Computer Networks (*RCOMP - Redes de Computadores*)

2024/2025

Team Project 2 – Network Applications Development

1. Calendar

This project starts on **2024-05-19** and ends on **2024-06-15** (the submission deadline). It's four weeks long and is developed jointly with **Sprint 3** of the **Integrative Project of the 4th Semester**.
The project presentation/demonstration is to be held in the week following the submission deadline, as scheduled by the RCOMP laboratory classes teacher.

2. Teams and teamwork

The teams for Project 2, are the **Integrative Project of the 4th Semester** teams, excluding the students that are not enrolled in RCOMP. For Project 2, it's acceptable teams with up to five members. Students that are not yet part of a team, should find, within the laboratory class, a team with less than five members enrolled in the RCOMP course.

In special cases where no other solution is possible, the RCOMP laboratory classes teacher may decide to accept a team without any student enrolled in the **Integrative Project of the 4th Semester**, for such a team there is an **alternative project** and backlog described in Chapter 5.

2.1. Repository

For Project 2, the repository to be used is the **Integrative Project of the 4th Semester** GitHub repository. Special teams without any student enrolled in the **Integrative Project of the 4th Semester**, undertaking the alternative project, should either keep using the same repository from Project 1 or create a new repository, either at Bitbucket or GitHub.

Whatever the repository, the team must grant the read permission to the **RCOMP laboratory classes teacher**.

2.2. Project Master

One member of the team takes the **project master** role, if possible, a student who has not taken the sprint master role in any sprint of Project 1.

The main missions for the project master are:

- Coordinating the work of the team (e.g., organizing coordination meetings).
- Submit the repository snapshot at the Moodle service (RCOMP area) before the deadline.
- Heading the project presentation/demonstration.

2.3. Project presentation/demonstration

The maximum duration for the presentation/demonstration is **15 minutes**. All team members are required to be present, they may not all take part in the presentation/demonstration, but afterwards they must all be available to answer questions.

Even though a live demonstration of the user stories is not mandatory, its absence without an acceptable justification will have a very negative impact in the teacher's assessment.

During live demonstrations, the applications are supposed to be executed outside of any IDE and in different network nodes. Client applications may be executed on students' personal computers, but server applications should be executed at remote nodes in a cloud.

3. Subjects and backlog

This is a software development project for network applications using the Berkeley Sockets API, which can be written in Java or C. The backlog for this project is a subset of the **Integrative Project of the 4th Semester** Sprint 3 user stories (Table 1).

User story	Nonfunctional requirements
US370 - Analyse a proposal	The solution should be written in Java or C. The implementation of these user stories must follow the components diagram in Figure 1 where network sockets are used to interconnect the encompassed applications. The application protocol to be used by the network sockets must follow the guidelines expressed in Chapter 4.
US371 - Accept/reject proposal	
US372 - Check shows dates	
US373 - Get show info	
US376 - Show testing	
US378 - Running test	

Table 1 - User stories for Project 2

Figure 1 represents the system's components. User stories US370, US371, US372, and US373 are implemented in the "Customer App" which communicates through the network with the "Customer App Server", which in turn accesses the Database. The "Customer App" is not allowed direct access to the Database.

US376 is implemented in the "Testing App" which accesses the Database and communicates through the network with the "Simulator". US378 is implemented at the "Drone Runner" that communicates through the network with the "Simulator".

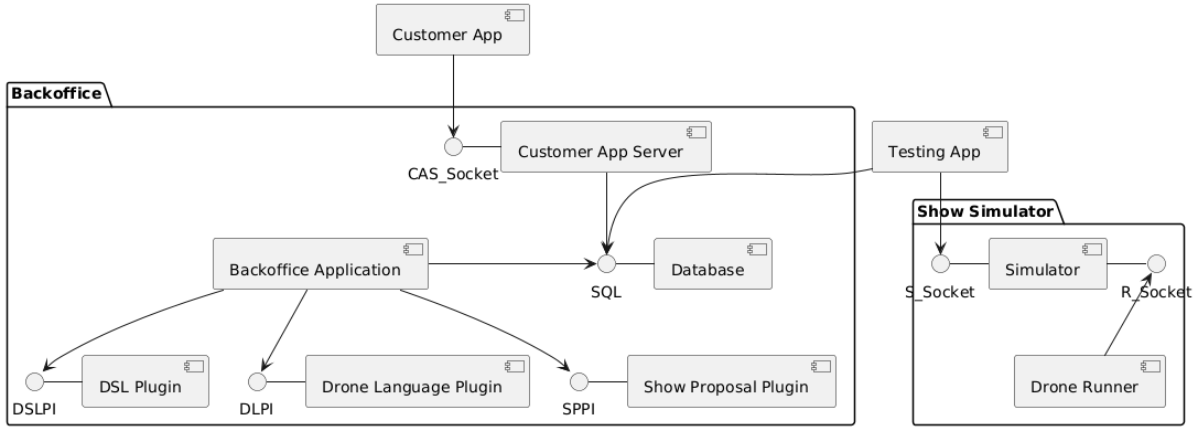

Figure 1- Components diagram (source: Integrative Project of the 4th Semester,2024/2025)

The "Costumer App Server" must be able to cope with several instances of "Costumer App" running at the same time, and the same goes for the "Simulator" application when it comes to several instances of the "Testing App" and "Drone Runner".

4. Application protocol

The communications between the applications must use TCP (Transmission Control Protocol) sockets in a client-server architecture.

Teams may use a standard client-server TCP based application protocol, like for instance HTTP (Hyper Text Transfer Protocol), or design their own client-server TCP based application protocol.

If the team chooses to design a new application protocol, then it must be formally described in the project's documentation. If the team chooses to use a standard application protocol, then the features of that protocol that are going to be used must be described in detail.

5. Alternative project and backlog

This chapter concerns only to special teams without any member enrolled in the **Integrative Project of the 4th Semester**.

5.1. Description

The final goal is implementing **prototype / proof of concept** for a file sharing system. The **Files Server** is a TCP server application that stores files uploaded by **Client App** client applications, Figure 2 represents the general view of the system.

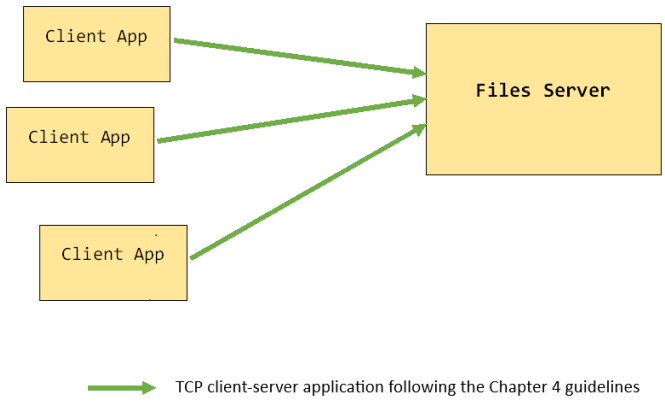

Figure 2 - Alternative project

The **Files Server** provides slots to store files, each slot can hold only one file, slots are numbered from one up to a maximum number of slots established by runtime configuration at the Files Server application. For each slot, the **Files Server** stores the filename and the file content, this data should be persistent.

For the sake of testing this prototype system, when it comes to authentication, there should be only one authorized user, with username and password provided by runtime configuration at the Files Server application.

After authentication, a client application may request several operations over the slots, when necessary, identifying the target slot by its number:

- List the slots (for non-empty slots, the filename should be presented).
- Upload a file (if the slot is not empty, the operation fails).
- Download a file (if the slot is empty, the operation fails).
- Remove a file from a slot.

5.2. Non-functional requirements

The communications between the Client App applications and the Files Server application must use network sockets and a TCP based client-server protocol as described in Chapter 4.

The Files Server application may be developed in C or Java.

Two different versions of the Client App are to be developed, one in Java and another in C.

5.3. Project backlog

In this backlog, the first tasks take precedence over the following tasks and teams are required to implement the number of tasks equal to the number of members in the team.

5.3.1. Develop and implement the Files Server application in C or Java, encompassing the **user authentication** process and **getting a list of the slots** features.

5.3.2. Develop and implement the Client App application in Java, encompassing the **user authentication** process and **getting a list of the slots** features.

5.3.3. Develop and implement the Client App application in C, encompassing the **user authentication** process and **getting a list of the slots** features.

5.3.4. Develop and implement the **upload a file into an empty slot** feature at the Files Server at and the two versions of the Client App.

5.3.5. Develop and implement the **download a file from a slot** feature at the Files Server and at the two versions of the Client App.

5.3.6. Develop and implement the **remove a file from a slot** feature at the Files Server and at the two versions of the Client App.