PROJECT REPORT

Tic Tac Toe

Powered by
C Web Server
Web Socket Server
HTML Javascript Client

ARIJIT MUKHERJEE
17305T002
HIMADRI SHEKHAR BANDOPADHYA
173050004

CS 744 Course Project

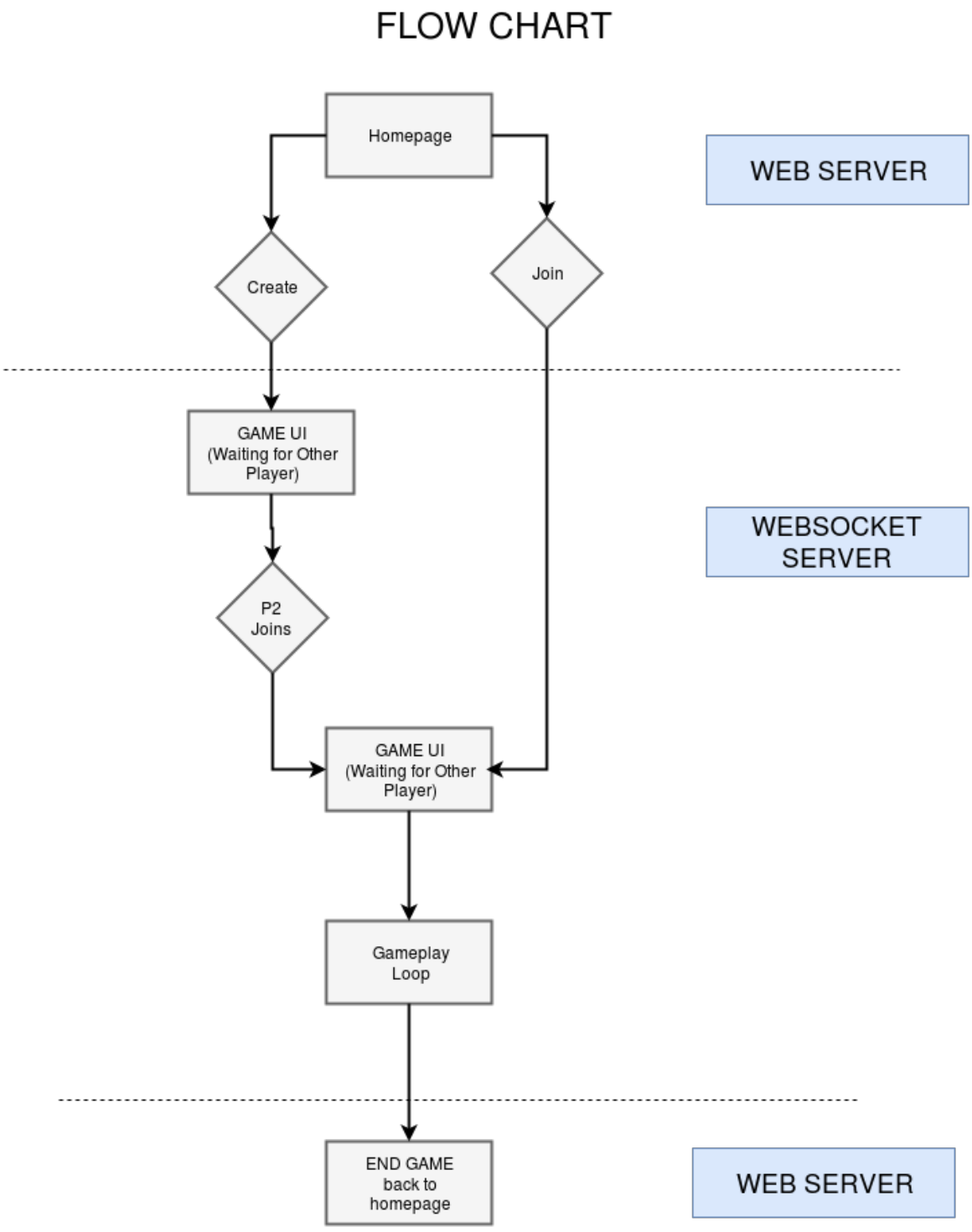
Brief Overview

Here we built a simple realtime Multiplayer Tic Tac Toe game using C , C++ in the server side and HTML Javascript on the Clinet Side . A Webserver in C is used to Serve the html , Javascript and CSS pages . Multi threaded architecure is followed in the WebServer using the Pthread Library in C . The Webserver is built from scratch using UNIX Sockets which follow the HTTP 1.0 protocol . The realtime Game between multiple players is achived using the modern websocket protocol . We built a Websocket Server from scratch . The Client uses Javascript and Jquery to communicate witht the webserver and Websocket to communcate with the web socket server .

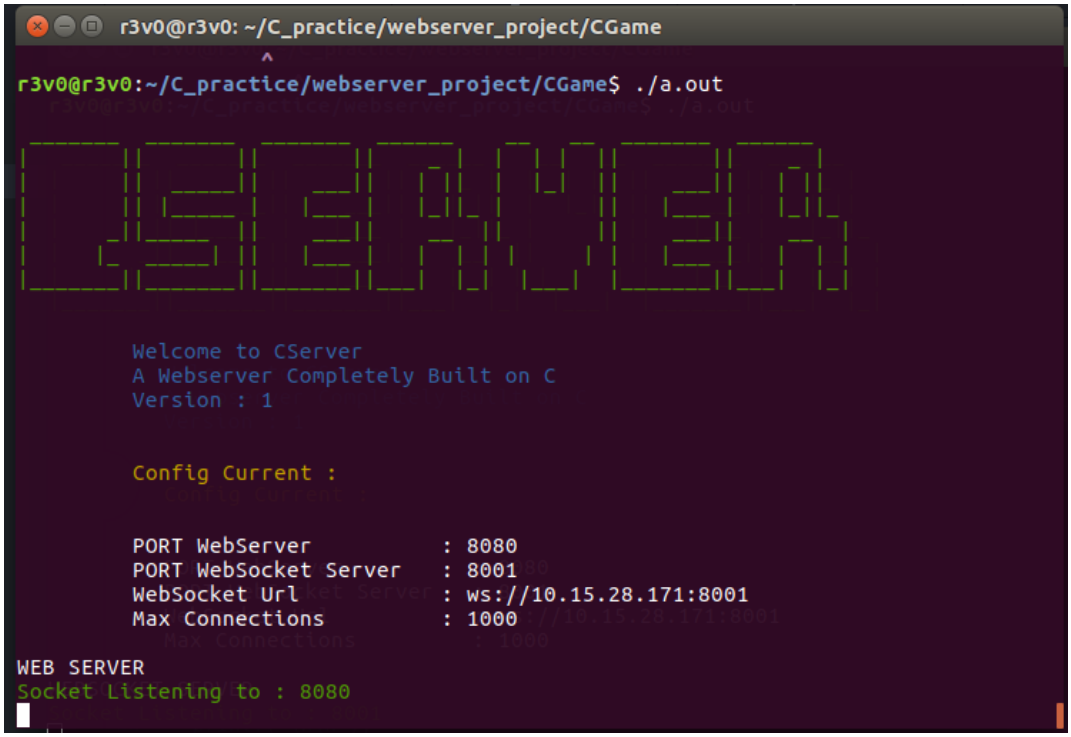
The game works in the following way,

- Player 1 creates a game genrates a game id
- Player 2 joins the game using the game id genreated by the player 1
- then both connect to the websocket server with game id and their username
- then the game is played over the websocket server
- when the game is ended the result is issued to all the clinets and they are disconnected from the websocket server .

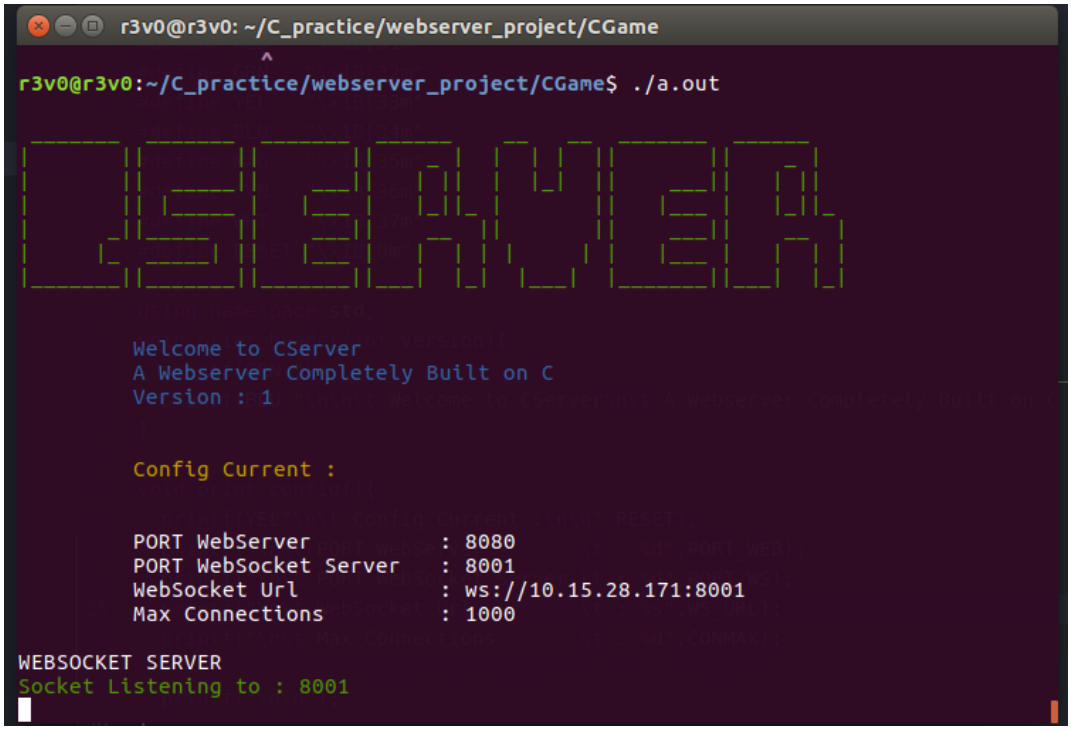
Application Workflow



Running Web Server and Web Socket Server

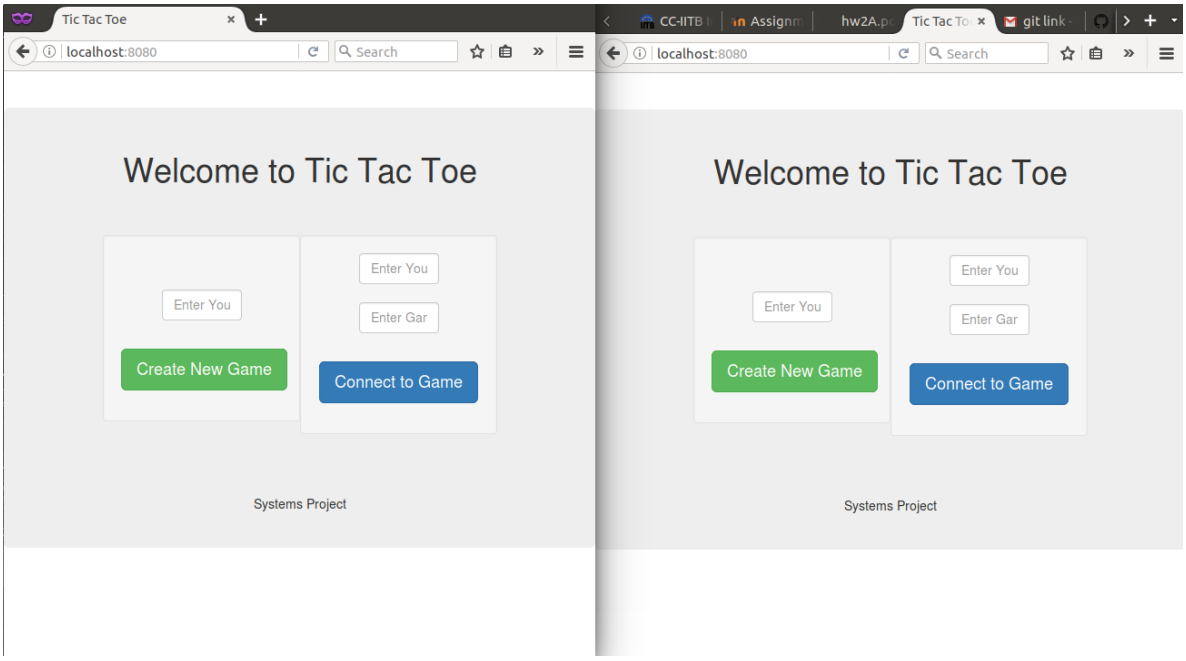


Starting Web Server from terminal

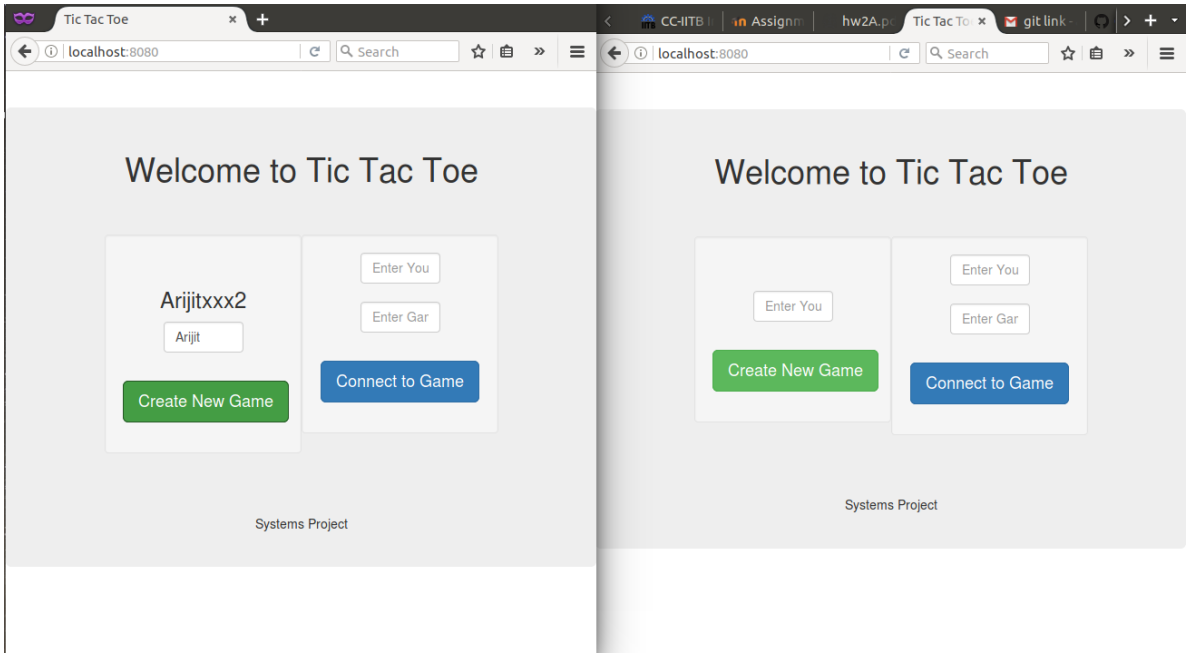


Starting Web Socket Server from terminal

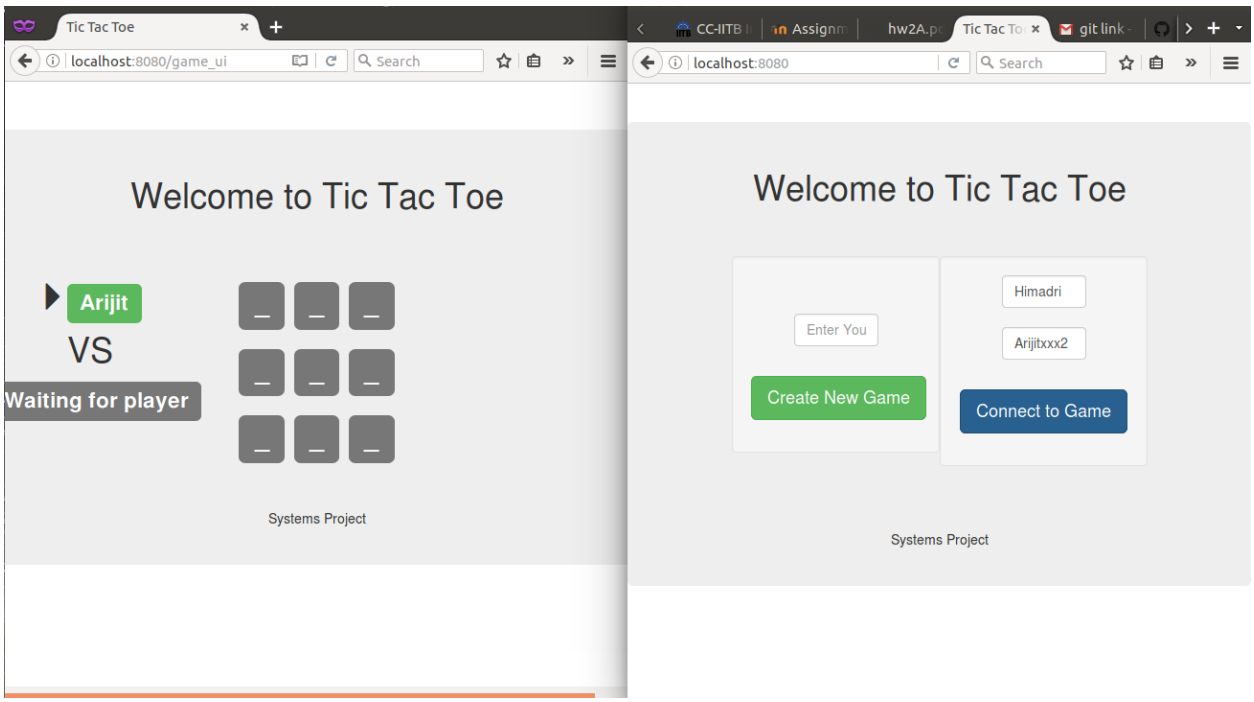
Client Side



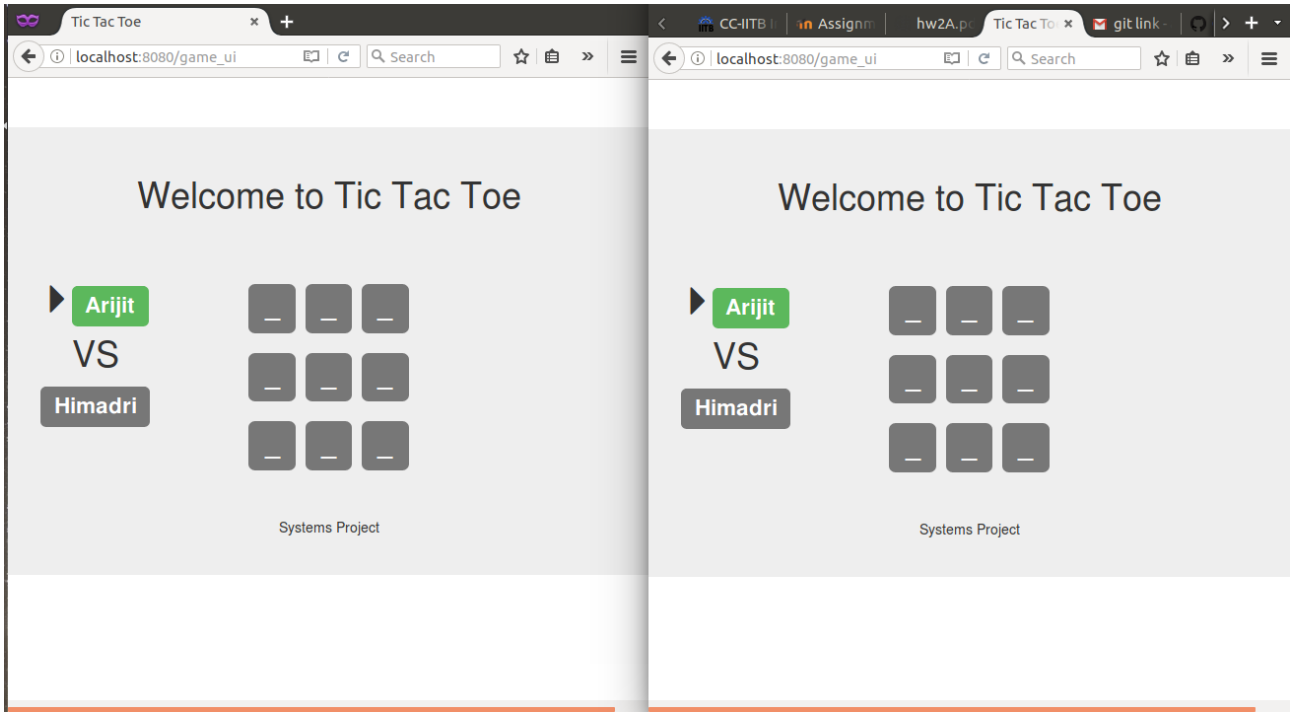
Client Side Home Page



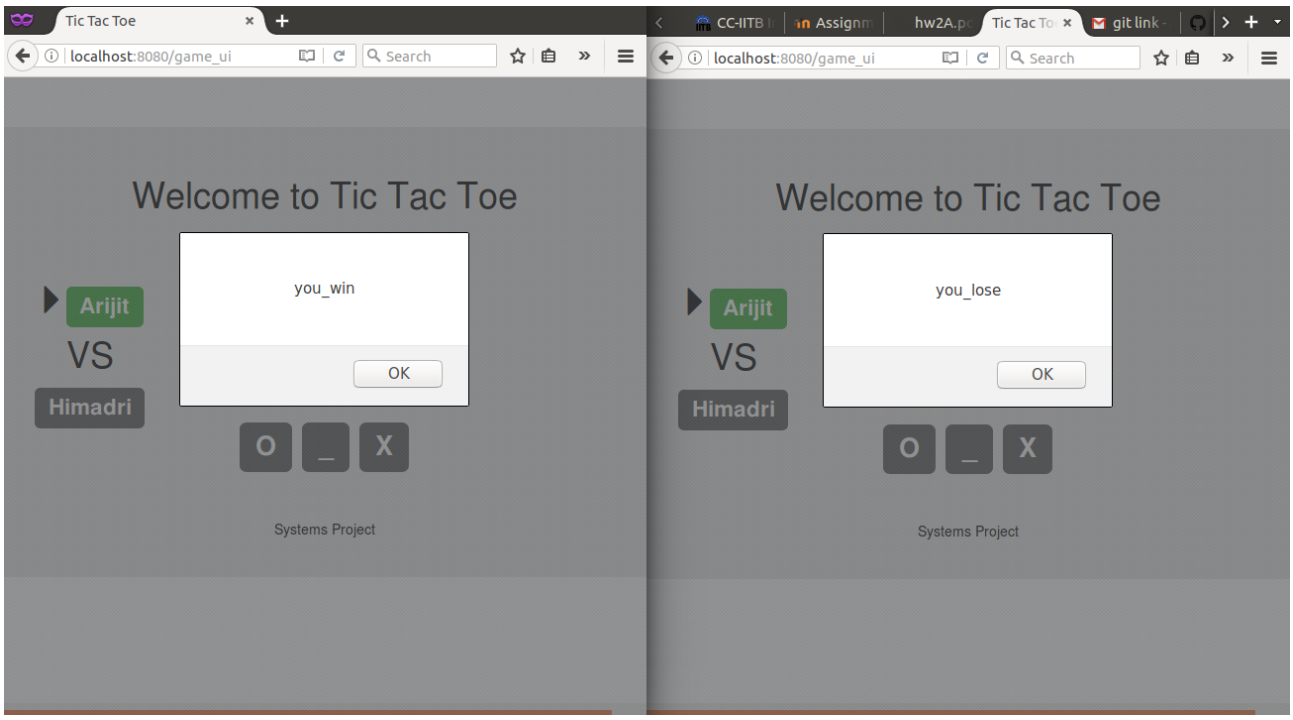
Player 1 Creates Game and Gets Game Id



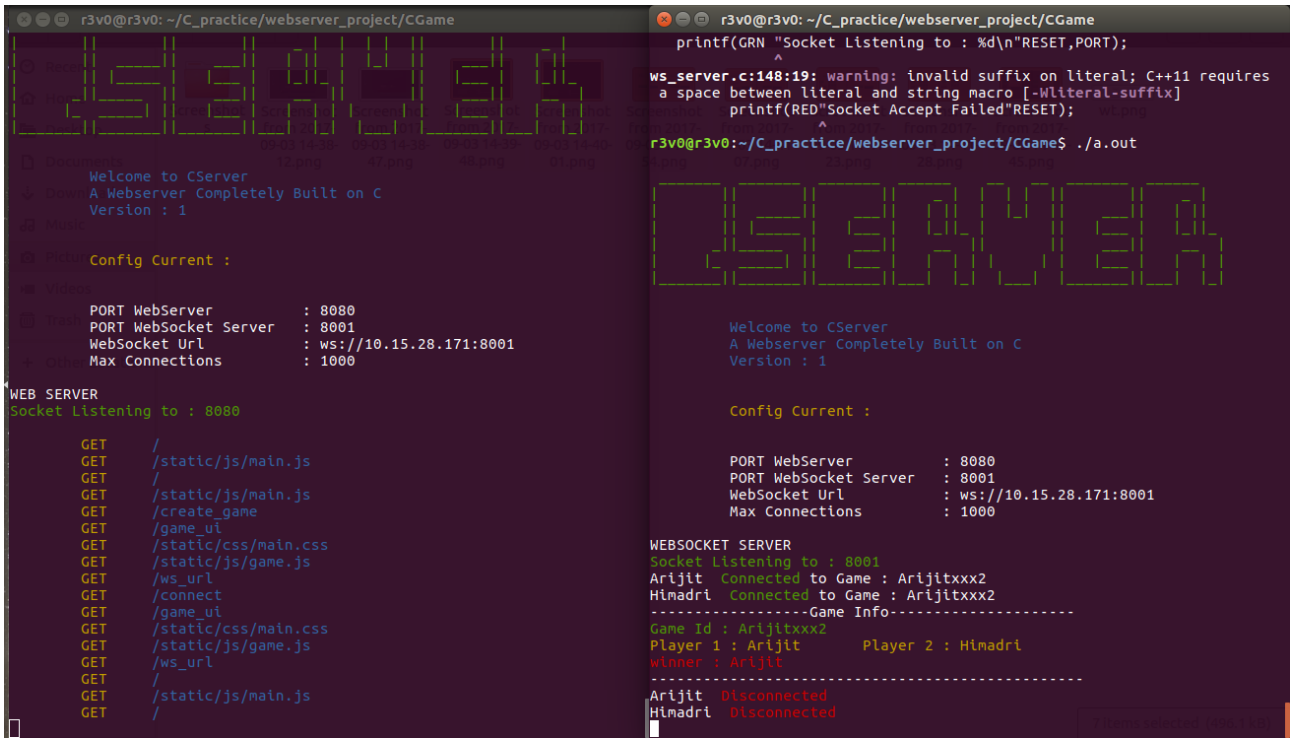
Player 1 Waiting while player to connecting to game with game id and username



Both Player Joins and Starts Playing the Game

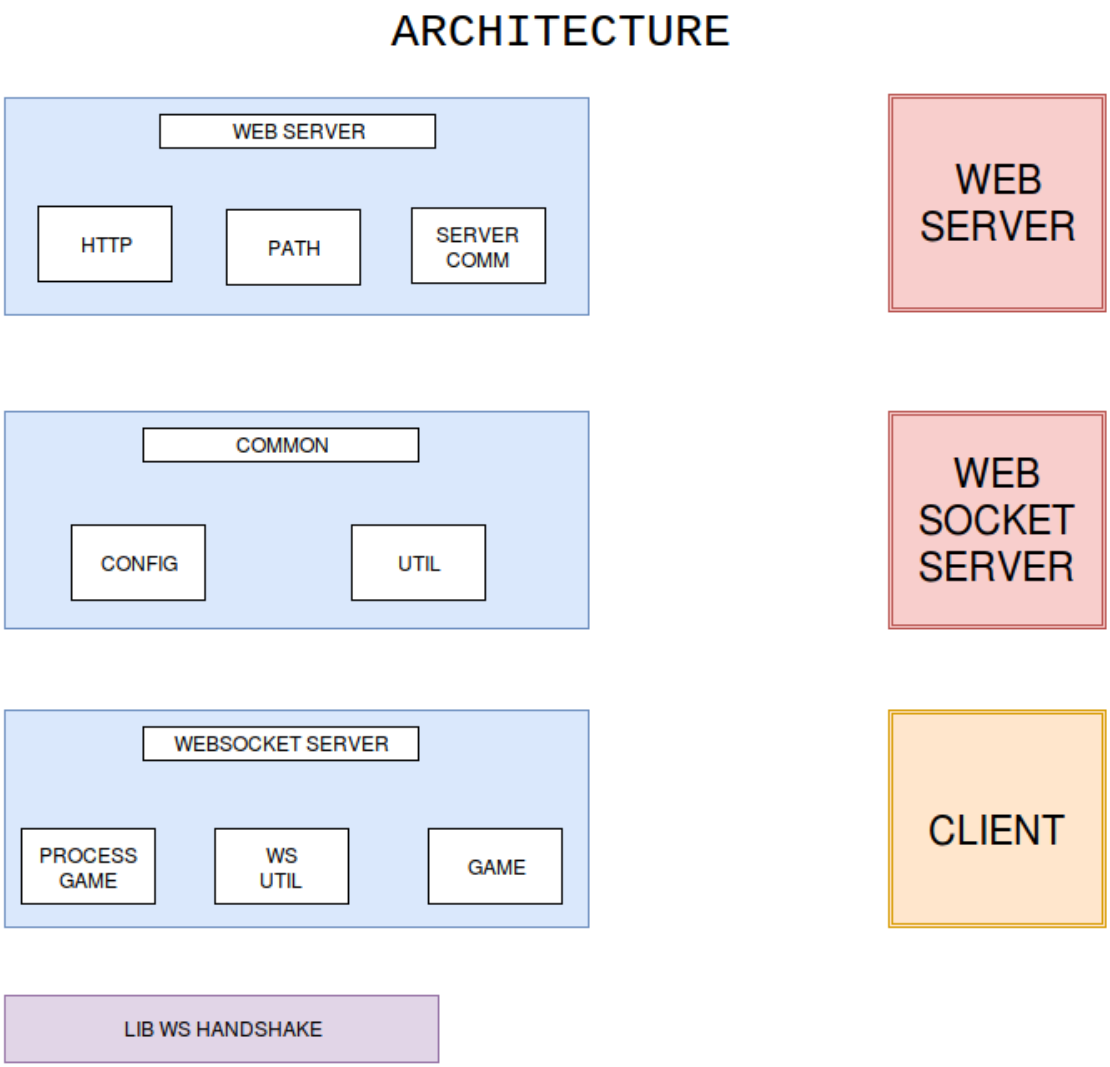


Game is ended when either one of them wins or out of move .

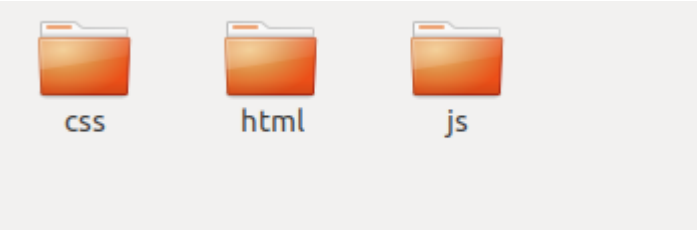


The Log of Game and HTTP requests Served displayed on the terminal

Architecture



The Static Folder Contains the HTML , CSS and Javascript Files which are served on request by the web server



How it Works
Webserver

The webserver creates a socket and binds it to a given port , now when from browser we send a http request the server returns either HTML file or JSON response , to efficiently handle a http request the server first reads the HTTP request from the client , then the HTTP module decodes the raw request and fetches the REQ_TYPE , REQ_URL , REQ_HEADER and REQ_DATA inside a HTTP object . When the request url is fetched the PATH module maps function for each url like any modern webserver .

```
//main path function
string process_req(string path,map<string,string> header,map<string,string> data){
  if(!path.compare("/")){
    return static_html("index.html");
  }
  if(!path.compare("/create_game")){
    return path_create_game(header, data);
  }
  if(path.find("/static/")!=string::npos){
    return static_file(path);
  }
  if(!path.compare("/connect")){
    return path_connect(header,data);
  }
  if(!path.compare("/game_ui")){
    return static_html("game_ui.html");
  }
  if(!path.compare("/ws_url")){
    return path_ws_url(header,data);
  }
  return static_html("404.html");
}
```

Http Requests are served independently of each other , each connection is treated as an individual connection , Sessions are implemented using cookies , but HTTP protocol is non persistant .

Now each function takes the header and data and processes the request and returns a string as response , which is then sent back the client , the reponse can be a HTML file or a JSON message .

HTTP module process the raw HTTP request and separates header and data , both header and Data are stored in a C++ STL map .

The server serves index.html when the endpoint is / .

The server serves files in the /static end point which includes html , css and Javscript files . If the endpoint is not given it serves the 404.html .

The /ws_url serves the url of the websocket server as a json object .

Now to create a Game the player 1 sends request with Player name to the /create_game end point and the server returns the game id to user as json encoded message .

Now to join a Game the player 2 sends request with Player name and Game Id to the endpoint /connect , if the game id exists the player is sent a status with success as a json encoded message.

Now upon receiving this /create_game request the Webserver sends a message to the Websocket server to create game for player 1 with a given game id and for /connect request the web server sends a request to the websocket server to add player 2 with given player name to game with given game id .

How it Works
Web Socket Server

The Web Socket Protocol allows the server and the client to have a persistent connection and binds an UNIX socket to a Javascript WebSocket , building a dedicated connection between the client and server , thus ensures a realtime communcation without burdening the server with polling from the client .

Intially the Websocket connection starts in a HTTP protocol , a Web Socket Secret Key is sent to the server and Server process the key , Initially the key is concatinated with a Magic Key , and SHA1 hash of the concatinated string is genrated , then the SHA1 hash is base64 enocoded and sent to the client . If the process is done properly a connection is established between the server and the client and the websocket handshake is complement , we used this to perform the websocket handshake .

After the Handshake is complement still the client sends data encoded in a frame and masked using specific masking key , and the server should also decode the data in such frames so that the client can read .

We implemented our own Websocket server and encode decode using this documentation .

Now when the websocket server received message to create a game from the webserver the websocket server creates a game object with given game id and given player 1 name , and puts it in a map with game id as key .

Now the object has lot of functions which allows to manipulate and update moves in the game . The game object saves the game state and implements additional functions for the game .

Module WS Util implements the encode and decode operations for Websocket frame .

LibWSHandhake implements the handshake for WebSocket .

Upon recieving instructions from the client the Process_game module updates the game object or the game state accordingly and sends both client the updated Game state .

When the Game ends either with an winner or a draw the end event is passed to the both connected clients and both of them are disconnected from the server .

In simple , the webserver serves the html , js and css contents and creates and adds player to an game , and the web socket server facilitates the real time game.