Software Requirements Specification

for a

socket-based  
storage system

Version 1.7 approved

Prepared by Martin Disch

23.04.2015

Table of Contents

1. Introduction 1

1.1 Purpose 1

1.2 Product Scope 1

1.3 References 1

2. Overall Description 1

2.1 Product Functions 1

2.2 Operating Environment 1

2.3 Design and Implementation Constraints 1

2.4 User Documentation 1

3. External Interface Requirements 2

3.1 User Interfaces 2

3.2 Communications Interfaces 2

4. System Features 2

4.1 Pinging/alive signal 2

4.2 Getting backup list 2

4.3 Getting backup 3

4.4 Storing data 3

4.5 Deleting a backup 3

4.6 Exception handling 3

5. Other Nonfunctional Requirements 4

5.1 Performance Requirements 4

5.2 Safety Requirements 4

5.3 Security Requirements 4

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Martin Disch | 08.04.15 | Phrasing | 1.1 |
| Martin Disch | 15.04.15 | Introduced JSON communication | 1.2 |
| Martin Disch | 15.04.15 | Handling of encoding over different platforms | 1.3 |
| Martin Disch | 16.04.15 | Format of response to ‘entries’ command | 1.4 |
| Martin Disch | 16.04.15 | Exception transmission over network | 1.5 |
| Martin Disch | 22.04.15 | Corrections for Unicode handling | 1.6 |
| Martin Disch | 23.04.15 | Documented ‘delete’ command | 1.7 |

# Introduction

## Purpose

This document describes a socket-based storage system, running on a Raspberry Pi microcomputer.

## Product Scope

The software specified provides functionality for KnowledgeBase, a note-taking application for Android, which will be referred to as the client (because the backup system is universally deployable and can be used by more than one specific application). The system described here will act as a remote backup storage.

## References

Being open source, the current version of KnowledgeBase, the application using the described service, can always be found on GitHub

<https://github.com/martindisch/KnowledgeBase>

The server software lives in a subfolder called “server” and is therefore also available in the same repository.

# Overall Description

## Product Functions

The server will listen to incoming connections and allow the client to check on its status, get a list of all available backups, get a certain backup’s data and store new data.

## Operating Environment

The software will run in a screen instance on a Raspberry Pi microcomputer, running a current build of Raspbian with Python 2.7.

## Design and Implementation Constraints

As there is need for port forwarding, only a single specified port can be used.

## User Documentation

Documentation will be provided in the form of comments in code and a detailed git history.

# External Interface Requirements

## User Interfaces

The user will only interact with the client application.

Status information of the server will be printed to standard console output.

## Communications Interfaces

Communication between the Android application and the server will be using the TCP/IP protocol and will be formatted in JavaScript Object Notation (JSON).

Due to the cross-platform nature of the project, different character encodings will pose a problem. This will be dealt with by rigorously implementing a type of “Unicode airlock”, making sure all internal data is in Unicode and encoded in UTF-8 for storage. Incoming traffic on the other hand, will be decoded back from UTF-8 to Unicode for internal usage while all outgoing data is sent in pure Unicode, as it’s easy to decode it into whatever is needed on the target platform.

# System Features

## Pinging/alive signal

4.1.1 Description & priority

When sending the ping command, the client will get a response to verify the server’s ability to save sent data.

4.1.2 Stimulus/Response Sequences

When receiving  
{“command”: “ping”}  
the server will respond with  
{“response”: “pong”}

## Getting backup list

4.2.1 Description & priority

When sending the entries command, the client will get a list of all stored backups.

4.2.2 Stimulus/Response Sequences

When receiving  
{“command”: “entries”}  
the server will respond by returning a list of saved backups like  
{“response”: [  
 “2015-03-25”,  
 “2015-04-03”  
 ]  
}

## Getting backup

4.3.1 Description & priority

When sending the get command followed by the date of storage, the client can get a certain backup’s data.

4.3.2 Stimulus/Response Sequences

When receiving a date like   
{“command”: “get”, “date”: “2015-03-25”}  
the server will respond by returning all encrypted data of said date like  
{“response”: “la8j9dskf9as8j2lkjsd8f9j2lkasdf32”}

## Storing data

4.4.1 Description & priority

When sending the store command followed by the date and data, the client can store the backup on the server.

4.4.2 Stimulus/Response Sequences

When receiving a store command like   
{“command”: “store”,  
 “date”: “2015-03-25”,  
 “data”: “la8j9dskf9as8”}  
the server will respond with an ok once everything has been safely stored  
{“response”: “ok”}

## Deleting a backup

4.3.1 Description & priority

When sending the delete command followed by the date of storage, the client can have a specific backup on the server removed.

4.3.2 Stimulus/Response Sequences

When receiving a date like   
{“command”: “delete”, “date”: “2015-03-25”}  
the server will respond with an ok once the backup has been removed  
{“response”: “ok”}

## Exception handling

4.5.1 Description & priority

Exceptions are caught in code and displayed locally on console output. Additionally, instead of a response, the exception message is sent to the client.

4.5.2 Stimulus/Response Sequences

When an exception occurs, the server will not respond with a JSON object containing a response, but one containing an “error” entry like  
{“error”: “No/unknown command received”}  
  
The client should always check if the response is of the type “error” and can then act accordingly.

# Other Nonfunctional Requirements

## Performance Requirements

The server does not need to handle multiple users at the same time, although it very well could.

## Safety Requirements

As the server already acts as backup, no further backup is required. The newest version will always be on the device running the client and failure of both systems is extremely unlikely.

## Security Requirements

As all data is already encrypted on the client. It remains in this state during transmission and while being stored, requiring no further security measures.

The server is by no means intended to be safe. It is secure in the sense that it should be very hard or near impossible to decrypt the data stored on it, but it is easy to just connect to it and have it remove all backups. It should be fairly easy to extend functionality to include some sort of authentication, but as this is hardly necessary for this hobby project, implementation is currently not planned.