Client-Server Network

Solution Design Document

University of Liverpool

Version: 1.00

Updated: 22/07/2022

Authors: Barbara Surowiecka, Chun Wong, Amr Ibrahim

Contents

[1. Overview 3](#_Toc109586047)

[2. Summary of Existing Functionality 3](#_Toc109586048)

[3. Requirement Details 3](#_Toc109586049)

[4. Assumptions and Prerequisites 3](#_Toc109586050)

[5. High-Level Design 3](#_Toc109586051)

[6. Low-Level Design 4](#_Toc109586052)

[7. Out-of-scope 5](#_Toc109586053)

[8. Risks and Mitigation 5](#_Toc109586054)

[Appendices 5](#_Toc109586055)

# Overview

This document is a work of Solution Architect and outlines the client/server network design and prerequisites.

# Summary of Existing Functionality

The simple client/network application does not have any predecessors and will be built from scratch.

# Requirement Details

Functional application requirements:

1. Build a simple client/server network. The client and server can be on separate machines or on the same machine.
2. Once the network is established, please complete the following tasks:
   1. Create a dictionary, populate it, serialize it and send it to a server.
   2. Create a text file and send it to a server.
3. With the dictionary, the user should be able to set the pickling format to one of the following: binary, JSON and XML. Also, the user will need to have the option to encrypt the text in a text file.
4. The server should have a configurable option to print the contents of the sent items to the screen and or to a file. Also, the server will need to be able to handle encrypted contents.

# Assumptions and Prerequisites

These are the following assumptions:

1. User interacting with the solution has some basic knowledge of how to open and use terminal (also known as command prompt, command shell, and command line) as well as knowledge of directory.
2. User knows how to install Python packages.
3. User is familiar with how to run Python scripts and code.

These are the following prerequisites:

1. 3.6 or higher version of the Python is installed.
2. Python packages that must be installed by user are listed in the **requirements.tx**

# [High-Level Design](https://softwaredominos.com/home/software-design-development-articles/high-level-solution-design-documents-what-is-it-and-when-do-you-need-one/)



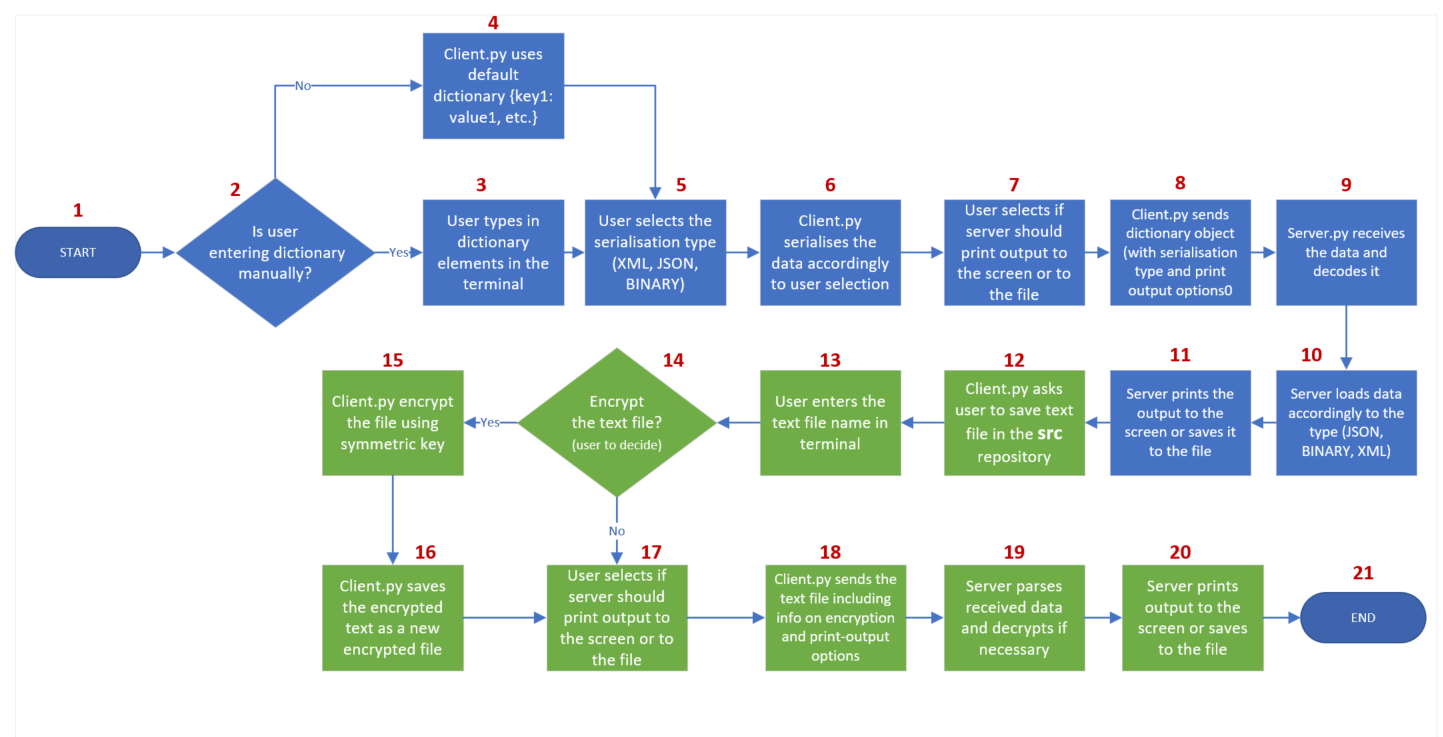


Figure 1. High-level application behaviour design

The Figure 1 illustrates application process flow, which comprises of the following steps:

1. Start.

2. Question: Is user entering dictionary manually?

3. Answer Yes: User types in dictionary elements in the terminal.

4. Answer No: Client.py uses default dictionary {key1: value1, etc.}.

5. User selects the serialisation type (XML, JSON, BINARY).

6. Client.py serialises the data accordingly to user selection.

7. User selects if server should print output to the screen or to the file.

8. Client.py sends dictionary object (with serialisation type and print output options).

9. Server.py receives the data and decodes it.

10. Server.py loads data accordingly to the type (JSON, BINARY, XML)

11. Server prints the output to the screen or saves it to the file.

12. Client.py asks user to save text file in the “src” repository.

13. User enters the text file name in terminal.

14. Question: Encrypt the text file?

15.Answer Yes: Client.py encrypt the file using symmetric key.

16. Client.py saves the encrypted text as a new encrypted file.

17. Answer No: User selects if server should print output to the screen or to the file.

18.Client.py sends the text file including info on encryption and print-output options.

19. Server parses received data and decrypts if necessary.

20. Server prints output to the screen or saves to the file.

21. End.

# Low-Level Design

Client-server session steps.

**Step 1.** Server and client create a stream socket using the socket library. The AF\_INET is an address family that is used to designate the type of addresses, such as IPV4, which the socket is able to communicate with. SOCK\_STREAM parameter determines how the data is streamed.

socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

**Step 2.** Servers uses the **bind()** function to bind the created socket to the local address (IP address and port).

**Step 3.** Servers uses calls the **listen()** function to detect upcoming connections.

**Step 4.** Client uses **connect()** call to connect socket to the server.

**Step 5.** Server accepts client’s connection through **accept()** call.

**Step 6.** Client prepares dictionary object by collecting user’s input data and with the use of the pickle module serializes the dictionary object, whereas server users **pickle** module to de-serialize the received data.

**Step 7.** Exchange of data between client and serveris possible through writing and reading data on theirs sockets with the help of **send()** and **recv()** functions.

**Step 8.** The connection between server and client is terminated by either of the sides through calling the **close()** function.

# Out-of-scope

* The graphical user interface is not part of the simple client-network application.

# Risks and Mitigation

Risks can arise in the form of:

* Specific machine settings, such as Firewall, may prevent the program from connecting client to server.

# Appendices