Advanced Programming Spring 2023

Assignment 3

Release date: March 26, 2023

Due date: 11:55 PM, April 14, 2023

My Game - Rang / UNO / 3 PATTI

Overview:

For this programming assignment, you will be coding one of the following 3 card games:

1. Rung
2. UNO
3. 3 Patti

Each student has been assigned one of the 3 apps which they have to code. You can find the distribution here .

For this assignment, you will be coding an entire MERN application from scratch.

Firstly, you may create the boiler for a MERN stack application with `typescript` (we have already provided some starter code - however, you can also create your own if you want to). This has already been covered in class as well. You can also use other methods to create your boiler code. One of such examples is, Vite. You can read more about it here and here.

After setting up the boiler code and testing it, you may start by introducing sockets to your application. For that you are required to use `Socket.IO` and you can find its docs here.

Installing node modules

In the boiler code provided to you, there are separate folders for backend and frontend. You are required to do `npm i` once in the `MyGame folder` and once inside the `frontend folder` for the node modules to be correctly installed.

Backend

This folder consists of a file, server.ts. This is where your server implementation would reside. Upon running this file, you will start a server on port number 3001 that shall listen and cater to all requests. Additionally, the file consists of an io object that listens to certain socket events using the socket.io library. You have been provided with a listener for the 'connection' event that is emitted by all clients when they open up a socket to this server. Hence, given the initial implementation ,upon every user connecting, you will be able to view `USER CONNECTED` on the server-sided console.

In your implementation, you can edit and add listeners of any type to suit your program.

You can review various options from the official `socket.io` documentation.

Frontend

This is the typical `React-based` frontend that shall be developed for interactive purposes. To see where sockets come into hand, navigate to the `App.tsx file` where `socket.io-client` has been used to create a client instance to hit requests on our server running at backend. This is a global socket that has been created per frontend/client running.

To have access to the **same socket** in various other components of your app, you are to pass it down as `props` to the child component. There exists one sample component, `HomePage`, that is rendered via react-router and passes the socket down to the child component. You can replicate the same for all the new components that you create.

A slight catch-

Everytime that a user reloads a page, that alone causes the socket on the client's end to emit a 'disconnect' event and create a new socket upon reloading that again emits a 'connection' event with a different socket-id on the server's end. While you may want to just let this pass by imagining that the refresh button does not exist, it would be convenient if you cater to this.

There can be multiple ways to cater to this. One of them being that upon every new connection, the server assigns a unique client id and emits it back to the client. At the same time the server maintains a map of each unique client id against the socket id. In case that an already existing client id emits an event with a different socket id, a page refresh can be detected. On the client's end, you would have to accordingly ensure that once assigned with an ID, it cannot be overwritten in the future by the server.

Once sockets are set in your application you may start coding the game.

Please note:

1. Each game requires 4 players to start in all cases. All 4 players are required before starting the game.
2. Your application will only support 1 game at a time.
3. To simulate the game, you can open 4 different tabs where each tab will represent a new user.
4. Initially, the turns will always be from user 1 to 4 in the correct order.
5. There is no fixed format for the messages being displayed. However, the messages mentioned in the documents need to be appropriate so that the game is understood.
6. Most of the `HTML`, `CSS`, for the game board, cards, deck, hands are provided to you. However, it's in the form of plain `html css`. It is your job to convert them into `react functional components`.
7. For the `button and input field`, you can use any 3rd party UI library or the plain html input and button tags.

8. `CSS` is global, so if you introduce any new css classes then their names should be unique! This problem can be solved using `CSS Modules` but we won't be doing that since it is not in the scope of our course module.
9. Once the game has ended, an appropriate message containing the name of the user who has won should be shown. The game should be disabled then meaning that no user would be able to take another turn. Similarly, in case of a draw, show an appropriate message and disabled the game. In case of an invalid move, please show an appropriate message to that user only.

Flow of the game:
The initial flow for all 3 games will be the same. This will be a `Single-Page-Application` but you can divide it into pages if you want to.

Initially, on opening `localhost:PORT`, the user will be shown an input field along with a button. (The default port is 3000 and 3001 for the frontend and backend respectively) You are required to take in the name of the user here and upon pressing enter, they will be shown the loading state which will say that the game is waiting for players to join.

The loading/waiting message will be shown to each user, until the 4th user enters their name and joins the game. Once the game starts the UI will change and the appropriate UI for the game you are coding will be shown to all the users. Please note that each user must have the same updated screen at all times.

As the game progresses, messages will be shown to all of the users as well in order to guide them through the game. For example, when the game starts, it will be player 1's turn so each user will be shown in the message box that it is player 1's turn. Once that player takes a turn, then everyone including them will be shown in the message box about the move they did along with the message whose turn it is next. For example, if player 1 throws an Ace of Spades, then the message box will say that "Player one played an Ace of Spades. It is player two's turn now".

When the game ends (when someone wins), the game must become unresponsive and a message should be shown stating the name(s) of the users that have won this round of the game.

HTML CSS - dividing the code into components:
We have provided you with the following html and css files:

1. Rang.html and rang.css
2. Uno.html and uno.css
3. Teenpatti.html and teenpatti.css
4. Playing-cards.css (required for rang and teen patti only)
5. Uno-cards.css (required for uno only)

Each html file contains the layout of the UI which will be shown while the game is being played. It is up to you on how you divide the UI into multiple components so the entire logic and UI isn't just in 1 single file.

You can open these html files in your browser to inspect them in detail.

The files playing-cards.css and uno-cards.css will be required for the UI of the cards and you don't have to open those. You can add them accordingly to the index.html file in your public folder (in frontend) by using the following code:

```
<link rel= "stylesheet" href= "uno-cards.css" />
Or
<link rel= "stylesheet" href= "playing-cards.css" />
```

By doing this you won't have to worry about including the css in your components etc.

Rules and flow for UNO:

Players:

- To get a particular UNO game started, you need to have exactly 4 players. Each player will be playing on its own meaning there is no coordination.

Setup:

- Initially, each player is given 7 cards and the rest of the cards are placed in the draw pile. Please note that there are a total of 104 cards for our game:
    - 19 Red cards – 0 to 9
    - 19 Blue cards – 0 to 9
    - 19 Green cards – 0 to 9
    - 19 Yellow cards – 0 to 9
    - 8 Skip cards – two cards of each colour
    - 8 Reverse cards – two cards of each colour
    - 8 Draw cards – two cards of each colour
    - 4 Wild Draw 4 cards –
- To start the game, the top most card from the draw pile will be moved to the discard pile meaning it will now be shown to all of the players and the game will start.
- It may be possible that the first card drawn from the draw pile put into the discard pile is an action card then the action will be performed by the player 1. If it is a wild card then the player 1 can throw any card they want to (no restrictions). If it is a wild draw 4 then normally the deck is reshuffled and the game is restarted but for the sake of simplicity, we are going to ignore it and the player can throw anything they want to with no restrictions.

Gameplay:

- The players then throw their cards in the correct order (player 1 then 2 then 3 and 4)
- Each player must throw a card that matches the number of the card on top of the discard pile or matches the colour of that card. For instance, if the Discard Pile has a red card that is an 8 you have to place either a red card or a card with an 8 on it. You can also play a Wild card (which can alter current colour in play).
- If the player has no matches or they choose not to play any of their cards even though they might have a match, they must draw a card from the Draw pile. If that card can be played, play it. Otherwise, keep the card, and the game moves on to the next person in turn.

Action Cards:

- **Skip card.** The skip card can only be played if it matches the colour of the card present in the discard pile. In that case, the next player's turn is skipped.

    - **Reverse card.** The reverse card card can only be played if it matches the colour of the card present in the discard pile. In that case, the direction of the gameplay is reversed.
    - **Draw cards (draw 2).** The draw card can only be played if it matches the colour of the card present in the discard pile or that *card is another draw 2 card of any colour*. In that case the next player has to draw 2 cards. Please note that the next player draws 2 cards and takes their turn as well. However, for simplicity, we will be ignoring the condition that a player can throw draw 2 if the previous player has thrown a draw 2 meaning that if player 1 throws a draw 2 then the player 2 will draw 2 and take their turn. They cannot get away with it by throwing another draw 2 card.
    - **Wild card.** The wild card can be used over a card of any colour and then the same player can throw another card of any colour. Please note that the wild card cannot be used on draw 2 or draw 4 cards.
    - **Wild draw 4 card**. This acts just like the wild card except that the next player also has to draw four cards as well as forfeit his/her turn. With this card, you must have no other alternative cards to play that matches the colour of the card previously played.

- The game ends when any player has no more cards remaining. Please note that normally in UNO, players have to say “UNO” when they have 1 card remaining but we will be ignoring that condition.
- If the deck ends before any of the players wins, then the game will be a draw!

Rules and flow for Teen Patti:

Players:

- To get a particular Teen Patti game started, you need to have exactly 4 players. Each player will be playing on its own meaning there is no coordination.

Setup:

- Initially, each player is given 3 face down cards and 3 face up cards. Each player will be able to see the 3 face up cards for all of the other users.
- Apart from that each player is given 3 cards in their hand. (which of course others cannot see).
- The rest of the deck is kept aside.

Gameplay:

- The game begins from the top card of the set aside deck meaning that the top card will be played in the centre of the table and all of the players will be able to see it.
- The players then throw their cards in the correct order (player 1 then 2 then 3 and 4)
- Every player must throw the card of the same value or higher than the topmost card.
- Once a player throws a card, then they must pick one from the set aside deck. Every player must have at least three cards during the game (this rule goes away once the deck finishes). We can automate the picking up cards from the deck.
- If the player has no valid card then they can use a **power card**:
  Power Cards
  - **2 is a refreshing card.** You can throw another card with it and the game restarts from that card. For example, at the top of the pile is 9; if the player throws a 2 followed by a 3, the next player will have to throw 3 or higher.
  - **7 is a low card.** When a player throws a 7, the next player must throw a 7 or lower to continue. This only applies to the next player, and the player after will follow the standard game rules (i.e., same or higher).
  - **8 is an invisible card.** The game continues from the card that was below 8.
  - **10 burns the pile.** The pile is set aside, and the next player can start with any card.
- If the player has **no valid card** to play and **no power card,** then they must pick up the whole pile and add it to their hand. Their turn is skipped. You can do this automatically since the server should/will know what cards the players have at all times. (maybe add a short timeout so it doesn't happen instantly)
- Once the player runs out of cards in their hand and there are no cards left in the set-aside deck, the 3 face-up cards come into play. The players select either one of them to play.
- Once the 3 face-up cards are used, the 3 face-down cards come into play. The players select the one they want to play (although they won't be able to see what that card is).

Rules and flow for RANG:

We will be implementing the basic (vanilla) version of Rang that abides by the following rules.

Players:

- To get a particular Rang game started, you need to have exactly 4 players who have coordinated in pairs of 2. By default, we will be pairing player 1 with player 3, and player 2 with 4.

Who decides the *Rang*? :

- From the deck of 52 cards, each of the four players are allotted a card randomly.
- Of the 4 cards displayed to each of the players, there will be one player who will get the least ranked card and there will be one player who gets the highest ranked card. You need to implement it such that there can be no two players who get a card of the same rank. This implies that a single go would be enough to choose the player who decides the *Rang*.
- The player who gets the highest ranked card decides the *Rang* at a later stage when the game begins. You can assume that this player is named as R1 and the corresponding partner of R1 is R2. On the parallel sides, the other two players can be A1 and A2 who are together playing as a team.
- All the cards are restored back to the deck and shuffled.

Distributing the cards:

- R1 is given 5 cards randomly from the deck of 52 cards and is asked to choose the *Rang* (one of hearts, diamonds, clubs and spades).
- The chosen Rang is fixed for the rest of the game.
- The remaining players are also given 5 cards each and this marks the end of round 1 of distribution.
- No two cards such that the two cards have the same suit and rank can be shared between players.
- Proceeding to round 2 of distribution, from the remaining 32 cards, each of the 4 players are given 8 cards each.

Starting the game:

- R1 takes the lead to select a card and make it visible to other players.
- The teams take alternating turns so one of A1 or A2 follows and takes turn after R1. Please see that the order of turns is fixed. It either has to be:
    - R1→ A1→R2→A2 in ALL of the rounds
      OR
    - R1→ A2→R2→A1 in ALL of the rounds
- One round is completed when all of the 4 players are done taking their turns and proposing their cards.

- The team whose player has put forth the highest ranked card or has cut off the thread by throwing a rang card wins the round and takes the lead.
    - For instance, if A1 throws the highest ranked card and team A wins the round, then A1 would take the lead for the following round. The order of turns would be:
        - A1→R2→A2 → R1 **if** you chose R1→ A1→R2→A2 for the first round.
          OR
        - A1 → R1→A2→ R2 **if** you chose R1→ A2→R2→A1 for the first round.

Who wins a round?:

- The player who leads a particular round decides what suit to play for the round.
- Every player is expected to propose a card of the same suit unless they do not have a card of the same.
- In case that a player does not have the same suit, they have the option to choose from a rang card or a non-rang card.
    - If they choose a rang card, they would be taking the lead unless some other player proposes a rang card of a higher rank in the following turns of the same round.
    - If they do not choose a rang card, they would not be taking the lead for the round.
- In case a player tries to propose a card of a different suit despite having a card of the same suit, they should be warned and asked to retry.
- The player who proposed the highest ranked card wins the round if every other player has a card of the initial suit proposed.
- If there is any player who proposes a Rang card because they do not have a card of the originally running suit, then the player with the highest ranked Rang card proposed wins.
- Given that the *Rang* chosen is *Hearts*, consider the following flow:
    - A1 takes lead from last round and proposes Ace of Diamonds
    - R1 has Diamonds, so they propose 3 of Diamonds
    - A2 does not have Diamonds so they propose 2 of Clubs
    - R2 has Diamonds, so they propose 5 of Diamonds
    - **Hence, A1 is clearly senior and team A wins the round.**
    - A1 is again leading since they came senior in the last round.
    - A1 proposes 5 of Spades
    - R1 has no Spades so they propose 5 of Hearts (which is the *Rang*)
    - A2 has Spades so they propose Ace of Spades
    - R2 has Spades so they propose 3 of Spades
    - **Hence, R1 is senior-most and team R wins the round.**
    - R1 proposes 4 of Clubs
    - A2 proposes 3 of Clubs
    - R2 proposes 5 of Clubs

    - A1 has no Clubs nor a Rang card, so they just propose Queen of Diamonds.
    - **Hence, R2 senior-most and team R wins the round.**

- **The above listed are just a few of the flows. You will be tested on all possible game flows that abide by the rules listed in the previous section.**

Who wins the game?:

- The pair who is first to win 7 rounds wins. 🎉 🎉

Guidelines:

Some Tips:

- Make use of the internet (stack overflow etc). However, there is a difference between copying code and understanding it before using it. It is always recommended to understand every piece of code you write or use from the internet.
- It is better to keep track of the deck - discard - draw pile - etc on the server side and then update each of the clients after every move.

Plagiarism:

- Refrain from collaborating with your peers whilst coding.
- Any similarity between components, and methods will be reported and strictly addressed.

Grading:

The total marks for this assignment are **100**. There will be partial marking for this assignment. The following will be the division of marks:

1. Correct use of sockets to communicate between the clients and the server. **[15 marks]**
2. Division of the UI into components. **[15 marks]**
3. Correct starting condition which includes waiting for other players and asking for their names. **[10 marks]**
4. Correct implementation of the gameplay. **[20 marks]**
5. Correct implementation of the game rules. **[20 marks]**
6. Correct winning / draw conditions. **[7.5 marks]**
7. Correct display of messages throughout the game. **[7.5 marks]**
8. Writing clean code which is properly formatted (use prettier) along with the correct usage of react hooks studied in class. **[5 marks]**

Submission guidelines:

You can either put their code in a private git repo and add your respective TAs to it or you can zip and submit the entire application (**please remove node modules before zipping it** 🙏). The filename of the final submission should be your **roll number**.

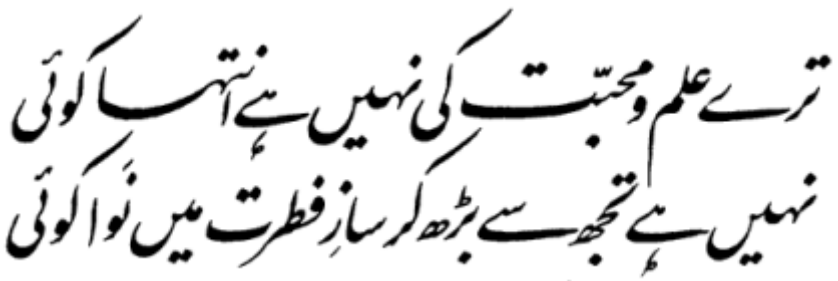
Goodluck 🎉

Random

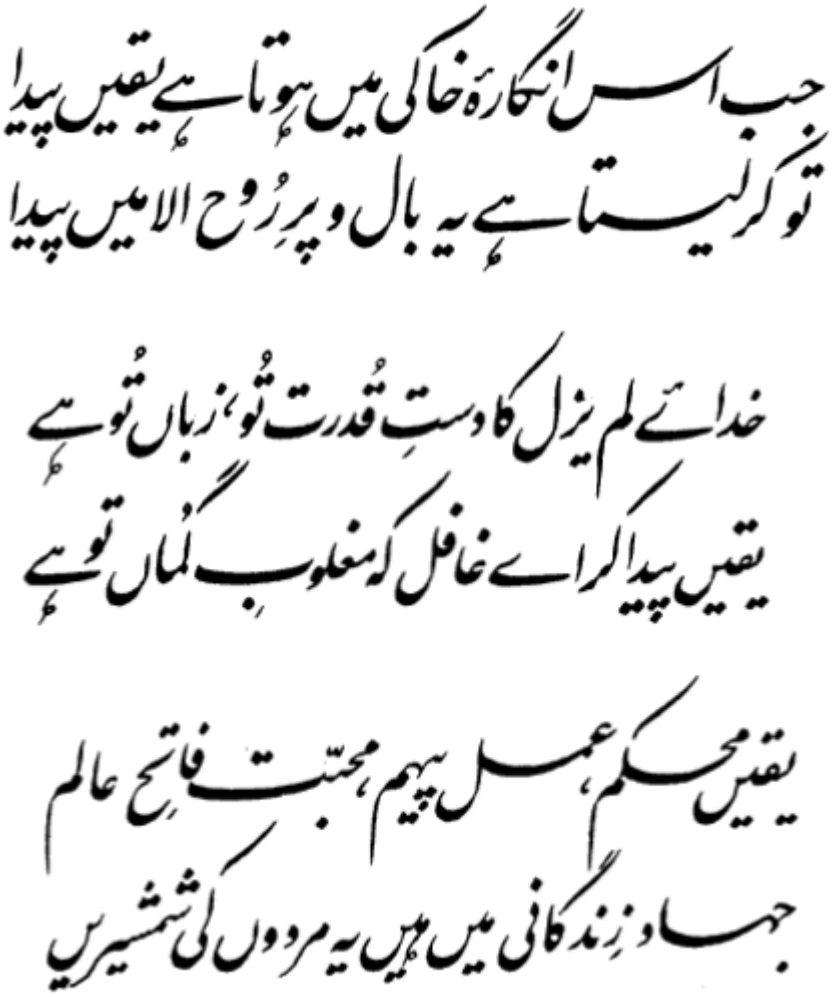
The instructions for the assignment are over. This is just Hassnain trying to write something.

CS 300 is definitely one of the most demanding course. Whether you are a senior, junior or a soph, you all had to put a lot of time and effort for this. I am certain some of you sacrificed your color days, some sacrificed sleep, some sacrificed gatherings with friends and family but now we are in the last phase. We all are very much proud of you all for all progress. The grades or marks you guys will get they really are no metric of the effort you have put or the learning you have had. It is unfair to grade you all equally since we all don't start from the same point, but please bear with us on this. You all have done a wonderful job and have come so far ahead of the person you were at the start of semester. Don't you ever underestimate yourself and never ever doubt you. This module can be very overwhelming but do not lose sight of who you are in front of it. You all are very much capable. We all are rooting for you. CS 300 teaches you a lot of things in a very short span of time MashAllah you all have done so wonderful in such a short span of time. Insha'Allah you all will do great things in life. Ziada tension nahi laini, in the end sab hou jata hai, the important thing is to never give up. I know sometimes everything looks really overwhelming and daunting, and you start to doubt yourself, in such cases you all should know that, I believe in you, and you all are capable of achieving great things. Best of luck everyone ^-^. (P.S : You all are welcomed to claim chocolates from Hassnain anytime for all the progress you have made).

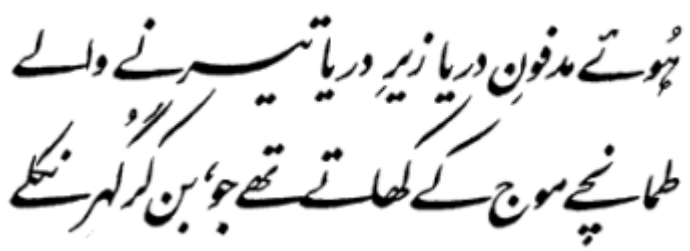
Here is some Iqbal, which I find really motivating often times :)

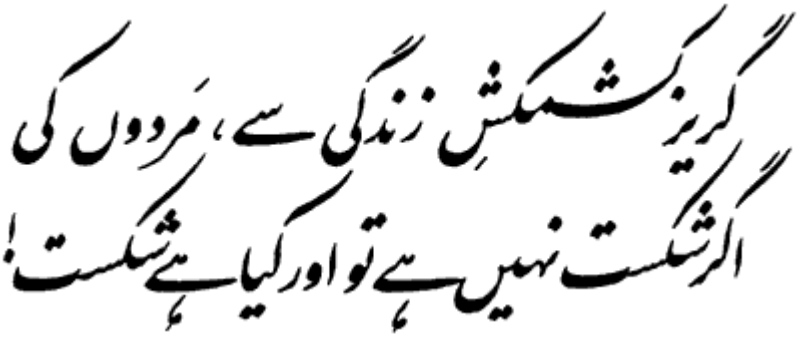


Just belief in yourself and stand firm and you are destined to achieve great things :)

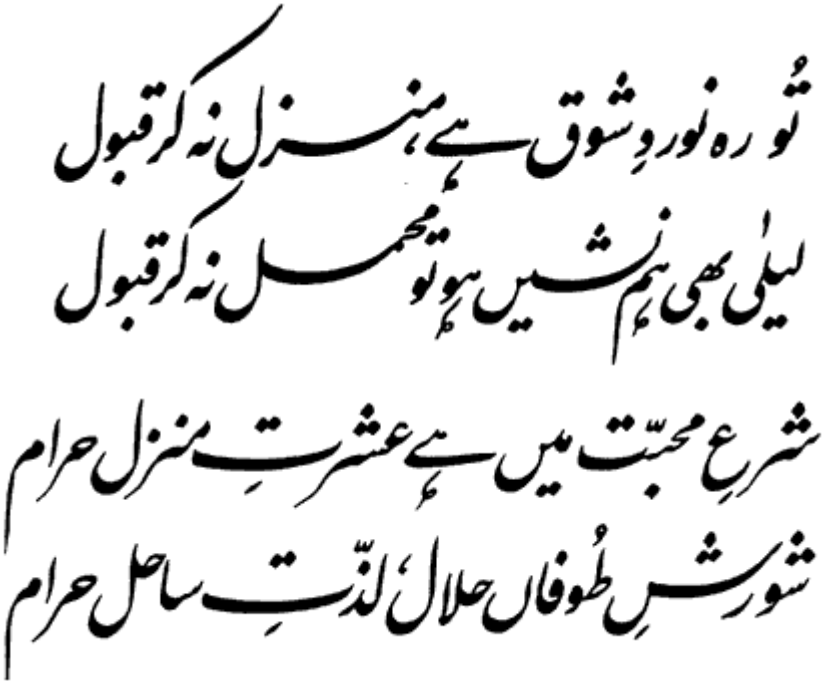


I know that, even with all the faith on oneself things can get very much overwhelming and in such cases you often want to quit , please don’t the temptation to quit is often the strongest when you are about to win :)





Often times, we achieve a certain target and we get lazy after that. Please do not fall in this trap as well :) You are always capable of achieving more. Never ever settle, you see as long as there is motion is life and as soon as they rest, it's over. No matter, how slow you go, always keep moving ahead. Progress is progress and I will always be proud of you for all kinds of progress. :)



(I want to go on and on but I think it's already a lot and I have made a point, the important thing is to believe in yourself, not give up, keep moving ahead and spreading peace and love ^-^ . If anyone of you ever want to talk about Iqbal, you are always welcomed)