Nonceblox - Round 1

(Github repo - <https://github.com/sarthakahuja07/nonceblox>)

Frontend

1) Write a custom hook to manage state for <input> tags.

Ans.)

We can create a custom hook that accepts type of input and initial value as parameter and implement State change functions inside the custom hook,.

An implementation of this is below -:

useFormField.js Hook

Text

Description automatically generated

App.js

Text

Description automatically generated

Here, I have created 3 inputs and their state is managed by the custom hooks. We can also add other fields such as label,name,errorMessage and isValid field to the custom hook and use the hook in our app.

2) What will be the output for the below code if myState is incremented 5 times before scrolling?

Ans.)

The updated state of the component will be 5, but still the output will always be “Scroll: 0” because when the addEventListener is binded to it, the state of the component is 0, and the useEffect with no dependancy only runs for the initial state. i.e when component gets mounted. And even if there was state in the dependancy of the useEffect, there would be 2 addEventListeners attatched to scrolling as there is no cleanup function in the useEffect, so the output would be “State: 0”and “State: 5”.

3) How to implement componentWillUnmount in functional components?

Ans.)

componentWillUnmount in functional component is triggered by the cleanup function in the useEffect, i.e a return function which has the logic for when a component unmounts.

Its implementation is as follows -:

Text

Description automatically generated

4) What is the difference between the below snippets?

Ans.)

Both the code snippets give the same result, but using the prevState argument can be helpful in the case of the state being in the form of an object and when you need to use or change just some properties of the object instead of making a new object and setting it to the state.

This is implemented as follows -:

A screenshot of a computer

Description automatically generated with medium confidence

5) How will you wait for all the promises to resolve/reject that are executed inside a loop? Answer with an example.

Ans.)

When there is a promises in a for loop, the loop does not wait for the promises to resolved or rejected. For using a promise in a for loop, we have to use async await. This is done by making the containing function of the loop to be async and using await for the promise call.

An example of this -:

Text

Description automatically generated

Another efficeint way of using promises in a loop is by using Promise.all method, for this we make an array of promises and push the promises using the loop, and then use the promise.all function to return a single promise and it resolves when all the promises in the array have been resolved

Text

Description automatically generated

6) Which component will be rendered when you visit /products ?

Ans.)

Here, path of the home page is ‘/’ and the component rendered against it is Home. When you visit ‘/products’, you would just see the “Home” component because v5 of the react-router always finds the first matching route, in this case “/” of “/products” is matched with the “/” route and hence the Home component gets rendered. In order to get the correct result, i.e to render the Product component on the “/products” route, we would have to put “exact” phrase in the “/” route, i.e.

<Route exact path="/" component={Home} />

Backend

1. What will be the order of the output?

Ans.)

There will be 3 processes running when the server is started, asyncronously reading the file, the asyncrononus setTimeout function and the main thread which console logs “Hello”. So, “Hello” will be logged first and then the first process which gets completed first out of the Timeout or Reading file will get executed (In my opinion reading a file takes longer than the 0 second timeout in setTimeout) and then the last one

2) Write a program to run 7 threads and create server instances for each thread using the http module with master and child relationships.

Ans.)

Child.js

Text

Description automatically generated

Index.js

Text

Description automatically generated

Output

Text

Description automatically generated

3) Write a middleware that does the following tasks:

Ans.)

authMiddleware.js

Text

Description automatically generated

4) What will happen with this command?

Ans.)

The command creates an index in the sessionlog collection of the db

So, the index key is “lastUpdateTime” and its value being “1”

And, { expireAfterSeconds: 1800 } is the options object that means the db retains documents in the collection for 1800 seconds

5) What is the fundamental difference between db.copyDatabase() and

db.repairDatabase()?

Ans.)

db.copyDatabse Copies a database from one mongod instance to the same mongod instance or from one mongod instance to another mongod instance within the current mongod. The fromdb and todb parameters are passed to the mongo shell method db.copyDatabase()

repairDatabase is a database repair utility. That is, it tries to clear up any corrupt documents in the database that might be preventing MongoDB from starting up. RepairDatabase could theoretically remove documents from the database, but how it works in detail depends on your storage engine.

6) Write the query to perform the following on a MongoDB collection name “Wonder”

Ans.)

a) Specify the maximum number of documents for a collection

-> db.createCollection( "Wonder", { capped: true, size:5242880, max:5000} )

b) Check if the collection is capped or not

-> db.Wonder.isCapped()

c) Converting a collection to capped Wonder collection

-> db.runCommand({"convertToCapped": " Wonder", size: 5242880});