Build a Triva Game using WebSockets with Typescript

Build the Server Code

First, we will need to create some types but what will they be?

- **IGame** - this is the main interface where other types will branch from

```
export interface IGame {
  id: string;
  currentQuestionId: string;
  done?: boolean;
  hasStarted: boolean;
  timer: number;
  clients: IClient[];
  questions: IQuestion[];
  settings: ISettings;
}
```

- **id** - a distinct identifier to represent each game
- **currentQuestionId** - keeps track of the current question
- **done** - boolean value to know if the game is finished
- **hasStarted** - boolean value to know if the game has started
- **timer** - how long a person has to answer each question
- **clients** - the person playing the game w/ other related info
- **questions** - all the questions for the game with other related info
- **settings** - settings related to the game

- **IClient** - each person (computer, phone, etc) at the url of the game

```
export interface IClient {
  id: string;
  name: string;
  questionsAnswered?: eQuestionAnswered[];
  score: number;
  ws?: WebSocket;
}
```

- **id** - a distinct identifier to represent the client
- **name** - the display name of the client
- **questionsAnswered** - keeps track of all answered questions where
  - 1 - answered correctly
  - 0 - answered incorrectly
  - -1 - not answered
- **score** - the current score

- **ws** - the websocket for each client, which allows us to send data to the correct client
- **IQuestion** - each question

```
export interface IQuestion {
  id: string;
  seq: number;
  text: string;
  answers: IAnswer[];
  done: boolean;
  hasFirstCorrectAnswer: boolean;
  clientIdsWhoAnswered: string[];
}
```

- **id** - a distinct identifier to represent the question
- **seq** - helps to keep the order of the questions
- **text** - display text
- **answers** - answers for each question and related info
- **done** - boolean value to know if the question is done
- **hasFirstCorrectAnswer** - boolean value to know if the question has been answered correctly at least once
- **clientIdsWhoAnswered** - list of clients who answered the question
- **IAnswer** - each answer per question

```
export interface IAnswer {
  id: string;
  text: string;
  isCorrect: boolean;
}
```

- **id** - a distinct identifier to represent the answer
- **text** - display text
- **isCorrect** - boolean value to know if answer is correct
- **ISettings** - settings for the game

```
export interface ISettings {
  questionsCount: number;
  timePerQuestion: number;
  timeBreakPerQuestion: number;
}
```

- **questionsCount** - how many questions per game
- **timePerQuestion** - the amount of time to answer each question
- **timeBreakPerQuestion** - the amount of time to break before the next question loads

server.ts - entry point for websocket

- has a games variable that is used to store all games data - this could be stored in a database or some sort but doing everything here for now
- once a connection is established, get a unique key for the client and set it as it's id

```
const clientId = req.headers['sec-websocket-key'];
ws.id = clientId;
```

- listen for all messages and send to the handleRoute method in the router.ts file

```
ws.on('message', async (message: string) => {
  await handleRoute(message, games, clientId, ws);
});
```

router.ts - routes the work to be done to the correct methods

- First we parse the message into json with `const obj = JSON.parse(message);`
- check for the following methods:
  - create - creates the game to be played
    - only create game if client is not already in a game
  - join - allows a client to join a created game
    - only join game is client is not already in a game
  - start - starts the game
  - guess - checks the answered question in a game

games/index.ts - routes are sent here to do the all the work - Let's discuss

methods

```
export const createGame = async (
  clientId: string,
  settings: ISettings,
  ws: WebSocket,
  name: string,
  games: IGame[],
) => {
  const questions = await getQuestions(settings.questionsCount);
  const game = {
```

```
    settings: settings,
    id: uuidv4(),
    timer: settings.timePerQuestion,
    clients: [
      {
        id: clientId,
        name,
        score: 0,
        ws,
        questionsAnswered: new Array(settings.questionsCount).fill(
          eQuestionAnswered.notAnswered,
        ),
      },
    ],
    questions,
    currentQuestionId: questions[0].id,
    hasStarted: false,
  };
  games.push(game);
  sendSocket(
    {
      method: eRouteMethods.create,
      gameId: game.id,
      clients: game.clients,
      clientId,
      isInGame: true,
    },
    ws,
  );
};

```