FSD Laboratory 07

Arin Khopkar, TY B.Tech CSE, Panel I, I2, 62, 1032221073

Aim: Develop a full stack web application using MERN stack to perform CRUD operations.
Objectives:

1. To develop full-stack web projects using the MERN stack.
2. To learn database connectivity using fetch api.
3. To perform insert, update, delete and search operations on database.

Theory:

1. What is MERN stack?

The **MERN stack** is a popular technology stack used for building web applications. It consists of four main components:

1. **MongoDB**: A NoSQL database that stores data in a flexible, JSON-like format. It allows for the storage of large amounts of data and can handle unstructured data effectively.
2. **Express.js**: A web application framework for Node.js that simplifies the process of building server-side applications. It provides a robust set of features for web and mobile applications and allows developers to build APIs easily.
3. **React.js**: A JavaScript library developed by Facebook for building user interfaces, especially single-page applications (SPAs). React allows developers to create reusable UI components and manage the application state efficiently.
4. **Node.js**: A JavaScript runtime that enables the execution of JavaScript code server-side. It allows developers to build scalable network applications and manage server-side logic.

2. Use of Fetch api.

The **Fetch API** is a modern JavaScript API for making HTTP requests. It is built into most modern web browsers and provides a more powerful and flexible feature set compared to the older `XMLHttpRequest` object.

Key Features of the Fetch API

- **Promise-Based**: The Fetch API is promise-based, which means it allows for cleaner and more readable asynchronous code.
- **Simplified Syntax**: The API has a simple and clean syntax, making it easy to understand and use.
- **Support for Various Request Types**: It can handle various HTTP request methods, such as GET, POST, PUT, DELETE, etc.

Here's a simple example that demonstrates how to use the Fetch API to make a GET request to retrieve data from an API:

```
// Making a GET request to fetch data from an API
fetch('https://api.example.com/data')
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok ' + response.statusText);
    }
    return response.json(); // Parsing the JSON response
  })
  .then(data => {
    console.log(data); // Handle the data received
  })
  .catch(error => {
    console.error('There has been a problem with your fetch operation:', error);
  });
```

Here's an example of how to use the Fetch API to send data using a POST request:

```
// Data to be sent in the POST request
const data = {
  name: 'John Doe',
  email: 'john@example.com'
};

// Making a POST request
fetch('https://api.example.com/users', {
  method: 'POST', // Specify the request method
  headers: {
    'Content-Type': 'application/json', // Set the content type to JSON
  },
  body: JSON.stringify(data), // Convert the JavaScript object to a JSON string
})
.then(response => {
  if (!response.ok) {
    throw new Error('Network response was not ok ' + response.statusText);
  }
  return response.json(); // Parse the JSON response
})
.then(data => {
  console.log('Success:', data); // Handle the response data
})
.catch(error => {
  console.error('Error:', error); // Handle errors
});
```

Client Code:

a7 > mern > client > src > JS App.js > ...

```javascript
import React, { useEffect, useState } from 'react';
import './App.css';
import InputField from './InputField';

const FormValidation = () => {
    const [formData, setFormData] = useState({
        username: '',
        email: '',
        phoneNo: '',
        password: '',
        confirmPass: ''
    });
    const [users, setUsers] = useState([]);
    const [editingEmail, setEditingEmail] = useState(''); // State to track email for editing and deleting
    const [editingUserId, setEditingUserId] = useState(null); // State to track the user being edited
    const [viewing, setViewing] = useState(false); // State to track whether to show users

    const handleChange = (e) => {
        const { name, value } = e.target;
        setFormData({
            ...formData, [name]: value
        });
    };

    const [errors, setErrors] = useState({});

    const handleSubmit = async (e) => {
        e.preventDefault();
        const validationErrors = validateForm(formData);
        setErrors(validationErrors);
```

a7 > mern > client > src > JS App.js > ...

```javascript
const FormValidation = () => {
    const handleSubmit = async (e) => {
        setErrors(validationErrors);

        if (Object.keys(validationErrors).length === 0) {
            try {
                const response = await fetch(`http://localhost:5050/record${editingUserId ? `/${editingUserId}` : ''}`, {
                    method: editingUserId ? 'PATCH' : 'POST',
                    headers: {
                        'Content-Type': 'application/json'
                    },
                    body: JSON.stringify(formData)
                });

                if (response.ok) {
                    alert(editingUserId ? "User Updated Successfully" : "Form Submitted Successfully");
                    resetForm(); // Reset form after submission
                    fetchUsers(); // Fetch updated user list
                } else {
                    alert("Failed to submit form");
                }
            } catch (error) {
                alert("An error occurred: " + error.message);
            }
        }
    };

    const resetForm = () => {
        setFormData({
            username: '',
```

(local var) error: any

a7 > mern > client > src > JS App.js > ...

```
5    const FormValidation = () => {
55       const resetForm = () => {
56           setFormData({
57               username: '',
58               email: '',
59               phoneNo: '',
60               password: '',
61               confirmPass: ''
62           });
63           setEditingUserId(null); // Reset editing state
64           setEditingEmail(''); // Clear the email for editing/deleting
65           setViewing(false); // Hide user list when resetting
66       };
67
68       const validateForm = (data) => {
69           const errors = {};
70           if (!data.username.trim()) errors.username = "Username is a required field.";
71           if (!data.email.trim()) {
72               errors.email = "Email is a required field.";
73           } else if (!/^[^\s@]+@[^\s@]{3,}\.[^\s@]{2,3}$/.test(data.email)) {
74               errors.email = "Invalid email entered.";
75           }
76           if (!data.phoneNo.trim()) {
77               errors.phoneNo = "Phone Number is a required field.";
78           } else if (!/^\d{10}$/.test(data.phoneNo)) {
79               errors.phoneNo = "Phone number should be 10 digits.";
80           }
81           if (!data.password.trim()) {
82               errors.password = "Password is a required field.";
83           } else if (!/^(?=.*[A-Z])(?=.*\d)(?=.*[&$#@]).{7,}$/.test(data.password)) {
```

a7 > mern > client > src > JS App.js > ...

```
5    const FormValidation = () => {
68       const validateForm = (data) => {
83           } else if (!/^(?=.*[A-Z])(?=.*\d)(?=.*[&$#@]).{7,}$/.test(data.password)) {
84               errors.password = "Password must be at least 7 characters long and contain at least one capital letter, one digit,
                 and one special character from the set (&,$,#,@).";
85           }
86           if (!data.confirmPass.trim()) {
87               errors.confirmPass = "Confirm Password is a required field.";
88           } else if (data.password !== data.confirmPass) {
89               errors.confirmPass = "Password does not match.";
90           }
91           return errors;
92       };
93
94       const fetchUsers = async () => {
95           try {
96               const response = await fetch('http://localhost:5050/record');
97               if (response.ok) {
98                   const data = await response.json();
99                   setUsers(data);
100              }
101          } catch (error) {
102              alert("Error fetching users: " + error.message);
103          }
104      };
105
106      const handleEdit = async () => {
107          if (!editingEmail) {
108              alert("Please enter an email to edit a user.");
```

a7 > mern > client > src > JS App.js > ...

```
5   const FormValidation = () => {
106     const handleEdit = async () => {
108             alert("Please enter an email to edit a user.");
109             return;
110         }
111         const userToEdit = users.find(user => user.email === editingEmail);
112         if (userToEdit) {
113             setEditingUserId(userToEdit._id);
114             setFormData({
115                 username: userToEdit.username,
116                 email: userToEdit.email,
117                 phoneNo: userToEdit.phoneNo,
118                 password: userToEdit.password, // Set to empty to prevent showing the password
119                 confirmPass: userToEdit.confirmPass // Set to empty to prevent showing the password
120             });
121         } else {
122             alert("User not found.");
123         }
124     };
125
126     const handleDelete = async () => {
127         if (!editingEmail) {
128             alert("Please enter an email to delete a user.");
129             return;
130         }
131         const userToDelete = users.find(user => user.email === editingEmail);
132         if (userToDelete) {
133             if (window.confirm("Are you sure you want to delete this user?")) {
134                 try {
135                     const response = await fetch(`http://localhost:5050/record/${userToDelete._id}`, {
```

a7 > mern > client > src > JS App.js > ...

```
5   const FormValidation = () => {
126     const handleDelete = async () => {
135                     const response = await fetch(`http://localhost:5050/record/${userToDelete._id}`, {
136                         method: 'DELETE',
137                     });
138                     if (response.ok) {
139                         alert("User Deleted Successfully");
140                         fetchUsers(); // Refresh user list
141                     } else {
142                         alert("Failed to delete user");
143                     }
144                 } catch (error) {
145                     alert("An error occurred: " + error.message);
146                 }
147             }
148         } else {
149             alert("User not found.");
150         }
151     };
152
153     const handleView = () => {
154         setViewing(true);
155         fetchUsers(); // Fetch users when viewing
156     };
157
158     useEffect(() => {
159         fetchUsers(); // Fetch users on component mount
160     }, []);
161
162     return (
```

a7 > mern > client > src > JS App.js > ...

```
const FormValidation = () => {
    }, []);

    return (
        <>
            <form onSubmit={handleSubmit}>
                <InputField
                    label="Username"
                    type="text"
                    name="username"
                    value={formData.username}
                    onChange={handleChange}
                    error={errors.username}
                />
                <InputField
                    label="Email"
                    type="email"
                    name="email"
                    value={formData.email}
                    onChange={handleChange}
                    error={errors.email}
                />
                <InputField
                    label="Phone Number"
                    type="text"
                    name="phoneNo"
                    value={formData.phoneNo}
                    onChange={handleChange}
                    error={errors.phoneNo}
                />
```

a7 > mern > client > src > JS App.js > ...

```
5   const FormValidation = () => {
187                 error={errors.phoneNo}
188             />
189             <InputField
190                 label="Password"
191                 type="password"
192                 name="password"
193                 value={formData.password}
194                 onChange={handleChange}
195                 error={errors.password}
196             />
197             <InputField
198                 label="Confirm Password"
199                 type="password"
200                 name="confirmPass"
201                 value={formData.confirmPass}
202                 onChange={handleChange}
203                 error={errors.confirmPass}
204             />
205             <button type="submit">{editingUserId ? "Update" : "Submit"}</button>
206         </form>
207
208         <h2>Email Operations</h2>
209         <InputField
210             label="Email"
211             type="email"
212             value={editingEmail}
213             onChange={(e) => setEditingEmail(e.target.value)}
214             error={null} // No error for this specific field
```

a7 > mern > client > src > JS App.js > ...

```
5   const FormValidation = () => {
209         <InputField
210             label="Email"
211             type="email"
212             value={editingEmail}
213             onChange={(e) => setEditingEmail(e.target.value)}
214             error={null} // No error for this specific field
215         />
216         <button onClick={handleEdit}>Edit</button>
217         <button onClick={handleDelete}>Delete</button>
218         <button onClick={handleView}>View Users</button>
219         <button onClick={resetForm}>Cancel</button> {/* New button to reset form and cancel editing/deleting */}
220
221         {viewing && (
222             <>
223                 <h2>User List</h2>
224                 <ul>
225                     {users.map(user => (
226                         <li key={user._id}>
227                             {user.username} - {user.email}
228                             {/* Password is not displayed */}
229                         </li>
230                     ))}
231                 </ul>
232             </>
233         )}
234     </>
235 );
236 }
237
```

Client CSS:

a7 > mern > client > src > # App.css > ...

```
/* General Styles */
body {
  font-family: Arial, sans-serif;
  background-color: #f4f7f6;
  color: #333;
  margin: 0;
  padding: 0;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
}

/* Form Container */
form {
  background-color: #fff;
  padding: 20px;
  border-radius: 8px;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
  max-width: 400px;
  width: 100%;
}

/* Form Fields */
form div {
  margin-bottom: 15px;
}

label {
  display: block;
```

a7 > mern > client > src > # App.css > ...

```
label {
  font-weight: bold;
  margin-bottom: 5px;
}

input[type="text"],
input[type="email"],
input[type="password"] {
  width: calc(100% - 10px);
  padding: 8px;
  border: 1px solid #ccc;
  border-radius: 4px;
  font-size: 14px;
  box-sizing: border-box;
}

input[type="text"]:focus,
input[type="email"]:focus,
input[type="password"]:focus {
  border-color: #007bff;
  outline: none;
}

/* Error Messages */
span {
  color: #d9534f;
  font-size: 12px;
  margin-top: -10px;
  display: block;
}
```

a7 > mern > client > src > # App.css > ...

```
span {
  margin-top: -10px;
  display: block;
}

/* Submit Button */
button[type="submit"] {
  background-color: #007bff;
  color: #fff;
  padding: 10px 15px;
  border: none;
  border-radius: 4px;
  font-size: 16px;
  cursor: pointer;
  width: 100%;
  box-sizing: border-box;
}

button[type="submit"]:hover {
  background-color: #0056b3;
}

/* Responsive Design */
@media (max-width: 480px) {
  form {
    padding: 15px;
  }

  button[type="submit"] {
    padding: 8px 10px;
```

a7 > mern > client > src > # App.css > ...

```
}

/* Responsive Design */
@media (max-width: 480px) {
  form {
    padding: 15px;
  }

  button[type="submit"] {
    padding: 8px 10px;
    font-size: 14px;
  }

  label {
    font-size: 14px;
  }

  input[type="text"],
  input[type="email"],
  input[type="password"] {
    padding: 6px;
    font-size: 12px;
  }
}
```

Server: Connection.js:

a7 > mern > server > db > JS connection.js > default

```javascript
import { MongoClient, ServerApiVersion } from "mongodb";

const uri = process.env.ATLAS_URI || "";
const client = new MongoClient(uri, {
    serverApi: {
        version: ServerApiVersion.v1,
        strict: true,
        deprecationErrors: true,
},
});

try {
  // Connect the client to the server
    await client.connect();
  // Send a ping to confirm a successful connection
    await client.db("admin").command({ ping: 1 });
    console.log(
        "Pinged your deployment. You successfully connected to MongoDB!"
);
} catch(err) {
    console.error(err);
}

let db = client.db("formDatabase");

export default db;
```

Record.js:

a7 > mern > server > routes > JS record.js > router.post("/") callback > newDocument

```javascript
import express from "express";

// This will help us connect to the database
import db from "../db/connection.js";

// This help convert the id from string to ObjectId for the _id.
import { ObjectId } from "mongodb";

// router is an instance of the express router.
// We use it to define our routes.
// The router will be added as a middleware and will take control of requests starting with path /record.
const router = express.Router();

// This section will help you get a list of all the records.
router.get("/", async (req, res) => {
    let collection = await db.collection("formEntry");
    let results = await collection.find({}).toArray();
    res.send(results).status(200);
});

// This section will help you get a single record by id
router.get("/:id", async (req, res) => {
    let collection = await db.collection("formEntry");
    let query = { _id: new ObjectId(req.params.id) };
    let result = await collection.findOne(query);

    if (!result) res.send("Not found").status(404);
    else res.send(result).status(200);
});
```

a7 > mern > server > routes > JS record.js > ...

```js

// This section will help you create a new record.
router.post("/", async (req, res) => {
    try {
        let newDocument = {
            username: req.body.username,
            email: req.body.email,
            phoneNo: req.body.phoneNo,
            password: req.body.password,
            confirmPass: req.body.confirmPass
        };
        let collection = await db.collection("formEntry");
        let result = await collection.insertOne(newDocument);
        res.send(result).status(204);
        } catch (err) {
        console.error(err);
        res.status(500).send("Error adding record");
    }
});

// This section will help you update a record by id.
router.patch("/:id", async (req, res) => {
    try {
        const query = { _id: new ObjectId(req.params.id) };
        const updates = {
            $set: {
            username: req.body.username,
            email: req.body.email,
            phoneNo: req.body.phoneNo,
            password: req.body.password,
```

a7 > mern > server > routes > JS record.js > router.delete("/:id") callback

```js
router.patch("/:id", async (req, res) => {
        const updates = {
            $set: {
            confirmPass: req.body.confirmPass
            },
        };

        let collection = await db.collection("formEntry");
        let result = await collection.updateOne(query, updates);
        res.send(result).status(200);
    } catch (err) {
    console.error(err);
    res.status(500).send("Error updating record");
    }
});
// This section will help you delete a record
router.delete("/:id", async (req, res) => {
    try {
        const query = { _id: new ObjectId(req.params.id) };

        const collection = db.collection("formEntry");
        let result = await collection.deleteOne(query);

        res.send(result).status(200);
    } catch (err) {
        console.error(err);
        res.status(500).send("Error deleting record");
    }
});
export default router;
```

Config.env file:

```
a7 > mern > server > config.env
ATLAS_URI=mongodb+srv://arinkhopkar:4ZMCI4RymaxdiRCM@cluster0.ftixv.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0
PORT=5050
```

Server.js:

```
a7 > mern > server > JS server.js > ...
import cors from "cors";
import express from "express";
import records from "./routes/record.js";

const PORT = process.env.PORT || 5050;
const app = express();

app.use(cors());
app.use(express.json());
app.use("/record", records);

// start the Express server
app.listen(PORT, () => {
    console.log(`Server listening on port ${PORT}`);
});
```

FAQ:

1. What makes MERN stack the fastest growing tech stack?

The MERN stack has emerged as one of the fastest-growing technology stacks for web development due to several compelling factors. Here's a breakdown of what makes it so appealing and popular among developers:

1. Full-Stack JavaScript

- Unified Language: The MERN stack uses JavaScript for both front-end (React.js) and back-end (Node.js and Express.js) development, simplifying the development process and allowing developers to work across the stack without needing to switch languages. This leads to improved productivity and reduced context switching.

2. Component-Based Architecture

- React's Flexibility: React's component-based architecture allows developers to create reusable UI components. This modularity leads to better organization of code, easier debugging, and enhanced maintainability, which are critical for scaling applications efficiently.

3. Performance and Scalability

- Asynchronous Nature: Node.js operates on a non-blocking, event-driven architecture, which makes it highly efficient and scalable for handling concurrent requests. This is particularly beneficial for applications expecting high traffic.
- MongoDB's NoSQL Structure: MongoDB's schema-less design allows for the flexible storage of data, which can be particularly advantageous for rapidly changing projects or applications that need to scale horizontally.

4. Strong Ecosystem and Community Support

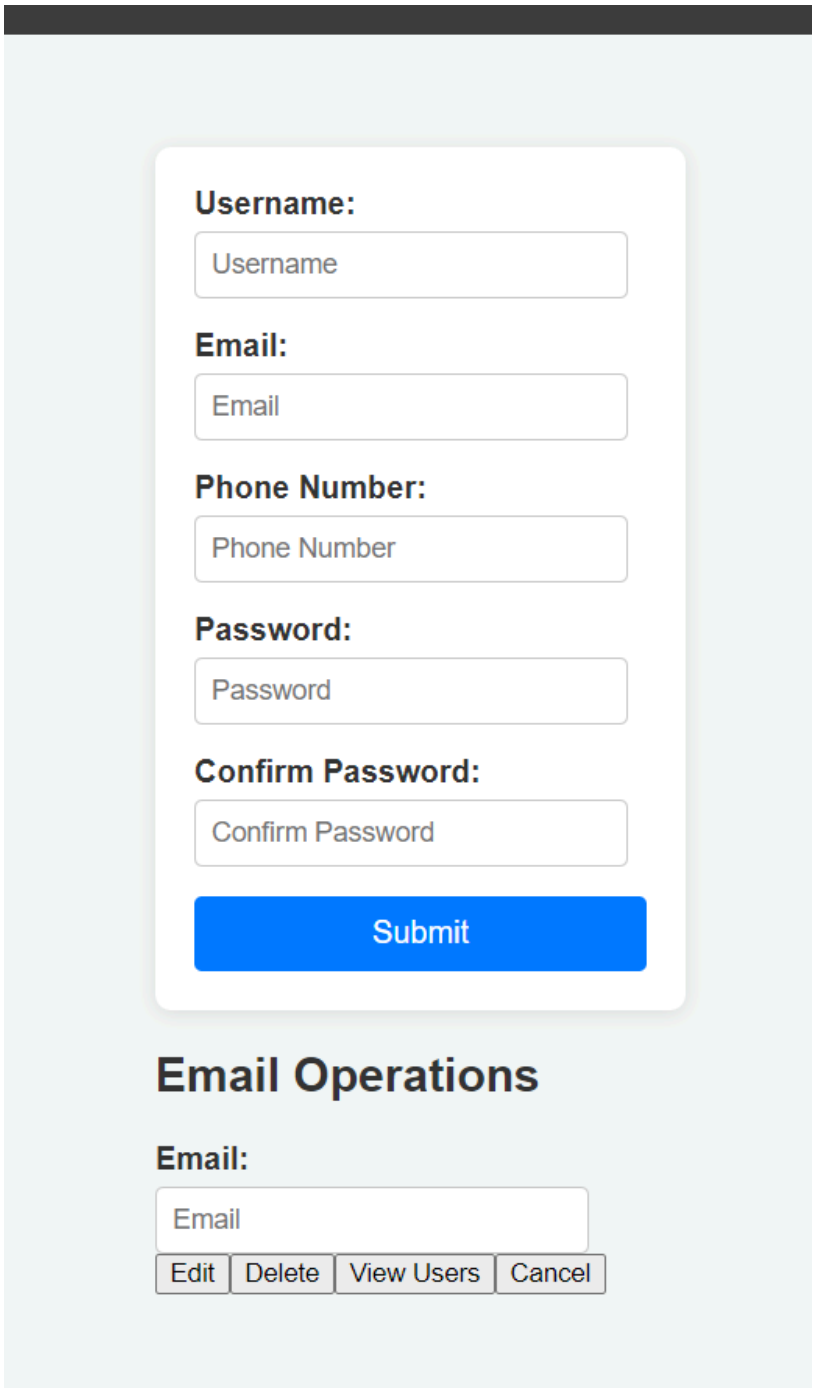
- Active Community: Each component of the MERN stack has a large and active community, contributing to a wealth of resources, libraries, and tools. This community support means developers can find solutions to problems quickly and share knowledge through forums, blogs, and GitHub repositories.
- Rich Libraries and Tools: The availability of numerous libraries (like Redux for state management) enhances functionality and speeds up the development process.

5. Ease of Learning and Adoption

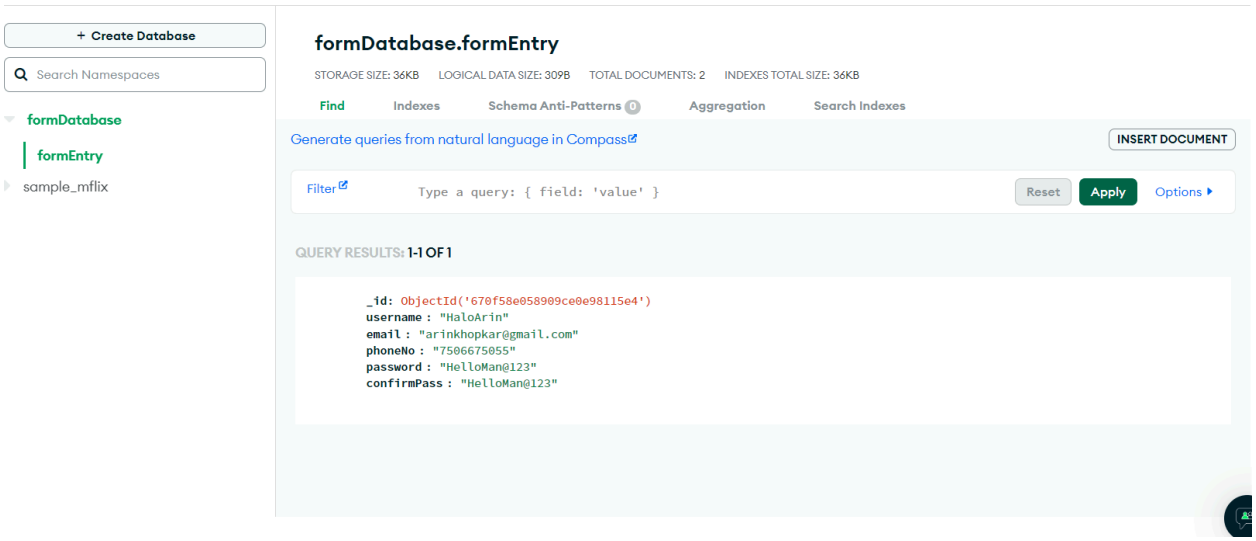
- JavaScript Familiarity: Given that JavaScript is one of the most widely used programming languages, many developers are already familiar with it. This lowers the barrier to entry for new developers wanting to learn the MERN stack.
- Well-Documented Frameworks: Each technology in the MERN stack is well-documented, with extensive tutorials and guides available, making it easier for newcomers to learn and adopt the stack.

Output: Screenshots of the output to be attached.

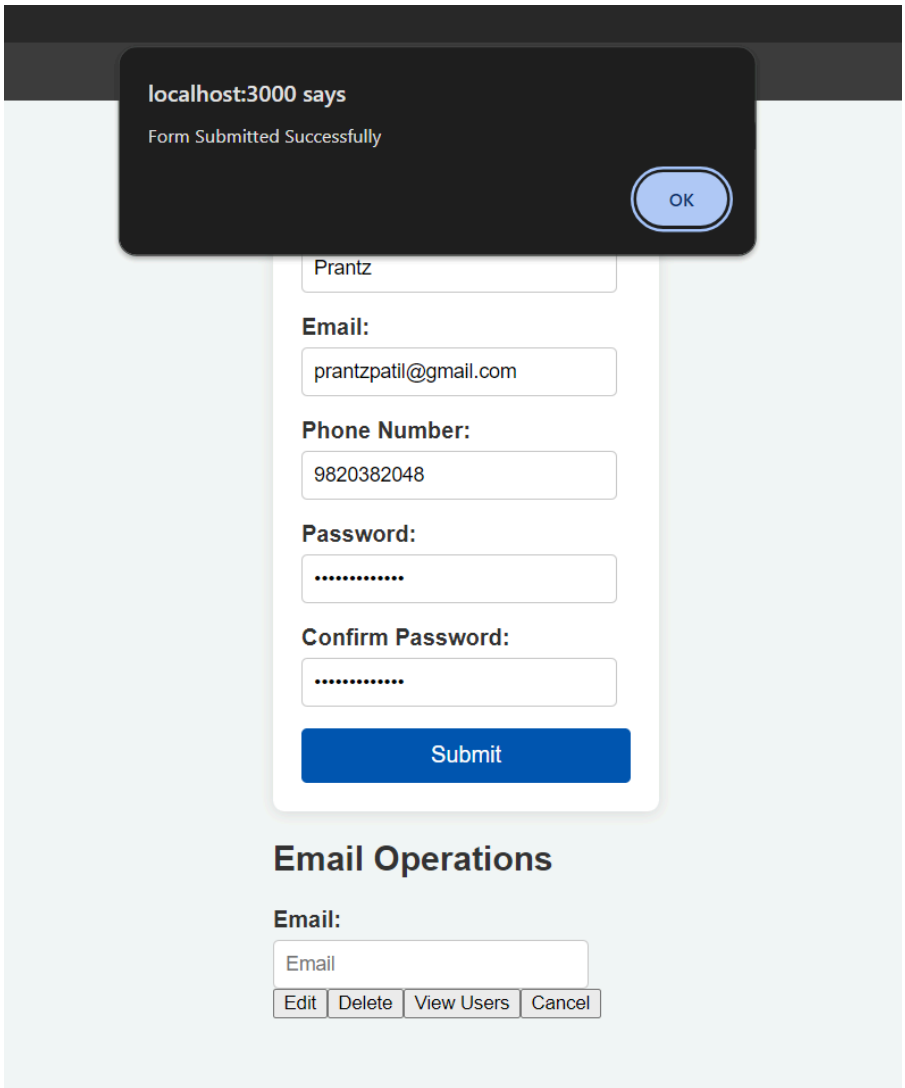
React App:



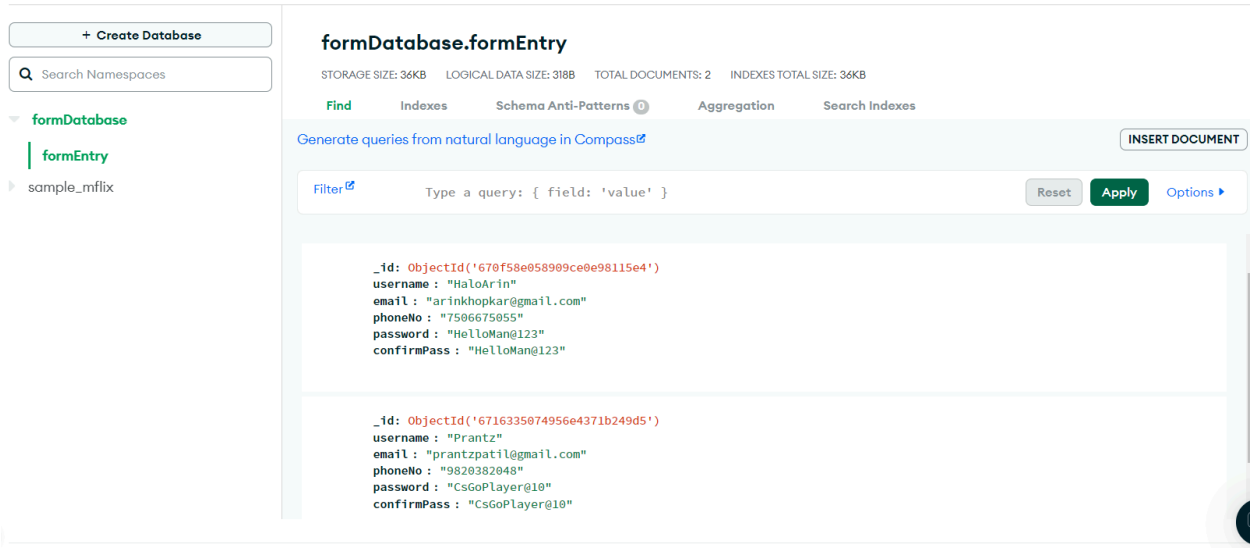
MongoDB:



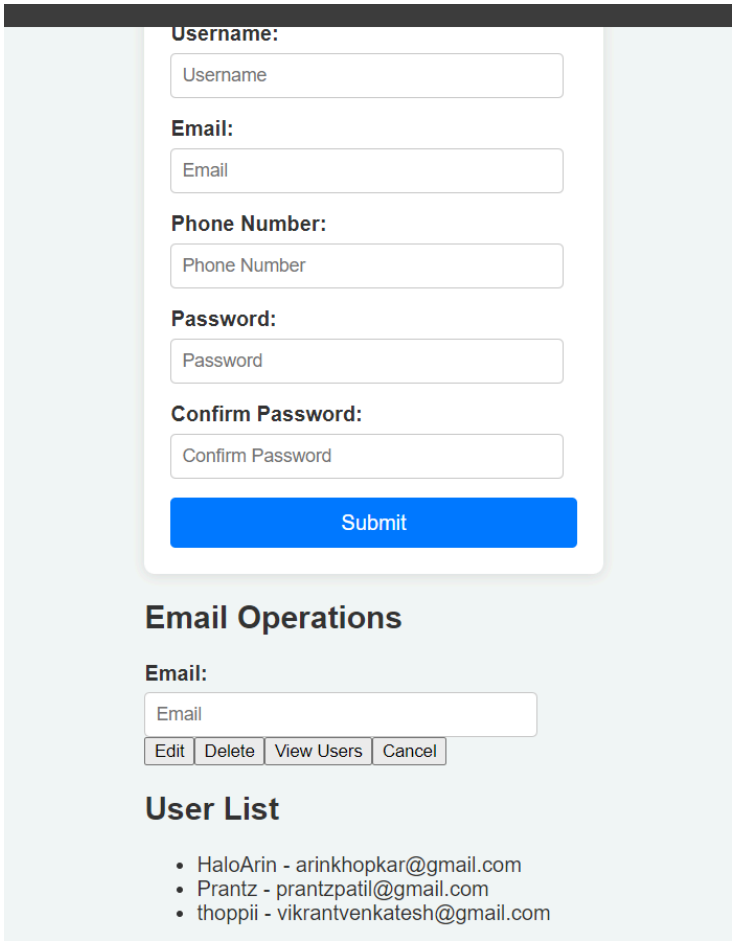
Insertion:



DB After insertion:



Viewing:



QUERY RESULTS: 1-3 OF 3

```
_id: ObjectId('670f58e058909ce0e98115e4')
username : "HaloArin"
email : "arinkhopkar@gmail.com"
phoneNo : "7506675055"
password : "HelloMan@123"
confirmPass : "HelloMan@123"
```

```
_id: ObjectId('6716335074956e4371b249d5')
username : "Prantz"
email : "prantzpatil@gmail.com"
phoneNo : "9820382048"
password : "CsGoPlayer@10"
```

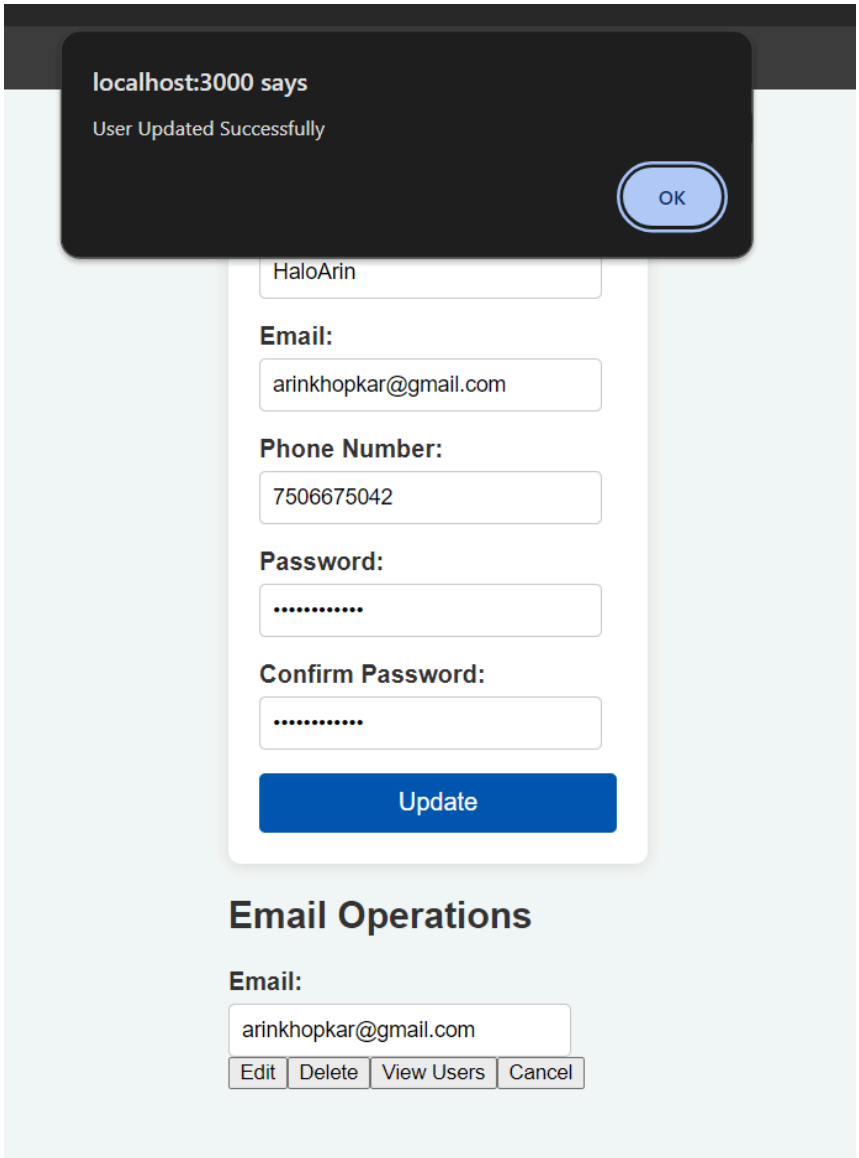
Filter

Type a query: { field: 'value' }

```
email : "prantzpatil@gmail.com"
phoneNo : "9820382048"
password : "CsGoPlayer@10"
confirmPass : "CsGoPlayer@10"
```

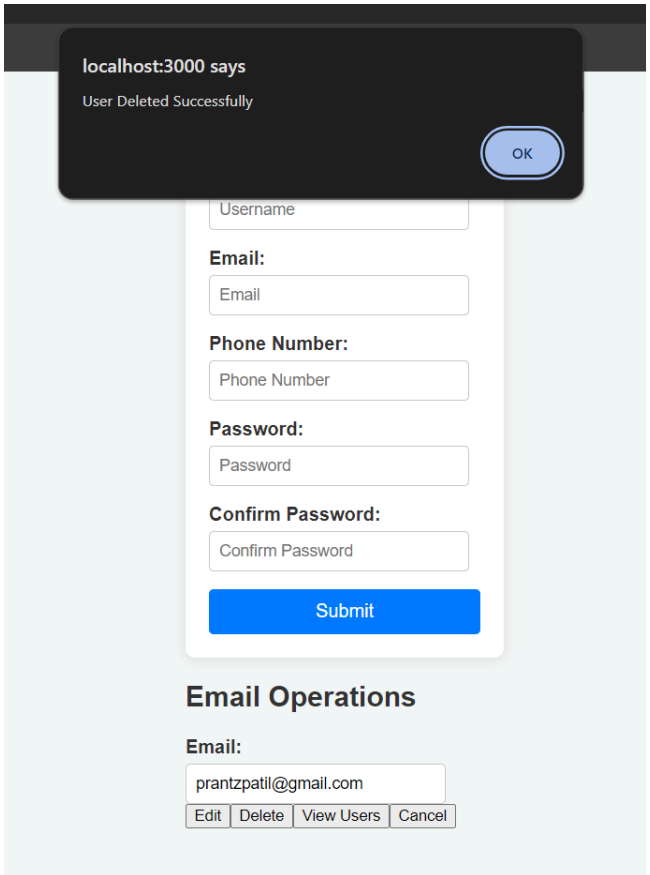
```
_id: ObjectId('671634a674956e4371b249d6')
username : "thoppii"
email : "vikrantvenkatesh@gmail.com"
phoneNo : "9349230422"
password : "BlastFromThePast@3000BC"
confirmPass : "BlastFromThePast@3000BC"
```

Updation:


localhost:3000 says
User Updated Successfully
OK
HaloArin
Email:
arinkhopkar@gmail.com
Phone Number:
7506675042
Password:
Confirm Password:
Update
Email Operations
Email:
arinkhopkar@gmail.com
Edit
Delete
View Users
Cancel


QUERY RESULTS: 1-3 OF 3
_id: ObjectId('670f58e058909ce0e98115e4')
username : "HaloArin"
email : "arinkhopkar@gmail.com"
phoneNo : "7506675042"
password : "HelloMan@123"
confirmPass : "HelloMan@123"

Deletion:



Sample Problem Statements:

CRUD Operations using MERN stack:

1.Student can create a React form or use existing/ implemented HTML form for Student's Registration System with the fields mentioned: First name, Last name, Roll No/ID, Password, Confirm Password,Contact number and perform following operations

1.Insert student details -First name,Last name, Roll No/ID, Password, Confirm Password,Contact number
2.Delete the Student records based on Roll no/ID
3.Update the Student details based on Roll no/ID- Example students can update their contact details based on searching the record with Roll no.
4.Display the Updated student details or View the Students record in tabular format.

2. Student can create a React form or use existing/ implemented HTML form for Library Management System with the fields mentioned: Book name, ISBN No, Book title, Author name, Publisher name and perform following operations

1.Insert Book details -Book name, ISBN No, Book title, Author name, Publisher name
2.Delete the Book records based on ISBN No
3.Update the Book details based on ISBN No- Example students can update wrong entered book details based on searching the record with ISBN No.
4.Display the Updated Book details or View the Book Details records in tabular format.

3. Student can create a React form or use existing/ implemented HTML form for Employee Management System with the fields mentioned: Employee name, Employee ID, Department_name, Phone number, Joining Date and perform following operations

1.Insert Employee details -Employee name, Employee ID, Department_name, Phone number, Joining Date
2.Delete the Employee records based on Employee ID
3.Update the Employee details based on Employee ID- Example students can update Employee details based on searching the record with Employee ID.
4.Display the Updated Employee details or View the Employee Details records in tabular format.

4. Student can create a React form or use existing/ implemented HTML form for Flight Booking Management System with the fields mentioned: Passenger name, From, to, date,Departure date,Arrival date, Phone number , Email ID and perform following operations

1.Insert Passenger details -Passenger name, From, to, date,Departure date,Arrival date, Phone number , Email ID
2.Delete the Passenger records based on Phone Number
3.Update the Passenger details based on Phone Number - Example students can update Flight Booking details based on searching the record with Phone Number.
4.Display the Updated Flight Booking details or View the Flight Booking Details records in tabular format.

Help Link:

https://www.mongodb.com/languages/mern-stack-tutorial