CPS630 – Web Applications

Lab 3

Setting up the lab

Step 1: Install Node.js and npm

Before starting, ensure you have Node.js installed on your computer.

1. Visit https://nodejs.org to download and install Node.js. Node.js version 14.x or newer is required.
2. Download the latest version for your operating system (Windows, macOS, or Linux).
3. Follow the installation instructions provided on the website.
4. Node.js comes with npm (Node Package Manager), which you'll use to install Create React App and manage dependencies.

Step 2: Install Visual Studio Code

1. Go to the Visual Studio Code website.
2. Download the latest version for your operating system (Windows, macOS, or Linux).
3. Follow the installation instructions provided on the website.
4. After installation, open Visual Studio Code to ensure it is installed correctly.

Step 3: Create a New Project

Create a new directory for this lab and within it, place the Battleship game code from Lab01.

Note: If you are using the provided sample code, extract the zip file into this directory.

Step 4: Open Project in VS Code

Navigate to your project directory using:
cd lab3

Then, open the project in Visual Studio Code by running:
code .

This opens VS Code with your project folder.

Step 4.1: If you are using your own code

- Initialize Node.js Project: In your project directory, open a terminal and run npm init to create a package.json file. Accept the default configurations or modify them as needed.
- Install WebSocket Library: Run npm install ws to install the WebSocket library needed to create your server.
- Create WebSocket Server: Create a new js file for the server and use the following basic structure to get started:

```
const express = require('express');
const http = require('http');
const WebSocket = require('ws');

const app = express();
const bin = http.createServer(app);
const wss = new WebSocket.Server({ server: bin });

wss.on('connection', function connection(ws) {
   console.log('A new client connected');
   ws.on('message', message => {
       console.log('Received message:', message);
       ws.send('You sent -> ' + message);
   });
});

app.use(express.static('public'));

const port = 3000;
bin.listen(port, () => {
   console.log(`Server is running on
http://localhost:${port}`);
});
```

Step 4.2: If you are using the provided code

- Extract the folder and do a npm install in the root directory of your project (where you see the package.json file).

Description

Overview

In this lab, you will evolve the Battleship game you developed in Lab 01 into a multiplayer version. This involves replacing the computer opponent with another human player and implementing a WebSocket Node.js server to manage communication between the two players. You have the option to use your own code from Lab 01 or a provided sample code. The main objective is to open the game up to two players playing from separate browsers.

Objectives

- Modify the single-player Battleship game to support two human players.
- Implement a WebSocket server using Node.js to handle real-time communication between players.
- Demonstrate the ability to manage game state either on the client or the server.

Required Tools and Technologies

- Node.js
- WebSocket
- 2 Different browsers (e.g., Firefox and Chrome)
- Text Editor or IDE (e.g., VSCode, Sublime Text)

1: Implement Connection Logic

Modify the server file to handle two players.
Assume the first two connections are the players and ignore any additional connections.

2: Game State Management

Decide where you want to manage the game state (client-side or server-side).
Managing it server-side will earn you bonus marks since in that case, you don't need to provide the opponent ship's placement to the user and basically the user is not able to cheat.
Implement the game state management.

Make sure you are able to distinguish different types of messages from clients (e.g., Ship placement message, move message, game finished message, …)

3: Remove Computer Player

Adjust your Battleship game's logic to remove the computer opponent and instead send moves to the WebSocket server.

4: Implement the Multiplayer Gameplay

Modify the game to connect to your WebSocket server and send/receive messages related to game moves.

Ensure the game can distinguish between the two players and correctly process and display each player's moves in real-time.

Deliverables:

1. Create a folder called *YourLastName_YourFirstName_YourStudent#_Lab3*
2. Create a brief PDF file, ideally a single page, as a report. Explain how and where (on the client or server side) you managed the game state.
3. In this folder, put the corresponding source code, the report, and a video demo in .mp4 format.
    a. Save the Lab3 folder in your TMU Google Drive and share (make sure you provide read permission to TMU members)
    b. Zip the Lab3 folder and upload it to D2L. Provide us with a link to your shared Google Drive in the comment section.
4. **Your submission must be complete according to the guidelines above. Failure to do so will result in a mark of Zero on the lab.**

Evaluation Criteria:

1. Functionality: Does the code work as expected without bugs?
2. Code Quality: Is the code well-organized, commented, and following best practices?
3. Design: How visually appealing and user-friendly is the interface?

Submission Deadline:

Wednesday, March 27, 2024 @ 11:59 pm