# Personhood.Online

Preliminary Work - due Feb 20

## Introduction

As part of the Proof-of-Personhood (PoP) project, you will be involved in developing a decentralized, distributed system, using a client-server communication paradigm and built across different technologies.

The initial development on this system has proceeded according to *N*-version programming (with N=2), where two back-end and two front-end systems have been developed independently of each other, by separate teams, around a shared specification.

As you pick up the project where it left off, you will be dealing with several programming languages, software libraries and communication protocols. This document will guide you through several tasks towards a practical exploration of some environments you should become familiar with (at least partially) for this project. The overall effort shouldn't take you more than 20 or 30 hours, but you should definitely consider investing more time if you're not familiar with a particular technology. In all cases, you should at least feel comfortable with one front-end and one back-end environment.

You can complete these tasks in any particular order and we encourage you to have a look at the environments you know least. Please do not hesitate to ask questions on Slack, play around, have fun, and most importantly: learn as much as possible.

⇒ Head to the next page for the objectives !

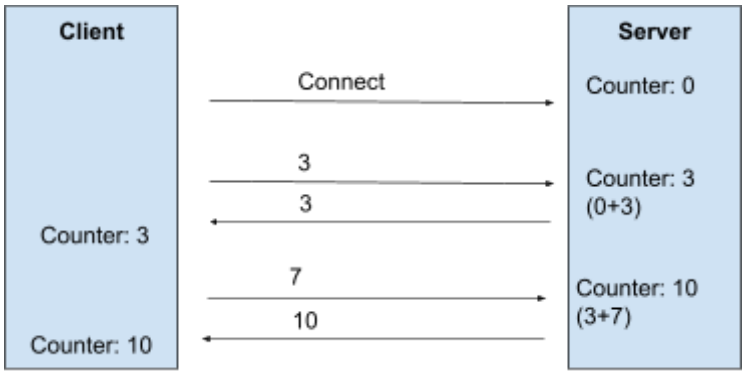
# Objective

You will be building an extremely simple WebSocket-based, client-server system across 4 different programming languages, familiarizing yourself with their libraries and specific paradigms.

- The server (Go, Scala) will need to be listening on port X (e.g. 11337) of the local machine for incoming WebSocket connections.
- A client (TypeScript/Web, Java/Android) can connect to the local machine on port X (when the user clicks on a “connect” button of your choice).
- Every time a client connects to the server, the server will initialize a new counter (at 0) for that WebSocket connection.
- The client can send messages containing raw numbers over the WebSocket connection. Whenever the server receives a number, it will increment the counter associated with the WebSocket connection by that number, and send back a message containing the current value of the counter.
- Both clients and servers should log all incoming/outgoing communications.

A schematic representation of this interaction is shown in the figure below.

The specification is voluntarily vague and you should feel free to extend it as you see fit. However, do not spend too much time on one specific implementation.




⇒ Head to the next pages for the tasks !

# Task 1 - Server: Go

As part of this task, you will be building an extremely simple WebSocket server in the Go programming language.

## Language resources

If you have limited or no familiarity with Go, we recommend starting from this tutorial, as it will provide you with all the basic knowledge about the Go programming language.
There's a lot more to Go than this simple tutorial will show you, and you will most likely need to dig deeper in the documentation (*Effective Go, Debugging, Data Race Detector, Error Handling, Testing, etc.*) as you tackle the PoP project.

## WebSocket library

As part of the PoP project, we use the Gorilla websocket library and we encourage you to try it out. You can follow its examples and documentation to ease yourself into its usage.

⇒ Head to the next pages for the other tasks !

# Task 2 - Client: TypeScript / React / Web

As part of this task, you will be developing an extremely simple WebSocket client in the TypeScript programming language according to the specifications above.

## Language resources

If you have never worked with TypeScript, a superset of JavaScript, or you have little familiarity with it, you should definitely start by looking at the documentation. It provides an introduction based on what languages you're familiar with, and the *handbook* will cover most concepts thoroughly.

As part of the PoP project, we use the React (incl. React Native) library to develop the front-end and produce both a Web-based interface and possibly a native mobile app. React is a large beast and it takes a while to master. If you have the time ahead of the semester, it's a worthwhile investment to go through its tutorial. You will also want to look at the Redux library, whose tutorial is here.

For both React and Redux, you can find dedicated Chrome plugins (React Developer Tools, Redux DevTools), enabling a much better debugging experience, on top of Chrome's debugging capabilities. Invest the time to get familiar with them and the possibilities they offer.

## WebSocket library

Browsers have native support for WebSocket libraries, but the PoP code should also be deployable as a native mobile app, without a browser. As such, we use a separate library to handle WebSockets, simply called 'websocket'. You can find more about it here.
You should definitely consider trying it out and using it as part of this task.

⇒ Head to the next pages for the other tasks !

# Task 3 - Client: Java / Android

As part of this task, you will be developing an extremely simple WebSocket client in the Java programming language, meant to run on Android.

## Language resources

We expect all of you have had some degree of exposure to Java as part of your education at EPFL and ETHZ, but do reach out to us if you do need help in locating learning resources.

## Android

While you can develop this client as a regular Java app, we encourage you to have a look at the Android SDK, Android Studio and its documentation. In particular, you might want to familiarize yourself with the notions of “Activities” and “Fragments”.

If you have time, try to develop this client as a “hello world” Android app and run it within the provided Android emulator.

## WebSocket library

As part of the PoP Spring 21 project, we plan to use Kotlin-based Scarlet websocket library and we encourage you to try it out. You can follow this tutorial to ease yourself into its usage. This library was developed for Kotlin, but Kotlin is designed to be fully interoperable with Java, and you can find information on how to use it with Java here.

⇒ Head to the next pages for the last task !

# Task 4 - Server: Scala

As part of this task, you will be building an extremely simple WebSocket server in the Scala programming language according to the specifications above.

## Language resources

If you have never worked with Scala before or you have little familiarity with it, you should definitely start by looking at one of the Scala courses and/or its tutorial. Scala combines object-oriented programming with functional programming language and the latter will require some getting used to if you don't have previous functional programming experience.

Even if you're familiar with Scala, you may not have developed large projects with it. You might consider investigating best practices, notably concerning architecture and code organization.

## WebSocket library

As part of the PoP project, we use the Akka HTTP library for its WebSocket support and we encourage you to try it out. This is a huge and powerful library, you can expect to spend a fair amount of time browsing through its documentations and examples.