I Know the Answer!

It was a late Friday night and we were all playing **Kahoot!** which somewhat had become a tradition at work, but suddenly everything stopped working. I was devastated for days, even weeks, but suddenly I realized that **Kahoot!** wasn't going to come back and it was time to take matters into my own hands. I was going to build a new platform providing an even better online quiz experience and I was going to call it **I Know the Answer!**

Endpoints

All endpoints are documented with Swagger and accessible at http://localhost:4567/docs. It is required for the server to be running to access the docs. The server is included in the assignment in **Canvas**.

Assignment description

In this assignment we are going to build an online quiz platform, similar to **Kahoot!** which makes use of **React, Redux** and **Socket.io**. See assignment description below for a detailed enlisting of the functionality that should be provided in this assignment:

(20%) Authentication

- (7.5%) Login
    - You are able to login by providing the correct credentials, username and password
    - Upon successfully logging in, you are redirected to the dashboard
    - A failed login attempt, should result in an error message stating that you provided the incorrect credentials and allow the user to try again
- (7.5%) Register
    - You are able to register by providing the following information (which are all required)
        - Username
        - Password
        - Display name
    - The registration can only be successful if it passes the following conditions
        - The username is not occupied and is more than 3 characters in length
        - The password is more than 8 characters in length
        - The display name is more than 3 characters in length
    - Upon successfully registering the user should be redirected to the login where he can use the newly created user to login
    - A failed register attempt, should result in an error message stating that you provided insufficient information and allow the user to try again

- (2.5%) Authenticated pages
    - The following should not be accessible if a user has not yet logged in:
        - Accessing the dashboard
        - Creating a new match
        - Participating in match
        - Viewing match summary

- (2.5%) Layout
    - All authenticated pages should have a navigation bar, which should contain:
        - The avatar image of the user
        - The display name of the user
        - A button, which allows the user to logout

(25%) Dashboard

- (5%) A list of all matches
    - All matches (regardless of the state) should display the following information
        - Title of the match
        - Image for the match
        - Number of participants, e.g. 1 / 4
        - Status of the match, where the only available status are the following:
            - Not started
            - Started
            - Finished
- (5%) Joining a match
    - A match can only be joined if it meets the following conditions:
        - The match is in the ‘Not started’ state
        - The match is not full
    - Only one match can be joined at a time, therefore if a player decides to join a match, he must leave his current match before entering another one
- (15%) Creating a match
    - A user is able to create a new match, which will be visible to others when it has been successfully created
    - Every match should contain the following:
        - A title
        - An image (*uploading an image yields more points, although URLs are accepted as a solution*)
        - A list of questions
            - Each question should contain the following:
                - The actual question
                - Available options, which are four and you must select one option as the correct option
    - Upon successfully creating a match, the user should receive a message that the match was successfully created and should be redirected to the dashboard

(15%) Waiting room (**Not started**)

- Up to 4 people can compete in the same game, but at first they wait in a waiting room. The minimum is 2 players to start the match.
- When all players have joined, the host is the one that starts the game. This will notify all players in the waiting room that the game is starting, which sends all players within the match to the game
- Players can only decide to leave the match during this period, when the match has not yet started
- Within the waiting room, there should be several live updates
    - When a new player joins the match, existing players should display the new player without having to refresh the page (see **Events**)
    - When a player leaves the match, existing players should display the changes without having to refresh the page (see **Events**)
    - When the host decides to start the match, notifying all players that the game has started (see **Events**)

(20%) Game (**Started**)

- The game consists of **x** many questions, which each has a time limit of 10 seconds
- When a player has given his answer to a question, other players within the match should see that he has given his answer. During this step, his answer should not be revealed, only display that he has given his answer
- When the time has passed, all answers should be revealed, where players can see what other players guessed and the correct answer should be marked
- This will go on until all questions are finished and the user is redirected to the game summary
- All updates within the game are live, meaning that no refreshes are involved and the game relies solely on sending and receiving events from the server (see **Events**)

(15%) Game summary (**Finished**)

- When all questions are finished a winner podium should display the 1st, 2nd and, 3rd place of the game
- More detailed information about each player's score should be displayed as well. The following information should be displayed for each player:
    - Place of the player
    - Avatar of the player
    - Display name of the player
    - Total score of the player (in the game)
- The formula that defines the points is the following:
    - $(Maximum\ time\ (s)\ -\ Time\ elapsed\ (s))\ *\ 10$, e.g. (10 - 1) * 10 = 90 points, if you answered the question correct after 1 seconds

(5%) Additional functionality

- Handle browser refreshes correctly
    - (2.5%) Refresh within all authenticated pages should check the logged in user. If the user has an invalid session, he should be redirected to the login page. If the user has a valid session, he should stay on the same page and still maintain state with the server
    - (2.5%) Refresh within all unauthenticated pages should check the logged in user. If the user has an invalid session, he should remain on the same site. If the user has a valid session, he should be redirected to the dashboard

Events

The underlying server is an Express server utilizing socket.io for event-based communications between client and server.

Outgoing events

Outgoing events are events which are sent from the server to connected clients. Below is a list of all outgoing events emitted from the server:

- **joinmatch** - The server emits this event to notify clients within a particular match that a new player has joined the match

```
socket.on('joinmatch', user /* {
  "id": "string",
  "username": "string",
  "displayName": "string",
  "avatar": "string"
} */ => {
  /* function body */
});
```

- **leavematch** - The server emits this event to notify clients within a particular match that a player has left the match

```
socket.on('leavematch', user /* {
  "id": "string",
  "username": "string",
  "displayName": "string",
  "avatar": "string"
} */ => {
  /* function body */
});
```

- **startmatch** - The server emits this event to notify clients within a match, that the match has been registered as started

```
socket.on('startmatch', () => {
  /* function body */
});
```

- **answer** - The server emits this event to notify clients within a match, that a certain user has given his answer to a question

```
socket.on('answer', user /* {
  "id": "string",
  "username": "string",
  "displayName": "string",
  "avatar": "string"
} */ => {
  /* function body */
});
```

- **updatetimer** - The server emits this event to notify clients within a match what the current timer for the current question is. This event is used to sync all players within the same match, so they don't have to keep track of the time left on each question. This event is required to display the time left for each question within a game

```
socket.on('updatetimer', counter /* number */ => {
  /* function body */
});
```

- **answers** - The server emits this event to notify clients within a match what the answers for each question were. This event is emitted after the time has passed for each question

```
socket.on('answers', answers /*
  [
    {
      user: {
        "id": "string",
        "username": "string",
        "displayName": "string",
        "avatar": "string"
      },
      answer: "number"
    }
  ]
*/ => {
  /* function body */
});
```

- **nextquestion** - The server emits this event to notify clients within a match that the next question can be displayed. This event is emitted 5 seconds after the time for the previous question has elapsed.

```
socket.on('nextquestion', questionNumber /* number */ => {
  /* function body */
});
```

- **finishedgame** - The server emits this event to notify clients within a match that the game has finished

```
socket.on('finishedgame', match /*
  {
    "id": "string",
    "title": "string",
    "titleImage": "string",
    "questions": [],
    "owner": {},
    "answers": [],
    "currentQuestion": 0,
    "players": [],
    "status": "finished"
  }
*/ => {
  /* function body */
});
```

Incoming events

Incoming events are events which the server listens to and reacts accordingly. Below is a list of all incoming events which can be emitted to the server:

- **joinmatch** - Whenever a client decides to join a match, he should notify other clients within the same match that he has joined the match

```
socket.emit('joinmatch', matchId /* number */, user /*
  {
    "id": "string",
    "username": "string",
    "displayName": "string",
    "avatar": "string"
  }
*/);
```

- **leavematch** - Whenever a client decides to leave a match, he should notify other clients within the same match that he has left the match

```
socket.emit('leavematch', matchId /* number */, user /*
  {
    "id": "string",
    "username": "string",
    "displayName": "string",
    "avatar": "string"
  }
*/);
```

- **startmatch** - When the host decides to start the match, he should notify other players within the same match that the match has been started

```
socket.emit('leavematch', matchId /* number */);
```

- **answer** - When a client decides to answer a question within a match, he should notify other players that he has given his answer

```
socket.emit('answer', match /*
  {
    "id": "string",
    "title": "string",
    "titleImage": "string",
    "questions": [],
    "owner": {},
    "answers": [],
    "currentQuestion": 0,
    "players": [],
    "status": "finished"
  }
*/, user /*
  {
    "id": "string",
    "username": "string",
    "displayName": "string",
    "avatar": "string"
  }
*/, answer /* number */, timer /* number */);
```