Part 6

Polling

The scenario: Multiple clients connect to the server and need to communicate with each other in real time.

Example: client 1 is a rover (R), client 2 is a laptop computer (C) displaying the map with the rover on it, server (S) has the backend with the database and other processing.

When R is navigating the terrain in auto mode, it needs to constantly send information, such as its current location, to the server. Meanwhile, C, needs to constantly receive this information (in some processed form) from the server. Similarly, when R is navigating in the manual mode, it needs to receive information from the server about change in direction, etc.

The Problem: HTTP is a simplex (i.e., not a duplex) protocol: when the client requests the server responds, whereas the scenario above requires duplex communication. The server needs to prompt the client with updates.

A solution: Polling is a workaround to this problem that can be used to implement approximately real time communication. The idea is simple (considering the auto mode): the client, C, can keep asking (polling) the server for updates at regular intervals of time. The client, R, can keep updating the server at regular intervals of time.

A truly duplex solution to the problem would require a different protocol than HTTP, such as WebSockets (a thin layer above HTTP). However, that would be a different approach than the stateless RESTful HTTP we have been implementing so far.

This guide shows the use of an open source React library to implement polling. In this example the client will poll the server for current time and display a clock in the browser. The clock only needs to poll at once a second, but generally the polling rate can be adjusted to make it much faster.

We add the following end-point to backend_server/index.js:

```
app.get("/pollServer", (req, res) => {
   var d = new Date();
   const json_res = {
     "time" : d.toTimeString()
   };

   res.send(json_res);
 });
```

This should appear before app.get("*",…

This is the end-point the client's polling component will fetch.

Following is the updated App.js of the React app. Some code has been commented out to keep the focus on the current topic.

We're using a polling library available here.

To install it, navigate to the react app folder and run:

```
npm i react-polling --save
```

Notice at the top of the file:

```
import ReactPolling from "react-polling/lib/ReactPolling";
```

App.js with polling

```
import React, {useState}  from "react";
import TableComp from './TableComp';
import ReactPolling from "react-polling/lib/ReactPolling";
//https://www.npmjs.com/package/react-polling

function App() {
  const [tableData33, updateTable33] = useState([[]]);
  const [polledData, updatePollData] = useState("");

  const fetchData = () => {
    return fetch("http://localhost:3001/pollServer/");
  }

  const pollingSuccess = (jsonResponse) => {
    const txtres = "Current time at server: " + jsonResponse.time;
    updatePollData(txtres);
    return true;
  }

  const pollingFailure = () => {
    //alert('Polling failed');
    //return true;
  }

  React.useEffect(() => {
    ///See CORS
    /*
    fetch("http://localhost:3001/personQuery/")
      .then((res) => res.json())
```

```
    .then((data) => alert(JSON.stringify(data)))
    .catch((err) => alert(err)
     );
     */
 }, []);

 //handleClick is our event handler for the button click
 const handleClick = (updateMethod) => {
  fetch("http://localhost:3001/tableData33/")
    .then((res) => res.json())
    .then((data) => updateMethod(data.tableData33))
    .catch((err) => alert(err)
     );
 };

 return (
  <div className="App">
    <ReactPolling
    url={'http://localhost:3001/pollServer/'}
    interval= {500} // in milliseconds(ms)
    retryCount={3} // this is optional
    onSuccess = {pollingSuccess}
    onFailure= {pollingFailure}
    promise={fetchData} // custom api calling function that should return a promise
    render={({ startPolling, stopPolling, isPolling }) => {
      return <div>{polledData}<br/><br/></div>;
    }}
   />

    <TableComp td = {tableData33}/>
    <button onClick={() => handleClick(updateTable33)}>Randomize ages</button>
  </div>
 );
}
export default App;
```

The component ReactPolling has been invoked inside App. It has several useful props.

The polling interval is set to 500 milliseconds (for a seconds' clock this is sufficiently fast).

The call back pollingSuccess is where the code goes to handle the response from the server. We have another state variable to set in this function as you can see.

Note: the *return true* at the end of pollingSuccess is important, as it signals to ReactPolling to continue the polling. *return false* (or no return) will stop the polling.

pollingFailure is optional but may be implemented for logging purposes.

fetchData is a callback we're providing to fetch the promise from the server. This uses our end-point /pollServer. This promise returned by fetch is processed by ReactPolling internally. ReactPolling expects the promise to contain JSON, therefore our end point in backend_server must return valid JSON. The JSON has been parsed and converted to a JSON object by ReactComponent and made available as the parameter jsonResponse in pollingSuccess.

We method used by default in ReactPolling is GET. We can use POST to send information from the client to server. The props *method* and *body* of ReactPolling are used for this purpose.

For more detail on the ReactPolling component visit the library's page.

*