1. Backend Setup (Node.js with WebSockets)

Overview of Key Components:

1. **Player Authentication**: Manage user signups and logins using JSON Web Tokens (JWT).
2. **Game Matchmaking**: Match players together based on criteria (like rank or game mode).
3. **Real-Time Game Updates**: Use WebSockets to update players on the game state in real-time.
4. **Player Stats Tracking**: Store player statistics (such as wins, losses, scores) in a database.

Step 1: Initial Setup with Node.js and Expres:

1.Install Dependencies:

```
npm init -y
npm install express socket.io jsonwebtoken bcryptjs
mongoose
```

2.Basic Server Setup (server.js):

```
const express = require('express');const http =
require('http');const socketIo = require('socket.io');const jwt =
require('jsonwebtoken');const bcrypt = require('bcryptjs');const
mongoose = require('mongoose');
const app = express();const server =
http.createServer(app);const io = socketIo(server);
const PORT = process.env.PORT || 3000;

app.use(express.json());
// Simple user model for authenticationconst UserSchema = new
mongoose.Schema({
   username: String,
   password: String,
```

```
    wins: { type: Number, default: 0 },
    losses: { type: Number, default: 0 },
});
const User = mongoose.model('User', UserSchema);
// MongoDB connection
mongoose.connect('mongodb://localhost:27017/multiplayer-
game', {
    useNewUrlParser: true,
    useUnifiedTopology: true
}).then(() => console.log('Connected to MongoDB'));
// Serve the frontend
app.get('/', (req, res) => {
    res.sendFile(__dirname + '/index.html');
});
// Player registration
app.post('/register', async (req, res) => {
    const { username, password } = req.body;
    const hashedPassword = await bcrypt.hash(password, 10);
    const newUser = new User({ username, password:
hashedPassword });
    await newUser.save();
    res.json({ message: 'User registered successfully' });
});
// Player login
app.post('/login', async (req, res) => {
    const { username, password } = req.body;
    const user = await User.findOne({ username });
    if (!user || !(await bcrypt.compare(password, user.password)))
{
        return res.status(400).json({ message: 'Invalid
credentials' });
    }
    const token = jwt.sign({ id: user._id }, 'secretkey', { expiresIn:
'1h' });
    res.json({ token });
});
```

```
// Middleware to verify JWT tokenconst verifyToken = (req, res, next) => {
    const token = req.headers['authorization'];
    if (!token) return res.status(403).json({ message: 'No token provided' });

    jwt.verify(token, 'secretkey', (err, decoded) => {
        if (err) return res.status(500).json({ message: 'Failed to authenticate token' });
        req.userId = decoded.id;
        next();
    });
};

server.listen(PORT, () => console.log(`Server is running on port ${PORT}`));
```

Step 2: Real-Time Communication with WebSockets

The WebSocket server will handle real-time communication between players, including the matchmaking process and sending game updates.

1.WebSocket Integration in Server (server.js):

```
// In-memory storage for waiting playersconst waitingPlayers = [];
// Handle WebSocket connections
io.on('connection', (socket) => {
    console.log('A player connected:', socket.id);

    // Handle matchmaking request
    socket.on('join_game', async (token) => {
        // Authenticate player using JWT token
        try {
            const decoded = jwt.verify(token, 'secretkey');
            const player = await User.findById(decoded.id);
```

```
            if (!player) return socket.emit('error', 'User not found');

            waitingPlayers.push({ socket, player });
            socket.emit('waiting', 'Waiting for another player...');

            if (waitingPlayers.length >= 2) {
                const player1 = waitingPlayers.shift();
                const player2 = waitingPlayers.shift();

                // Notify players that they are matched
                player1.socket.emit('match_found', { opponent:
player2.player.username });
                player2.socket.emit('match_found', { opponent:
player1.player.username });

                // Initiate game logic here (send initial game state, etc.)
                startGame(player1, player2);
            }
        } catch (error) {
            socket.emit('error', 'Authentication failed');
        }
    });

    // Handle disconnection
    socket.on('disconnect', () => {
        console.log('Player disconnected:', socket.id);
    });
});
function startGame(player1, player2) {
    const gameState = {
        player1: { id: player1.socket.id, username:
player1.player.username, score: 0 },
        player2: { id: player2.socket.id, username:
player2.player.username, score: 0 },
    };

    // Send initial game state to both players
```

```
    player1.socket.emit('game_start', gameState);
    player2.socket.emit('game_start', gameState);

    // Real-time game logic: listen for moves from both players
    player1.socket.on('make_move', (move) => {
        gameState.player1.score += move;
        io.to(gameState.player2.id).emit('update_game',
gameState);
    });

    player2.socket.on('make_move', (move) => {
        gameState.player2.score += move;
        io.to(gameState.player1.id).emit('update_game',
gameState);
    });
}
```

Step 3: Frontend Setup

The frontend will handle game rendering, sending moves, and receiving real-time updates from the server.

1. **Basic Frontend (index.html)**:

```
<!DOCTYPE html><html lang="en"><head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Multiplayer Game</title>
    <style>
        body { font-family: Arial, sans-serif; text-align: center; }
        #game { display: none; }
    </style></head><body>
    <h1>Multiplayer Online Game</h1>
    <div id="auth">
        <h2>Login</h2>
        <input id="username" placeholder="Username"><br><br>
```

```
        <input id="password" type="password"
placeholder="Password"><br><br>
        <button onclick="login()">Login</button>
    </div>

    <div id="game">
        <h2>Game</h2>
        <p>Opponent: <span id="opponent"></span></p>
        <p>Your Score: <span id="your-score">0</span></p>
        <p>Opponent's Score: <span id="opponent-
score">0</span></p>
        <button onclick="makeMove()">Make Move</button>
    </div>

    <script src="/socket.io/socket.io.js"></script>
    <script>
        const socket = io();
        let token;

        function login() {
            const username =
document.getElementById('username').value;
            const password =
document.getElementById('password').value;

            fetch('/login', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ username, password })
            }).then(res => res.json())
              .then(data => {
                  if (data.token) {
                      token = data.token;
                      joinGame();
                  }
              });
        }
```

```
        function joinGame() {
            socket.emit('join_game', token);
        }

        socket.on('waiting', message => {
            alert(message);
        });

        socket.on('match_found', data => {
            document.getElementById('auth').style.display = 'none';
            document.getElementById('game').style.display = 'block';
            document.getElementById('opponent').innerText =
data.opponent;
        });

        socket.on('game_start', gameState => {
            document.getElementById('your-score').innerText =
gameState.player1.score;
            document.getElementById('opponent-score').innerText =
gameState.player2.score;
        });

        socket.on('update_game', gameState => {
            document.getElementById('your-score').innerText =
gameState.player1.score;
            document.getElementById('opponent-score').innerText =
gameState.player2.score;
        });

        function makeMove() {
            const move = Math.floor(Math.random() * 10);
            socket.emit('make_move', move);
        }
    </script></body></html>
```

1. Backend Setup (Node.js with WebSockets)

Overview of Key Components:

1. **Player Authentication**: Manage user signups and logins using JSON Web Tokens (JWT).
2. **Game Matchmaking**: Match players together based on criteria (like rank or game mode).
3. **Real-Time Game Updates**: Use WebSockets to update players on the game state in real-time.
4. **Player Stats Tracking**: Store player statistics (such as wins, losses, scores) in a database.

Step 1: Initial Setup with Node.js and Express:

1.Install Dependencies:

```
npm init -y
npm install express socket.io jsonwebtoken bcryptjs
mongoose
```

2.Basic Server Setup (server.js):

```
const express = require('express');const http =
require('http');const socketIo = require('socket.io');const jwt =
require('jsonwebtoken');const bcrypt = require('bcryptjs');const
mongoose = require('mongoose');
const app = express();const server =
http.createServer(app);const io = socketIo(server);
const PORT = process.env.PORT || 3000;

app.use(express.json());
// Simple user model for authenticationconst UserSchema = new
mongoose.Schema({
    username: String,
    password: String,
    wins: { type: Number, default: 0 },
```

```
    losses: { type: Number, default: 0 },
});
const User = mongoose.model('User', UserSchema);
// MongoDB connection
mongoose.connect('mongodb://localhost:27017/multiplayer-
game', {
    useNewUrlParser: true,
    useUnifiedTopology: true
}).then(() => console.log('Connected to MongoDB'));
// Serve the frontend
app.get('/', (req, res) => {
    res.sendFile(__dirname + '/index.html');
});
// Player registration
app.post('/register', async (req, res) => {
    const { username, password } = req.body;
    const hashedPassword = await bcrypt.hash(password, 10);
    const newUser = new User({ username, password:
hashedPassword });
    await newUser.save();
    res.json({ message: 'User registered successfully' });
});
// Player login
app.post('/login', async (req, res) => {
    const { username, password } = req.body;
    const user = await User.findOne({ username });
    if (!user || !(await bcrypt.compare(password, user.password)))
{
        return res.status(400).json({ message: 'Invalid
credentials' });
    }
    const token = jwt.sign({ id: user._id }, 'secretkey', { expiresIn:
'1h' });
    res.json({ token });
});
// Middleware to verify JWT tokenconst verifyToken = (req, res,
next) => {
```

```
    const token = req.headers['authorization'];
    if (!token) return res.status(403).json({ message: 'No token
provided' });

    jwt.verify(token, 'secretkey', (err, decoded) => {
        if (err) return res.status(500).json({ message: 'Failed to
authenticate token' });
        req.userId = decoded.id;
        next();
    });
};

server.listen(PORT, () => console.log(`Server is running on port
${PORT}`));
```

Step 2: Real-Time Communication with WebSockets

The WebSocket server will handle real-time communication between players, including the matchmaking process and sending game updates.

1.WebSocket Integration in Server (server.js):

```
// In-memory storage for waiting playersconst waitingPlayers =
[];
// Handle WebSocket connections
io.on('connection', (socket) => {
    console.log('A player connected:', socket.id);

    // Handle matchmaking request
    socket.on('join_game', async (token) => {
        // Authenticate player using JWT token
        try {
            const decoded = jwt.verify(token, 'secretkey');
            const player = await User.findById(decoded.id);
            if (!player) return socket.emit('error', 'User not found');
```

```
            waitingPlayers.push({ socket, player });
            socket.emit('waiting', 'Waiting for another player...');

            if (waitingPlayers.length >= 2) {
                const player1 = waitingPlayers.shift();
                const player2 = waitingPlayers.shift();

                // Notify players that they are matched
                player1.socket.emit('match_found', { opponent:
player2.player.username });
                player2.socket.emit('match_found', { opponent:
player1.player.username });

                // Initiate game logic here (send initial game state, etc.)
                startGame(player1, player2);
            }
        } catch (error) {
            socket.emit('error', 'Authentication failed');
        }
    });

    // Handle disconnection
    socket.on('disconnect', () => {
        console.log('Player disconnected:', socket.id);
    });
});
function startGame(player1, player2) {
    const gameState = {
        player1: { id: player1.socket.id, username:
player1.player.username, score: 0 },
        player2: { id: player2.socket.id, username:
player2.player.username, score: 0 },
    };

    // Send initial game state to both players
    player1.socket.emit('game_start', gameState);
    player2.socket.emit('game_start', gameState);
```

```
    // Real-time game logic: listen for moves from both players
    player1.socket.on('make_move', (move) => {
        gameState.player1.score += move;
        io.to(gameState.player2.id).emit('update_game',
gameState);
    });

    player2.socket.on('make_move', (move) => {
        gameState.player2.score += move;
        io.to(gameState.player1.id).emit('update_game',
gameState);
    });
}
```

Step 3: Frontend Setup

The frontend will handle game rendering, sending moves, and receiving real-time updates from the server.

1.Basic Frontend (index.html):

```
<!DOCTYPE html><html lang="en"><head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Multiplayer Game</title>
    <style>
        body { font-family: Arial, sans-serif; text-align: center; }
        #game { display: none; }
    </style></head><body>
    <h1>Multiplayer Online Game</h1>
    <div id="auth">
        <h2>Login</h2>
        <input id="username" placeholder="Username"><br><br>
        <input id="password" type="password"
placeholder="Password"><br><br>
```

```
        <button onclick="login()">Login</button>
    </div>

    <div id="game">
        <h2>Game</h2>
        <p>Opponent: <span id="opponent"></span></p>
        <p>Your Score: <span id="your-score">0</span></p>
        <p>Opponent's Score: <span id="opponent-
score">0</span></p>
        <button onclick="makeMove()">Make Move</button>
    </div>

    <script src="/socket.io/socket.io.js"></script>
    <script>
        const socket = io();
        let token;

        function login() {
            const username =
document.getElementById('username').value;
            const password =
document.getElementById('password').value;

            fetch('/login', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ username, password })
            }).then(res => res.json())
              .then(data => {
                  if (data.token) {
                      token = data.token;
                      joinGame();
                  }
              });
        }

        function joinGame() {
```

```
            socket.emit('join_game', token);
        }

        socket.on('waiting', message => {
            alert(message);
        });

        socket.on('match_found', data => {
            document.getElementById('auth').style.display = 'none';
            document.getElementById('game').style.display = 'block';
            document.getElementById('opponent').innerText =
data.opponent;
        });

        socket.on('game_start', gameState => {
            document.getElementById('your-score').innerText =
gameState.player1.score;
            document.getElementById('opponent-score').innerText =
gameState.player2.score;
        });

        socket.on('update_game', gameState => {
            document.getElementById('your-score').innerText =
gameState.player1.score;
            document.getElementById('opponent-score').innerText =
gameState.player2.score;
        });

        function makeMove() {
            const move = Math.floor(Math.random() * 10);
            socket.emit('make_move', move);
        }
    </script></body></html>
```