IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 2 Issue 4, April 2015.
www.ijiset.com
ISSN 2348 – 7968

Remote System Applications Access

Jaskirath Singh Makol[1], Manish Yadav[2], Nitesh Kumar[3] and Vijay Dhama[4]

[1]Computer Engineering, Army Institute of Technology,
Pune, Maharashtra, India

[2]Computer Engineering, Army Institute of Technology,
Pune, Maharashtra, India

[3]Computer Engineering, Army Institute of Technology,
Pune, Maharashtra, India

[4]Computer Engineering, Army Institute of Technology,
Pune, Maharashtra, India

Abstract

In this paper, we will describe a system that allows remote access of a PC to a mobile phone. The proposed system is a simple, fast and secure one that allows mobile devices to interact with personal computers remotely on a network. The basic purpose of the system is to allow a mobile device to remotely access and control some of the features of the computer it is connected to, in real-time. This is like using mobile devices as remote control for personal computers. The system is based on simple client-server architecture wherein the client is the mobile device and server is the personal computer. The client requests to access the running processes on the server and to perform certain operations on them. The server responds by providing remote access to the client. The client is not given unfettered access to the server, so it can access and control a limited set of features only. The user access is kept as simple as possible with minimal setting up time.The system is user friendly and provides faster access leading to better performance than most existing systems.

Keywords:*Remote access; HTTP – Hypertext Transfer Protocol; REST – Representational State Transfer; JSON – JavaScript Object Notation; Client - server architecture*

1. Introduction

Remote access is simply a connection formed via a network to a computer in a remote location. This remote connection enables the local computer to interact with the remote computer. The local computer is the one that controls the connection and the remote computer is the one that is being accessed. So through remote connection a computer one can access another computer without being physically present on its keyboard. Remote access can be on a local network or on the internet. That means you can have access to computers of different departments in the same office remotely [1]. Also, you can have access to your work computer from your home computer by using remote desktop sharing software.

Most of the remote desktop software are based on Virtual Network Computing (VNC) technology. Virtual Network Computing (VNC) is a graphical desktop sharing system that uses the Remote Frame Buffer protocol (RFB) to remotely control another computer [1]. Although VNC is platform independent, there are some limitations:-

a) The VNC based remote solutions are slow performance wise.
b) These systems provide unfettered access to the local host [9].

There are other applications that use SSH protocol for securing the remote desktop sharing and file access. SSH is a remote login protocol used for interacting with a remote device. Although SSH is more secure than Telnet, it has some drawbacks:-

a) Usage via SSH is very complex, it is not user friendly.
b) With many different implementations of the protocol, interoperability has become an issue, e.g. different implementations of the server may crash the client and vice versa [1].

In this paper, we propose implementation of the system that transcends all the above limitations. In our system, the scope is limited to smartphones and laptops or desktop devices connected on a local network such as Wi-Fi. The remote access connection is based on client server model using HTTP. The system basically acts as a remote control with limited functionalities. The system allows only certain applications to be controlled that ensures the user does not have unfettered access to system. The server first authenticates the client that logs into the system. This involves validating the user's credentials by the server. Once authenticated, the client can then select a particular application and request the server to change its state, for e.g. pausing a running song on a media player. The server interprets this request and performs operation on the particular process. This allows the mobile device to interact with the system in real time. All this is achieved by implementing a RESTful API for communication between the client and server. REST i.e. Representational State Transfer is a simple stateless architecture that runs over HTTP and is a lightweight, easy to use alternative to web services like SOAP [2][3].

This paper is organized into six sections: Section 1 specifies introduction of the topic, Section 2 defines the technology stack of the system, Section 3 defines the

design of the system, Section 4 describes the methodology of the system, Section 5 depicts the future work and Section 6 states the conclusion.

2. Technology Stack

The technologies that are used in developing the system are discussed. On server side the following technology stack is used.

2.1. Linux

The remote server is being implemented on a machine running the Ubuntu distribution of GNU/Linux OS. Ubuntu is open source, easy to use and offers better customizability. The APIs that we want to implement for the working for our system can be easily coded in a Linux distribution.

2.2. Rails Server

Ruby on rails or simply Rails is an open source web application framework. It's a software library that is built on Ruby. It enables easy creation of advanced database-driven websites using scaffolding and code generation, following convention over configuration, which means that if you stick to a certain set of conventions, a lot of features will work right out of the box with very little code.Ruby on Rails uses the Model-View-Controller (MVC) architecture to organize the web application programming. It is extremely productive, with Rails a web application could be developed at least ten times faster than with a typical Java framework [6].

The Rails web server that is being used in our system is Unicorn. Unicorn's principle is doing only what needs to be done by a web application server and delegating other responsibilities to modules that perform them better. Unicorn's master process spawns workers, as per user's requirements, to serve the requests. This process also monitors the workers in order to prevent memory and process related staggering issues [5]. It uses the OS for load balancing, not allowing the requests to pile up against busy workers spawned. Also, all workers run within a given isolated address space, serving only one request at a time. Unicorn has the ability to listen to multiple interfaces.

2.3. Sidekiq

Sidekiq is a background message processing framework for Ruby. It is simple enough to integrate with any modern Rails application and delivers much higher performance than other existing solutions. Sidekiq allows you to move jobs into the background for asynchronous processing. The Sidekiq message format is quite simple and stable, it's a Hash in JSON format.This requires three parts:-

1. Client

The Sidekiq client i.e. Rails Unicorn in our system, runs in the web application process and allows to push jobs into the background for processing. The Sidekiq client API uses JSON.dump to send the data to Redis.

2. Redis

Redis provides data storage for Sidekiq. It holds all the job data along with runtime and historical data in a queue to power Sidekiq's Web UI.

3. Server

The Sidekiq is designed for asynchronous processing of jobs that can be completed in isolation and independent of each other. The Sidekiq server pulls jobs from the queue in Redis in the order in which they were pushed and processes them. Like web processes, Sidekiq boots Rails so that jobs and workers have full Rails API, including ActiveRecord, available for use. The server will instantiate the worker and pass the arguments. A 'worker' is defined as a thread currently processing a job. The Sidekiq server pulls the JSON data from Redis and uses JSON.load to convert the data back into Ruby types [7].

2.4. Ruby

Ruby is a dynamic, object oriented programming language created by Yukihiro Matsumoto. It's a scripting language used mainly for web development. Some of the features of ruby are:-

a) *High Level:* Ruby is a high level language, syntax is very similar to English.
b) *Interpreted:* It means that there is no requirement of compiler to write and run ruby. It has the ability to make operating system calls directly. It includes string operations and regular expressions. It also provides immediate feedback during development.
c) *Object oriented:* It allows users to manipulate data structures called objects to build and execute programs. Everything in ruby is treated as an object.
d) *Quick and easy:* In ruby, variable declarations are unnecessary. Syntax is simple and consistent and memory management is automatic.

All the server-side scripting is done in ruby.

On the client side the Android platform is used for the GUI.

2.5. Android

Android is an open source Linux-based operating system for smartphones and tablets created by Google. It supports various applications, available through the Google Play Store. The Android platform allows end users to develop, install and use their own applications on top of the Android framework [8]. The Android architecture is divided into five layers:-

1. Kernel

The Android OS is derived from Linux kernel 2.6 compiled for mobile devices. This provides basic system functionality like process management, memory management, device management like camera, keypad,

ISSN 2348 – 7968

display etc. Also, the kernel handles the networking part and all the device drivers.

2. Libraries

This layer consists of Android native libraries. These libraries are written in C/C++. Some of the major libraries are:-

a) *Surface Manager:* It manages the display and compositing windowing manager.
b) *Media framework:* This supports different audio and video formats and codecs.
c) *System C Libraries:* Standard C library like libc targeted for ARM devices.
d) *OpenGL ES Libraries:* These are the graphics libraries for rendering 2D and 3D graphics.
e) *SQLite:* It's a database engine for Android.
f) *SSL libraries:* These libraries are responsible for Internet security.
g) *WebKit:* It's an open-source Web browser engine.

3. Android Runtime

This layer consists of the Dalvik Virtual Machine which is like the Java Virtual Machine, but designed and optimized specially for Android. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine.The Android Runtime layer also consists of set of core libraries which enable the development of Android apps in Java.

4. Application Framework

The application framework provides a set of APIs in the form of Java classes. These APIs can be used by developers for various standard purposes.

5. Applications

This is the top-most layer of the architecture. The various Android apps e.g. contacts, music player, games, etc. that are installed on the device are present on this layer [8]. Anyone can develop these apps and make them available on Google Play Store. These apps are installed directly, without the need of integration with the Android OS.

For our system, the Android app development has been done in Android Studio, which is the official IDE for Android application development.

3. Design of System

The design of the proposed system is focussed to provide an easy to access and interactive system that supports multiple users and not only peer-to-peer. This real-time system has an Android app as GUI and Ruby on Rails to provide the necessary API backend.

3.1. GUI

The Graphical User Interface has been developed on the Android platform. The Android app presents a login system that is used to validate the user using the backend API [8]. After successful authentication, the user can access the application stack, through the rails server. Application manipulation in this system would be just one click away increasing the ease of access to the user.

3.2. Working

The working of the system is explained in the following four steps:-

a) When a new user interacts with the system, she first registers herself by filling her credentials. These credentials include a Username and a Password. This information is stored in a database. So the user uses the Android app to fill the necessary credentials directly in the database. Refer Figure1.

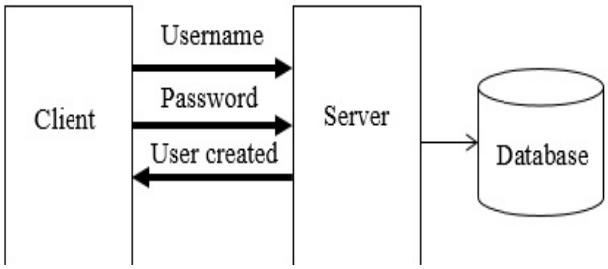

Figure 1. Client Server Interaction

b) The second step is user login. The user enters her credentials which are validated by the main server. For authentication, the Rails server checks the username and password combination of the user against the details stored in the database. If authenticated, the user is given access to the system, as shown below in Figure 2. If user enters wrong combination of username and password, an appropriate message is displayed.

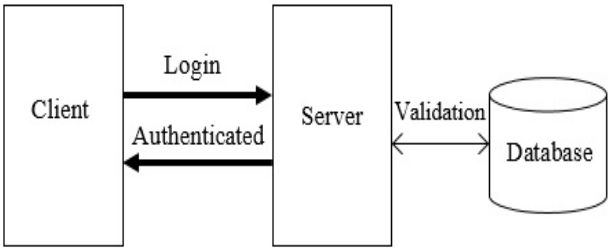

Figure 2. Client Login and Authentication

c) Once the user has been authenticated, the user sends a HTTP request message to the server asking for a list of all the processes running at the server-side. The request message uses the GET request method to retrieve information from the server. The server then sends the list of all the currently running processes to the client in the JSON format, refer Figure 3.

ISSN 2348 – 7968

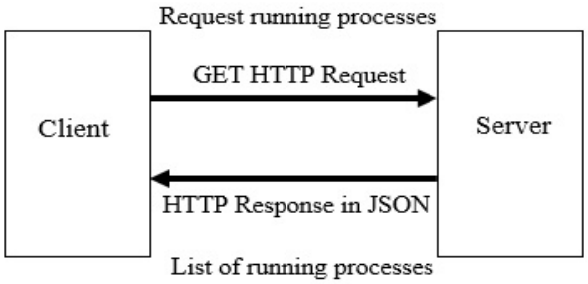

Figure 3. Client requests list of running processes

d) The client is now able to see a list of running processes in the GUI. The client again sends a HTTP request to the server, this time, requesting to change the state of a running process. Again, GET request method is used in the request message. An example of the request sent by client to pause a running song in VLC media player: http://root/api/vlc/?op=pause.
The Rails server performs the requested operation with the help of Sidekiq. The server then sends the response to the client stating that the requested operation has been completed, this is described in Figure 4.

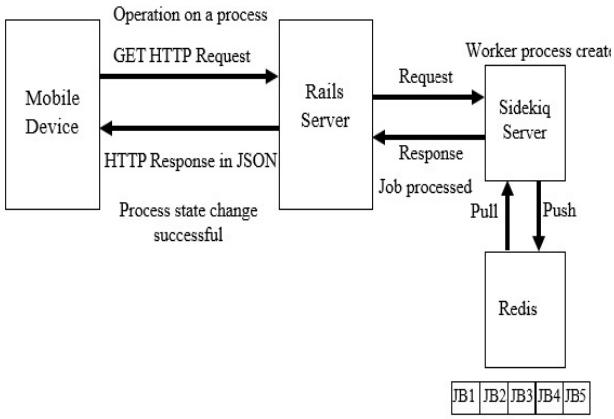

Figure 4. Client requests list of running processes

3.3. Approach to Implementation

The system is broadly based on the REST services architecture. It is implemented in following three steps:-

a) The client side is a mobile device running the Android OS. The user uses the Android app for remote access.
b) At the server side, the Rails server is implemented on a Linux machine running the Ubuntu OS. This is the main server handling all the client requests.
c) The Sidekiq is a secondary server responsible for processing the state change requests by client.

4. Methodology

The system is a client-server system, essentially using the REST based architecture for application development [4]. HTTP is used for communication between client and server. The above mentioned technology stack is used for implementation of the system.

The first interaction of the user is with the Android based GUI wherein the user registers herself by entering the required credentials i.e. username and password. These details are stored in the PostgreSQL database directly, Figure 5. The user then logins to the system by typing in her username and password. The server validates the user's credentials and on authentication provides access to the system. The client is given limited access to the system meaning only some of the server's processes can be accessed by the client. A default list of all such processes is present with the server. When the client requests a list of running processes, the server matches all the currently running processes with the default list and returns the list of matched processes to the client. The user then sends a GET HTTP request to change the state of a running process [3]. When the Rails server receives such a request, it further sends this request to the Sidekiq server. The Sidekiq creates a worker process for this job and pushes it to a queue at the Redis database. Sidekiq performs asynchronous processing of jobs. It pulls out a particular job from Redis queue following the FIFO rule, instantiates the worker for that job and processes it [7]. The result is returned to the main server, which sends the JSON response to the client. The user is able to see a success message on the interface, Figure 6. The user can then log out from the system.

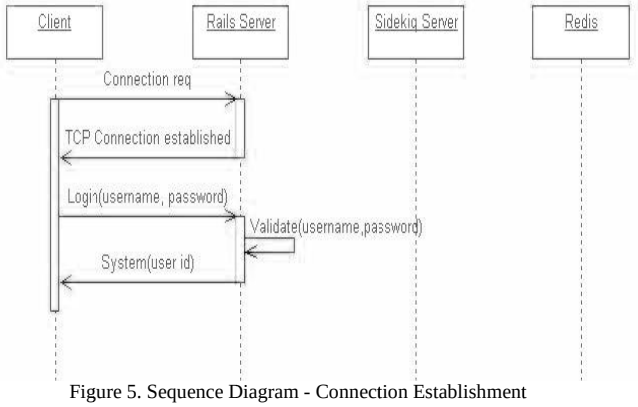

Figure 5. Sequence Diagram - Connection Establishment

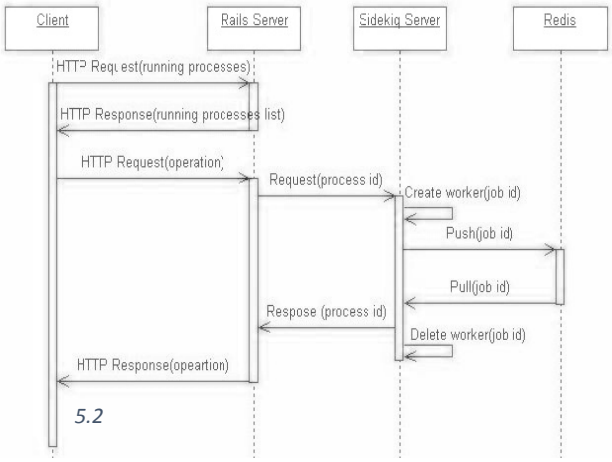

Figure 6. Sequence Diagram - Client Server Remote Access Mechanism

5. Future Work

ISSN 2348 – 7968

The system is currently designed to work in the local network only. The system can be extended to work on the internet using a proxy system which will work as middleware for the mobile device and the personal computer. Then the Android app of the system can be released in the user space for free download from the web. The system will then be made open source so that the source code is available to anyone who can then contribute any enhancement or bug fix to the system. Also, encryption protocol like HTTPS can also be implemented to make the request-response mechanism more secure.

6. Conclusion

This system being RESTful in nature is very lightweight i.e. requiring less memory in comparison to other conventional web applications that use SOAP. It provides faster remote access to client as requests are handled asynchronously in complete isolation and independent of each other. Most importantly, the user is not given unfettered access to the system. The user can only change only certain limited processes which makes the system secure from the administrator's perspective.

Acknowledgements

Prof. Nikita Gupta, project guide at Army Institute of Technology, Pune wishes to acknowledge Jaskirath Singh Makol, Manish Yadav, Nitesh Kumar and Vijay Dhama for developing a system to support remote applications access. The development of this system is totally attributed to the guidance of above mentioned.

References

[1] Bradley Mitchell, What Is Remote Access to Computer Networks?,http://compnetworking.about.com/od/internetaccessbestuses/f/what-is-network-remote-access.htm
[2] M. Vaqqas, RESTful Web Services: A Tutorial, http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069?pgno=1
[3] Dr. M. Elkstein,
Learn REST: A Tutorial, http://rest.elkstein.org/
[4] Margaret Rouse,
REST(representational state transfer), http://searchsoa.techtarget.com/definition/REST
[5] O. S. Tezer, A Comparison of (Rack) Web Servers for Ruby Web Applications, https://www.digitalocean.com/community/tutorials/a-comparison-of-rack-web-servers-for-ruby-web-applications
[6] Daniel Kehoe, What is Ruby on Rails, http://railsapps.github.io/what-is-ruby-rails.html
[7] Mike Perham, https://github.com/mperham/sidekiq/wiki
[8] Rupali Sharma, Developing for Android - An Introduction, http://www.cprogramming.com/android/android_getting_started.html
[9] R.Manikandasamy, "Remote Desktop Connection Using Mobile Phone", International Journal of Science, Engineering and Technology Research (IJSETR) Vol. 2, Issue 8, August 2013, pp. 1629-1633

Jaskirath Singh Makolis currently pursuing Computer Engineering from Army Institute of Technology, Pune affiliated to Savitribai Phule Pune University.

Manish Yadav is also currently pursuing Computer Engineering from Army Institute of Technology, Pune affiliated to Savitribai Phule Pune University.

Nitesh Kumar is also currently pursuing Computer Engineering from Army Institute of Technology, Pune affiliated to Savitribai Phule Pune University.

Vijay Dhama is also currently pursuing Computer Engineering from Army Institute of Technology, Pune affiliated to Savitribai Phule Pune University.
He is also a freelance coder.