جامعة التقنية والعلوم التطبيقية
University of Technology and Applied Sciences

CSSE3101 – ADVANCED WEB TECHNOLOGIES
MERN STACK DEVELOPMENT

Objective:

In this lab activity, you are going learn DELETE and UPDATE operations the document in the MongoDB database.

`PART 1- APPLICATION SETUP for DELETE Operation.`

In the client folder, Create `ManageStudents.js` component by doing the following:

1) Import the following:

```
import React, { useState, useEffect } from "react";
import Axios from "axios";
```

2) Create state variables using `useState` hook.

```
const [listOfStudents, setlistOfStudents] = useState([]);
const [countRecords, setcountRecords] = useState(0);
```

3) Create a `useEffect` hook to accept the response from the server.

```
useEffect(() => {
     Axios.get("http://localhost:3001/getAllStudents")
        .then((response) => {
                setlistOfStudents(response.data.students);
                setcountRecords(response.data.count);
            })
        .catch((err) => {
                console.log(err);
        });
 }, []);
```

4) In the `<tbody>` use the `map()` function to iterate over the response and display the data.

```
listOfStudents.map((s) => {
                            return (
                                <tr>
                                    <td>{s.studId}</td>
                                    <td>{s.studName}</td>
                                    <td>{s.dept}</td>

                                </tr>
                            )
                        }
                        )
```

Display the number of records.

```
<div>
  <h3>Number of Records: {countRecords}</h3>
 </div>
```

5) Add the Update and Delete buttons in the table body section.

```
listOfStudents.map((s) => {
                           return (
                               <tr>
                                   <td>{s.studId}</td>
                                   <td>{s.studName}</td>
                                   <td>{s.dept}</td>
                                   <td>
                                    <button
                                       type= 'button'
                                       className= 'btn btn-info'>
                                     Update
                                    </button>
                                    <button
                                       type= 'button'
                                       className='btn btn-warning'
                                       onClick={()=>deleteStudent(s._id)}>
                                      Delete
                                    </button>

                                 </td>

                               </tr>
                           )
                       }
                       )
```

6) Add the event handler for the Delete button, passing as parameter the student id.

```
                <button
                          type= 'button'
                          className='btn btn-warning'
                          onClick={()=>deleteStudent(s._id)}>
                          Delete
                  </button>
```

7) Create the `deleteStudent` function that sends an Axios request to the server to delete the selected record.

```
 const deleteStudent = (async (id) => {
          Axios.delete(`http://localhost:3001/delete/${id}`)
         .then((response) =>{
             setlistOfStudents(listOfStudents.filter((val) => { return val._id !== id
}));
             setcountRecords(response.data.count);
         });
     });
```

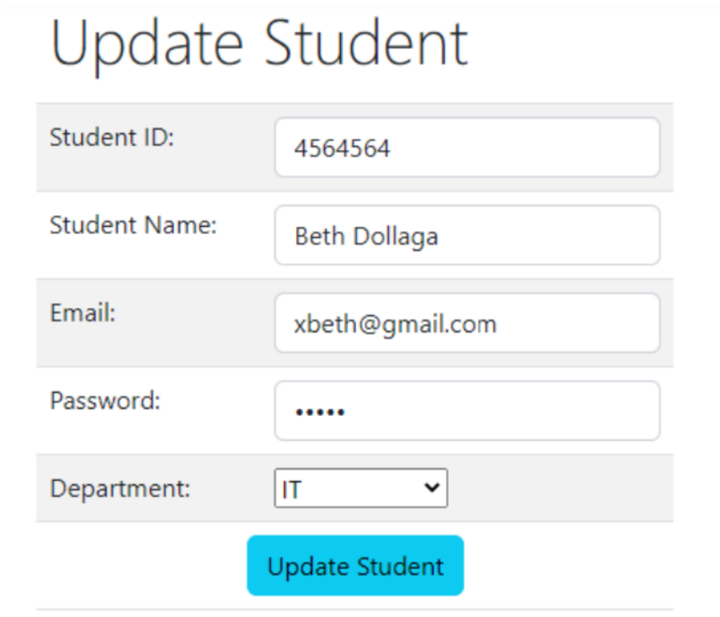
PART 2- Express DELETE Route in index.js

8) In your server folder, update the **index.js** to add a new Express DELETE route to delete the selected record.

```
 app.delete('/delete/:id', async(req,res) =>{
           const id = req.params.id;
           await StudentModel.findByIdAndRemove(id).exec();
           const count = await StudentModel.countDocuments({});
           const msg = 'Item Deleted ';
           res.send({msg,count});
});
```

PART 3- APPLICATION SETUP for UPDATE Operation.

1) Create a new Component UpdateStudent.js which renders the same user interface as StudentRegister. You may just save a copy of the StudentRegister component and do the necessary changes.



Update Student

Student ID:	4564564
Student Name:	Beth Dollaga
Email:	xbeth@gmail.com
Password:	•••••
Department:	IT

2) Update your App.js by adding a new Route for the update function.

```
<Route path="/update/:sid" element={<UpdateStudent />} />
```

3) In the ManageStudent.js component, add this import statement:

```
import {Link } from 'react-router-dom';
```

4) In the ManageStudent.js component, Use the Link component to convert the Update button into a link that will navigate to the update route.

```
<Link to={`/update/${s._id}`}>
   <button type= 'button' className= 'btn btn-info'>Update </button>
</Link>
```

5) Edit `UpdateStudent.js` component to implement the update functionality by doing the following:
   a) Add this import statement.
   ```
   import { useParams } from 'react-router-dom';
   ```
   b) After the state variables, declare a variable to store the parameter from the URL.
   ```
   const [studId, setstudId] = useState("");
   const [studName, setstudName] = useState("");
   const [email, setemail] = useState("");
   const [password, setpassword] = useState("");
   const [dept, setDept] = useState("IT");
   const [responseMsg, setresponseMsg] = useState("");
   let { sid } = useParams();
   ```
   c) Create a `useEffect` hook which will be executed when the page renders or is loaded. The function will send a request to the server, adding in the URL the id of the selected record. When the server sends a response, assign the result to the setter methods of the state variables.
   ```
   useEffect(() => {
        Axios.get(`http://localhost:3001/getStudentForUpdate/${sid}`)
            .then((response) => {
                setstudId(response.data.student.studId);
                setstudName(response.data.student.studName);
                setemail(response.data.student.email);
                setpassword(response.data.student.password);
                setDept(response.data.student.dept);
            })
            .catch((error) => { console.log(error); });
     }
        , []);
   ```
   d) Add the attribute value in the form controls and assign the corresponding state variable. Do it for all the controls.
   ```
       <input
            type="text"
            value={studId }
            className="form-control"
            onChange={(e) => setstudId(e.target.value)}
      />
   ```
6) In the server folder, update `index.js` and create a new Express GET route for the getStudent endpoint. At this point, the values are now displayed in the form controls.
   ```
   app.get("/getStudentForUpdate/:id", async (req, res) => {
       try {
           const id = req.params.id;
           const student = await StudentModel.findById(id);
           const count = await StudentModel.countDocuments()
           res.send({student,count});
           } catch (err) {
           console.error(err);

           }
   });
   ```

7) To save the updates/changes to the database the user will click the Update button. To implement this, do these steps:

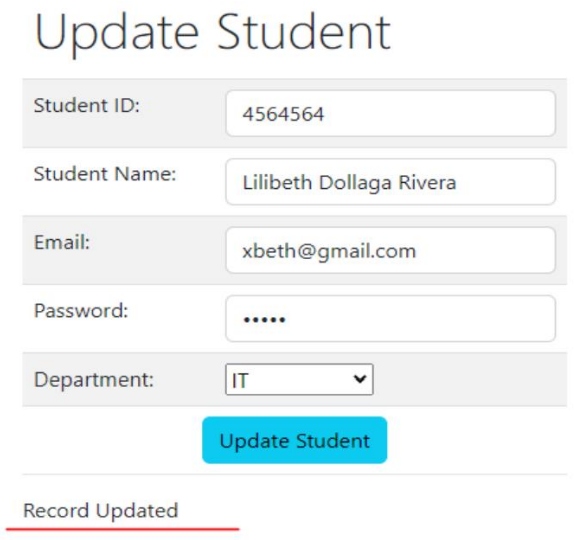
a) Edit `UpdateStudent.js` component, Add the event handler in the Update button to call the function updateStudent.

```
<tr> <td colSpan="2">
      <button className="btn btn-info" onClick={updateStudent}>
           Update Student
      </button>
     </td>
 </tr>
```

b) Edit `UpdateStudent.js` component,

```
const updateStudent = () => {
        Axios.put("http://localhost:3001/updateStudent", {
            studId: studId,
            studName: studName,
            email: email,
            password: password,
            dept: dept
        })
        .then((res) => {
            setresponseMsg(res.data.msg);
        })
      .catch((err) => { console.log(err);});
};
```

c) Display the value of the `responseMsg` state variable at the end of the table.



Update Student

Student ID:	4564564
Student Name:	Lilibeth Dollaga Rivera
Email:	xbeth@gmail.com
Password:	•••••
Department:	IT

Record Updated

d) In the server folder, update index.js and add a new Express PUT route to handle the update endpoint.

```
//Express PUT route to update student documents in database
app.put("/updateStudent", async (req, res) => {
    const studId = req.body.studId;
    try {
        const studentUpdate = await StudentModel.findOne({ studId: studId });
        studentUpdate.studId = String(req.body.studId);
        studentUpdate.studName = String(req.body.studName);
        studentUpdate.email = String(req.body.email);
        studentUpdate.password = String(req.body.password);
```

```
            studentUpdate.dept = String(req.body.dept);
            await studentUpdate.save();
            res.send({ msg: "Record Updated successfully" })
        }
        catch (err) {
            res.send({ error: "Failed to update student" });
        }
    });
```

Required Submission.

Once you complete the lab activity, you are required to upload the database model file `student.js`, `server` folder and `src` folder of the client app.