



# BASIC STORAGE SERVER FOR APARTMENT BUILDINGS

FEBRUARY 8, 2014

*Aryamman Jain, Pranav Mehndiratta & Vaibhav Vijay*  
*Team ID: CD-037*

# TABLE OF CONTENTS

## Contents

Introduction	1
Software Architecture - UML Diagrams	2
System Requirements	4
Design Decisions	6
Conclusion	7
Bibliography	7
Appendices	9

# INTRODUCTION

## Introduction



For our storage server project, we are implementing a database for an apartment building in Downtown Toronto. This document's focus is to provide the reader with an overview of the client requirements and the design's ability to fulfil them. Our primary goal is to cater the needs of the management office by providing them with a centralized database. The end users would include the authoritative staff of the property; specifically, the superintendent and the security department.

Residential buildings generate vast amounts of data every day that need to be tracked for security and managerial purposes. This data includes tenant details, rent status, agreements, maintenance requests, and pending notices for residents. Secure and efficient storage of this data is imperative to provide a decent communication bandwidth between the tenant, management staff and landlord.



The current system lacks the necessary automation of record keeping as it is entirely paper based. The prime duty of security guards, at the building in question, is to record and store a variety of data in large log books kept at his desk [1]. The management office mentioned that the only method to contact the tenants is by dropping letters at each apartment [2]. Whereas, there is no form of direct communication between the superintendent and tenant, except for coincidental meetings [3].



Our proposed system will implement a centralized server that will store records in a particular data structure. This database can be accessed and modified by the client through a given interface that is capable of sending requests to the server using a specific protocol. To maintain the privacy of this data, server access is restricted through the use of login credentials that are configured as the server connection is established. The server itself may also internally store the database in an encrypted form. The client will be able to run queries on this database for tasks such as communicating with residents and securely obtaining resident(s) information.



# SOFTWARE ARCHITECTURE - UML DIAGRAMS

## Software Architecture - UML Diagrams

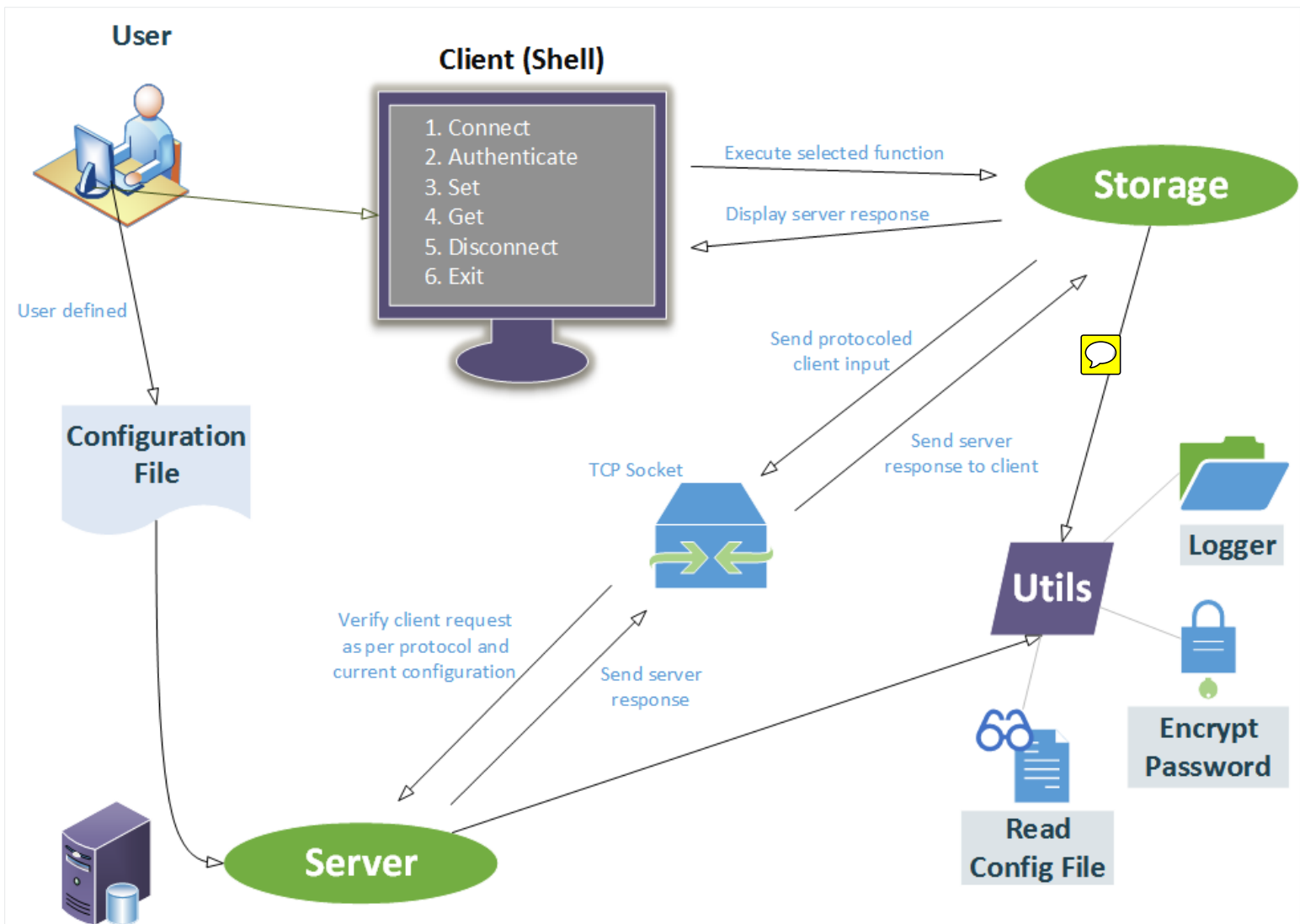


Figure 1: UML Component Diagram

# SOFTWARE ARCHITECTURE - UML DIAGRAMS

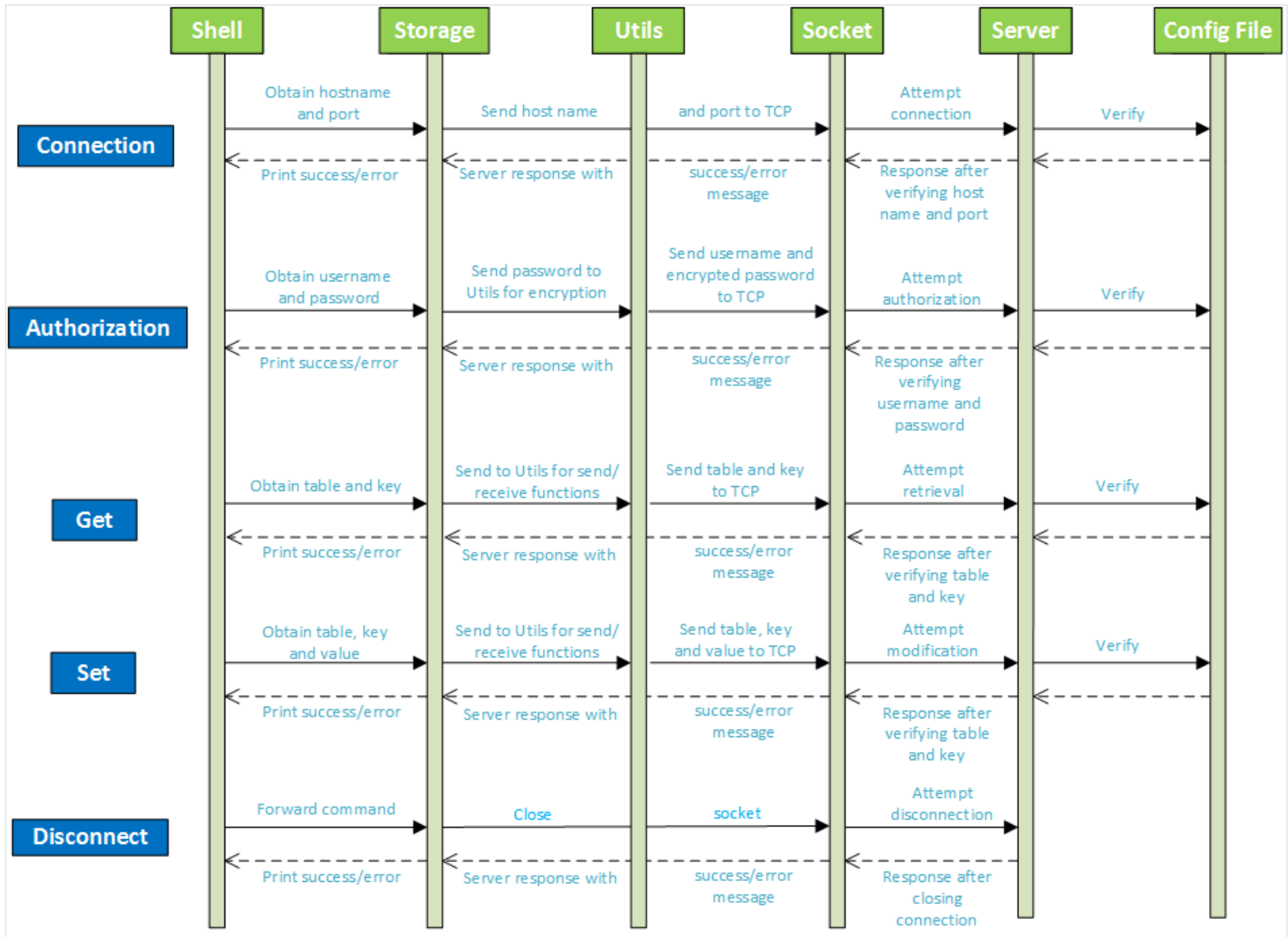


Figure 2: UML Sequence Diagram



# SYSTEM REQUIREMENTS

## System Requirements

### FUNCTIONS

- Primary
  - Centralize all data to one location and provide multi user connections
  - Retrieve an existing resident record (`storage_get`)
  - Insert/Update/Delete a resident record (`storage_set`)
  - Parse server configuration file set by user and set-up server accordingly (`read_config`)
- Secondary
  - Reduce amount of paper work that needs taking care of
  - Log all client-server interactions in time-logged files
  - Keep track of utility usage: hydro, water and heat
  - Save paper: benefit the environment

### OBJECTIVES

- Implement the shell and protocol to allow client to communicate with the storage server
- Allow client to store data: using the set command
- Allow client to retrieve data: using the get command
- Implement a communication protocol between server and client
- Ensure server doesn't prematurely respond with success messages
- Manage tenant contact details and facilitate rapid access
- Possess an appropriate data structure for multiple record insertion in a hierarchical manner
- Provide means of effective communication among landlord, management and tenant using proper querying commands
- Facilitate debugging of client-server operations with proper error messages and logging history

### CONSTRAINTS

- Client-based
  - Shall not use any personal details without tenant's permission
  - Server shall have enough memory to store the entire building's details
  - Access shall be restricted to building personnel

# SYSTEM REQUIREMENTS

- Technical-based



- Code must be written in C
- Client interface file 'storage.h' shall not be changed
- Server shall respond to client requests running on separate machines
- Server shall only accept configuration files of given format (Appendix A)
- Client-server communication is governed by Transmission Control Protocol (TCP) [4]

## CRITERIA



- Performance of database in sending and receiving data (Metric: time needed to store and access records)
- Reliability of stored data (Metric: data consistency following modification of records)
- Intuitiveness of client-server interface (Metric: time and support required to teach usage of new software)

# DESIGN DECISIONS

## Design Decisions



Hash tables will be used to store records and tables in memory using apartment numbers as keys. This will ensure unique hash values, hence minimizing collisions [5] and avoiding a complex hashing function. An alternative data structure is arrays; however, an additional field will be required to store the apartment number. A tree data structure is not desirable either as it increases the complexity to  $O(\log n)$  instead of  $O(1)$  in hash tables [5]. This happens in the worst case scenario, where the required node is at the bottom of the tree. This problem, however, does not occur in a hash table structure.



The connection state between the client and the server will be represented using a `struct` with details such as connection status, hostname/port used, authorization credentials, and socket file descriptor. This connection will then be reused for multiple get/set requests. The server is disconnected when requested to do so, or upon any error with respect to establishing a connection (invalid configuration file or credentials). In the event of an error occurring in the midst of a client request, an exception is thrown and the user is alerted to check the client/server log files to support their debugging process.



If multiple clients attempt to connect to the server simultaneously, they will be notified of the busy nature of the connection and will be requested to try again later. Moreover, it is possible to extend this to maintain and manage a queue of multiple client connection requests.



We will be using the protocol for the client and server communication as defined in Table 1. This protocol is subject to change as we progress through the design process.



Function	Protocol*	
	Client Request	Server Response
storage_connect	-	-
storage_auth	AUTH #username #enc_password	AUTH #pass AUTH #fail
storage_get	GET #table #key	GET #fail GET #value
storage_set	SET #table #key #value	SET #fail SET #modified SET #deleted SET #created
storage_disconnect	-	-

\*All strings are terminated by a new line ('\n') character

Table 1: Client Server Communication Protocol (Rev 1)



# CONCLUSION

## Conclusion


This document has highlighted the primary components and architecture of the state-of-the-art electronic system that will digitize data generated at residential buildings. The main goal of this storage server is to provide a secure, accessible and efficient method to store essential data. The client's requirements are iteratively assessed against the system's present functionality to best meet the necessary implementation.

The storage system design is formed by the decisions taken in this document; however, it only represents the current state of the design and will reflect any future changes in the client's requirements throughout the development process. For the time being, the major implementations are the hash table data structure, client-server protocol, error handling, and user shell interface.

Using this approach for record keeping at residential buildings, we hope to facilitate managerial tasks such as tracking a tenant's rent payments, maintenance requests and effective communication between different entities.

# BIBLIOGRAPHY

## Bibliography

- [1] M. R. Ravi, Interviewee, *Initial Interaction with Client Personnel*. [Interview]. 2 February 2014.
- [2] A. Williams, Interviewee, *Detailed Discussion with the Management Personnel*. [Interview]. 3 February 2014.
- [3] V. Hacuman, Interviewee, *Further Research with Authoritative Staff*. [Interview]. 2 February 2014.
- [4] RFC Sourcebook, "TCP, Transmission Control Protocol," 2012. [Online]. Available:  
 <http://www.networksorcery.com/enp/protocol/tcp.htm>. [Accessed February 2014].
- [5] C. E. L. R. L. R. a. C. S. Thomas H. Cormen, "Introduction to Algorithms," MIT Press, 2009, pp. 253-285.

# APPENDICES

## Appendices

### APPENDIX A – CONFIGURATION FILE FORMAT

Name	Value
server_host	A string that represents the IP address (e.g., 127.0.0.1) or hostname (e.g., localhost) of the server.
server_port	An integer that represents the TCP port the server listens on.
username	A string that represents the only valid username for clients to access the storage server.
password	A string that represents the encrypted password for the username.
table	A string that represents the name of a table. There may be more than one table parameter, one for each table.

Table 2: Configuration file parameters (from design specifications)

#### Sample configuration files

Correct Configuration	Wrong Configuration	Wrong Configuration
server_host localhost server_port 1111 username admin password xxxnq.BMCifhU table marks	server_host localhost server_port 1111 server_port 2222 username admin password xxxnq.BMCifhU table marks	server_host localhost server_port 1111 table marks table marks



# APPENDICES

## APPENDIX B – SECTION WISE AUTHOR/REVISION/PROOFREADING

Contributor	Name	Student Number
<b>AJ</b>	Aryamman Jain	999554076
<b>PM</b>	Pranav Mehndiratta	999480725
<b>VV</b>	Vaibhav Vijay	1000073029

Section	Authored	Revision	Proofread
Introduction	<b>AJ</b> <b>PM</b> <b>VV</b>	<b>AJ</b> <b>PM</b> <b>VV</b>	<b>PM</b> <b>VV</b>
System Architecture – UML Diagrams	<b>PM</b>	<b>PM</b> <b>VV</b>	<b>AJ</b> <b>PM</b>
System Requirements	<b>PM</b> <b>VV</b>	<b>AJ</b> <b>PM</b> <b>VV</b>	<b>AJ</b> <b>VV</b>
Design Decisions	<b>AJ</b> <b>PM</b>	<b>AJ</b> <b>PM</b> <b>VV</b>	<b>AJ</b> <b>PM</b> <b>VV</b>
Conclusion	<b>AJ</b>	<b>AJ</b>	<b>AJ</b> <b>PM</b> <b>VV</b>