

A SIMPLE WEB SERVER

Prepared By

Julia Brozovich

Narayana Reddy

Abdulhadi Alalmai

Ibrahim Alqahtani

Supervised By

Dr. Ayad Barsoum

28 March 2021

Table of Contents

[Introduction 3](#_Toc67857167)

[Project Specifications 4](#_Toc67857168)

[Design 4](#_Toc67857169)

[Algorithm 5](#_Toc67857170)

[User Manual 5](#_Toc67857171)

[Summary and Conclusion 8](#_Toc67857172)

[Future Work 8](#_Toc67857173)

[References 9](#_Toc67857174)

[Code Listing 10](#_Toc67857175)

# Introduction

A web server is software and hardware as shown in Figure 1 that responds to client requests via the World Wide Web using HTTP (Hypertext Transfer Protocol) and other protocols. A web server's main responsibility is to display website content by storing, processing, and delivering webpages to users.

Web server hardware links to the internet and enables data to be shared with other connected devices, while web server software governs how users access hosted files. The client/server model is illustrated by the web server operation. Web server software is needed on all computers that host websites.

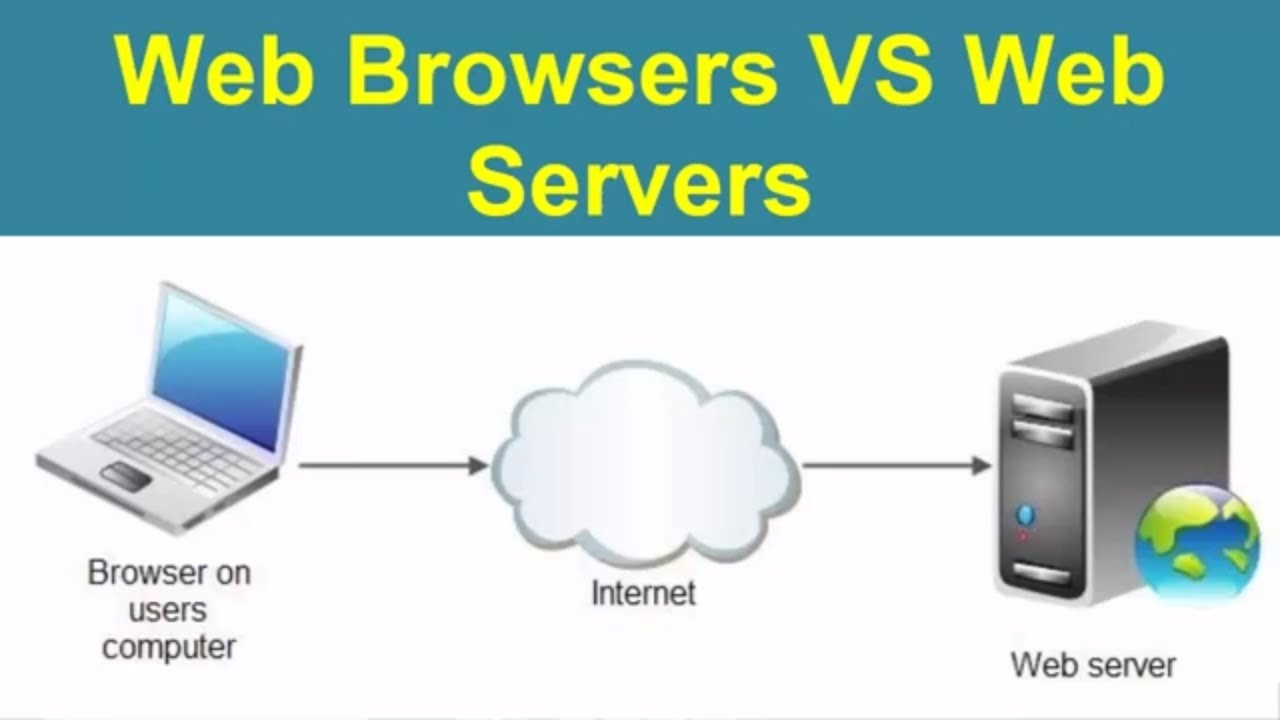


Figure 1 Web Server

In this report, the implementation of a simple web server in Python programming language is presented. The web server is capable of processing one client request at a time. The web server listens for incoming connections from clients which are Internet browsers and then returns the requested documents from the web server file system. if the requested file is not found in the web server file system, then the web server responds with a 404 File Not Exist message. This message is then displayed on the client web browser and then the web server terminated the connection.

# Project Specifications

The software application developed in this project is a simple web server that listens for incoming connections from clients through web browsers. Then, the web server looks for the requested file in the file system of the web server and returns it if it is existing or returns a file not exist error message.

# Design

The block diagram of the web server is shown in Figure 2. The module socket is used in the project to create socket and listen for and communicate with requests from clients. The web server listens from requests at the localhost and at a port number 8080. The library sys is used by terminate the web server program. The datetime library is used to add the current date and time to the response header of HTTP response messages.

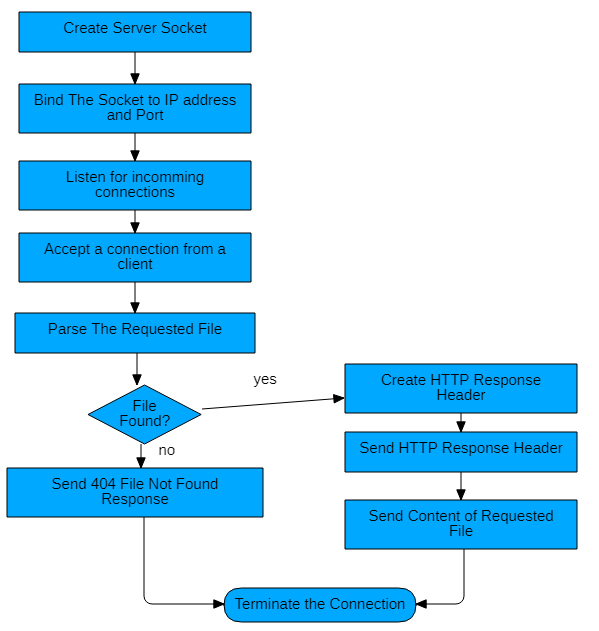


Figure 2 Block diagram of the web server

# Algorithm

The web server uses the following steps to communicate and respond to requests coming from clients:

1. The web server establishes a server socket
2. It listens from request at localhost at port number 8080
3. It receives the HTTP request from client’s web browser
4. It parses the request to determine the specific file being requested
5. It gets the requested file from the server’s file system
6. It creates an HTTP response message consisting of the requested file preceded by header lines
7. It sends the response over the TCP connection to the requesting browser.
8. It returns a “404 Not Found” error message if a browser requests a file that is not present in file system of the server.

# User Manual

The code of the web server was written using python programming language. The Spyder program which is installed with Anaconda was used to run the web server.

The following screenshots show the web server running in Spyder and show the response from the web server when requesting some existing and non-existing files in the server.

Firefox web browser was used to test the web server. It was connected to the localhost at port number 8080 followed by file name.

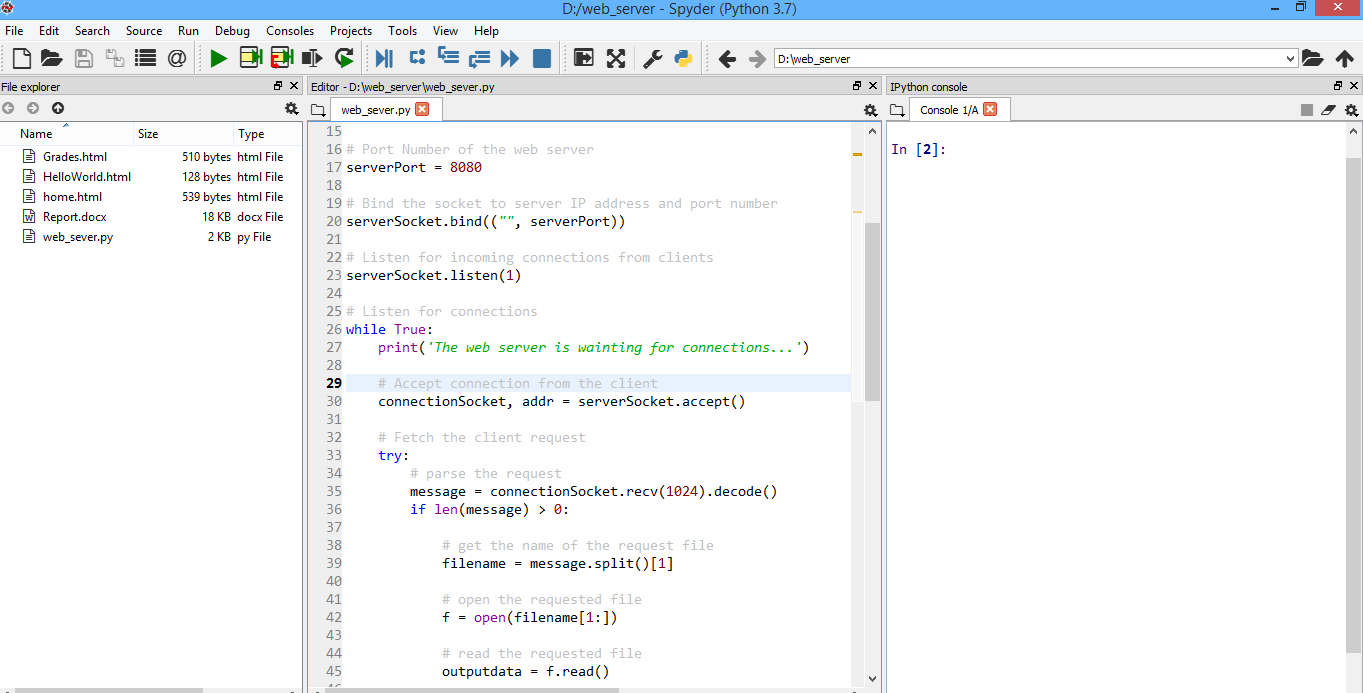


Figure 3 Web server application

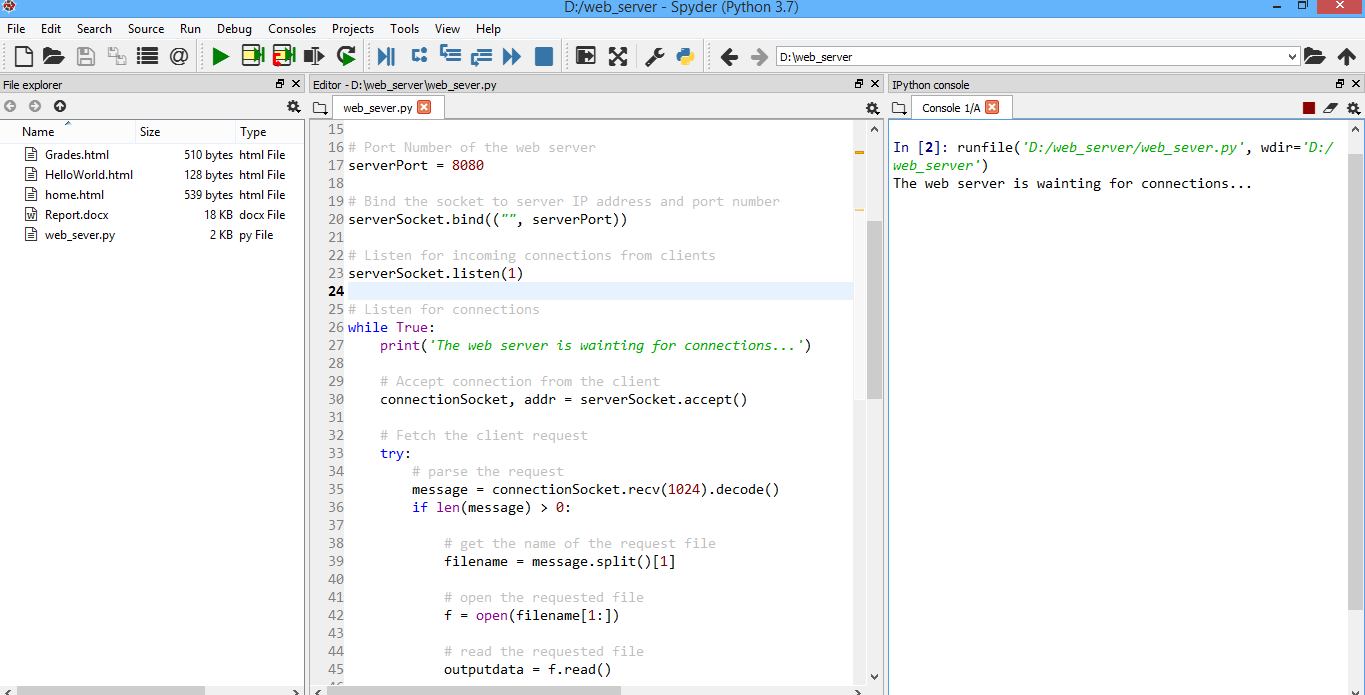


Figure 4 Web server running in Spyder

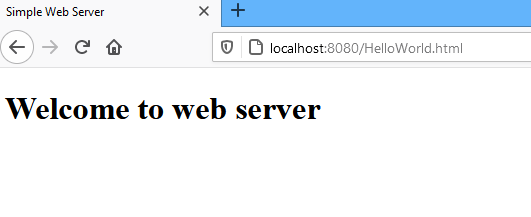


Figure 5 Request to the web server for file

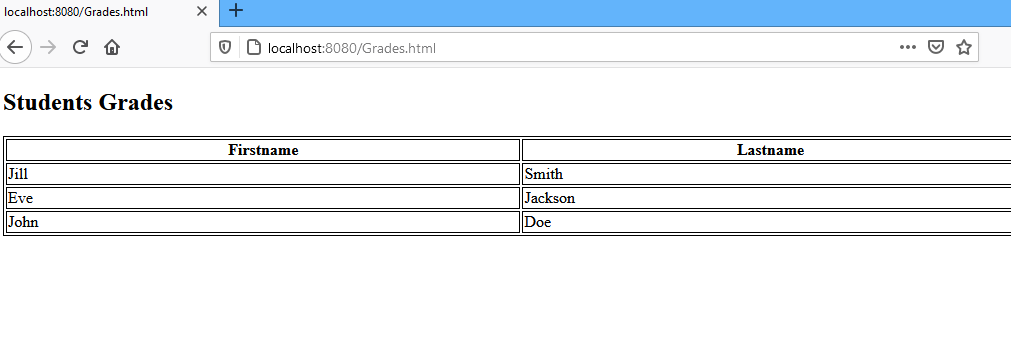


Figure 6 Request to the web server for a second file

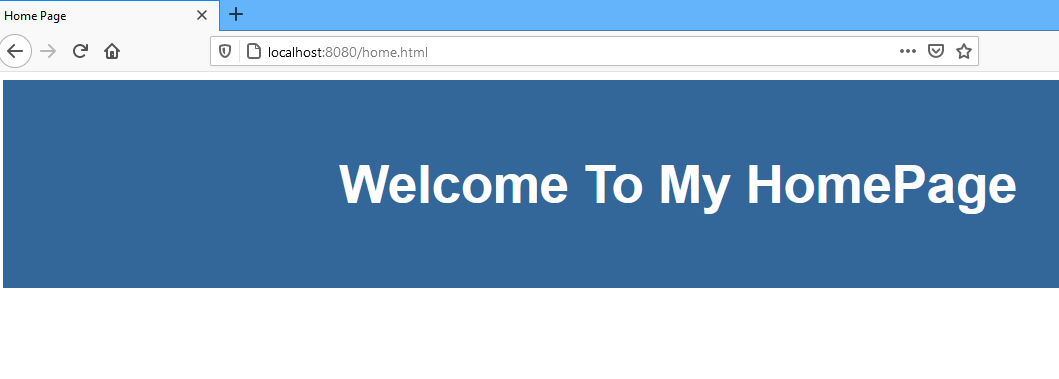


Figure 7 Request to the web server for a third file

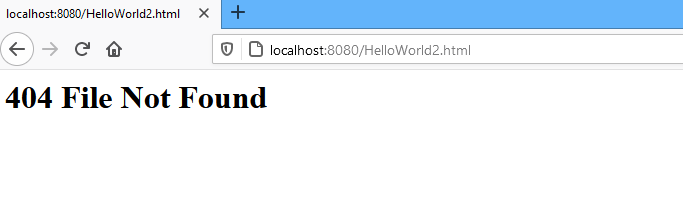


Figure 8 Request to the web server for non-existing file

# Summary and Conclusion

In this project, a simple web server was implemented using python language. The web server serves request from one client at a time. The web server was implemented using Spyder software program. The web server was tested for from a client running Firefox Mozilla web browser.

# Future Work

The future work will be improving the existing simple web server to support different types of web pages. Also, the current web server will be improved to support other protocols.

# References

1. Alexander S. Gillis, “web server”, https://whatis.techtarget.com/definition/Web-server
2. <https://www.python.org/>
3. <https://www.mozilla.org/en-US/firefox/new/>
4. <https://www.anaconda.com/>
5. <https://www.spyder-ide.org/>
6. <https://docs.python.org/3/howto/sockets.html>
7. <https://docs.python.org/3/library/sys.html>
8. <https://docs.python.org/3/library/datetime.html>
9. Create a Python Web Server, <https://pythonbasics.org/webserver/>

# Code Listing

# Simple Web Server

# This web server listens for requests from clients and responses with the requested file.

# import socket module

**from** socket **import** **\***

# import datatime module to add the date and time to the header of the response

**import** datetime

# import sys module to exit the program

**import** sys

# Establish a TCP server socket

serverSocket **=** socket**(**AF\_INET**,** SOCK\_STREAM**)**

# Port Number of the web server

serverPort **=** 8080

# Bind the socket to server IP address and port number

serverSocket**.**bind**((**""**,** serverPort**))**

# Listen for incoming connections from clients

serverSocket**.**listen**(**1**)**

# Listen for connections

**while** **True:**

**print(**'The web server is wainting for connections...'**)**

# Accept connection from the client

connectionSocket**,** addr **=** serverSocket**.**accept**()**

# Fetch the client request

**try:**

# parse the request

message **=** connectionSocket**.**recv**(**1024**).**decode**()**

**if** len**(**message**)** **>** 0**:**

# get the name of the request file

filename **=** message**.**split**()[**1**]**

# open the requested file

f **=** open**(**filename**[**1**:])**

# read the requested file

outputdata **=** f**.**read**()**

# Create the HTTP header

# get the current date and time

now **=** datetime**.**datetime**.**now**()**

# create the status line of the response

statusLine **=** "HTTP/1.1 200 OK\r\n"

# create the header of the response

headerInfo **=** **{**"Date"**:**now**.**strftime**(**"%Y-%m-%d%H:%M:%S"**),**

"Content-Type"**:**"text/html"**,**

"Charset="**:** "uuiltf-8"**,**

"Content-Length"**:** len**(**outputdata**),**

"Keep-Alive"**:** "timeout=%d,Max%d"**%(**10**,**100**),**

"Connection"**:** "Keep-Alive:"**}**

headerLines **=** "\r\n"**.**join**(**"%s:%s"**%(**item**,** headerInfo**[**item**])** **for** item **in** headerInfo**)**

HTTPResponse **=** statusLine **+** headerLines **+** "\r\n\r\n"

# Send the HTTP response header line to the clinet

connectionSocket**.**send**(**HTTPResponse**.**encode**())**

# Send the content of the requested file to the client

**for** i **in** range**(**0**,** len**(**outputdata**)):**

connectionSocket**.**send**(**outputdata**[**i**].**encode**())**

connectionSocket**.**send**(**"\r\n"**.**encode**())**

# Terminate the connection

connectionSocket**.**close**()**

**except** IOError**:**

# Send HTTP response error message for file not found if the file does not exists in the server

connectionSocket**.**send**(**"HTTP/1.1 404 File Not Found\r\n\r\n"**.**encode**())**

connectionSocket**.**send**(**"<html><head></head><body><h1>404 File Not Found</h1></body></html>\r\n"**.**encode**())**

# Terminate the connection

connectionSocket**.**close**()**

serverSocket**.**close**()**

sys**.**exit**()** # Terminate the program