5-1-2020

RUST API IMPLEMENTATION FOR KOMODO BLOCKCHAIN FOR EFFICIENT AND SECURE E-COMMERCE TRANSACTION

Ashish Mainali

RUST API IMPLEMENTATION FOR KOMODO BLOCKCHAIN FOR EFFICIENT AND SECURE E-COMMERCE TRANSACTION

by

ASHISH MAINALI

Presented to the Faculty of the Honors College of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

HONORS BACHELOR OF COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2020

ACKNOWLEDGMENTS

I would like to thank faculty at Honors College, friends and family for continuous support and encouragement. I would also like to thank my parents for everything including this life. I am grateful to my girlfriend for bearing with me throughout this journey and making this possible. I would not have been able do this without her continuous support. I am grateful to Dr. Sajib Datta for giving me the opportunity to work on the Komodo Rust API project.

I have learnt through this project about Rust programming, blockchain, and building web applications using rocket framework. This project was made possible with the help of my genius teammates Sudip Ghale, Kenny Hyunh, and Gouldsbrough Scott who spent countless hours to make this project a success. I am thankful for their diligence and patience while working with me during this process.

May 3, 2020

ABSTRACT

RUST API IMPLEMENTATION FOR KOMODO BLOCKCHAIN

FOR EFFICIENT AND SECURE E-COMMERCE

TRANSACTION

Ashish Mainali, B.S. Computer Science

The University of Texas at Arlington, 2020

Faculty Mentor: Christopher McMurrough

Komodo is an open source blockchain server that is available to developers. However, different developers have preference over different languages. As an emerging blockchain technology, it is imperative to have the technology available across different medias available to the world. Application Program Interface are translator programs which work between two different technologies used by using a common language used by both sides. In the case of Komodo blockchain, Hyper Text Transport Protocol (abbreviated as HTTP).

Komodo Rust Application Program Interface (API) is a bridge program that Rust developers can use to communicate with the Komodo blockchain node using HTTP through the internet.

The problem with the current model was that Komodo had APIs for popular languages like C, C++, Python, Java, etc. However, there was no API available for Rust programming language. As Rust is an emerging technology currently in top ten list of most popular languages, it was very imperative to provide the Rust developers with the tools to connect with Komodo blockchain for fast, secure and efficient communication.

The API is available to download using a package manager for Rust. Adding the API will decrease the complexity of setting up the connection with Komodo server and increase productivity. The security of the API makes it attractive to the blockchain and Rust developers. The API is then demonstrated using an app called KPay which is an online web wallet management program. KPay uses the Komodo Rust API to communicate with the Komodo server. For the demonstration purposes, a crypto currency named Kenny Coins was introduced in the Komodo blockchain software to perform the transactions in KPay application.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Vision

The vision is to create a secure and globally accessible digital world using block chain technology.

1.2 Mission

Performance and security have always been a tradeoff in programming languages, and Rust is a new programming language that has performance and security. The mission is to build an API with Rust programming language; it will enable developers to connect to the Komodo block chain to create secure chain networks. For the following semester, our mission is to build an application that will demonstrate how developers can use Rust Application Programming Interface to connect to the Komodo block chain. The program should be able to securely communicate with the client and server through public medium.

1.3 Success Criteria

Upon completion of the Komodo block chain API, we expect the following success indicators to be observed:

- 75% of Rust developers should be able to connect to Komodo block chain using the API.
- 25% reduction in development cost.
- 75% secure HTTP connection from any Rust application to the Komodo block chain.

- 50% reduction in average development time.
- Increase in number of developers using Rust API over 100

Within 6 months after the prototype delivery date, we expect the following success indicators to be observed:

- All Rust developers should be able to connect to Komodo block chain using the API
- An additional 15% reduction in development cost
- An additional 15% increase in security using an HTTP connection
- An additional 10% reduction in average development time
- Increase in number of developers using Rust API over 100

Within 12 months after the prototype delivery date, we expect the following success indicators to be observed:

- An additional 10% reduction in development costs
- An additional 10% increase in security using an HTTP connection
- An additional 5% reduction in average development time
- Helpful to over 1000 new developers using the Rust Komodo API
- Porting of the system to additional hardware platforms, such as Super kiosk and Alpha Pay.

1.4 System Overview

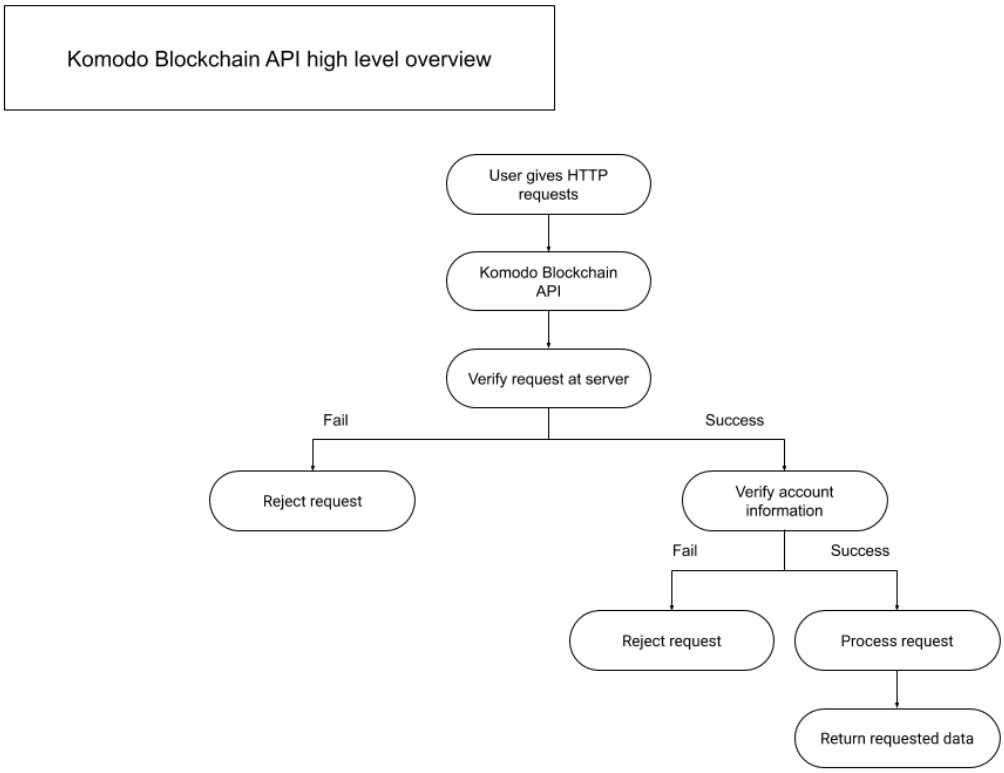

Figure 1.1: Diagram of Major Components of the System

As Komodo requires an API for Rust programs to implement their blockchain, we will have to create one. In order to do this, we will have to go through the Komodo documentation to see what kind of functions can be called and what possible returns will come through. Komodo has a very well-made documentation page, giving us categories of functions and specific functions that might not belong in one. Since these functions display what their parameters are and what they return, we can use that information to figure out how to fill in the functions. A lot of the functions appear to be converting the input that comes through JSON into a useable output that can be sent to the user, and possibly vice versa.

1.5 Purpose and Use

Our product should let users call their functions when they want to access certain information of the Komodo Block chain. If a user imports this API into their code, they can use the given functions that are supplied to check wallet info, address info, transactions, and everything else the Komodo Block chain provides to its users.

1.6 Intended Audience

The intended audience for this API would be Rust developers who wish to create a program using the Komodo block chain. The Komodo Rust API will be open-sourced once it is ready to be used. In addition, it will be passed to Komodo for further analysis and verification.

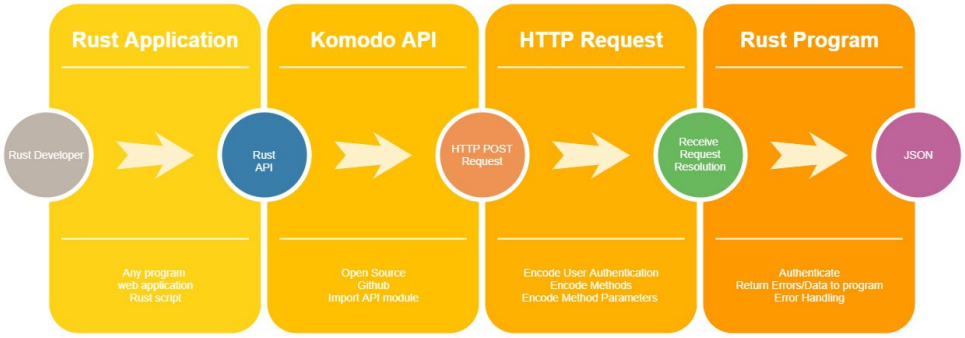

Figure 1.2: Komodo API Overview

CHAPTER 2

LITERATURE REVIEW

2.1 Background

The internet is dependent on a client-server architecture. The problem with the client-server architecture is that if a hacker gets into the server then they will have full control of the whole system. Another limitation with client-server is scalability in real-time. A server needs more resources to keep up with the demand of growing clients. Komodo is an open-source blockchain software that is available on multiple platforms. Its goal is to be developer-friendly by providing developers from all platforms with resources to develop easily in their respective fields. Blockchain is the technology behind bitcoin that has a promising secure future for our vulnerable digital world. Blockchain is a decentralized technology, that validates all the data using a cryptographic algorithm and adds it to the chain of other blocks. In other words, blockchain doesn't have any central agent or server, it distributes the copy of the blockchain ledger to all the node in that chain and adds or updates the ledger when all the nodes validate the transaction. With millions of nodes around the world, it is impossible to hack the blockchain, providing us the most secure digital world. Blockchain technology can distribute the client-server roles to each and every node in the network allowing users to share resources. This project will give an opportunity to create a tool to connect Rust application to the Komodo blockchain and set a smooth communication between them. Komodo is sponsoring this project in order to add more ways to connect to the Komodo blockchain promoting versatility and diversity.

Developing a Rust API and adding it to the Komodo will allow all Rust application developers to easily set function calls on their application in order to communicate with the Komodo blockchain, reducing the development cost and time.

2.2 Related Work

We are planning on creating an API for connecting to the Komodo Blockchain using the Rust programming language. With Komodo gradually growing as a platform, there is an increasing need in a generalized API for any interested developers to use. There are already some existing APIs in some programming languages for this platform. However, while the bigger languages such as Python, C, and Java already have APIs created, the growing community of Rust developers could benefit significantly from this interface dedicated to their language. All of these APIs are freely available in open-source. Facebook has created a Blockchain platform called Libra that is created in Rust itself. While this is also useful for Rust developers, giving them access to another platform to choose from will work in the same sense as offering a new programming language. Developers that already use Komodo will have another language to use, and developers that use Rust will have another platform to use. Bitcoin also has its own API developed in Rust for processing transactions with Bitcoin, which also shows that there is a need for more Rust APIs in other platforms.

2.3 Documentation

Komodo website has a smart chain essential documentation that will be used to build the API. The documentation will also be used to find the essential input and outputs for the API which will be implemented throughout the project. The documentation will be

used to approximate the inputs and outputs of the program and uses appropriate variable types for type guarding.

2.4 Application

Study of web-based application will give a perspective in the development of the application in the Rust programming language. The application will be built using the rocket framework for the Rust programming language. Popular demonstration of the secure application built using rocket framework show enhanced security built into the language framework providing security to the application from the core. Individual study of guide for rocket framework shows security measures like type guarding, path traversal prevention and buffer overflow protection for the application. The default features of the program will be used to secure the KPay against attacks like buffer overflow, cross site scripting, SQL injection, etc.

CHAPTER 3

PRODUCT DESIGN

3.1 Product Concept

Developers who use this API will be able to create Rust programs and applications that communicate with the Komodo blockchain without having to know the way Komodo blockchain communicates with the applications. The API will communicate with the Komodo Server and handle all the requests from the application.

3.2 Product Description

The Komodo Rust API connects developers using Rust programming language with the Komodo daemon running on a local or remote server using HTTP requests. The user will import the Komodo Rust API and call the API to connect with the Komodo server. The key features of the Komodo Rust API are RPC_Util and Config files which will contain the Utility to connect using RPC port and the configuration information like username, password, hostname, etc. Each of the smart chain essential components will be configured in their respective files with similar filename.

3.2.1 Features and Functions

A Rust developer can connect to the Komodo server using the Komodo Rust API. It doesn't connect to the server which is not available within the network or which cannot handle HTTP requests. As shown in Figure 1, any Rust developer can connect with Komodo Rust API. Rust API will require the developer to provide a username, password, host address, RPC port for establishing a connection and authenticating users.

The Rust API will handle smart chain essential commands of Komodo software. All the operations documented under the Komodo website for smart chain essentials will be supported by the API. Developers will be able to pass methods of smart chain with valid parameters.

3.2.2 External Inputs and Outputs

The basic input for the Rust API includes a username and password for authentication. The API connects the Rust application using HTTP over the internet. So, the user must provide the Universal Resource Locator (URL) resolving to an IP address and the Remote Procedure Call (RPC) port number to remotely connect to the Komodo server. Furthermore, the API expects the user to input the correct method and relevant method parameters. The output of the program is either an error from invalid input or a JavaScript Object Notation (JSON). The JSON object could be parsed using a JSON parser to provide string output.

3.2.3 Product Interfaces

There will be methods that will take parameters and output either a result or an error. The code will be available on GitHub as open source. Any developers trying to maintain or update code will be able to see the source code with comments.

3.3 Customer Requirements

3.3.1 Build the API

This Komodo blockchain API shall be built in the Rust programming language. Given user input from the console, the API will send the requested data to the server and retrieve the requested information to be sent back to the user.

3.3.2 Build an Application

The application should be built in a way that will effectively utilize the Komodo blockchain API. The type of application was unspecified by the product owner. The requirements that shall be consistent across any type of application that we, the Senior Design team, choose to build are the following, but not limited to: the user's username and password for authentication, access to the internet, user input and accessibility support. The user will provide information that will be sent to the server and will return the requested data.

3.4 Packaging Requirements

This section provides the reader with an overview of the packaging requirements. These requirements are from our product owner and sponsor, Dr. Datta. The Rust Komodo blockchain API will be open sourced and its pre-built distribution library will be available on GitHub to download and install and on the crates.io, Rust's crate distribution tool. Before using the API, the user shall install the Komodo smart chain software available on Komodo's website.

3.4.1 Project Download

The finished product will be available on GitHub and on the crates.io to download. Rust developers will be able to include the API on their projects by cloning the pre-built library from the GitHub repository, and by installing the project cargo crate files on their project workspace.

The end-users will have two options to use the product. The first will be using GitHub. The finalized pre-built distribution library will be available on GitHub to download and use. In addition, the same library will be published on crates.io, Rust's package distribution tool for the developers. As most of the expected end-users of the

product are developers, the recommended option will be using crates.io. Installing project distribution library through crates.io is very convenient but the Rust application deliverables are expected to have Rust, Cargo, and other system dependencies that the project required installed.

3.5 Performance Requirements

This section provides the reader with an overview of the performance requirements. These requirements are from the Komodo documentation available on their website. The current requirement for the product is defined here. Depending on the configuration of the user's computer, the time required to install the Komodo software is roughly around two hours. The main configurations that will have a great impact on the duration of the installation are a processor, the amount of RAM and storage device type. When connecting to any chain network, the process could take some time to synchronize all the data.

3.5.1 RAM Specifications

A computer with at least 8 GB of RAM with a 64-bit processor is highly recommended to install the Komodo software. Installation is also possible with 4 GB of RAM. The computer may also become unresponsive during the installation if there is an insufficient amount of RAM. Our product owner and the sponsor have directed us to a documentation on the Komodo website to refer to.

3.5.2 Storage Specifications

A computer with a solid-state drive installed, rather than a hard disk drive, is recommended to speed up the installation process. A minimum of 32 GB storage space is recommended. Dr. Datta, our product owner, and the sponsor have directed us to a documentation on the Komodo website to refer to.

3.5.3 Power Specifications

If the installation is done on a laptop, it is highly recommended for the laptop to be connected to an outlet. The installation process may take a few hours and may exceed the current battery charge. The user will have to start over the installation process if the computer shuts down.

3.5.4 Temperature

During the installation of the Komodo software, it is preferred for a room to be at 22 degrees Celsius, or 72 degrees Fahrenheit or cooler. The computer may enable its overheating protocol in order to reduce its temperature and maximize performance. The computer may also become unresponsive during the installation.

3.6 Maintenance and Support Requirements

Experience and knowledge about the Rust programming language are critical in maintaining this Komodo blockchain API. Rust programming language must be installed to run Rust applications. IDE's that support Rust may be used to find defects. There are online resources available to learn more about Rust programming language and also to help troubleshoot errors that may not have a straightforward fix. The Komodo website has documentation stating the input and output requirements of each method. The customer will be responsible for updating this Komodo blockchain API upon major changes in new versions of the Rust programming language.

3.6.1 Rust Programming Language Version

Languages may change over time. The names of built-in functions may change, or the syntax is different compared to a previous iteration of a language. It is vital for the customer to check and note the changes in the Rust programming language for this API.

3.7 Other Requirements

This section provides the reader with an overview of other requirements of this product that were not categorized under previous requirement sections. The network and OS specifications are critical components for the Komodo software to be functional. The Komodo daemon needs to be running and reachable from the developer's machine in order to run the commands on the Komodo server. The program returns a JSON file which can be further tokenized and passed as a string. User can run any operating system that supports Rust programming language and have a Rust programming environment setup.

3.7.1 Network Specifications

The Komodo daemon connects to individual nodes using a P2P port. The network IP must be reachable for connecting peers with the Komodo blockchain. Users need to be on the same subnet or public IP to run the Komodo daemon.

3.7.2 OS Specification

Installation on the Linux OS, Ubuntu or other Debian-based distributions is recommended. If the user chooses to install on Ubuntu, the Komodo's documentation recommends using only 16.04 or 18.04 releases. If the user would like to install on a MacOS, it is recommended to install OS X 10.11 (El Capitan) or newer. The Komodo program on Windows OS must be compiled from a Linux machine and transferred. The user may also use a Linux Virtual Machine to install and use the Komodo software.

3.7.3 Return Result as a JSON Object

The return functionality for these methods would return a JSON object rather than a string. In doing so, the developers may simply call any methods they wish. This would take the effort by the developer to process the string to the JSON object on their end.

CHAPTER 4

DESIGN SPECIFICATION

4.1 Introduction

The product will incorporate all the smart chain essential functionalities provided by Komodo blockchain application. The Rust API for the Komodo Blockchain will enable Rust application developers to integrate their applications with Komodo asset-chain network easily using the functions and methods provided by Rust API. Rust developers will be able to import Komodo Rust API by adding a Komodo API crate to the project. Rust developers will call upon the functions provided by Komodo Rust API which will provide functionalities of Smart Chain by connecting the Rust application with the Komodo server.

4.2 System Overview

Rust API has four main layers to manage the smooth interaction between the Rust application and the Komodo asset-chain network. The four main layers are RPC Config, RPC Request, API Modules and Rust Applications. Developers will be able to configure the connection by providing host URL, RPC port, and authentication information to the RPC Config Layer. After the connection is set, the application will call Smart Chain API Modules which will in turn check the RPC config to find the URL of the Komodo Server along with authentication. The Smart Chain API Module will then make a request to the server by passing methods and parameters provided by the application to the RPC Request Layer. The RPC Request Layer will make HTTP request and handle the response

appropriately and pass it back to Smart Chain API Module which will then pass it back to the Application Layer itself.

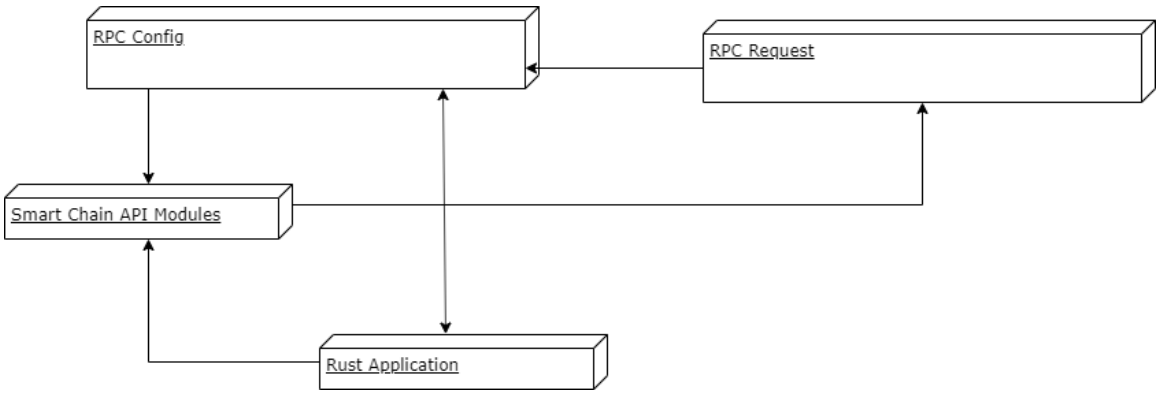

Figure 4.1: A Simple Architectural Layer Diagram

4.2.1 RPC Config Layer

The RPC Config layer holds all the user information received from the Rust Application Layer and hold user authentication credentials. User needs to input RPC address, RPC port number, username, and user password, which is then stored into the RPC Config layer. This layer then checks user authentication prior to handling any request from the Rust Application.

4.2.2 RPC Request Layer

RPC Request layer handles the request received from the Smart Chain API Modules. The received request is sent to the Komodo asset-chain network and the response is then returned to the Smart Chain API Module in the proper format.

4.2.3 Smart Chain API Modules

API Module layer consists of eleven different subsystem modules which contains all the RPC request functions that Komodo Blockchain provides to the Application Layer. Subsystem modules includes Address Module, Blockchain Module, Control Module, Disclosure Module, Generate Module, Mining Module, Jumblr Module, Network Module,

Raw Transactions Module, Util Module and Wallet Module. API Modules layer gets the RPC request calls from the Rust Application Layer, which is then processed through the RPC Request Layers to get the response. The received response is then returned back to the application.

4.2.4 Rust Application Layer

Rust Application Layer represents the end user application which sends the user information to the RPC Config Layer and sends the RPC request to Smart Chain API Modules.

4.3 Subsystem Definitions and Data Flow

The Rust application will get input from the user and validate it. After validation, the application will use Set RPC Config Info method to set the user info and server address to the Komodo RPC Config file in the RPC Config module. The RPC methods will be invoked to get data from the Komodo RPC config. This data will be used by Smart Chain API modules, Request Modules, Rust Application. The Smart Chain API modules will use RPC methods to get data from RPC config and send it to the RPC request module to make a request to the server. The Request module will use the RPC Config methods to generate body from the given RPC config instance. The Rust application will use the data from the RPC Config methods to validate the data later, send the config to the Smart Chain API modules to make requests from individual users.

4.4 RPC Config Layer

The RPC Config layer handles information from the user. Given user data; it will attempt to authenticate, retrieve specific data, then return a prepared string that is ready to be sent as HTTP requests.

4.4.1 RPC Methods Subsystem

This subsystem acts as an interface between RPC Config layer and other layers. Smart Chain API layer calls the functions in RPC Methods subsystem to request data from Komodo RPC Config. The request layer calls RPC module to get username, password, request URL, and generate body using the RPC methods.

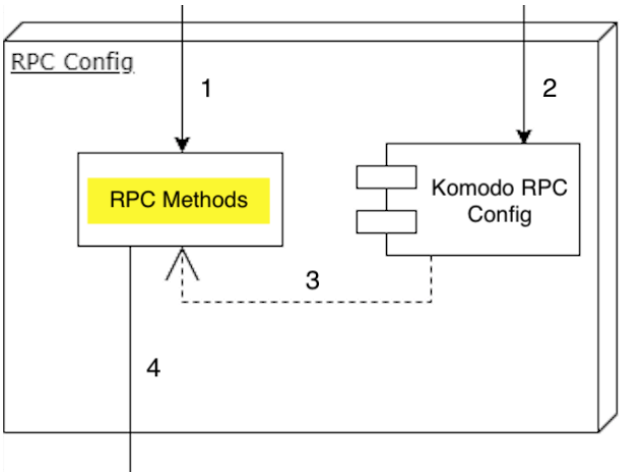

Figure 4.2: Diagram of RPC Config Layer. The RPC Methods Subsystem is Highlighted.

Assumptions: The users have provided valid data to the Komodo RPC Config file.

Responsibilities: The subsystem will return URL, username and password to the Request layer when requested. The method name and parameters are given to this function which returns the body of HTTP request.

4.4.1.1 Subsystem Interfaces

Each of the inputs and outputs for the subsystem are defined here. Create a table with an entry for each labelled interface that connects to this subsystem. For each entry, describe any incoming and outgoing data elements will pass through this interface.

4.4.2 Komodo RPC Config System

The Komodo RPC subsystem holds data about the user. It will be interacted by the interface subsystem of the RPC configuration layer, authentication subsystem and the getter function subsystem.

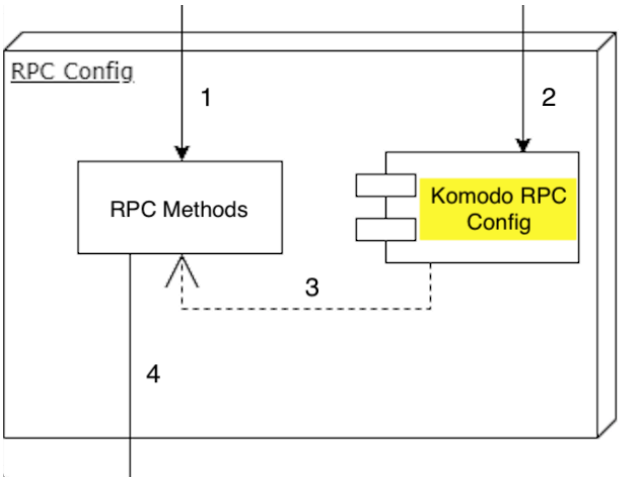

Figure 4.3: Diagram of RPC Config Layer. The Komodo RPC Config Subsystem Is Highlighted

Assumptions: The information provided by Rust Application is validated by the application. All the information passed to the RPC config file is valid.

Responsibilities: The Komodo RPC config is the object that holds information of the user such as username, password. Also, it holds information about the server like server IP address, RPC port number. It also provides URLs ready for request by putting the user info in HTTP body format.

4.5 RPC Request

The RPC Request layer handles and processes requests that are sent to and from the server. Given a method call, it will prepare an HTTP request to a server and will return some value back.

4.5.1 RPC_Request Subsystem

The RPC_Request subsystem will build an HTTP request string that will send information about the method.

Assumptions:

We are making the following assumptions:

1. The request dependency is added to the cargo.toml.

2. The system is connected to the internet.

3. The username, password, IP address, port number are valid in the config file.

Responsibilities: For each reqwest, the RPC_Request subsystem will authenticate the user credential and then build the string that will be sent to the server using the Rust http handling library called reqwest (reqwest = "0.9.22"). The responded string is then sent back to the API that called the request.

4.5.1.1 Subsystem Interfaces

Table 4.1: RPC_Request Subsystem Interface

ID	Description	Inputs	Outputs
#1	The RPC_Request subsystem will build and send an HTTP string to the RPC Methods.	N/A	Output 1
#2	The Smart Chain API Modules subsystem will require the RPC_Request subsystem to return the string result.	Input 5	N/A
#3	The Reqwest subsystem sends the HTTP string request to the server.	N/A	Output 6

4.5.2 Request Subsystem

The Request subsystem is using the dependence of reqwest ="0.9.22". The subsystem runs the finalized string to the server. The subsystem interacts with the RPC_Request subsystem.

Assumptions: We assume this version, 0.9.22, of this package, Request, will be used.

Responsibilities: The Reqwest subsystem will send the request to the server after all processing is successfully completed.

4.6 Smart Chain API

The API modules are the individual groupings of functions that will interact with the developer and create the requests that will get sent to the RPC request function. These will all be tailored to create strings in the format of Komodo requests and return what is given from the requests.

4.6.1 Address

This subsystem consists of the functions that specifically have to do with the different chain addresses. As such, they also require at least one address to be taken for the parameters. Address contacts the API module itself but it does not communicate with the other subsystems.

Assumptions: Assume that there is no way for address to communicate with other modules of its level, such as wallet info or mining.

Responsibilities: The functions within address need to take any parameters required by Komodo and create the string that will be used to send a request to the Komodo site.

4.6.2 Blockchain

This subsystem deals with the interactions regarding individual blocks. With these functions the developer can request different information regarding specific blocks.

Assumptions: As with all subsystems for this module, this does not interact with the other subsystems.

Responsibilities: The functions within blockchain need to take any parameters required by Komodo and create the string that will be used to send a request to the Komodo site. After this it will return the output from Komodo.

4.6.2.1 Subsystem Interfaces

Table 4.2: Blockchain Subsystem Interface

ID	Description	Inputs	Outputs
#1	The RPC_Request subsystem of the RPC Request layer will send and retrieve information from the Komodo RPC Config subsystem.	Input 2	N/A
#2	The Komodo RPC Config subsystem will provide information to the RPC Methods.	N/A	Output 3

4.6.3 CC_LIB

This subsystem consists of the functions that deal with the Antara modules that make use of the -acc_lib parameter.

Assumptions: Assume that there is no way for address to communicate with other modules of its level, such as wallet info or mining.

Responsibilities: The functions within address need to take any parameters required by Komodo and create the string that will be used to send a request to the Komodo site. After this it will return the output from Komodo.

4.6.4 Control

This subsystem mainly exists as an assistant to the user, giving help and state information, and stopping the daemon server.

Assumptions: As with all subsystems for this module, this does not interact with the other subsystems.

Responsibilities: The functions within control need to take any parameters required by Komodo and create the string that will be used to send a request to the Komodo site. After this it will return the output from Komodo.

4.6.5 Cross-Chain API

This subsystem is used for allowing a user to transfer any assets from one chain to another, which includes tokens and coins.

Assumptions: As with all subsystems for this module, this does not interact with the other subsystems.

Responsibilities: The functions within Cross-Chain need to take any parameters required by Komodo and create the string that will be used to send a request to the Komodo site. After this it will return the output from Komodo.

4.6.6 Disclosure

This subsystem is used for generating and validating payment disclosures with only the two methods to it.

Assumptions: As with all subsystems for this module, this does not interact with the other subsystems.

Responsibilities: The functions within Disclosure need to take any parameters required by Komodo and create the string that will be used to send a request to the Komodo site. After this it will return the output from Komodo.

4.6.7 Generate

This subsystem is used for setting up the generation of coins as well as seeing whether it is currently in generation mode.

Assumptions: As with all subsystems for this module, this does not interact with the other subsystems.

Responsibilities: The functions within Generate need to take any parameters required by Komodo and create the string that will be used to send a request to the Komodo site. After this it will return the output from Komodo.

4.6.8 Mining

This subsystem interacts with the mining process with Komodo, offering methods that get current mining info and the respective blocks.

Assumptions: As with all subsystems for this module, this does not interact with the other subsystems.

Responsibilities: The functions within Mining need to take any parameters required by Komodo and create the string that will be used to send a request to the Komodo site. After this it will return the output from Komodo.

4.6.9 Jumblr

This subsystem deals with the zero-knowledge transactions that Komodo offers. Depending on the specific smart chain in the Komodo ecosystem, this might not be available.

Assumptions: As with all subsystems for this module, this does not interact with the other subsystems.

Responsibilities: The functions within Jumblr need to take any parameters required by Komodo and create the string that will be used to send a request to the Komodo site. After this it will return the output from Komodo.

4.6.10 Network

This subsystem offers methods that help with adding new nodes and dealing with bans, specifically for IP addresses. It mainly deals with nodes on the network, offering information about each one.

Assumptions: As with all subsystems for this module, this does not interact with the other subsystems.

Responsibilities: The functions within Network need to take any parameters required by Komodo and create the string that will be used to send a request to the Komodo site. After this it will return the output from Komodo.

4.6.11 Raw Transactions

This subsystem offers methods that can be called when dealing with raw transactions within Komodo.

Assumptions: As with all subsystems for this module, this does not interact with the other subsystems.

Responsibilities: The functions within Raw Transactions need to take any parameters required by Komodo and create the string that will be used to send a request to the Komodo site. After this it will return the output from Komodo.

4.6.12 UTIL

This subsystem offers several utility methods, such as verifying signed messages, estimating memory fees, and additional block information.

Assumptions: As with all subsystems for this module, this does not interact with the other subsystems.

Responsibilities: The functions within Util need to take any parameters required by Komodo and create the string that will be used to send a request to the Komodo site. After this it will return the output from Komodo.

4.6.13 Wallet

This subsystem consists of the largest number of methods, containing everything that has to do with a user's wallet. It has info for the wallet as well as numerous parts of the address corresponding to a wallet.

Assumptions: As with all subsystems for this module, this does not interact with the other subsystems.

Responsibilities: The functions within Wallet need to take any parameters required by Komodo and create the string that will be used to send a request to the Komodo site. After this it will return the output from Komodo.

CHAPTER 5

SECURITY

5.1 Web Application Security

As databases are the backbone of any dynamic program, SQL injection is a very imminent threat. However, the Komodo server handles the input validation for all the remote procedure call (RPC) requests received. In addition to the Komodo sever, there is a layer of input validation in the Komodo Rust API which gives an additional layer of security in the client side before sending it to the server. The SQL injection is not a real threat in case of blockchain because of its distributed data structure. In order to combat SQL injection in KPay, a program filters all the input through a validation method to determine any SQL queries and stop any SQL injection attacks. There were also attacks like Cross Site Scripting (XSS) which were prevented by filtering inputs for html tags.

5.2 Komodo API Security

The API connects the user to the Komodo server using the publicly available network also known as internet. As it passes sensitive information like usernames, passwords along with urls and available RPC port numbers, a security measure is necessary to avoid any security breaches. The Komodo API uses a basic HTTP authentication/encryption method to send the sensitive data over the internet. The Komodo blockchain also relies on the data from the user transported through the Komodo Rust API. The data should be in proper format as expected by server. Various datatypes like text, numbers, decimals are implemented to achieve proper formats for the inputs that are sent to the Komodo server.

CHAPTER 6

CONCLUSION

Komodo API will help users to connect with the Komodo server and save developers the time to connect to the server manually. After the implementation of the Komodo Rust API, the developers will find it easy to connect with the Komodo server in Rust programming language. This will increase the technological advance of blockchain with the Rust programming language. Both Rust development environment and Komodo blockchain will see increase in the numbers of developers and user alike because of easy implementation of the application programming interface.

The secure program KPay developed will provide online wallet to users for secure online transactions. Web applications have various security flaws inherent, however Rust programming language helped cover many of the inherent security flaws and create a dynamic website. Various security features included with Rust like type guarding were essential to creating a secure website

REFERENCES

Aumasson, J.-P. (2013). Password Hashing: The Future is Now.

Katkar Anjali, S., & Kulkarni Raj, B. (2012). Web vulnerability detection and security mechanism. *International Journal of Soft Computing and Engineering (IJSCE), ISSN, 223,* 1-2307.

Manaa, M. E., & Hussein, R. (2016). Preventing cross site scripting attacks in websites. *Asian Journal of Information Technology,* 15(6), 2797-2804.

Scholte, T., Robertson, W., Balzarotti, D., & Kirda, E. (2012, March). An empirical analysis of input validation mechanisms in web applications and languages. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing* (pp. 1419-1426).

Thiyab, R. M., Ali, M. A., & Basil, F. (2017, April). The impact of SQL injection attacks on the security of databases. *In Proceedings of the 6th International Conference of Computing & Informatics* (pp. 323-331).

"1.0.0[−][Src]Crate Std." *Rust*, doc.Rust-lang.org/std/index.html.

"A Gentle Introduction to Rust." *Introduction - A Gentle Introduction to Rust*, stevedonovan.github.io/Rust-gentle-intro/.

Benitez, Sergio. "Guide." *Programming Guide - Rocket Web Framework*, rocket.rs/v0.4/guide/

"Komodo Developer Documentation." *Komodo Documentation*, developers.komodoplatform.com/basic-docs/smart-chains/smart-chain-api/address.html.

Murkute, V. (2019, June 27). V413H4V - Overview. Retrieved May 15, 2020, from https://github.com/V413H4V

BIOGRAPHICAL INFORMATION

Ashish is a computer security enthusiast. He has been interested in the field as long as he can remember. He graduated high school with a major in computer science and physics. He was enrolled in Computer Science at The University of Texas at Arlington. He is graduating with an Honors Bachelor's degree in Computer Science and Engineering. During his academic time at The University of Texas at Arlington, Ashish grew his interest in assembly language, block chain, network and information security. He took most of his honors projects in information security sector. A few of his noteworthy projects include developing custom hashing algorithm and building custom complier for a simple programming language.

Ashish knows his journey doesn't end here; he will be pursuing more challenges in his career as a network security professional. He also plans to go back to college for his further education in computer security field. He plans to research in homogenous encryption algorithms for his future projects.