Vision statement

Charter

Project title	Potato Chat
Project manager	@JacopoWolf
Start date	12/10/2019
End Date	06/12/2019
Purpose	A simple, local-network confined, chatting system.
Goals & Objective	Design a chat protocol and implement useful APIs for said protocol, with a focus on a functioning, optimized and easy-to-use server.
Project cost	Time of the developers' lives.
Approach	Scrum
Schedule	view project's sprints
Priorities	Develop a functioning, bug-free, server.
Development team	@JacopoWolf - @gfurri20 - @Alessio789

Note: dates are in dd/mm/yyyy format

Scope

For who

PCP and its official implementation are aimed at small groups who need a completely local, controlled and simple way to chat with others in the same network.

What

The protocol

- **open source**: everyone can contribute with new useful ideas
- **extremely simple**: meaning low resources needed for parsing and easy custom implementations
- **local**: nothing goes through a server located who-knows-where, data is controlled in your network.

The server

- **open source**: bugs and flaws will have a short life
- **simple usage**: the server is extremely simple to boot up and in very short time, without the need to learn complicated interfaces and confusing configuration files.
- **optimized**: requires low resources on the host machine
- **expandable**: want to implement a custom version of the protocol? Documented and easy to use interfaces for every component are given!

Why

because it's an extremely easy, portable and expandable protocol, perfectly good for small realities who need a way to talk between rooms withouth yelling and withouth losing control of their messages.

Software Requirements Specification

For **PotatoChatProtocol**

Version 1.0 approved

Prepared by @gfurri20

Server Group:

- @JacopoWolf - Team leader
- @gfurri20
- @Alessio789

November 25, 2019

1 Introduction

1.1 Purpose

The purpose of this document is to present a detailed description of the PotatoChatProtocol standard. It will explain the purpose and features of the software, the interfaces of the software, what the software will do and the constraints under which it must operate. This document is intended for developers who want to implement this protocol.

1.2 Document Conventions

This document is based on the IEEE template for System Requirement Specification (SRS) document.

1.3 Intended Audience and Reading Suggestions

- Typical Users, such as developers, who want to use PCP to develop their chat applications.
- Students who are interested to discover how an extremely simple chat works.
- Small realities who need a way to talk between rooms withouth yelling and withouth losing control of their messages.

1.4 Project Scope

PotatoChatProtocol is a an extremely easy, portable and expandable protocol which can be used as base to develop a chat application. The protocol defines all the constraints to create a functioning client capable of interacting with the server implemented in the project.

1.5 References

PCP's main page: https://jacopowolf.github.io/PotatoChatProtocol
PCP's Github page: https://github.com/JacopoWolf/PotatoChatProtocol
PCP's Jitpack page: https://jitpack.io/#JacopoWolf/PotatoChatProtocol

2 Overall Description

2.1 Product Perspective

PCP is a protocol developed in order to have the possibility to implement an easy and portable chat application. It could be useful when a team needs to exchange information as quickly as possible.

2.2 Product Functions

Mainly PCP takes care of supplying the protocol standards in a clear and simple way, then it implements a server related to the last release version.

Protocol specifications:

- the structure of the packets that will be exchanged between the server and client-side applications.
- error and exception handling
- the procedures necessary for logging, disconnecting and exchanging messages between client and server.

Server side:

- build a server ready to manage connected clients and exchange messages between them.

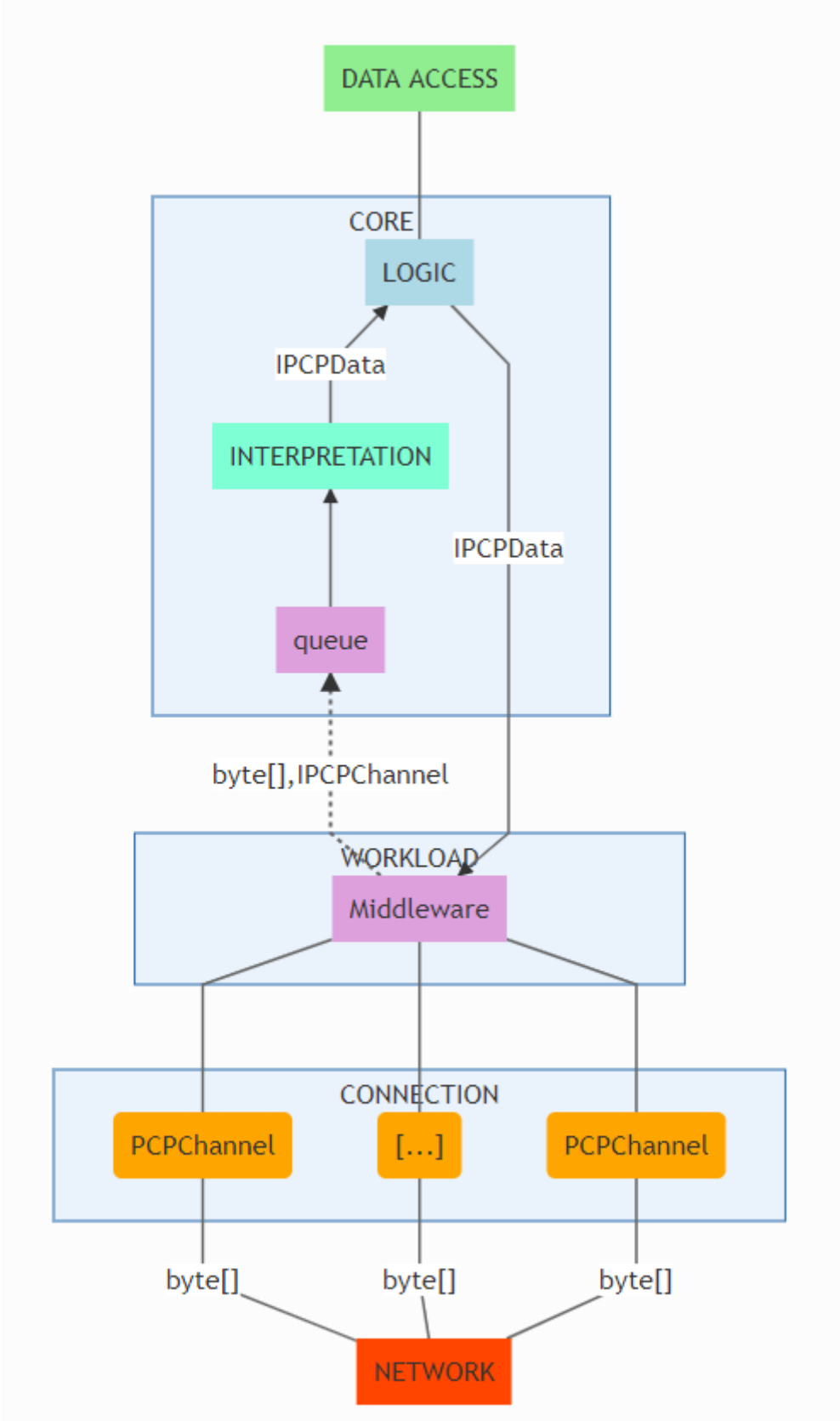
2.3 Operating Environment

All modern operating systems that have a java virtual machine installed.

2.4 Design and Implementation Constraints

This project is developed with Java using NetBeans as IDE. The packages are organized according to the protocol versions. It uses a modular design where every feature is wrapped into a separate module and the modules depend on each other through well-written APIs.

This is the structure of the API:



2.5 User Documentation

There is a quick guide which explains how to initialize a PCPServer:
https://github.com/JacopoWolf/PotatoChatProtocol/blob/master/docs/usage.md

Moreover there is the Javadoc, hosted on JitPack:
https://javadoc.jitpack.io/com/github/jacopowolf/potatochatprotocol/Min.a.3/javadoc/index.html

2.6 Assumptions and Dependencies

PCP is developed in Java so to start-up a server the machine requires a Java version 8 (recommended) or higher and JDK version 8 (recommended) or higher.

3 External Interface Requirements

The server implemented by PCP only print the logs. No graphical interfaces have been implemented. Clients that rely on PCP will implement it.

3.2 Hardware Interfaces

The protocol is very light, if you implement a small chat you can start the server on most multi cores processors. When the size increases accordingly the number of threads used will increase, this requires better performing processors. The minimum amount of RAM required is 4 GB.

4 System Feature

4.1 Server

PCP offers the possibility to start a server which is able to manage the communication between several connected users.

A simple guide is avaible:
https://github.com/JacopoWolf/PotatoChatProtocol/blob/master/docs/usage.md

5 Other Nonfunctional Requirements

5.1 Safety Requirements

To keep the server at the latest version, you need to update the software each time an update is released. This allows you to keep the server updated to avoid bugs or problems.

5.2 Security Requirements

PCP does not have any specification from a security point of view, however, it is a good practice to install and start the server on suitable and limited access machines.

Software project management plan

For **PotatoChatProtocol**, version 1.0 approved

Prepared by @JacopoWolf

Server Group:

- @JacopoWolf
- @gfurri20
- @Alessio789

Overview

purpose and scope

We are interested in implementing the server side of the **PCP Minimal** protocol specification.

The server application has the purpose of allowing the clients to chat between each other using the protocol.

The ideal target machine for this implementation should be servers, but due to the simple nature of the protocol target machines are going to be average PCs, with at least a multi core processor to handle parallelism.

Being a server, instead of an UI it uses a logging system, expandable to allow persistent logs.

goals and objectives

- develop easy-to-use APIs for the PCP protocol
- create a server with the following characteristics
  - functioning
  - easy-to-use
  - performant
  - light
  - working public chat
- instruct the group about:
  - Git
  - GitHub

- scrum
- normal development workflow

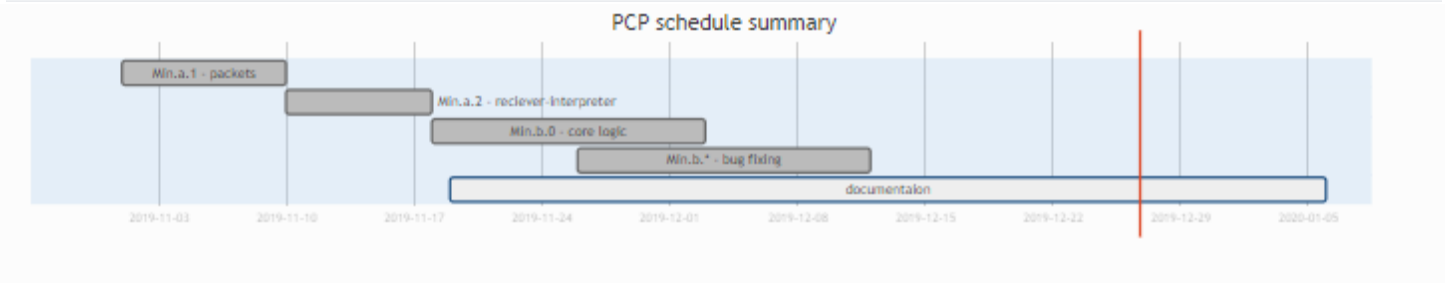
project deliverables

Deliverable	Date
Iteration plan	20/10/2019
packets - Min.a.1	14/11/2019
interpretation layer - Min.a.2	18/11/2019
core logic - Min.b.0	03/12/2019
final version - Min.1.0	12/12/2019
complete documentation	06/01/2019

constraints

- project manager will be Maven

Schedule



budget

on paper, based on the average cost of the respective work position:

name	role	frequency	hrs	€/h	cost [€]
@JacopoWolf	project manager / junior developer	6 to 10 h/week * 8 weeks	64	60	3.840
@gfurri20	junior developer	5 to 9 h/week * 8 weeks	56	40	2.240
@Alessio789	junior developer	5 to 9 h/week * 8 weeks	56	40	2.240
					8.320

success criteria

A working server.

definitions

term	definition
Server	the APIs used to instantiate a PCP server
"the protocol"	the Potato Chat Protocol specification

Startup plan

team organization

Role	Actors	Responsability
Project manager	Comparin Jacopo	Call team meetings, coordinate communications within group, coordinate communications outside group, break out tasks, assign them to teammates
Developer	Comparin Jacopo, Geremia Furri, Trentin Alessio	Develop software based on requirement and architect specifications
Debugger	"	Fix reported errors in the code
Tester	"	Write test cases, perform unit testing of test cases against incremental release of code, perform integrated testing of test cases against incremental release of code, report issues

project comunications

Event	Information	Audience	Format	Frequency
Team meating	Task status: completed since last meeting & planned for next; obstacles encountered; change requests in process	All team members	Informational meeting upon starting to work on the project in the same location	As needed

Event	Information	Audience	Format	Frequency
Notifications	A Telegram notificaion on the group specifically created	All team members	A telegram message	as needed
End users doubts	Informal meeting whith beta users to discuss bugs and code enhacements	End users and team members	Infomational meeting	As needed

technical process

An iterative and incremental development process is planned. Feedback will be used from each iteration to improve the next. The first iteration will focus on basic functionality of the application. Subsequent iterations will build upon that and incorporate more features as time allows.

tools

- Programming languages:
  - Markup
  - Java
- JDK 1.8 (platform)
- GitHub (VCS)
- NetBeans (IDE)

Work plan

release plan

The release plan is in a specific document

Iteration plans

The iteration plan is in a specific document

Project Success Criteria

The PCP (PotatoChatProtocol) Project will be considered a success if:

- **A basic version test system is delivered on (or before) December 13th 2019.**

  Version 1.0 must be published by the specified date. To consider the version 1.0 completed, it must allow users to use all the basic functions of the server through a client application.

  These main funtions are:

  - login and disconnection
  - exchange of private and public messages
  - change of alias

  Furthermore errors and exceptions must be handled appropriately.

- **Concerns raised during the test period are addressed according to the problem resolution plan contained in the Software Project Management Plan.**

  All major bugs related to basic functions will need to be fixed by the end of the trial period.

  Each problem encountered will be documented through the use of Github, a VCS software. The bugs will be solved in order of time according to their weight which will be established by the development team. The project documentation will contain all the information about the discovered and solved bugs.

  Users shall be notified of their rights and responsibilities under this policy prior to the start of acceptance test.

- **Community can easily use the protocol to develop a client application.**

  This specifically means that every developer who wants to implement a chat application based on this protocol must have the possibility to do it quickly and easily thorugh the study of the related documentation.

  Furtheremore the server that is already implemented must be easy to configure and start.

- **The system is properly documented**

  First, a user manual shall be produced and included. This user manual must present how to initialize a new PCP server which allows the exchange of messages between several connected user. The server configuration must be explained in order to be clear.

  Second, a system manual shall be produced. This system manual shall consist of the architecture document, as well as any additional material required to provide a technician with all necessary information needed to maintain and update the system. A single read through of this document and no more than 2 hours looking at the code should provide an appropriately skilled technician with all the information necessary to understand the system.

- **Known defects are entered into the Github issues tracking system.**

  After December 13th defects that are found but not fixed immediately will be entered into the Github issues tracking system.

Memorandum Of Understanding

For **PotatoChatProtocol**, version 1.0 approved

Prepared by @Alessio789

Server Group:

- @JacopoWolf
- @gfurri20
- @Alessio789

---

1. Purpose

The purpose of this Memorandum of Understanding (MOU) is to serve as a written understanding between the developers of PotatoChatProtocol (PCP) and the customers. This MOU documents the responsibilities of the principal organizations involved in the deployment of PotatoChatProtocol.

2. Objectives, Scope, and Major Activities

2.1 Objectives

The objectives are: - develop easy-to-use APIs for the PCP protocol - create a server with the following characteristics - functioning - easy-to-use - performant - light - working public chat

2.2 Scope

The scope of the PCP APIs is to provide: - a simple protocol that allows multiple users to communicate with each other in a local network, in public and private rooms. - an optimized and easily expandable server that allows the correct interaction to different users.

2.3 Major Activites

The main activity that allow PCP APIs to be carried out is to manage a simple chat system in a local network. It uses the PCP protocol and allows private and group chats.

3. Responsibilities

3.1 Customers

1. Respect the APIs license, without distributing them, free of charge or for commercial purposes, to third parties.
2. Provide all feedback on the use of APIs and on the use of the PCP protocol to developers.

3.2 PCP developers

1. Maintain the application and continue its development in order to improve it.
2. Develop new features on customer request.
3. Provide all the support and assistance necessary for the proper functioning of the APIs to customers.

4. Services

All services provided under this MOU shall be highlighted in this section, including but not limited to: - Description of the PCP APIs architecture; - Description of communication links; - Advanced notification of any planned APIs modifications - Support and resource required by both the developers and the customers; - APIs reliability plans

5. Duration and Amendments of the MOU

This MOU will be reviewed annually from the date signed.

Task Name			Role	Owner	Estimated by task	Effort Subtotals	Actual by task	Actual Subtotals
Preliminary tasks								
	Work Plan					1		1
		Define work plan	Project Manager	**Jacopo Comparin**	1		1	
	Protocol					2		2
		Define the protocol	Project Manager, Developers	**Jacopo Comparin, Geremia Furri, Alessio Trentin**	2		2	
	Server structure					2		1
		Define the server structure	Project Manager	**Jacopo Comparin**	2		1	
Iteration 1								
	Packet Classes					7		8
		Plan the structure of the classes	Project Manager	**Jacopo Comparin**	1		1	
		Creation of service packets classes	Developers	**Jacopo Comparin, Geremia Furri, Alessio Trentin**	2		2	
		Creation of message packets classes	Developers	**Jacopo Comparin, Geremia Furri, Alessio Trentin**	3		4	
		Create tests and fix bugs	Testers	**Jacopo Comparin, Geremia Furri, Alessio Trentin**	1		1	
Iteration 2								
	PCPManager					10		10
		Implement PCPManager	Project Manager	**Jacopo Comparin**	10		10	
	PCPInterpreter					6		6
		Implement PCPInterpreter	Developers	**Geremia Furri, Alessio Trentin**	6		6	
	Testing					3		4
		Create tests and fix bugs	Testers	**Jacopo Comparin, Geremia Furri, Alessio Trentin**	3		4	
Iteration 3								
	Logic Core					18		17
		Implement PCPMinLogicCore	Project Manager	**Jacopo Comparin**	10		10	
		Implement PCPMinCore	Developers	**Geremia Furri, Alessio Trentin**	8		7	
	Memory Manager					4		5
		implement server-side memory manager	Project Manager	**Jacopo Comparin**	4		5	
	Testing					3		3
		Create tests and fix bugs	Testers	**Jacopo Comparin, Geremia Furri, Alessio Trentin**	3		3	
Documentation								
	Documentation Part 1					10		10
		Vision Statement	Project Manager	**Jacopo Comparin**	1		1	
		Software Project Management Plan	Project Manager	**Jacopo Comparin**	3		3	
		Software Requirements Specification	Developers	**Geremia Furri**	3		3	
		Project Model Canvas	Developers	**Alessio Trentin**	1		1	
		Business Model Canvas	Developers	**Alessio Trentin**	2		2	
	Documentation Part 2					17		17
		Coding Standards	Project Manager	**Jacopo Comparin**	1		1	
		Project Closure Report	Project Manager	**Jacopo Comparin**	3		3	
		Architecture	Project Manager	**Jacopo Comparin**	1		1	
		Project Success Criteria	Developers	**Geremia Furri**	3		3	
		Testing Specification	Developers	**Geremia Furri**	3		3	
		Memo of Understending	Developers	**Alessio Trentin**	1		1	
		System Documentation	Developers	**Alessio Trentin**	1		1	
		Status Report	Developers	**Alessio Trentin**	3		3	
		Release Plan	Developers	**Alessio Trentin**	1		1	
					Totale	**83**		**84**

Project Closure Report

Results

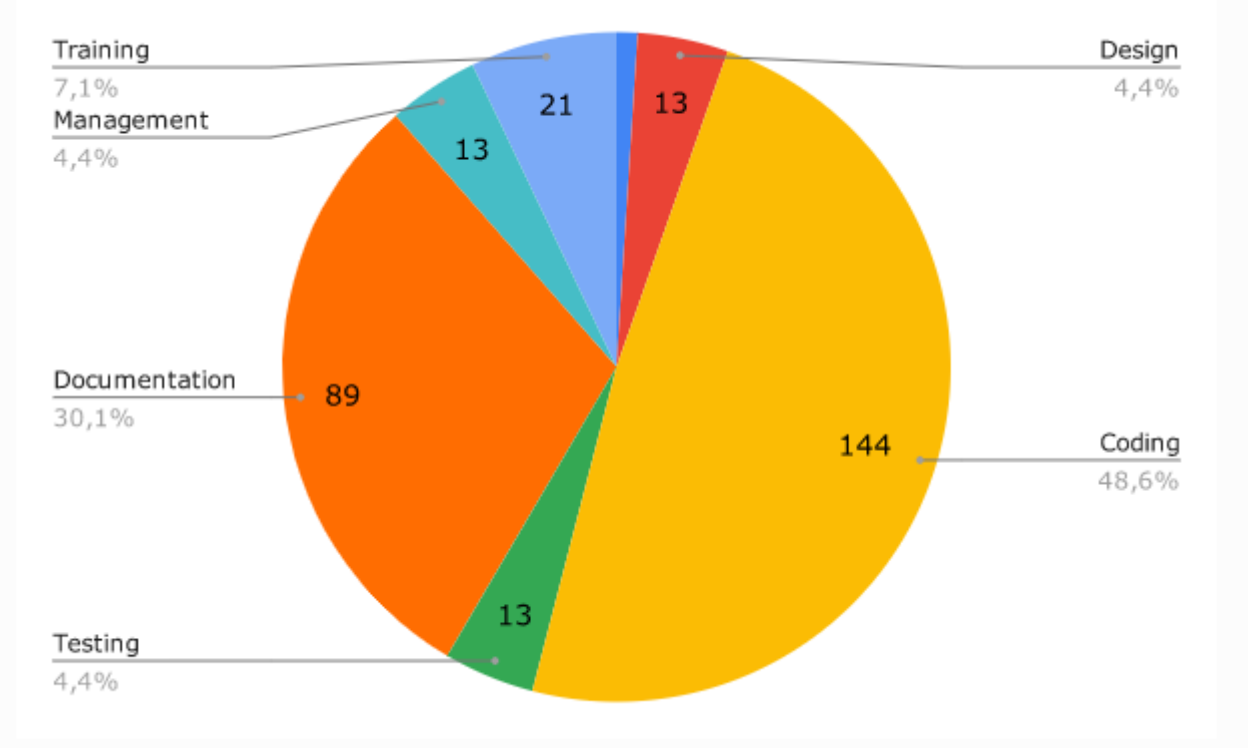
Project Metrics

Project Name	Potato Chat Protocol	
Manager	@JacopoWolf	
Scope	see the vision statement	
Project Type	New Development and protocol Design	*New development, Mantainence, Learning*
Problem Domain	Comunications	*Comunications, Finance, Healtcare*
Initial Technology Familiarity	4	*0-10 where 10 = very familiar*
Start-Finish Dates	20/10/2019 - 12/12/2019	**relative to development phase*
Duration (weeks)	7,6	

Effort Distribution

Effort is calculated assigning numbers on the fibonacci sequence (1,2,3,5,8,13,...)

Requirements	3
Design	13
Coding	144
Testing	8
Documentation	89
Proj. Management	13
Training	21
TOTAL	296



Staff

See the **budget** section in the software project management plan

Size

code size 143 kB

repo size 2 MB

Cost

The final effective monetary cost of the project is **0€**

Pulse

See the Pulse graph on **GitHub**

https://github.com/JacopoWolf/PotatoChatProtocol/graphs/contributors

Release Plan

For **PotatoChatProtocol**, version 1.0 approved

Prepared by @Alessio789

Server Group:

- @JacopoWolf
- @gfurri20
- @Alessio789

Iteration 1

Objective: the goal of this iteration is to outline the work plan and to define the protocol to be used for the chat.

Deliverable	Estimated Effort	Actual Effort
define work plan	1 h	1 h
define the structure of the server	2 h	1 h
define the protocol	2 h	2 h
Total	5 h	4 h

Iteration 2

Objective: the goal of this iteration is to create the classes that represent the packets.

Deliverable	Estimated Effort	Actual Effort
plan the structure of the classes	1 h	1 h
creation of service packets classes	2 h	2 h
creation of message packets classes	3 h	4 h
create tests and fix bugs	1 h	1 h
Total	7 h	8 h

Iteration 3

Objective: the goal of this iteration is to develop the part of the application that receives and interprets the incoming packets.

Deliverable	Estimated Effort	Actual Effort
implement PCPManager	10 h	10 h
implement PCPInterpreter	6 h	6 h
create tests and fix bugs	3 h	4 h
Total	19 h	20 h

Iteration 4

Objective: the goal of this iteration is to develop the core logic of the application.

Deliverable	Estimated Effort	Actual Effort
implement PCPMinLogicCore	10 h	10 h
implement PCPMinCore	8 h	7 h
implement server-side memory manager	4 h	5 h
create tests and fix bugs	3 h	3 h
Total	25 h	25 h

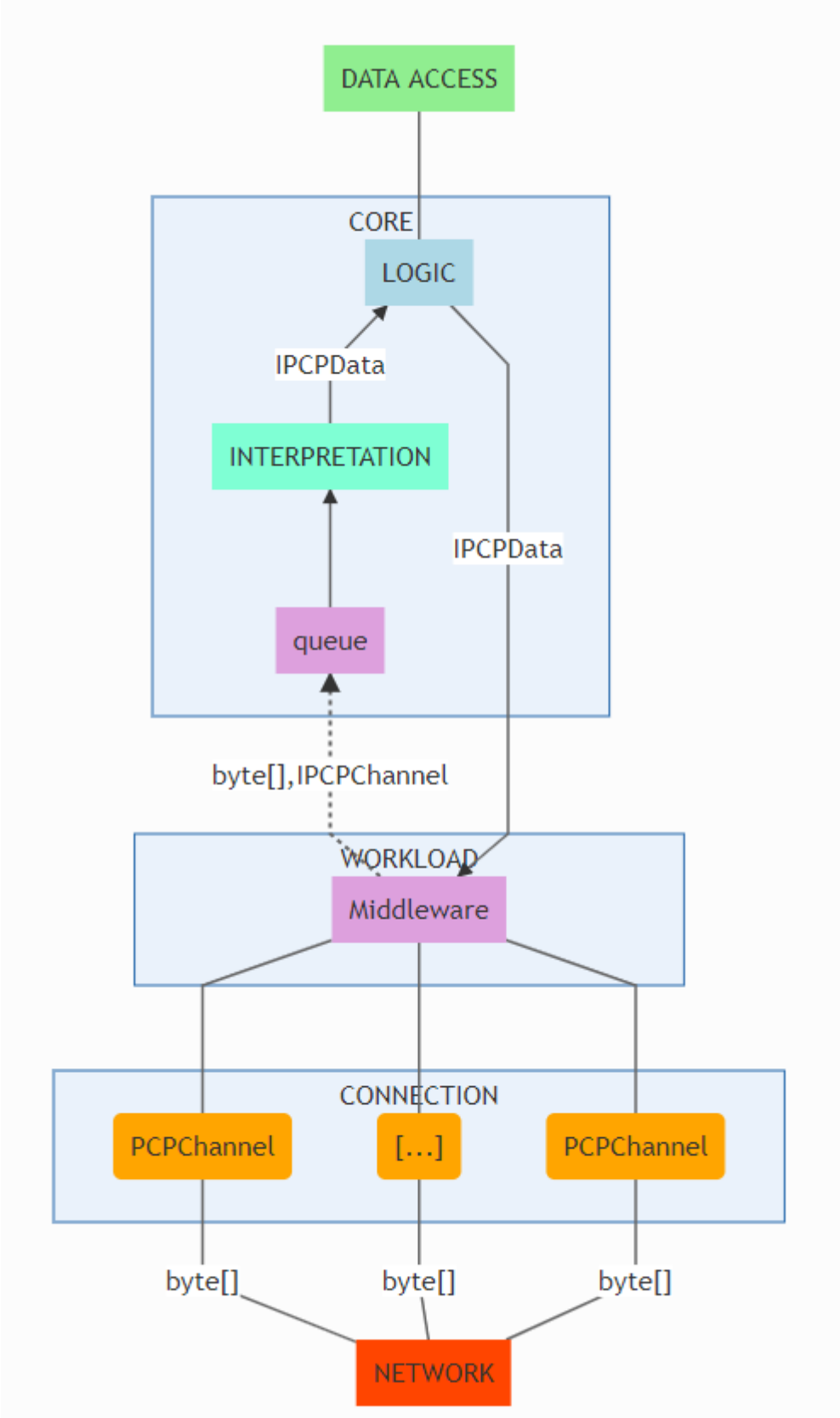
Iteration 5

Objective: the goal of this iteration is to complete the project documentation.

Deliverable	Estimated Effort	Actual Effort
documentation part 1	6 h	6 h
documentation part 2	8 h	8 h
Total	14 h	14 h

Architecture

Hierarchy



NETWORK

The network component is responsable to **mange I/O** from the outside using the TCP protocol.

This layer listens for new connections and recieves data from already established ones.

It is completely *asyncronous*, and runs on a series of threads that manage the incoming bytestreams. Every bytestram is associated with a `PCPChannel` .

When new data is recieved, it is queued to be executed by the middleware, in our implementation called `PCPManager` , which runs on a single thread.

CONNECTION

The scope of this layer is to **keep reference of useful variables during the TCP connection** and it's core classes are `PCPChannel` s. Those are managed by the server's middleware.

A `PCPChannel` keeps reference of the `IPCPUserInfo` object, used to keep the necessary information about a connection. It also keeps a reference to the `AsyncronousSocketChannel` used and a `ByteBuffer` , encapsulating a `byte[]` of the maximum lenght recievable by the connection.

WORKLOAD

This layer is responsable of **sorting the workload between threads**, keep them alive and decide when to kill them, optimize execution, map connections, and manage data flow.

Implemented by `PCPManager` , this layer is the "bottleneck" of the whole server, because for *ensuring data consinstency* it's executed on only one thread; the operations it performs are, anyway, extremely simple. Consisting mostly of checking an `HashMap` to direct incoming data.

The `PCPManager` implementation is also responsable to keep track of all existing connections. Because of that, it's the layer data must pass through for leaving the server.

INTERPRETATION

Tier responsable to **interpret incoming `byte[]` into simple to use at higher level, `IPCPData` objects.**

Those are version specific.

LOGIC

The logic layer is the core of the application, using the whole stack from the server to **perform business logic operations**.

This layer operates based on the results from the interpretation layer, and based on the reference to the channel that send the recieved data.

What it does is simply an opcode based set of instructions, handling requests and answering to them. Data generated from this tier are then sent through the `IPCPManager`

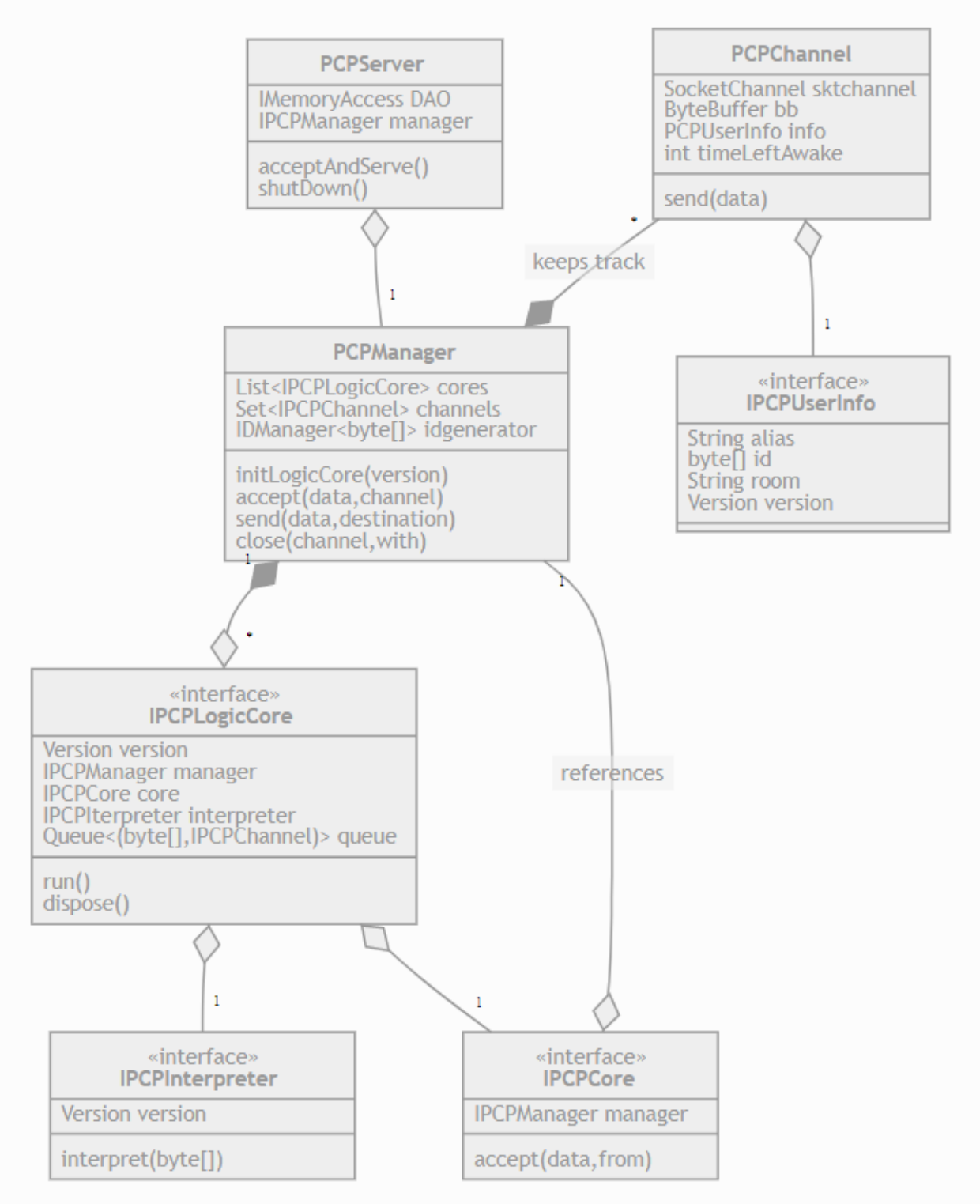
DATA ACCESS

Performs data access operations.

In this implementation, it is embedded in the logic core since the Minimal version of the protocol doesn't require long term storage. It is composed of the data structures necessary to keep the server working.

Class scheme

Below is rappresented the main class scheme. Data and secondary classes have been omitted for semplicity.



Execution

Execution is completely asyncronous, and is managed between **3 main thread goups**

- network

  Uses at least 3 threads to manage incoming data streams and direct them to the right channel and pass them to the upper tier; and 1 thread to manage outgoing data. 4 threads in total

- workload

  Single thread with the purpose of sorting out incoming data to the most optimal thread.

  Also manages a cache cleaning daemon thread, running on schedule; with the purpose of clean and remap connections to optimize resource usage.

- Logic

  number of thread in execution is variable and managed by the middleware. Usually the number is 1 even on mid-high data flow levels.

In total, the number of concurrent threads in this implementation is around 7. Meaning a multi-core processor is better for running the library.

Coding standards

Coding standards

Below are described the standards for writing code.

Those are based on the java's coding conventions.

Braces

Opening braces are always on a new line, and at the same indentation level as the declaration.

```
int example( int a )
{
    return a + 1;
}
```

Indentations

Indentations are 4 spaces long.

Elements on the same scope are on the same indentation level, elements in sub-scopes are an indentation level higher.

When a method or a statement is too long, it returns with a single additional indentation level. When an operator is in between a too long statement, it goes to its own line with an additional indentation.

```
void longMethodDeclaration
    (
        int parameter1,
        double parameter2,
        String anotherparameter
    )
    {
        anotherparameter =
            "" + parameter1 * 256
                +
            parameter2 * parameter1;
    }
```

Comments

Summarys of what the code below have a reserved line

```
// says hello
String str = "Hello";
System.out.println(str);
```

Instruction specification are on the same line

```
if (a<b||foo>el.param) // explaination on what this does
    return;
```

Naming

classes

- classes follow the Camel case notation.
- Interfaces always begin with a capital `I` and it's always the first letter in an interface name.
- PCP specic classes start with capital `PCP`

methods

methods follow camel case with lower first letter.

Documentation standards

A document title is written in html at the beginning og the document.

The highest header in the document is**h2**, written as `## Title` in markdown.

Before every section an additional empty line must be inserted.

```
<center>
<h1>Title here</h1>
</center>

## Section
section content

### Subsection
subsection content
```

System Documentation

For **PotatoChatProtocol**, version 1.0 approved

Prepared by @Alessio789

Server Group:

- @JacopoWolf
- @gfurri20
- @Alessio789

---

1. Introduction

PotatoChatProtocol is a simple protocol that allows multiple users to communicate with each other in a local network, in public and private rooms. It also offers APIs for an optimized and easily expandable server that allows the correct interaction to different users.

---

2. Install on Simulator or Device

2.1 Required Components

To start-up a PCP server the device requires a Java version 8 (recommended) or higher and JDK version 8 (recommended) or higher. The protocol is very light, if you implement a small chat you can start the server on most multi cores processors. When the size increases accordingly the number of threads used will increase, this requires better performing processors. The minimum amount of RAM required is 4 GB.

2.2 Installation Code

Note: the word "TAG" must be replaced with the version (eg "Min.1.0")

2.2.1 Maven

1. Add the repository:

```
<repositories>

    <repository>
        <id>jitpack.io</id>
        <url>https://jitpack.io</url>

    </repository>

</repositories>
```

2. Add the dependency

```
<dependencies>

    <dependency>

        <groupId>com.github.JacopoWolf</groupId>
        <artifactId>PotatoChatProtocol</artifactId>
        <version>TAG</version>

    </dependency>

</dependencies>
```

2.2.2 Gradle

1. Add the JitPack repository in your root build.gradle at the end of repositories `groovy allprojects { repositories { ... maven { url 'https://jitpack.io' } } }`
2. Add the dependency: `groovy dependencies { implementation 'com.github.JacopoWolf:PotatoChatProtocol:Tag' }`

2.2.3 sbt

1. Add it in your build.sbt at the end of resolvers:

```
resolvers += "jitpack" at "https://jitpack.io"
```

2. Add the dependency

```
libraryDependencies += "com.github.JacopoWolf" % "PotatoChatProtocol" % "Tag"
```

2.2.4 Leiningen

1. Add it in your project.clj at the end of repositories:

```
 :repositories [["jitpack" "https://jitpack.io"]]
```

2. Add the dependency

```
:dependencies [[com.github.JacopoWolf/PotatoChatProtocol "Tag"]]
```

2.2.5 After adding the dependency

1. Create a main class in your project and type: `import PCP.services.PCPServer;`
2. Instantiate the server and run it with

```
PCPServer server = new PCPServer();
```

`server.acceptAndServe();`

---

3. System Maintenance

The PCPServer has no graphical interface. Once the application has started, you can see the server logs. The server logs show whatever the server is doing: new connections, disconnections, messages sent...

Testing specifications

Introduction

This document outlines the test plan for the PotatoChatProtocol. The testing activities discussed in this document will verify that the software is capable of manage a chat between different client developed by different team following the structure of the protocol.

Items and features tested

The testing routine will test the management of the users in the server, exchange of messages between a client and another, the structure of the packets, log in and log out by the users. The results of this testing procedure will enable the creators of this system to gauge project success.

Approach

The overall method to this testing procedure is manual system testing using Junit library which implement testing tools.

Manual system testing will continue throughout the second and third iteration of the project. For each iteration, both old and newly implemented features will be tested. Adding new features or functionality can sometimes interfere with the functionality of old features and to ensure product/project success, all features implemented should function as intended throughout the life of the software.

Item Pass/Fail Criteria

The minimum requirements for this software system were laid out project success criteria.

Features that contain major defects will fail the testing procedure and will be turned over to the developer for investigation and revision.

Test Deliverables

In addition to the Test Plan, other test deliverables include the Test Specification which outlines the specific test cases and expected results of each test, and Test reports which is comprised of Incidents, Defects and Changes.

Testing tasks

The following list the testing deliverables and the activities required to produce the deliverable.

Deliverables	Activities
Test Plan	Analyze Requirements for System Features, Determine TestableNon-Testable Features, Develop Approach/Method for testing, Determine Task and Estimate Efforts, Develop Schedule for Testing
Test specifications	Analyze Requirements, Define Test Cases for Testable Features
Test reports	Implement Test Cases as Outlined by the Test Specifications, Document Incidents and Defects, Determine Severity of Incidents and Defects, Determine Changes that Need to be Made to System, Document and Submit Change Request to Developer

Test cases

Asynchronous management

Test ID	test1
Title	Asynchronous packets management
Objective	Confirm that the server can handle packets asynchronously
Setup	Junit testing on Netbeans IDE
Test Data	Packets automatically generated
Test actions	Allows the server to receive sent packets and process them
Expected Result	Server ends proccess data succesfully

Login and logout

Test ID	test2
Title	Login and logout by users
Objective	Confirm that the server can handle succesfully the communications which permit login and logout actions by users
Setup	Junit testing on Netbeans IDE
Test Data	User entitits automatically generated
Test actions	Simply log in and log out of a user

Expected Result	Server handles these two operations without errors

Messages exchange

Test ID	test3
Title	Management of communication between users
Objective	Confirm that the server can handle single or broadcast communcation
Setup	Manual, with one physical machine on which to run the server and two clients
Test Data	Packets manually generated
Test actions	Private exchange of messages and broadcast exchange of messages (in a room)
Expected Result	Every users can send and see rights messages

Usage guide

Overview

Description

The PCP-Minimal version of the PCP protocol is its stupid version, implementing an extremely easy, unprotected, channel-based and volatile LAN chat.

This is a short tutorial on how to use the APIs of this simple chat.

javadoc

This project is published with JitPack, JavaDocs and releases are hosted on their servers for everyone to use.

API structure

The root package, `PCP` contains the common important variables:

- `PCP`: static class containing common protocol variables, like **PCP.PORT** or the **Versions** enumerator
- `OpCode`: enumerator conaining all operational codes, convertable to their respective *byte*, and a function to retrive the opcode from a byte.
- `PCPException`: manages all possible errors planned by the protocol.

The API is then divided in 4 main generic packages:

- `services`: high level APIs who expose simple-to-use services, like **PCPServer**
- `logic`: logic and data layers, includes interfaces to implement correctly the logical level of the server and to access data.
- `data`: common data types. **PCPVariablePayload** is a really useful class to extend.
- `net`: net, connection, and worklaod layers. contains **PCPChannel** and **PCPManager**

Version-specific packages, named with their version name, like `Min` contain their specific implementation of said root packages. Again, they are organized in the same way. For example the PCP-Min implementation of **IPCPUserInfo** is located under `PCP.Min.logic`.

Services

PCPServer

`PCP.services.PCPServer` is the main implementation of a PCP server.

It uses the default gobal logger to log messages, wich can be obtained with `Logger.GetGlobal()`.

If you have performance problems, it might be your system is not good at managing threads, as this implementation allocates at least 7 of them.

start

To start a server write down in a meaven project these three lines of code.

In the below example, a PCPServer is started on the loopback interface.

```
 InetAddress address = InetAddress.getByName("127.0.0.1");
PCPServer server = new PCPServer( address );
server.acceptAndServe();
```

be careful as all of those calls can throw an `IOException`. If the reson of this exception is a port already in use, make sure you don't have any other application unfortunately already occupying the port **53101**. If that's the case, restarting your computer might solve the issue.

logging

PCPServer uses Java's `Logger.getGlobal()` to log every operation it does. By default, the global logger prints on the console with an `INFO` logging level.

If you want to change logging output, go look at Oracle's official documentation for complete usage information.

To change the logging level to `FINEST`, allowing detailed printing of every action performed by the server, write those two instructions:

```
 // changes default console logging level
Logger.getLogger("").getHandlers()[0].setLevel(Level.FINEST);
// changes global configuration logging level
Logger.getGlobal().setLevel(Level.FINEST);
```

interrupt

to interrupt the server the proceedure is simple:

```
 server.shutDown();
```

this is a blocking call that will safely shutdown the server, meaning it might take some time.