NATIONAL UNIVERSITY

NATIONAL UNIVERSITY
COLLEGE OF COMPUTING AND
INFORMATION TECHNOLOGIES

Lab Activity 7

Objectives

- To create and design a server to implement MERN Stack Development
- To revise the code relatively

Materials Needed:

- A computer with Node.js, MongoDB Compass, and React JS installed.
- IDE: VS Code, Github Codespaces, or Code Sandbox
- Optional: Postman [API Testing]

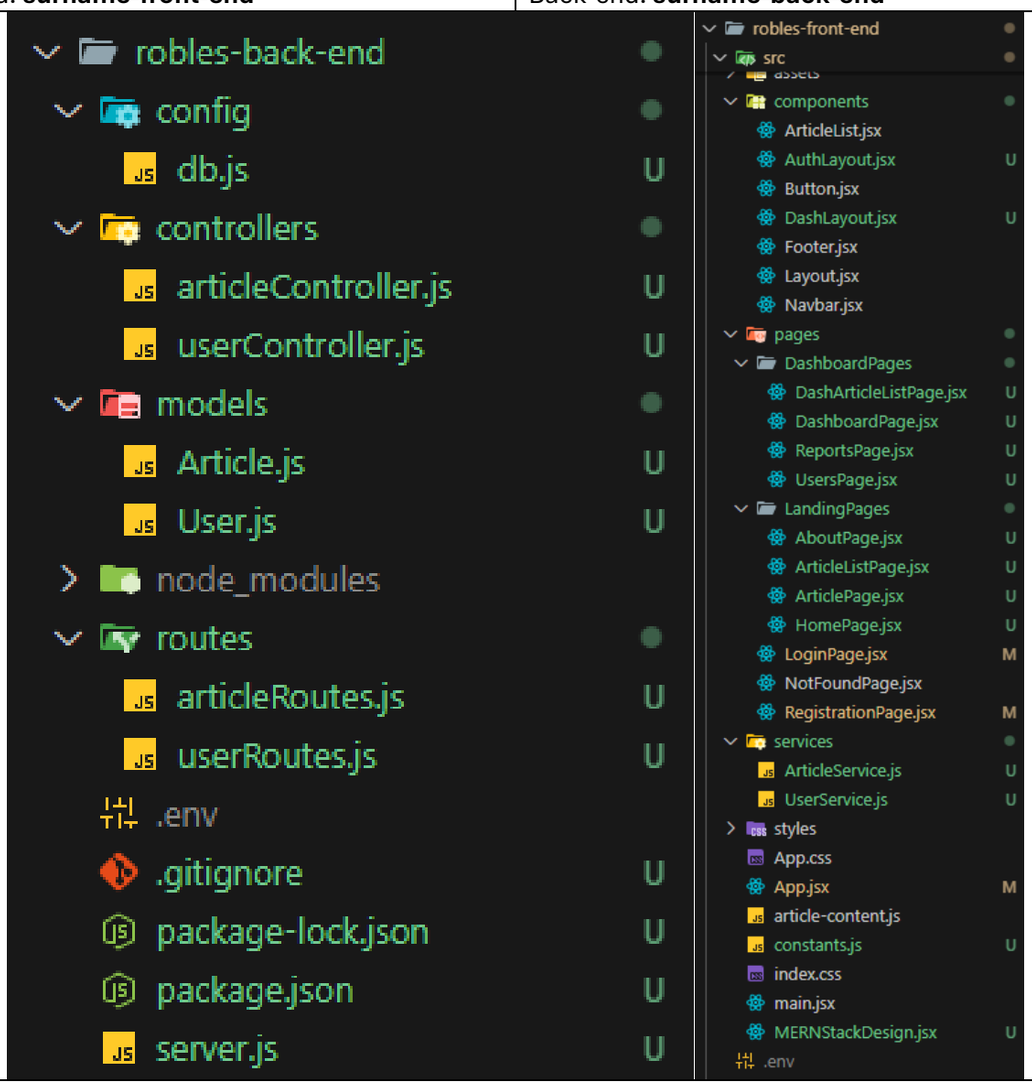
Instructions:

Replicate the code snippet.
Follow and implement the enhancement instructions.
Disregard the **MERNStackDesign.**
Install the following npm packages:

- For backend:
  - bcryptjs
  - cors
  - dotenv
  - express
  - jsonwebtoken
  - mongoose
- For frontend
  - axios
  - dotenv

Web App Initialization:

Sample design pattern [component based]	
Front-end: **surname-front-end**	Back-end: **surname-back-end**



Code Snippets

Back-end

db.js

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

const connectDB = async () => {
    // Connect MongoDB at default port 27017.
    try {
        const conn = await mongoose.connect(process.env.MONGO_URI, {
            useNewUrlParser: true,
            useUnifiedTopology: true,
        });
        console.log(`MongoDB Connected: ${conn.connection.host}`);
    } catch (error) {
        console.error(`Error: ${error.message}`);
        process.exit(1); // Exit process with failure
    }
};

module.exports = connectDB;
```

User.js

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

const userSchema = new mongoose.Schema({
  firstName: { type: String, required: true },
  lastName: { type: String, required: true },
  age: { type: String, required: true },
  gender: { type: String, required: true },
  contactNumber: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  type: { type: String, enum: ['admin', 'editor', 'viewer'], default: 'editor' },
  username: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  address: { type: String, required: true },
  isActive: { type: Boolean, default: true },
});

module.exports = mongoose.model.userSchema || mongoose.model('User', userSchema);
```

User.js

```
const express = require('express');
// import functions
const { getUsers, createUser, updateUser, deleteUser, loginUser, } = require('../controllers/userController');

const router = express.Router();

router.route('/').get(getUsers).post(createUser);

router.route('/:id').put(updateUser).delete(deleteUser);

router.post('/login', loginUser);

module.exports = router;
```

userController.js

```
const User = require('../models/User');
const bcrypt = require('bcryptjs'); // For password hashing
const jwt = require('jsonwebtoken'); // For generating tokens

const getUsers = async (req, res) => {
  try {
    const users = await User.find({}, '-password'); // Exclude the password field
    res.json({ users });
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};

const createUser = async (req, res) => {
  try {
    // Ensure the password is included in the request body
    if (!req.body.password) {
      return res.status(400).json({ message: 'Password is required' });
    }

    // Hash the password
    const hashedPassword = await bcrypt.hash(req.body.password, 10);

    // Create the user with the hashed password
    const user = await User.create({ ...req.body, password: hashedPassword });

    res.status(201).json(user);
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
};

const updateUser = async (req, res) => {
  try {
    // Check if the password is being updated
    if (req.body.password) {
      // Hash the new password
      req.body.password = await bcrypt.hash(req.body.password, 10);
    }

    // Update the user with the new data
    const user = await User.findByIdAndUpdate(req.params.id, req.body, { new: true });

    res.json(user);
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
};

const deleteUser = async (req, res) => {
  try {
    await User.findByIdAndDelete(req.params.id);
    res.json({ message: 'User deleted successfully' });
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
};

const loginUser = async (req, res) => {
  try {
    const { email, password } = req.body;

    // Find the user by email
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    // Check if the user is active
    if (!user.isActive) {
      return res.status(403).json({ message: 'Your account is inactive. Please contact
support.' });
    }

    // Compare the provided password with the hashed password
    const isPasswordValid = await bcrypt.compare(password, user.password);
    if (!isPasswordValid) {
      return res.status(401).json({ message: 'Invalid credentials' });
    }

    // Generate a JWT token
    const token = jwt.sign(
      { id: user._id, email: user.email, type: user.type }, // Include type in the
token
      process.env.JWT_SECRET,
      { expiresIn: '1h' }
    );

    res.json({ message: 'Login successful', token, type: user.type, firstName:
user.firstName }); // Include type in the response
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};

module.exports = { getUsers, createUser, updateUser, deleteUser, loginUser };
```

.env

```
MONGO_URI=mongodb://localhost:27017/nu-class-materials
PORT=5000
# JWT_SECRET=your_secret_key
JWT_SECRET=ZQQVPv4laSLJ1rfj
NODE_ENV=production
```

server.js

```
require('dotenv').config();
const express = require('express');
const cors = require('cors');
const path = require("path");
const bodyParser = require("body-parser");
const jsonParser = bodyParser.json();
const connectDB = require('./config/db');
const userRoutes = require('./routes/userRoutes');
const articleRoutes = require('./routes/articleRoutes');


const app = express();

// Database Connection
connectDB();

app.use(express.json());

//Middleware
app.use(jsonParser);
app.use(bodyParser.urlencoded({ extended: true }));
app.use(cors());

// Curb Cores Error by adding a header here
app.use((req, res, next) => {
    res.setHeader("Access-Control-Allow-Origin", "*");
    res.setHeader(
      "Access-Control-Allow-Headers",
      "Origin, X-Requested-With, Content, Accept, Content-Type, Authorization"
    );
    res.setHeader(
      "Access-Control-Allow-Methods",
      "GET, POST, PUT, DELETE, PATCH, OPTIONS"
    );
    next();
  });

// Routes
app.use('/api/users', userRoutes);
app.use('/api/articles', articleRoutes);

//Getting UI
if (process.env.NODE_ENV === "production") {
    const root = path.join(__dirname, '../robles-front-end/dist');
    app.use(express.static(root));
    app.all('/{*any}', (req, res, next) => {
        res.sendFile(path.join(root, 'index.html'));
    })
    // app.get('*', (req, res) => {
        // res.sendFile(path.join(root, 'index.html'));
    // });
}

// Error Handling
app.use((err, req, res, next) => {
    console.error(err.stack);
    res.status(500).json({ message: 'Server Error' });
});

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

Front-end
.env

```
VITE_LOCAL_HOST=http://localhost:5000
```

UserService.js

```
import axios from 'axios';
import constants from '../constants';

// API Access to Front-end JSON data transformation or decoder
const API = axios.create({
  baseURL: `${constants.HOST}/users`,
});

// Fetch users
export const fetchUsers = (user) => API.get('/', user);

// Create user
export const createUser = (user) => API.post('/', user);

// Update user
export const updateUser = (id, user) => API.put(`/${id}`, user);

// Delete user
export const deleteUser = (id) => API.delete(`/${id}`);

// Login user
export const loginUser = (credentials) => API.post('/login', credentials);
```

Login.js [Please adopt this code with caution. Revise your code relatively**]**

```
import React, { useState } from 'react';
import Button from '../components/Button';
import { Link, useNavigate } from 'react-router-dom';
import { loginUser } from '../services/UserService';

function LoginPage() {
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const [error, setError] = useState('');
    const navigate = useNavigate();

    const handleLogin = async (e) => {
        // navigate('/dashboard');
        e.preventDefault();
        try {
            // Call the login API
            const { data } = await loginUser({ email, password });
            console.log('Login successful:', data);

            localStorage.setItem('token', data.token);
            localStorage.setItem('firstName', data.firstName);
            localStorage.setItem('type', data.type); // user this for dynamic rendering

            // Navigate to the dashboard with the user's email and type
            navigate('/dashboard', { state: { firstName: data.firstName, type: data.type } });
        } catch (err) {
            console.error('Login failed:', err.response?.data?.message || err.message);
            setError(err.response?.data?.message || 'Login failed. Please try again.');
        }
    };

    return (
        <div>
            <h2>Login</h2>
            {error && <p style={{ color: 'red' }}>{error}</p>}
            <form onSubmit={handleLogin}>
                <div>
                    <label htmlFor="email">Email:</label>
                    <input
                        type="email"
                        id="email"
                        value={email}
                        onChange={(e) => setEmail(e.target.value)}
                        required
                    />
                </div>
                <div>
                    <label htmlFor="password">Password:</label>
                    <input
                        type="password"
                        id="password"
                        value={password}
                        onChange={(e) => setPassword(e.target.value)}
                        required
                    />
                </div>
                <Button type="submit">Login</Button>
            </form>
            <Link to="/register">If you do not have an account, register here.</Link>
        </div>
    );
}

export default LoginPage;
```

NATIONAL UNIVERSITY

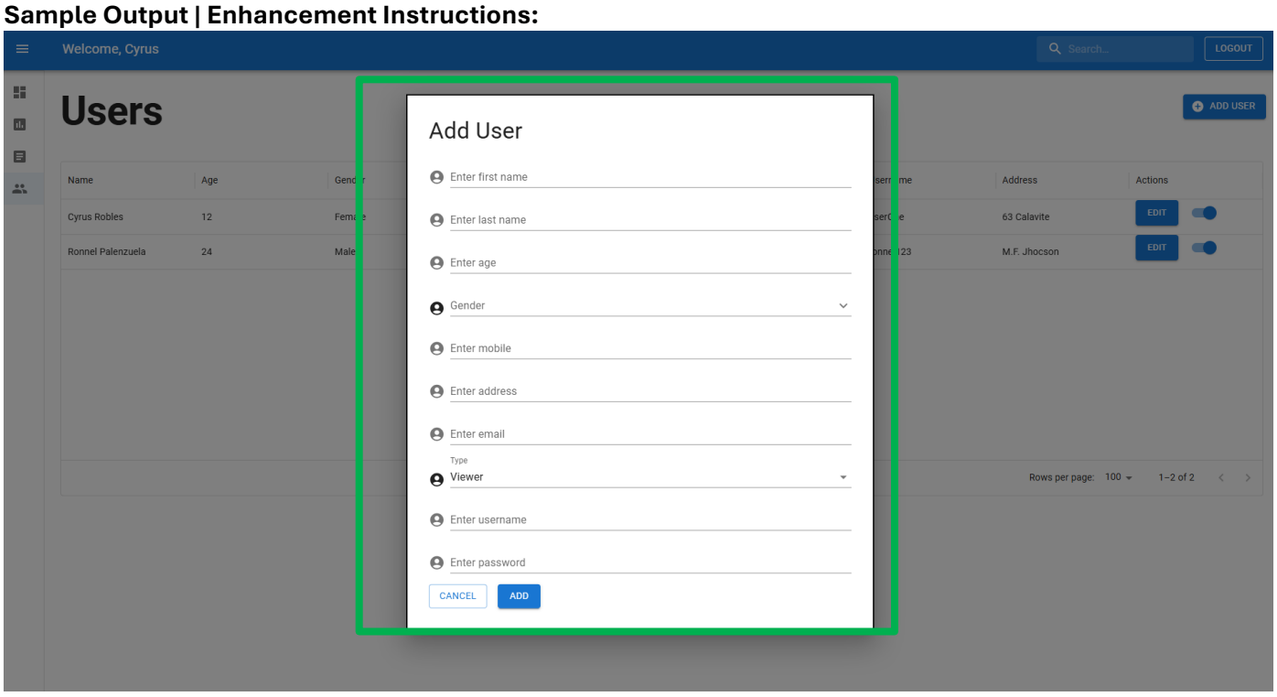
NATIONAL UNIVERSITY
COLLEGE OF COMPUTING AND
INFORMATION TECHNOLOGIES

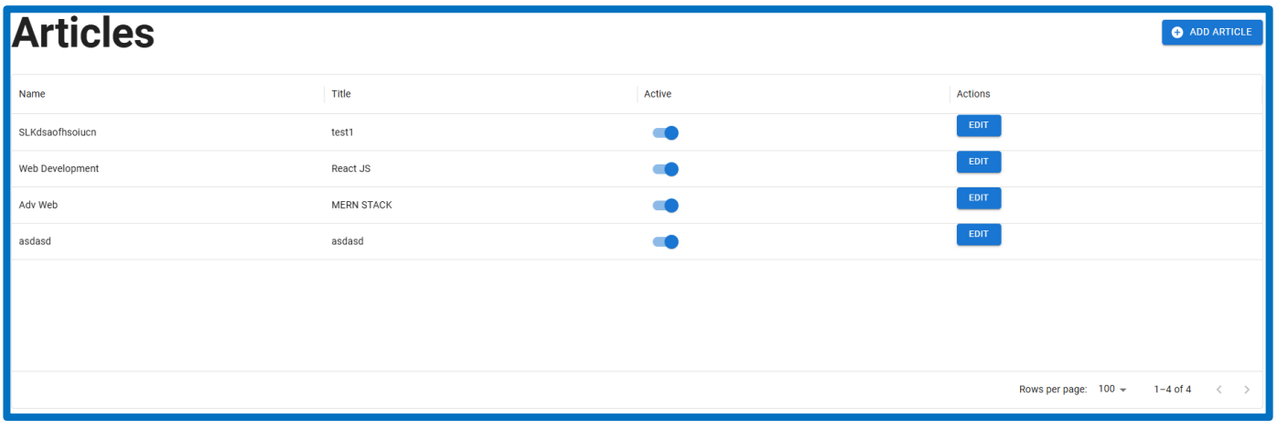
Sample Revision for
UsersPage.jsx [Please adopt this code with caution. Revise your code relatively**]**

```jsx
const modalStyle = {
    position: 'absolute',
    top: '50%',
    left: '50%',
    transform: 'translate(-50%, -50%)',
    width: 700,
    bgcolor: 'background.paper',
    border: '2px solid #000',
    boxShadow: 24,
    p: 4,
};

const UsersPage = () => {
    const [open, setOpen] = useState(false);
    const [isEditing, setIsEditing] = useState(false); // Track if editing
    const [editUserId, setEditUserId] = useState(null); // Track the user being edited
    const [users, setUsers] = useState([]);
    const [loading, setLoading] = useState(true);
    const [newUser, setNewUser] = useState({
        firstName: '',
        lastName: '',
        age: '',
        gender: '',
        contactNumber: '',
        email: '',
        username: '',
        password: '',
        address: '',
        isActive: true,
    });

    const loadUsers = async () => {
        try {
            setLoading(true);
            const { data } = await fetchUsers();
            setUsers(data.users);
        } catch (error) {
            console.error('Error fetching users:', error);
        } finally {
            setLoading(false);
        }
    };

    useEffect(() => {
        loadUsers();
    }, []);

    const handleOpen = () => {
        setIsEditing(false); // Reset to "Add" mode
        setNewUser({
            firstName: '',
            lastName: '',
            age: '',
            gender: '',
            contactNumber: '',
            email: '',
            username: '',
            password: '',
            address: '',
            isActive: true,
        });
        setOpen(true);
    };

    const handleClose = () => {
        setOpen(false);
        setIsEditing(false);
        setEditUserId(null);
    };

    const handleEdit = (id) => {
        const userToEdit = users.find((user) => user._id === id);
        if (userToEdit) {
            setNewUser({ ...userToEdit, password: '' }); // Set password to an empty string
            setEditUserId(id); // Track the user being edited
            setIsEditing(true); // Switch to "Edit" mode
            setOpen(true); // Open the modal
        }
    };

    const handleSaveUser = async () => {
        try {
            if (isEditing) {
                // Update user
                const updatedUser = { ...newUser };
                if (!updatedUser.password) {
                    delete updatedUser.password; // Exclude password if it's empty
                }
                await updateUser(editUserId, updatedUser);
            } else {
                // Add new user
                await createUser(newUser);
            }
            loadUsers(); // Reload users
            handleClose(); // Close modal
        } catch (error) {
            console.error('Error saving user:', error);
        }
    };

    const handleToggleActive = async (id, isActive) => {
        try {
            await updateUser(id, { isActive: !isActive });
            loadUsers(); // Reload users after toggling
        } catch (error) {
            console.error('Error toggling user status:', error);
        }
    };

    const columns1 = [
        {
            field: 'name',
            headerName: 'Name',
            flex: 1,
            valueGetter: (value, params) => `${params.firstName || ''} ${params.lastName || ''}`,
        },
        { field: 'age', headerName: 'Age', flex: 1, sortable: true },
        { field: 'gender', headerName: 'Gender', flex: 1, sortable: true },
        { field: 'email', headerName: 'Email', flex: 1 },
        { field: 'type', headerName: 'Type', flex: 1, sortable: true },
        { field: 'contactNumber', headerName: 'Contact', flex: 1 },
        { field: 'username', headerName: 'Username', flex: 1 },
        { field: 'address', headerName: 'Address', flex: 1 },
        {
            field: 'actions',
            headerName: 'Actions',
            flex: 1,
            renderCell: (params) => (
                <Box sx={{ display: 'flex', gap: 1 }}>
                    <Button
                        variant="contained"
                        size="small"
                        onClick={() => handleEdit(params.row._id)}
                    >
                        Edit
                    </Button>
                    <Switch
                        checked={params.row.isActive}
                        onChange={() => handleToggleActive(params.row._id, params.row.isActive)}
                        color="primary"
                    />
                </Box>
            ),
        },
    ];

    return (
        <>
            <Stack direction="row" sx={{ marginBottom: 5, display: 'flex', justifyContent: 'space-between',
alignItems: 'center' }}>
                <Typography variant="h2" fontWeight="bold">
                    Users
                </Typography>
                <Button
                    variant="contained"
                    color="primary"
                    startIcon={<AddCircleIcon />}
                    onClick={handleOpen}
                    sx={{
                        position: 'fixed',
                        right: '20px',
                        top: '100px',
                        zIndex: 1000,
                    }}
                >
                    Add User
                </Button>
            </Stack>
```

```jsx
            {/* Modal for Add/Edit User */}
            <Modal
                keepMounted
                open={open}
                onClose={handleClose}
                aria-labelledby="add-user-modal"
                aria-describedby="add-user-modal-description"
            >
                <Box sx={modalStyle}>
                    <Typography id="keep-mounted-modal-title" variant="h4" component="h2">
                        {isEditing ? 'Edit User' : 'Add User'}
                    </Typography>
                    <Stack id="transition-modal-description" direction="column" spacing={3} sx={{ mt: 2 }}>
                        <FormControl fullWidth variant="standard"  >
                            <Box sx={{ display: 'flex', alignItems: 'flex-end', mb: 2 }}>
                                <AccountCircle sx={{ color: 'action.active', mr: 1, my: 0.5 }} />
                                <TextField fullWidth id="input-with-sx" label="Enter first name"variant="standard"
                                    value={newUser.firstName}
                                    onChange={(e) =>
                                        setNewUser({ ...newUser, firstName: e.target.value })
                                    }
                                />
                            </Box>
                            <Box sx={{ display: 'flex', alignItems: 'flex-end', mb: 2 }}>
                                <AccountCircle sx={{ color: 'action.active', mr: 1, my: 0.5 }} />
                                <TextField fullWidth id="input-with-sx" label="Enter last name" variant="standard"
                                    value={newUser.lastName}
                                    onChange={(e) =>
                                        setNewUser({ ...newUser, lastName: e.target.value })
                                    }
                                />
                            </Box>
                            <Box sx={{ display: 'flex', alignItems: 'flex-end', mb: 2 }}>
                                <AccountCircle sx={{ color: 'action.active', mr: 1, my: 0.5 }} />
                                <TextField fullWidth id="input-with-sx" label="Enter age" variant="standard"
                                    value={newUser.age}
                                    onChange={(e) =>
                                        setNewUser({ ...newUser, age: e.target.value })
                                    }
                                />
                            </Box>

                            <Stack direction='row' sx={{ display: 'flex', alignItems: 'flex-end', mb: 2 }}>
                                <AccountCircle color={color} sx={{ mr: 1 }} />
                                <FormControl fullWidth variant="standard">
                                    <InputLabel id="demo-simple-select-standard-label">Gender</InputLabel>
                                    <Select
                                        IconComponent={ExpandMoreIcon}
                                        labelId="demo-simple-select-standard-label"
                                        id="demo-simple-select-standard"
                                        value={newUser.gender}
                                        onChange={(e) =>
                                            setNewUser({ ...newUser, gender: e.target.value })
                                        }
                                    >
                                        <MenuItem value="Male">Male</MenuItem>
                                        <MenuItem value="Female">Female</MenuItem>
                                    </Select>
                                </FormControl>
                            </Stack>

                            <Box sx={{ display: 'flex', alignItems: 'flex-end', mb: 2 }}>
                                <AccountCircle sx={{ color: 'action.active', mr: 1, my: 0.5 }} />
                                <TextField fullWidth id="input-with-sx" label="Enter mobile" variant="standard"
                                    value={newUser.contactNumber}
                                    onChange={(e) =>
                                        setNewUser({ ...newUser, contactNumber: e.target.value })
                                    }
                                />
                            </Box>
                            <Box sx={{ display: 'flex', alignItems: 'flex-end', mb: 2 }}>
                                <AccountCircle sx={{ color: 'action.active', mr: 1, my: 0.5 }} />
                                <TextField fullWidth id="input-with-sx" label="Enter address" variant="standard"
                                    value={newUser.address}
                                    onChange={(e) =>
                                        setNewUser({ ...newUser, address: e.target.value })
                                    }
                                />
                            </Box>
                            <Box sx={{ display: 'flex', alignItems: 'flex-end', mb: 2 }}>
                                <AccountCircle sx={{ color: 'action.active', mr: 1, my: 0.5 }} />
                                <TextField fullWidth id="input-with-sx" label="Enter email" variant="standard"
                                    value={newUser.email}
                                    onChange={(e) =>
                                        setNewUser({ ...newUser, email: e.target.value })
                                    }
                                />
                            </Box>
                            <Stack direction='row' sx={{ display: 'flex', alignItems: 'flex-end', mb: 2 }}>
                                <AccountCircle color={color} sx={{ mr: 1 }} />
                                <FormControl fullWidth variant="standard">
                                    <InputLabel id="type-label">Type</InputLabel>
                                    <Select
                                        labelId="type-label"
                                        value={newUser.type || 'viewer'}
                                        onChange={(e) => setNewUser({ ...newUser, type: e.target.value })}
                                    >
                                        <MenuItem value="admin">Admin</MenuItem>
                                        <MenuItem value="editor">Editor</MenuItem>
                                        <MenuItem value="viewer">Viewer</MenuItem>
                                    </Select>
                                </FormControl>
                            </Stack>
                            <Box sx={{ display: 'flex', alignItems: 'flex-end', mb: 2 }}>
                                <AccountCircle sx={{ color: 'action.active', mr: 1, my: 0.5 }} />
                                <TextField fullWidth id="input-with-sx" label="Enter username" variant="standard"
                                    value={newUser.username}
                                    onChange={(e) =>
                                        setNewUser({ ...newUser, username: e.target.value })
                                    }
                                />
                            </Box>
                            <Box sx={{ display: 'flex', alignItems: 'flex-end', mb: 2 }}>
                                <AccountCircle sx={{ color: 'action.active', mr: 1, my: 0.5 }} />
                                <TextField
                                    fullWidth
                                    id="input-with-sx"
                                    label="Enter password"
                                    variant="standard"
                                    type="password"
                                    value={newUser.password}
                                    onChange={(e) =>
                                        setNewUser({ ...newUser, password: e.target.value })
                                    }
                                />
                            </Box>
                        </FormControl>
                    </Stack>
                    <Stack spacing={2} direction="row">
                        <Button variant="outlined" onClick={handleClose}>
                            Cancel
                        </Button>
                        <Button variant="contained" onClick={handleSaveUser}>
                            {isEditing ? 'Save Changes' : 'Add'}
                        </Button>
                    </Stack>
                </Box>
            </Modal>

            <Box sx={{ height: 500, width: '100%', mb: 5 }}>
                <DataGrid
                    rows={users}
                    columns={columns1}
                    getRowId={(row) => row._id}
                    loading={loading}
                    pageSize={10}
                    rowsPerPageOptions={[10, 20, 50]}
                    disableSelectionOnClick
                />
            </Box>

        </>
    )
}

export default UsersPage
```

Sample Output | Enhancement Instructions:





Enhancement 1: Enhance the UI of Modal User. The editors cannot access the UsersPage.
Enhancement 2: Base on UsersPage create a DashArticleListPage with this consideration:

- The articles will be available on ArticleListPage.

NATIONAL UNIVERSITY

NATIONAL UNIVERSITY
COLLEGE OF COMPUTING AND
INFORMATION TECHNOLOGIES

Name	
Section	
Date	

Lab Activity Rubric

Criteria	**Excellent (4 pts)**	**Good (3 pts)**	**Fair (2 pts)**	**Poor (1 pt)**	**Points**
Project Setup & Environment	Project is set up correctly with all dependencies installed. The environment is well-configured and ready for development.	Project is set up with minor issues that do not impact core functionality.	Project setup is attempted but has errors affecting development.	Project setup is incomplete or non-functional.	
Navigation & Routing	Navigation is well-structured, intuitive, and includes a responsive design. Routing is properly implemented with dynamic paths if applicable.	Navigation works correctly but may lack responsiveness or design refinements.	Navigation is functional but may have broken links or a confusing structure.	Navigation is missing or not working.	
Component Structure & Reusability	Components are well-structured, reusable, and follow best practices (e.g., props, state, hooks). Code is modular and scalable.	Components are properly used but may lack optimization or modularity.	Components are used but with poor structure, making the code difficult to scale.	Components are poorly implemented, leading to excessive redundancy or poor organization.	
Styling & UI Design	UI is visually appealing, well-structured, and fully responsive. Consistent styling is applied across the project.	UI is functional and mostly styled well but may have minor inconsistencies or responsiveness issues.	UI is present but lacks proper styling, consistency, or responsiveness.	UI is missing or poorly designed, making navigation difficult.	
Content & Functionality	The app displays dynamic content effectively, and all functionalities work as expected without bugs.	Most functionalities work well, but minor issues exist. Content is displayed correctly.	Some functionalities are incomplete, and content may not load properly.	Major functionalities are missing or broken.	
Code Quality & Best Practices	Code follows best practices, is clean, well-commented, and easy to maintain. Proper use of hooks, state, and props is evident.	Code is well-written but may have minor inefficiencies or inconsistent formatting.	Code runs but is cluttered, inefficient, or difficult to read.	Code is disorganized, hard to understand, and does not follow best practices.	
	Other Comments/ Observations:			**Total Score**	
				Rating: (Total Score/24) * 100	