CodeAlpha_ Multiplayer Online Game Server

Here is a comprehensive guide and sample code to develop a **Multiplayer Online Game Server** with the required features:

Features

1. **Player Authentication**
2. Allow players to log in and register using email and password.
3. **Game Matchmaking**

Match players with others based on their skill level or preferences.

4. **Real-time Game Updates**

Use WebSockets to facilitate live game interactions and updates.

5. **Player Stats Tracking**

Store and retrieve player stats such as scores, levels, and achievements.

Tech Stack

- **Frontend:** HTML, CSS, JavaScript
- **Backend:** Node.js with Express or Django
- **Real-time Communication:** WebSocket (via `Socket.IO` for Node.js or Django Channels for Django)

- **Database:** MongoDB (NoSQL) or PostgreSQL (SQL)

Step-by-Step Guide

1. Backend Setup (Node.js with Express)

Install necessary dependencies:

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

Directory Structure:

```
- backend/
  - server.js
  - models/
    - User.js
  - routes/
    - auth.js
  - controllers/
    - game.js
  - config/
    - db.js
```

server.js (Main server file):

```
const express = require('express');
const http = require('http');
const { Server } = require('socket.io');
const mongoose = require('./config/db');
const cors = require('cors');

const app = express();
const server = http.createServer(app);
const io = new Server(server, {
  cors: {
    origin: "*",
    methods: ["GET", "POST"]
  }
});

const authRoutes = require('./routes/auth');

app.use(cors());
app.use(express.json());
app.use('/api/auth', authRoutes);

let players = [];
io.on('connection', (socket) => {
  console.log('Player connected:', socket.id);
  socket.on('join_game', (data) => {
    players.push({ id: socket.id, ...data });
    io.emit('player_list', players);
  });

  socket.on('disconnect', () => {
    players = players.filter(player => player.id !==
socket.id);
    io.emit('player_list', players);
    console.log('Player disconnected:', socket.id);
  });
});

server.listen(3000, () => {
  console.log('Server is running on
http://localhost:3000');
});
```

config/db.js (Database connection):

```
const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost:27017/game', {
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(() => console.log('Database connected'))
  .catch(err => console.log('Database connection
failed', err));

module.exports = mongoose;
```

models/User.js (User Schema):

```
const mongoose = require('mongoose');
const bcrypt = require('bcrypt');

const UserSchema = new mongoose.Schema({
  username: { type: String, required: true, unique:
true },
  email: { type: String, required: true, unique:
true },
  password: { type: String, required: true }
});

UserSchema.pre('save', async function(next) {
  if (!this.isModified('password')) return next();
  this.password = await bcrypt.hash(this.password,
10);
  next();
});

module.exports = mongoose.model('User', UserSchema);
```

routes/auth.js (Authentication routes):

```
const express = require('express');
const jwt = require('jsonwebtoken');
const User = require('../models/User');

const router = express.Router();
const SECRET_KEY = 'mysecretkey';

router.post('/register', async (req, res) => {
  try {
    const user = new User(req.body);
    await user.save();
    res.status(201).send('User registered');
  } catch (err) {
    res.status(400).send(err.message);
  }
});

router.post('/login', async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ email });
    if (!user) return res.status(404).send('User not
found');

    const isValid = await bcrypt.compare(password,
user.password);
    if (!isValid) return
res.status(401).send('Invalid credentials');

    const token = jwt.sign({ id: user._id },
SECRET_KEY);
    res.json({ token });
  } catch (err) {
    res.status(500).send(err.message);
  }
});

module.exports = router;
```

2. Frontend Setup

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; }
    #players { margin-top: 20px; }
  </style>
</head>
<body>
  <h1>Multiplayer Online Game</h1>
  <div id="game">
    <button id="join">Join Game</button>
    <div id="players"></div>
  </div>
  <script
src="https://cdn.socket.io/4.0.0/socket.io.min.js"></
script>
  <script>
    const socket = io('http://localhost:3000');

document.getElementById('join').addEventListener('cli
ck', () => {
      socket.emit('join_game', { username:
'Player1' });
    });
    socket.on('player_list', (players) => {
      const playerDiv =
document.getElementById('players');
      playerDiv.innerHTML = '<h3>Players:</h3>';
      players.forEach(player => {
        playerDiv.innerHTML +=
`<p>${player.username}</p>`;
      });
    });
  </script>
</body>
</html>
```

3. Testing

1. Start the backend server: node `server.js`
2. Open `index.html` in your browser.
3. Test joining the game and real-time updates.

Next Steps

- Add **matchmaking logic** based on player skill levels.
- Implement **secure JWT-based authentication** for game sessions.
- Store and retrieve **player stats** in the database.
- Deploy the application using cloud services (e.g., AWS, Heroku, or DigitalOcean).