Báo cáo Checkpoint #2

Nhóm 13

Mô tả dự án

🎨 Scribble

Scribble là một trò chơi multiplayer real-time với lối chơi giống gameshow nhìn hình đoán chữ được xây dựng hoàn toàn bằng ngôn ngữ C cho phần backend (server + client proxy) kết hợp với Web UI hiện đại.

Lối chơi của Scribble

Người chơi tham gia phòng chơi

Hệ thống tự động thêm người chơi vào phòng chơi hợp lệ

Đợi đến khi game bắt đầu

Khi đã có đủ số người để có thể bắt đầu, hệ thống sẽ đếm ngược đến khi bắt đầu game

Vẽ

Mỗi vòng chơi sẽ có một người vẽ. Họ sẽ phải vẽ từ được hiển thị trên màn hình của họ vào một cái bảng ảo

Đoán

Những người còn lại sẽ phải đoán từ đang được vẽ là gì. Ai đoán được càng sớm sẽ càng có nhiều điểm.

Các usecase của hệ thống

Use Case: Join Game qua Auto Matchmaking			
Actor	Precondition	Main Flow	Postcondition
Người chơi	Server đang chạy, browser hỗ trợ WebSocket	1. Người chơi mở `http://localhost:8080` 2. Nhập username và click "Play Now" 3. Client gửi `MSG_REGISTER` → Server 4. Server trả về `MSG_REGISTER_ACK` với `player_id` và `session_token` 5. Client gửi `MSG_JOIN_ROOM` với room_id = 0 (auto matchmaking) 6. Server tìm room phù hợp hoặc tạo room mới 7. Server trả về `MSG_ROOM_JOINED` với thông tin room 8. Khi đủ 2+ người chơi, countdown 15 giây bắt đầu 9. Server broadcast `MSG_COUNTDOWN_UPDATE` mỗi giây 10. Sau 15 giây, game bắt đầu với `MSG_GAME_START`	Người chơi vào room và chờ game bắt đầu

Các usecase của hệ thống

Use Case: Drawing Phase (Pha vẽ)			
Actor	Precondition	Main Flow	Postcondition
Người chơi đang vẽ (Drawer)	Game đang chạy, đến lượt người chơi vẽ	1. Server gửi `MSG_YOUR_TURN` và `MSG_WORD_TO_DRAW` cho drawer 2. Drawer nhận được từ cần vẽ 3. UI hiển thị drawing tools (color palette, brush size) 4. Drawer vẽ trên canvas: - Mouse move → tạo stroke {x1, y1, x2, y2, color, thickness} - Client gửi `UDP_STROKE` qua WebSocket 5. Server nhận stroke và broadcast cho tất cả players khác 6. Other players nhận stroke và vẽ trên canvas của họ 7. Drawing đồng bộ real-time cho tất cả players (Drawer click "Clear Canvas" → Server broadcast `UDP_CLEAR_CANVAS`)	

Các usecase của hệ thống

Use Case: Guessing Phase (Pha đoán)			
Actor	Precondition	Main Flow	Postcondition
Người chơi đang đoán (Guesser)	Game đang chạy, người khác đang vẽ	1. Guesser nhìn canvas và nhập guess vào chat 2. Client gửi `MSG_CHAT` với message 3. Server so sánh guess với current_word (case-insensitive) 4. Nếu đúng: - Đánh dấu `player.has_guessed = true` - Tính điểm dựa trên thời gian còn lại: `points = 10 + (time_remaining * 90 / ROUND_TIME)` - Server broadcast `MSG_GUESS_CORRECT` với username và score - Client hiển thị notification màu xanh: "Player X guessed correctly!” 5. Nếu sai: - Server broadcast `MSG_CHAT_BROADCAST` để hiển thị chat bình thường. 6. Special Case:** - Tất cả players đã đoán đúng → End round sớm	

Các usecase của hệ thống

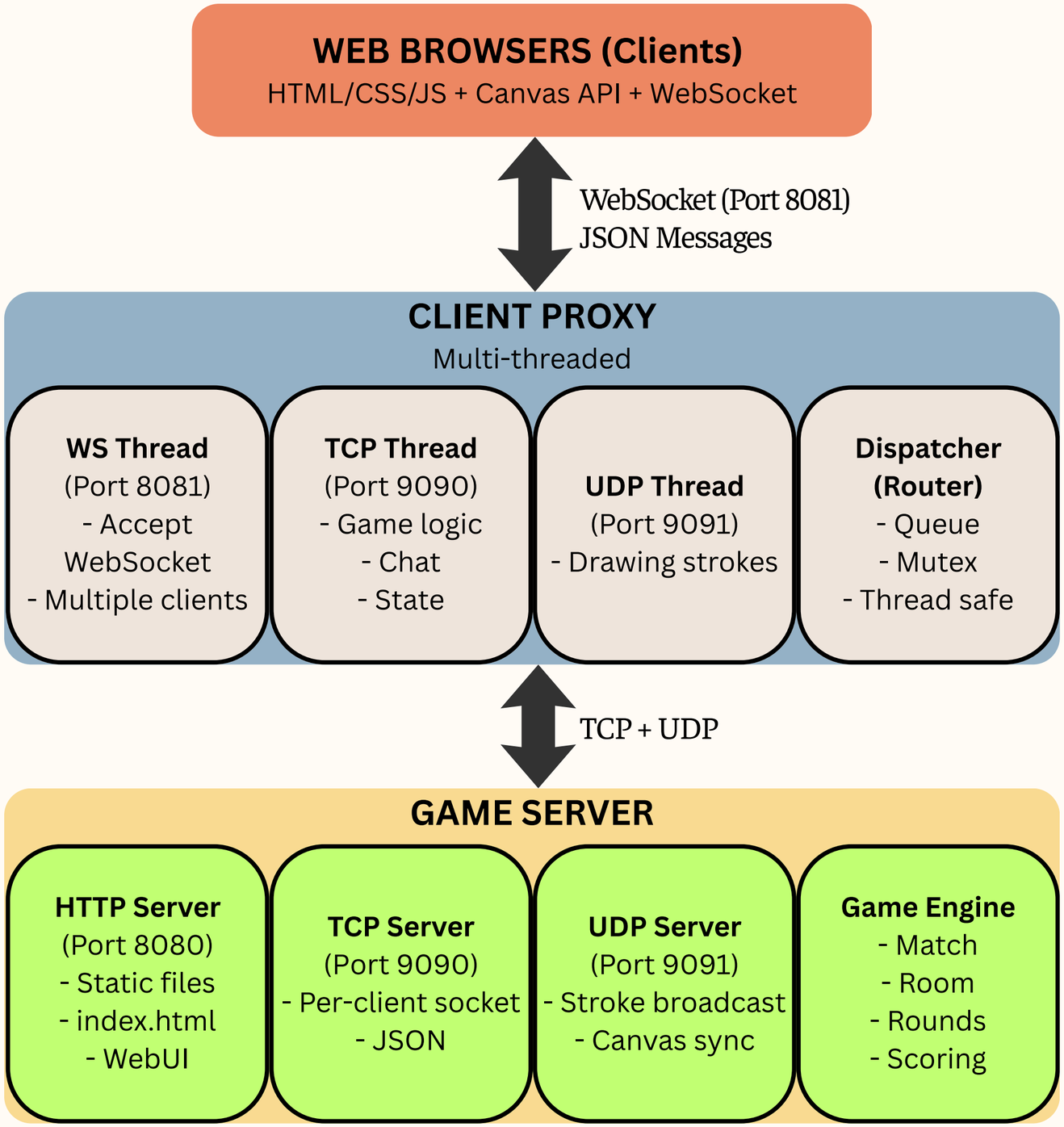
Use Case: Player Disconnect			
Actor	Precondition	Main Flow	Postcondition
Người chơi bị mất kết nối	Game đang chơi, kết nối bị gián đoạn, nhưng tab vẫn được mở	[Disconnect] 1. Client proxy phát hiện connection lost 2. Gửi `MSG_DISCONNECT` hoặc timeout 3. Server lưu player state với `session_token` 4. Server broadcast `MSG_PLAYER_LEAVE` cho players còn lại 5. Server giữ player state trong 5 phút 6. Game tiếp tục với players còn lại [Reconnect] 1. Player mở lại browser và connect 2. Client có `session_token` đã lưu trong localStorage 3. Gửi `MSG_RECONNECT_REQUEST` với token 4. Server validate token và restore player state 5. Server gửi `MSG_RECONNECT_SUCCESS` với full game state 6. Client restore UI: canvas, scores, timer, turn 7. Player tiếp tục game từ nơi đã dừng	Player trở lại game với state đúng

Các usecase của hệ thống

Use Case: Kết thúc game và xếp hạng			
Actor	Precondition	Main Flow	Postcondition

Tất cả người chơi	Tất cả rounds đã hoàn thành	1. Round cuối cùng kết thúc
		2. Server tính toán:
		- Tìm player có điểm cao nhất (winner)
		- Sắp xếp players theo score giảm dần
		3. Server broadcast `MSG_GAME_END` với json:
		{
		"players": [
		{"player_id": 1, "username": "Alice", "score": 850},
		{"player_id": 2, "username": "Bob", "score": 720},
		{"player_id": 3, "username": "Carol", "score": 650}
		]
		}
		4. Client hiển thị dialog "Game Ended" với rankings:
		- #1 Alice - 850 pts (Gold gradient)
		- #2 Bob - 720 pts (Silver gradient)
		- #3 Carol - 650 pts (Bronze gradient)
		5. Player click "Return to Home" → quay về landing page	

Cấu trúc của hệ thống



Cấu trúc tệp của hệ thống

```
server/
├── main.c                    # Entry point, khởi tạo các servers
│
├── http/                     # HTTP Server cho static files
│   ├── http_server.c/.h      # HTTP server chính (port 8080)
│   ├── router.c/.h           # URL routing
│   └── mime.c/.h             # MIME type detection
│
├── tcp/                      # TCP Server cho game logic
│   ├── tcp_server.c/.h       # TCP socket management
│   ├── tcp_handler.c/.h      # Message handlers
│   └── tcp_parser.c/.h       # JSON message parsing
│
├── udp/                      # UDP Server cho drawing
│   ├── udp_server.c/.h       # UDP socket management
│   └── udp_broadcast.c/.h    # Broadcast strokes
│
├── game/                     # Game Logic
│   ├── game_logic.c/.h       # Core game mechanics
│   ├── matchmaking.c/.h      # Room management
│   └── reconnection.c/.h     # Reconnection handling
│
└── utils/                    # Utilities
    ├── logger.c/.h           # JSON logging
    ├── timer.c/.h            # Timer thread
    ├── json.c/.h             # JSON utilities
    └── endian_compat.h       # Cross-platform byte order
```

Cấu trúc tệp của hệ thống

```
client_proxy/
├── main.c     # Entry point, spawn 4 threads
│
├── threads/   # Worker Threads
│   ├── ws_thread.c/.h # WebSocket listener thread
│   ├── tcp_thread.c/.h # TCP connection thread
│   ├── udp_thread.c/.h # UDP handler thread
│   └── dispatcher.c/.h # Message dispatcher thread
│
└── utils/ # Thread-safe utilities
    ├── queue.c/.h # Thread-safe message queue
    ├── state_cache.c/.h # Connection state management
    └── json.c/.h # JSON parsing
```

Cấu trúc tệp của hệ thống

```
webui/
├── index.html          # Main HTML structure
├── style.css           # Responsive styling
├── websocket.js        # WebSocket connection class
├── drawing.js          # Canvas drawing class
└── main.js             # Game controller class
```

Các thành phần chính (server)

HTTP Server (http/http_server.c)

Vai trò: Serve các static files (HTML, CSS, JS) cho browser clients

Chức năng chính:
- Bind socket tại port 8080
- Listen và accept HTTP connections
- Parse HTTP requests (GET method)
- Serve files từ `build/webui/` directory
- Detect MIME types (text/html, text/css, application/javascript)
- Return HTTP responses với proper headers

```
int start_http_server(int port);                // Khởi tạo
HTTP server
void* http_server_thread(void* arg);            // Thread
handler
void handle_http_request(int client_fd);        // Xử lý
request
```

Các thành phần chính (server)

TCP Server (tcp/tcp_server.c)

Vai trò: Quản lý game connections và per-client TCP sockets

Chức năng chính:
- Bind socket tại port 9090
- Maintain mảng players với TCP socket cho mỗi client
- Accept new connections và tạo Player struct
- Use `select()` để handle multiple clients
- Parse messages với 4-byte length prefix
- Route messages đến appropriate handlers

```c
typedef struct {
    int fd;                     // TCP socket file descriptor
    char username[32];
    uint32_t player_id;
    int score;
    PlayerState state;
    bool is_drawing;
    bool has_guessed;
    bool has_drawn;             // Track if had turn
    char session_token[64];     // For reconnection
    char recv_buffer[4096];     // Partial message buffer
    int recv_buffer_len;
} Player;
```

Các thành phần chính (server)

TCP Server (tcp/tcp_server.c)

Vai trò: Quản lý game connections và per-client TCP sockets

Chức năng chính:
- Bind socket tại port 9090
- Maintain mảng players với TCP socket cho mỗi client
- Accept new connections và tạo Player struct
- Use `select()` để handle multiple clients
- Parse messages với 4-byte length prefix
- Route messages đến appropriate handlers

```
void* tcp_server_thread(void* arg);              // Main TCP
thread
void handle_tcp_message(Player* player, ...); // Message
router
void send_tcp_message(int fd, ...);              // Send with
length prefix
void broadcast_to_room(Room* room, ...);         // Broadcast
to all players
```

Các thành phần chính (server)

UDP Server (udp/udp_server.c)

Vai trò: Low-latency transmission cho drawing strokes

Note: Hiện tại UDP không được sử dụng trực tiếp từ browser (browsers không hỗ trợ UDP). Strokes được gửi qua WebSocket → TCP thay thế. Code UDP vẫn giữ cho tương lai hoặc native clients.

Chức năng:
- Bind socket tại port 9091
- Receive stroke data
- Broadcast đến room members
- Binary protocol cho performance

Các thành phần chính (server)

Game Logic (game/game_logic.c)

Vai trò: Core game mechanics và room management

```c
typedef struct {
    uint32_t room_id;
    char room_code[16];              // 6-char code for
    private rooms
    Player* players[MAX_PLAYERS];    // Array of player
    pointers
    int player_count;
    RoomState state;                 // WAITING, PLAYING,
    ENDED
    int current_drawer_idx;
    char current_word[MAX_WORD_LEN];
    int round_number;
    int total_rounds;                // Dynamic: equals
    player_count
    uint64_t round_start_time;
    int time_remaining;
    Stroke strokes[MAX_STROKES];
    int stroke_count;
    bool is_private;
    uint64_t game_start_countdown;
    bool countdown_active;
} Room;
```

Các thành phần chính (server)

Game Logic (game/game_logic.c)

Vai trò: Core game mechanics và room management

```c
void start_game(Room* room);
  // - Set total_rounds = player_count
  // - Reset scores, has_drawn flags
  // - Start first round

void start_next_round(Room* room);
  // - Increment round_number
  // - Find next player who hasn't drawn
  // - Skip disconnected players
  // - Select random word
  // - Reset round timer
  // - Broadcast MSG_ROUND_START

void end_round(Room* room);
  // - Broadcast MSG_ROUND_END
  // - Call start_next_round()
  // - Or call end_game() if all done

void end_game(Room* room);
  // - Find winner
  // - Broadcast MSG_GAME_END with rankings
  // - Set state to ROOM_ENDED
```

Các thành phần chính (server)

Game Logic (game/game_logic.c)

Vai trò: Core game mechanics và room management

```c
int process_guess(Room* room, Player* player, const char*
guess);
  // - Case-insensitive comparison
  // - Calculate score based on time
  // - Mark player.has_guessed = true
  // - Broadcast MSG_GUESS_CORRECT
  // - Check if all guessed → end round early

void update_timer(Room* room);
  // - Calculate elapsed time
  // - Update time_remaining
  // - Broadcast MSG_TIMER_UPDATE
  // - If time_remaining <= 0 → end_round()

void check_game_start_countdown(Room* room);
  // - Check if countdown active
  // - Broadcast MSG_COUNTDOWN_UPDATE every second
  // - After 15 seconds → start_game()

int add_player_to_room(Room* room, Player* player);
int remove_player_from_room(Room* room, Player* player);
  // - Adjust drawer_idx if current drawer leaves
  // - Recalculate total_rounds
  // - End round if drawer disconnects
```

Các thành phần chính (server)

Matchmaking (game/matchmaking.c)

Vai trò: Auto matchmaking và room management

Chức năng:
- Maintain array of 100 rooms
- Auto matchmaking: tìm room có < MAX_PLAYERS
- Private room creation với 6-character code
- Join by room code
- Activate countdown khi 2nd player joins
- Prevent joining after round 1
- Iterate active rooms cho timer updates

```
int join_matchmaking(Player* player);
  // - Find best available room
  // - Add player to room
  // - If player_count == 2 → activate countdown

int create_private_room(Player* player);
  // - Generate unique 6-char code
  // - Initialize room
  // - Add creator

int join_private_room(Player* player, const char* code);
  // - Find room by code
  // - Check if joinable (round <= 1)
  // - Add player

void iterate_active_rooms(void (*callback)(Room*));
  // - Loop through ROOM_WAITING and ROOM_PLAYING
  // - Call callback for each active room
```

Các thành phần chính (server)

Timer Thread (utils/timer.c)

Vai trò: 1-second tick timer cho game updates

Chức năng:
- Run trong separate thread
- Mỗi giây gọi callback:
 + `check_game_start_countdown()` cho countdown
 + `update_timer()` cho round timer
 + Broadcast `MSG_TIMER_UPDATE` và `MSG_COUNTDOWN_UPDATE`
- Thread-safe với mutex

Implementation

```
void* timer_thread(void* arg) {
    while (timer_running) {
        sleep(1);

        iterate_active_rooms([](Room* room) {
            check_game_start_countdown(room);
            update_timer(room);
        });
    }
}
```

Các thành phần chính (client proxy)

TCP Thread (threads/tcp_thread.c)

Vai trò: Maintain persistent TCP connection đến game server

Chức năng:
- Connect to server port 9090
- Send messages với 4-byte length prefix
- Receive and parse server messages
- Enqueue responses đến Dispatcher
- Auto-reconnect nếu connection lost

Các thành phần chính (client proxy)

Dispatcher Thread (`threads/dispatcher.c`)

Vai trò: Thread-safe message routing hub

Chức năng:
- Maintain 2 thread-safe queues:
 + `to_server_queue`: Messages từ WS → Server
 + `to_client_queue`: Messages từ Server → WS
- Use mutex và condition variables
- Route messages dựa trên type và player_id
- Ensure message ordering

```
typedef struct {
    MessageType type;
    uint32_t player_id;     // Target client
    char* json_data;
    size_t data_len;
} QueuedMessage;
```

TCP Message Type

Định danh cho các loại tin nhắn được trao đổi thông qua giao thức TCP giữa Client Proxy và Server

```
typedef enum {
    MSG_PING = 0,
    MSG_PONG = 1,
    MSG_REGISTER = 2,
    MSG_REGISTER_ACK = 3,
    MSG_JOIN_ROOM = 4,
    MSG_CREATE_ROOM = 5,
    MSG_ROOM_CREATED = 6,
    MSG_ROOM_JOINED = 7,
    MSG_ROOM_FULL = 8,
    MSG_ROOM_NOT_FOUND = 9,
    MSG_GAME_START = 10,
    MSG_YOUR_TURN = 11,
    MSG_WORD_TO_DRAW = 12,
    MSG_ROUND_START = 13,
    MSG_CHAT = 14,
    MSG_CHAT_BROADCAST = 15,
    MSG_GUESS_CORRECT = 16,
    MSG_GUESS_WRONG = 17,
    MSG_TIMER_UPDATE = 18,
    MSG_COUNTDOWN_UPDATE = 19,
    MSG_ROUND_END = 20,
    MSG_GAME_END = 21,
    MSG_PLAYER_JOIN = 22,
    MSG_PLAYER_LEAVE = 23,
    MSG_SCORE_UPDATE = 24,
    MSG_RECONNECT_REQUEST = 25,
    MSG_RECONNECT_SUCCESS = 26,
    MSG_RECONNECT_FAIL = 27,
    MSG_ERROR = 28,
    MSG_DISCONNECT = 29
} MessageType;
```

Cấu trúc
TCP Message

Cấu trúc các tin nhắn được trao đổi thông qua giao thức TCP giữa Client Proxy và Server theo định dạng JSON

Ví dụ:

[0x00, 0x00, 0x00, 0x3A] {"type":2,"data":{"username":"Alice"}}

Checklist

Các chức năng, công việc

Đã hoàn thành

- Xây dựng luồng cho WebSocket
- Xây dựng luồng TCP
- Xây dựng HTTP Server
- Cơ chế broadcast nét vẽ
- Xây dựng engine xử lý game logic
- Xây dựng Dispatcher Router
- Cấu trúc tin nhắn dạng JSON

Checklist

Các chức năng, công việc

Đã hoàn thành

- Cơ chế định danh phòng chơi, người chơi
- Xử lý thông báo, tin nhắn
- WebUI
- Server logging

Checklist

Các chức năng, công việc

Chưa hoàn thành

- Chức năng tìm phòng ‘cân bằng’
- Chức năng phòng chơi private
- Chức năng khôi phục trạng thái chơi khi ngắt kết nối
- Chức năng khôi phục trạng thái chơi khi ngắt kết nối

Demo