I have chosen option three, some other distributed system. My project is an online quiz game that you can play along with your friends as opponents or team but restricted within the local area network; if you want to play on two separate devices, it must be connected on the same network. Also, I have a chat system that you can use to do whatever you want to do.

The rules are pretty simple; you enter your name, and you start solving questions, and if anyone else gets connected to the server, you will be notified in the chat. And if someone gets connected, you get updates from time to time of what other players are doing. Consider a two-player quiz game; the gist of the quiz questions or the project demonstrates that if someone solves(by pressing the 'check' button and it is correct) some question 'X,' first then, they receive more points than the other person.

I have tried to solve the main two distributed systems problems, how does synchronization work between more than two clients with message passing using the server as the main truth point. Another problem is timing the events according to some global clock or famously known as 'happened before' problem, i.e., event order based on the timestamp of some global clock. Both issues are fundamental in the design of a distributed system.

The first part mainly deals with informing all the clients at almost all the time about what a particular client is doing. In the project, I have 2 top bars that show updates on what other people are doing, such as typing in the chat window, typing in the answer window, solving a particular question, and a few others. I am using web sockets API using 'ws' module in node js to communicate back and forth with the server, which further communicates with all the other clients and thus synchronizes the state of all the clients. Also, I am sending JSON objects back and forth from the server and client as well.

I am using node.js for the backend server. The server is created with the help of 2 modules in node.js 'express' and 'http'. Now, node.js is a single-threaded application and is forming a queue of requests in and itself.

For the second problem, I have used a priority queue to solve the problem, which is in increasing order of the received timestamp requests. To show what event came first, after receiving an event, I am using setTimeOut to wait for a few seconds to see if any event is happening or has already happened relatively at the same time. If the event has happened then, it enqueues it in the priority list in the proper place and responds to both the requests accordingly. The request that came first was timestamped and therefore, will be executed first, and that player will be awarded relatively more points. Even though node js is a single-threaded application, I am adding all the events to the queue before executing, thus maintaining event ordering.

In a nutshell, I have tried to demonstrate two main problems in the design of distributed systems; synchronizing the state and ordering events or 'happened before.' Overall, I had fun doing the project; I learned something new and never had any experience with the technologies utilized in the project.