Personal Notes Manager

Frontend (React)

1)SetUp React

- **Create React App**: Start by setting up a new React project using create-react-app

CODE:

```
npx create-react-app personal-notes-manager
cd personal-notes-manager
```

Creating Components

NoteList Component

This component will fetch and display the notes list. It will include options to **edit** and **delete** each note.

CODE:

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


const NoteList = ({ searchQuery, categoryFilter }) => {
const [notes, setNotes] = useState([]);


useEffect(() => {
  axios.get('/api/notes', { params: { search: searchQuery, category: categoryFilter } })
    .then(response => {
      setNotes(response.data);
    })
    .catch(error => console.error('Error fetching notes:', error));
},


[searchQuery, categoryFilter]);


const handleDelete = (id) => {
  axios.delete(`/api/notes/${id}`)
    .then(response => {
      setNotes(notes.filter(note => note.id !== id));
    })
    .catch(error => console.error('Error deleting note:', error));
};


return (
  <div>
    <h2>Notes</h2>
    {notes.map(note => (
      <div key={note.id} className="note-item">
        <h3>{note.title}</h3>
        <p>{note.description}</p>
        <span>{note.category}</span>
        <button onClick={() => handleDelete(note.id)}>Delete</button>
        <button>Edit</button> {/* Implement Edit functionality */}
      </div>
    ))}
  </div>
);
};
```

NoteForm Component

This component allows users to add or edit a note

CODE:

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


const NoteForm = ({ noteToEdit, onSave }) => {
```

```
const [note, setNote] = useState({ title: '', description: '', category: 'Others' });


useEffect(() => {
  if (noteToEdit) {
    setNote(noteToEdit);
  }
}, [noteToEdit]);


const handleSubmit = (e) => {
  e.preventDefault();
  if (noteToEdit) {
    // Edit note
    axios.put(`/api/notes/${note.id}`, note)
      .then(response => onSave())
      .catch(error => console.error('Error updating note:', error));
  } else {
    // Create new note
    axios.post('/api/notes', note)
      .then(response => onSave())
      .catch(error => console.error('Error adding note:', error));
  }
};


return (
  <form onSubmit={handleSubmit}>
    <input type="text" name="title" value={note.title}   onChange={(e) => setNote({ ...note, title: e.target.value })  placeholder="Title"  required/>
    <textare  name="description value={note.description}
      onChange={(e) => setNote({ ...note, description: e.target.value })}
      placeholder="Description" required />
    <select
```

```
      name="category"
      value={note.category}
      onChange={(e) => setNote({ ...note, category: e.target.value })}    >
      <option value="Work">Work</option>
      <option value="Personal">Personal</option>
      <option value="Others">Others</option>
    </select>
    <button type="submit">Save</button>
  </form>
);
};
```

SearchBar Component

This component allows users to filter notes by title or category

```
import React from 'react';
```

CODE:

```
const SearchBar = ({ setSearchQuery, setCategoryFilter }) => {
return (
  <div>
    <input
```

```
      type="text"
      placeholder="Search by title"
      onChange={(e) => setSearchQuery(e.target.value)}
    />
    <select onChange={(e) => setCategoryFilter(e.target.value)}>
      <option value="">All Categories</option>
      <option value="Work">Work</option>
      <option value="Personal">Personal</option>
      <option value="Others">Others</option>
    </select>
  </div>
);
};
```

Styling

For styling, use a simple CSS framework like **TailwindCSS** or **Bootstrap** to ensure responsiveness

COMMEND of VS: npm install tailwindcss

Connecting to Backend

Use axios or the **fetch API** to connect React components to the backend for data fetching and CRUD operations.

COMMEND:npm install axios

Backend (Node.js with Express)

Setup Express App

Create an Express application for the backend

COMMEND:

```
mkdir project---backend
npm init -y
npm install express
```

NEEDED PACKEGS COMMEND:

```
npm i sqlite sqlite3 path bcrypt jsonwebtoken
```

- API Endpoints

Create a new note (POST /notes)

CODE:

```
const express = require('express');
const Note = require('./models/note');
const router = express.Router();


router.post('/notes', async (req, res) => {
const { title, description, category } = req.body;


if (!title || !description) {
  return res.status(400).send('Title and description are required');
}


const note = new Note({
  title,
  description,
  category: category || 'Others',
  created_at: new Date(),
  updated_at: new Date(),
});


try {
  await note.save();
  res.status(201).send(note);
} catch (error) {
  res.status(500).send('Error creating note');
}
});
```

Get all notes (GET /notes)

CODE:

```
router.get('/notes', async (req, res) => {
const { search, category } = req.query;


try {
  const filter = {};
  if (category) filter.category = category;
  if (search) filter.title = { $regex: search, $options: 'i' };


  const notes = await Note.find(filter).sort({ created_at: -1 });
  res.status(200).send(notes);
} catch (error) {
  res.status(500).send('Error fetching notes');
}
});
```

Update a note (PUT /notes/:id)

CODE:

```
router.put('/notes/:id', async (req, res) => {
const { title, description, category } = req.body;


if (!title || !description) {
  return res.status(400).send('Title and description are required');
}


try {
  const updatedNote = await Note.findByIdAndUpdate(
    req.params.id,
    { title, description, category, updated_at: new Date() },
    { new: true }
  );
  if (!updatedNote) {
    return res.status(404).send('Note not found');
  }
  res.status(200).send(updatedNote);
} catch (error) {
  res.status(500).send('Error updating note');
}
});
```

Delete a note (DELETE /notes/:id)

CODE:

```
router.delete('/notes/:id', async (req, res) => {
try {
  const deletedNote = await Note.findByIdAndDelete(req.params.id);
  if (!deletedNote) {
    return res.status(404).send('Note not found');
  }
  res.status(200).send('Note deleted');
} catch (error) {
  res.status(500).send('Error deleting note');
}
});
```

- Note Model (Mongoose Schema)

If using MongoDB, create a note model

```
const mongoose = require('mongoose');


const noteSchema = new mongoose.Schema({
title: { type: String, required: true },
description: { type: String, required: true },
category: { type: String, default: 'Others' },
created_at: { type: Date, default: Date.now },
updated_at: { type: Date, default: Date.now },
});


const Note = mongoose.model('Note', noteSchema);


module.exports = Note;
```

Note Model (Mongoose Schema)

If using MongoDB, create a note model.

CODE:

```
const mongoose = require('mongoose');


const noteSchema = new mongoose.Schema({
title: { type: String, required: true },
description: { type: String, required: true },
category: { type: String, default: 'Others' },
created_at: { type: Date, default: Date.now },
updated_at: { type: Date, default: Date.now },
});
```

```
const Note = mongoose.model('Note', noteSchema);


module.exports = Note;
```

SERVER FILE

COMMEND: FILENAME : SERVER.JS

set up your Express app and MongoDB connection.

CODE:

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const bodyParser = require('body-parser');
const noteRoutes = require('./routes/notes');


const app = express();
```

```
mongoose.connect('mongodb://localhost:27017/notesManager', { useNewUrlParser: true, useUnifiedTopology: true })
.then(() => console.log('MongoDB connected'))
.catch((err) => console.log('MongoDB connection error:', err));


app.use(cors());
app.use(bodyParser.json());
app.use(noteRoutes);


app.listen(5000, () => {
console.log('Server running on http://localhost:5000');
});
```

Database Setup

For **MongoDB**, no further setup is required beyond defining the schema and establishing a connection as shown above.

For **SQLite**, you'll need to install Sequelize or use raw SQL queries

COMMEND: npm install sequelize sqlite3

Testing

- **Backend testing**: Use **Postman** or **Insomnia** to test the backend API.
- **Frontend testing**: Test each feature in your React app, ensuring the notes display correctly, CRUD operations work, and filters function as expected.