints? Were there any general themes for the sprints,

phases of the project? (1min)

4. The architecture of your solution. What nodes are communicating? What is the responsibility of each? Preferably, include model's diagrams here. (1-2min)
5. Your communication protocol. Summarize it in a clear yet concise way. (3-4min)
6. Your solution and result. Explain what is working. Show a demo of the system. (2-4min)
7. Explain what extra work you have done (if any) for this to be considered an excellent project. (1-3min)
8. Reflect on potential improvements and future work. (1 min)

Note: it is not a big problem if your video is 17 or 18 minutes, but don't make it to 35 minutes! Ability to present your ideas and results concisely is a general skill you must master. While the video creation can be as simple as recording your screen during a Discord call, for better results it is suggested that you plan your presentation beforehand, try how you would explain the concepts, how much time that would take, etc. In other terms - prepare for the video presentation in the same way as you would prepare for a physical presentation in the class.

6 POSSIBLE EXTRAS

If you cover all the fundamental requirements, the project is considered "good" and deserves a C-level grade. If you want to get an A-level or B-level grade, you need to implement something extra. Only extras related to computer networks will be considered. The purpose of this restriction is to focus the efforts on the computer network concepts and avoid spending too much energy in topics of other courses (such as creating SQL databases, web interfaces, etc.). You can come up with your ideas on extras! Here are some suggestions for potential extras:

1. Resilience in case of network outages. The solution functions when the network connection is temporarily lost. This means buffering data, retransmissions, reconnecting, etc.
2. Data encryption. You can think of different methods of integrating security into your solution, either using public-key cryptography or other methods.
3. Automated generation of unique identifiers (addresses) for sensor nodes. By default, the programmer can assign static addresses to sensor nodes when running them (as a command-line argument). But you can design automated-address assignments as part of your protocol. For example, look at DHCP as an inspiration.
4. Images/files as sensor data. Imagine a scenario when a web camera is attached to a sensor node and the image frames it captures could be transmitted to the control panel. Image transfer poses some extra challenges. It is therefore considered an extra feature if you manage to integrate it in your protocol and implement it in your source code.
5. Support of more flexible actuator commands. By default, it is expected that a command is sent to a specific sensor node, specific actuator. If you manage to support also either broadcast commands (to all sensor nodes at a time), or multicast (to specific groups of sensor nodes), this is considered an extra.
6. Support data of different resolutions. For example, the sensor nodes could buffer and aggregate data, and the actuator nodes could request the 1- 1-minute averages, 1-hour averages, etc.

7 DELIVERABLES IN INSPERA

The project is handed in as a group-portfolio in the Inspera exam system. Each group delivers a single hand in. The portfolio consists of the following elements:

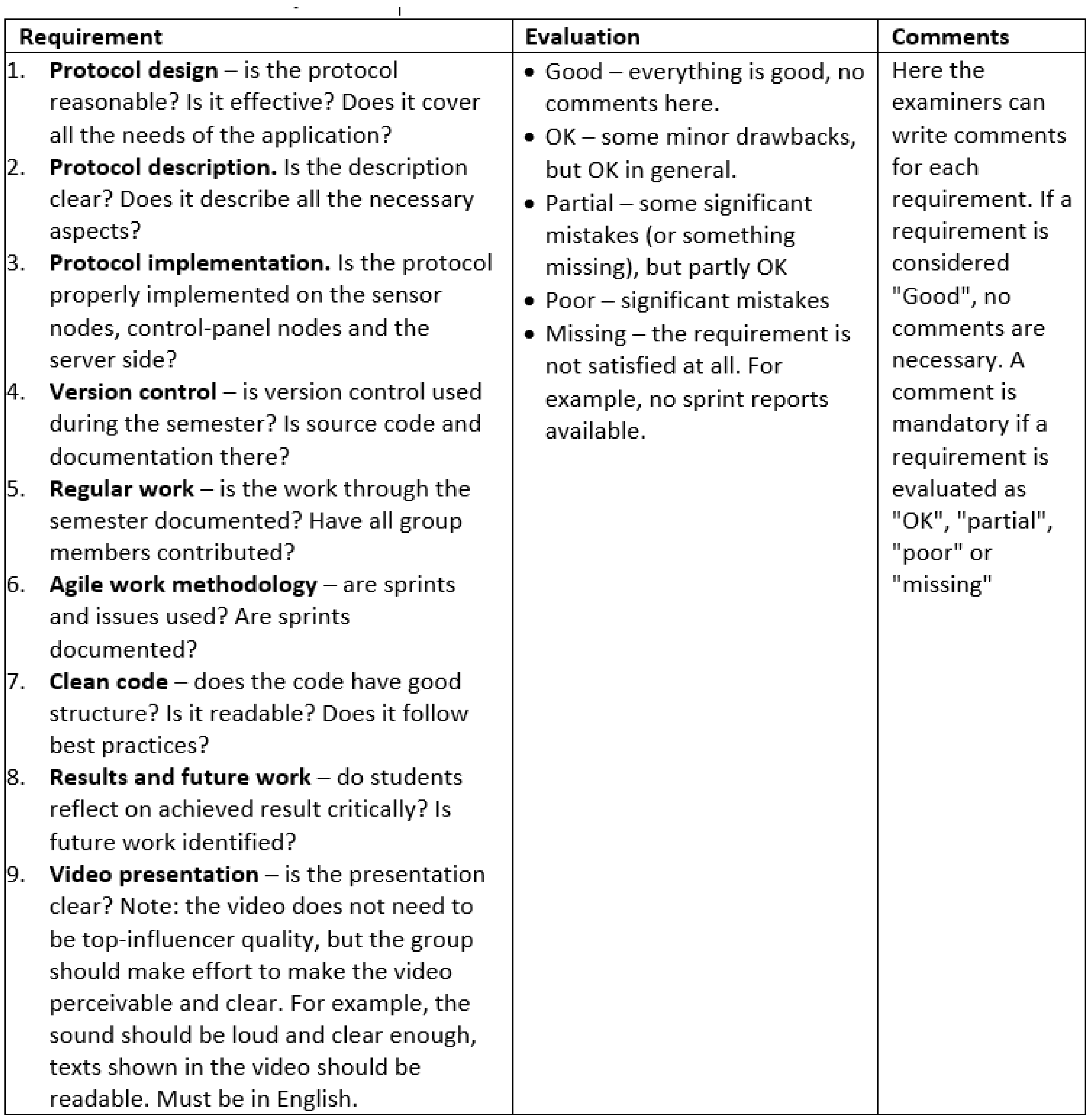
1. Link to the GIT repository where the source code and documentation is stored. The preferred way - use a public repository on GitHub.
2. A mapping between candidate numbers and git usernames. While we don't need to know the student names (the project can still be anonymous), the examiners need to understand the contribution of each candidate. The contribution will be evaluated based on the provided documentation: Git history and sprint reports. Both these artifacts will (most likely) include the GitHub nicknames, not the actual candidate names. Therefore, you need to specify which candidate was using which Github usernames.
3. Link to the protocol description document. By default, this should be protocol.md file stored at the root of your repository. You hand in a link just in case the protocol document is somewhere else (for example, in a sub-folder).
4. Link to sprint reports. These can be either .md files, .pdf files or other files, stored online.
5. Video presentation - a video file (.mp4 or other popular video formats).

8 GRADING

The projects will be graded as a whole, by humans. This means that there is no specific formula which an artificial intelligence could use to grade all projects. Here is a proposed grading schema, which will be used by the examiners (this same schema will be used by potential external examiners in a case of the complaint). The grading will happen in three phases:

1. Detect if all the **fundamental requirements** are met.
2. Identify the **excellence** in the project.
3. Decide whether the group gets the same grade or **individual grades** are needed

8.1 FUNDAMENTAL REQUIREMENT CHECK



Requirement	Evaluation	Comments
1. **Protocol design** – is the protocol reasonable? Is it effective? Does it cover all the needs of the application? 2. **Protocol description.** Is the description clear? Does it describe all the necessary aspects? 3. **Protocol implementation.** Is the protocol properly implemented on the sensor nodes, control-panel nodes and the server side? 4. **Version control** – is version control used during the semester? Is source code and documentation there? 5. **Regular work** – is the work through the semester documented? Have all group members contributed? 6. **Agile work methodology** – are sprints and issues used? Are sprints documented? 7. **Clean code** – does the code have good structure? Is it readable? Does it follow best practices? 8. **Results and future work** – do students reflect on achieved result critically? Is future work identified? 9. **Video presentation** – is the presentation clear? Note: the video does not need to be top-influencer quality, but the group should make effort to make the video perceivable and clear. For example, the sound should be loud and clear enough, texts shown in the video should be readable. Must be in English.	• Good – everything is good, no comments here. • OK – some minor drawbacks, but OK in general. • Partial – some significant mistakes (or something missing), but partly OK • Poor – significant mistakes • Missing – the requirement is not satisfied at all. For example, no sprint reports available.	Here the examiners can write comments for each requirement. If a requirement is considered "Good", no comments are necessary. A comment is mandatory if a requirement is evaluated as "OK", "partial", "poor" or "missing"

If all the fundamental requirements are evaluated as "Good", the project deserves a "C-grade". If some requirements are not met, the grade is reduced, based on the severity of the detected failures. The examiners may decide to write down a "temporary fundamental grade". The evaluation continues with the excellence check.

8.2 EXCELLENCE CHECK

When the fundamental evaluation is done, the "temporary fundamental grade" can be adjusted (upwards) based on the excellence demonstrated by the project.

8.3 FINAL PROJECT GRADE

After doing both the fundamental requirement check and excellence check, the examiners decide the final grade for the project.

8.4 INDIVIDUAL GRADES

By default, all the group members will get the same grade. However, individual (different) grades can be given to members of the same group if any of the following conditions are met:

- During the semester some group members report lack of contribution of some other member(s) to the teacher. A typical case - one of team members has disappeared, is not answering to messages from other students, etc.
- During the analysis of the final portfolio, it is revealed that some group members have little to no documented contribution to the project. This can be concluded either from the video presentation, or report, or the Git-log. For example, one student does not have any Git commits.

9 DEADLINE

Submission on Inspera:

Submission open: 26 November 2024

Submission close: 02 December 2024