BÀI TẬP FULLSTACK - QUẢN LÝ NGƯỜI DÙNG

MỤC LỤC

I. TỔNG QUAN BÀI TẬP

1. MỤC TIÊU

Xây dựng ứng dụng web Fullstack quản lý người dùng với đầy đủ chức năng CRUD, tìm kiếm, phân trang và giao diện responsive.

2. KIẾN THỨC ÁP DỤNG

- **Backend**: Node.js, Express.js, MongoDB, Mongoose
- **Frontend**: React (CDN), HTML, CSS
- **Kỹ năng**: REST API, State Management, Pagination, Search

3. YÊU CẦU THỰC HIỆN

- **ĐƯỢC PHÉP**: Hỏi AI về lỗi, khái niệm, giải thích syntax
- **KHÔNG ĐƯỢC**: Yêu cầu AI sinh toàn bộ code, copy project từ internet
- Giảng viên sẽ hỏi lại về code để kiểm tra hiểu biết
- Làm theo cấu trúc dữ liệu, API...theo tài liệu. Giảng viên có thể sử dụng CSDL khác hay trang giao diện khác để thử nghiệm.
- Mã nguồn hướng dẫn trong file cung cấp có tính tham khảo, một số phần có sự giản lược để phù hợp thời gian thực hành. Sinh viên lưu ý các yêu cầu mở rộng ở cuối mỗi phần.

II. YÊU CẦU CHỨC NĂNG CHI TIẾT

1. CẤU TRÚC DỮ LIỆU USER

Trường	Kiểu	Backend Validation	Frontend Validation
name	String	required, minlength: 2	required, trim(), length >= 2
age	Number	required, min: 0	required, >= 0
email	String	required, match: /^\S+@\S+\.\S+$/	required, chứa '@' và '.'
address	String	không bắt buộc	không bắt buộc

Schema MongoDB bắt buộc:

```
const UserSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Tên không được để trống'],
```

```
    minlength: [2, 'Tên phải có ít nhất 2 ký tự']
  },
  age: {
    type: Number,
    required: [true, 'Tuổi không được để trống'],
    min: [0, 'Tuổi phải >= 0']
  },
  email: {
    type: String,
    required: [true, 'Email không được để trống'],
    match: [/^\S+@\S+\.\S+$/, 'Email không hợp lệ']
  },
  address: {
    type: String
  }
});
```

2. BACKEND API ENDPOINTS

2.1. GET - Lấy danh sách (có phân trang + tìm kiếm)

Format: GET /api/users?page=1&limit=5&search=nguyen

Request Query Params:

- page: Số trang (mặc định = 1)
- limit: Số user/trang (mặc định = 5)
- search: Từ khóa tìm kiếm (tùy chọn)

Response Success (200):

```
{
  "page": 1,
  "limit": 5,
  "total": 23,
  "totalPages": 5,
  "data": [
    {
```

```
    "_id": "507f1f77bcf86cd799439011",
    "name": "Nguyễn Văn A",
    "age": 25,
    "email": "nva@example.com",
    "address": "Hà Nội"
  },
  ...
 ]
}
```


Yêu cầu Backend:

- Tìm kiếm theo name, email, address (regex, không phân biệt hoa thường)
- Phân trang
- Trả về cả totalPages để Frontend hiển thị

2.2. POST - Tạo user mới

POST /api/users

Content-Type: application/json


```
{
 "name": "Nguyễn Văn A",
 "age": 25,
 "email": "nva@example.com",
 "address": "Hà Nội"
}
```


Response Success (201):


```
{
 "message": "Tạo người dùng thành công",
 "data": {
   "_id": "507f1f77bcf86cd799439011",
   "name": "Nguyễn Văn A",
   "age": 25,
   "email": "nva@example.com",
   "address": "Hà Nội"
 }
```

}

Response Error (400):

{

"error": "Validation failed: age: Tuổi phải >= 0"

}

2.3. PUT - Cập nhật user

PUT /api/users/:id

Content-Type: application/json

{

"name": "Nguyễn Văn B",

"age": 30,

"email": "nvb@example.com",

"address": "Hà Nội"

}

Response Success (200):

{

"message": "Cập nhật người dùng thành công",

"data": { ... }

}

Response Error (404):

{

"error": "Không tìm thấy người dùng"

}

Lưu ý: Phải dùng { new: true, runValidators: true } trong findByIdAndUpdate() để trả về thông tin người dùng sau cập nhật và có hợp lệ hóa dữ liệu đầu vào.

2.4. DELETE - Xóa user

DELETE /api/users/:id

Response Success (200):

{

"message": "Xóa người dùng thành công"

}

Response Error (404):

{

"error": "Không tìm thấy người dùng"

}

3. FRONTEND REQUIREMENTS

3.1. Giao diện chính - Bảng danh sách

Hiển thị:

- Bảng với các cột: STT | Họ tên | Tuổi | Email | Địa chỉ | Thao tác
- Mỗi dòng có 2 nút: Sửa | Xóa

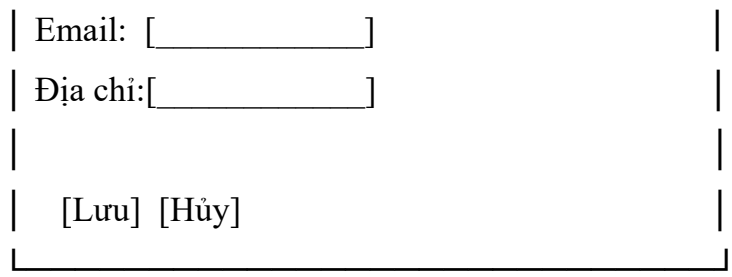
Các thành phần:

```
┌──────────────────────────────────────────────────────────┐
│ [Tìm kiếm theo tên, email, địa chỉ...]                   │
│ [+ Thêm người dùng]                                      │
├──────────────────────────────────────────────────────────┤
│  ┌──────────────────────────────────────────────────┐    │
│  │ STT │ Họ tên  │ Tuổi │ Email │ Thao tác        │    │
│  │  1  │ Nguyễn  │  25  │ nva@  │ Sửa Xóa         │    │
│  └──────────────────────────────────────────────────┘    │
├──────────────────────────────────────────────────────────┤
│ Hiển thị: [5▼] dòng/trang                                │
│ [Prev]  Trang 1/5  [Next ]                               │
└──────────────────────────────────────────────────────────┘
```

3.2. Modal Thêm/Sửa User

Giao diện:

```
┌──────────────────────────────────────┐
│  Thêm người dùng                [X]  │
├──────────────────────────────────────┤
│ Họ tên: [_____________]              │
│ Tuổi:   [_____________]              │
```



Validation Frontend:

- Tên: không rỗng, >= 2 ký tự
- Tuổi: số nguyên >= 0
- Email: có chứa '@' và '.'
- Hiển thị thông báo lỗi màu đỏ dưới input nếu sai

3.3. Chức năng Tìm kiếm

Yêu cầu:

- Input search ở trên cùng
- Tìm kiếm theo Backend (gọi API với query search)
- Tự động gọi API khi input thay đổi (có thể debounce 500ms)
- Kết quả tìm kiếm vẫn có phân trang

3.4. Chức năng Phân trang

Yêu cầu:

- Dropdown chọn số dòng/trang: 3, 5, 10
- Hiển thị: "Trang X/Y"
- Nút Previous: disabled khi ở trang 1
- Nút Next: disabled khi ở trang cuối
- Khi đổi page hoặc limit → gọi lại API

State cần quản lý:

```
const [page, setPage] = useState(1);
const [limit, setLimit] = useState(5);
const [search, setSearch] = useState("");
const [users, setUsers] = useState([]);
const [total, setTotal] = useState(0);
```

```
const [totalPages, setTotalPages] = useState(0);
```

3.5. Responsive Design

Breakpoint: 768px

Desktop (≥768px):

- Bảng đầy đủ các cột
- Modal width: 400px, center

Mobile (<768px):

- Bảng có thanh cuộn ngang (overflow-x: auto)
- Hoặc hiển thị dạng cards (khuyến khích)
- Modal width: 90%, padding nhỏ
- Font size nhỏ hơn
- Nút có kích thước tối thiểu 44x44px (dễ chạm)

CSS mẫu:

```
@media (max-width: 768px) {
  table {
    display: block;
    overflow-x: auto;
    font-size: 12px;
  }
  td, th {
    padding: 8px 4px;
  }
  .modal-content {
    width: 90%;
    padding: 15px;
  }

  button {
    min-width: 44px;
    min-height: 44px;
  }
```

}

III. HƯỚNG DẪN THỰC HIỆN

BƯỚC 1: BACKEND (90 PHÚT)

1.1. Khởi tạo project

mkdir user-management

cd user-management

mkdir backend frontend

cd backend

npm init -y

npm install express mongoose cors

1.2. Tạo file server.js

Cấu trúc cơ bản:

```
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");

const app = express();

// Middleware
app.use(cors());
app.use(express.json());

// Kết nối MongoDB với username là MSSV, password là MSSV, dbname là it4409
mongoose
  .connect("mongodb+srv://username:password@cluster.mongodb.net/dbname")
  .then(() => console.log("Connected to MongoDB"))
  .catch((err) => console.error("MongoDB Error:", err));

// TODO: Tạo Schema
const UserSchema = new mongoose.Schema({ ... });
const User = mongoose.model("User", UserSchema);
```

```
// TODO: Implement API endpoints
app.get("/api/users", async (req, res) => { ... });
app.post("/api/users", async (req, res) => { ... });
app.put("/api/users/:id", async (req, res) => { ... });
app.delete("/api/users/:id", async (req, res) => { ... });

// Start server
app.listen(3001, () => {
  console.log("Server running on http://localhost:3001");
});
```

1.3. Implement GET với Pagination + Search

Format: GET /api/users?page=1&limit=5&search=nguyen

Gợi ý:

```
app.get("/api/users", async (req, res) => {
  try {
    // Lấy query params
    const page = parseInt(req.query.page) || 1;
    const limit = parseInt(req.query.limit) || 5;
    const search = req.query.search || "";

    // Tạo query filter cho search
    const filter = search
      ? {
          $or: [
            { name: { $regex: search, $options: "i" } },
            { email: { $regex: search, $options: "i" } },
            { address: { $regex: search, $options: "i" } }
          ]
        }
      : {};
```

```
    // Tính skip
    const skip = (page - 1) * limit;

    // Query database
    const users = await User.find(filter)
      .skip(skip)
      .limit(limit);

    // Đếm tổng số documents
    const total = await User.countDocuments(filter);
    const totalPages = Math.ceil(total / limit);

    // Trả về response
    res.json({
      page,
      limit,
      total,
      totalPages,
      data: users
    });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

Giải thích:

- $or: Tìm trong nhiều field
- $regex: Pattern matching
- $options: "i": Không phân biệt hoa thường
- skip(): Bỏ qua N documents đầu
- limit(): Chỉ lấy M documents

1.4. Implement POST

Format: POST /api/users

Content-Type: application/json

Gợi ý:

```
app.post("/api/users", async (req, res) => {
  try {
    const { name, age, email, address } = req.body;

    // Tạo user mới
    const newUser = await User.create({ name, age, email, address });

    res.status(201).json({
      message: "Tạo người dùng thành công",
      data: newUser
    });
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});
```

1.5. Implement PUT

Format: PUT /api/users/:id

Content-Type: application/json

Gợi ý:

```
app.put("/api/users/:id", async (req, res) => {
  try {
    const { id } = req.params;
    const { name, age, email, address } = req.body;

    const updatedUser = await User.findByIdAndUpdate(
      id,
      { name, age, email, address },
```

```
      { new: true, runValidators: true } // Quan trọng
    );

    if (!updatedUser) {
      return res.status(404).json({ error: "Không tìm thấy người dùng" });
    }

    res.json({
      message: "Cập nhật người dùng thành công",
      data: updatedUser
    });
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});
```

1.6. Implement DELETE

Format: DELETE /api/users/:id

Gợi ý:

```
app.delete("/api/users/:id", async (req, res) => {
  try {
    const { id } = req.params;

    const deletedUser = await User.findByIdAndDelete(id);

    if (!deletedUser) {
      return res.status(404).json({ error: "Không tìm thấy người dùng" });
    }

    res.json({ message: "Xóa người dùng thành công" });
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
```

});

1.7. Test Backend với Postman

Test cases:

1. GET /api/users → Trả về danh sách
2. GET /api/users?page=2&limit=3 → Phân trang
3. GET /api/users?search=nguyen → Tìm kiếm
4. POST /api/users với data hợp lệ → Status 201
5. POST /api/users với age=-5 → Status 400, có error message
6. PUT /api/users/:id → Cập nhật thành công
7. DELETE /api/users/:id → Xóa thành công

Dữ liệu tối thiểu

- Dữ liệu bản thân ở vị trí đầu tiên (chính xác toàn bộ thông tin)
- Dữ liệu các thành viên trong nhóm BTL (chính xác tên, email)
- Dữ liệu các bạn ngồi xung quanh (chính xác tên)

1.8. Bổ sung thêm

- Giới hạn page/limit, tránh việc FE truyền giá trị page <0, hay limit quá lớn
- Chuẩn hóa giá trị đầu vào (loại bỏ khoảng trắng trong input, Tuổi là số nguyên, email duy nhất, ID hợp lệ trước khi xóa…)
- Khi cập nhật dữ liệu, chỉ cập nhật trường dữ liệu được truyền vào, tránh ghi null cho trường thiếu thông tin
- Sử dụng Promise.all cho truy vấn song song phần Get (find và countDocument)

BƯỚC 2: SETUP FRONTEND (90 PHÚT)

2.1. Tạo file index.html

Cấu trúc HTML cơ bản:

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Quản lý Người dùng</title>

  <!-- React CDN -->
  <script src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>

  <style>
    /* TODO: Thêm CSS */
  </style>
</head>
<body>
  <div id="root"></div>

  <script type="text/babel">
    // TODO: Viết React components
  </script>
</body>
</html>
```

2.2. Component Structure

App

├── SearchBar (input search)

├── AddButton (nút thêm)

├── UserTable

│ └── User Rows (map users)

├── Pagination Controls

└── UserModal (form thêm/sửa)

2.3. Implement Component App

```
function App() {
  // States
  const [users, setUsers] = React.useState([]);
  const [page, setPage] = React.useState(1);
  const [limit, setLimit] = React.useState(5);
  const [search, setSearch] = React.useState("");
  const [total, setTotal] = React.useState(0);
  const [totalPages, setTotalPages] = React.useState(0);
  const [showModal, setShowModal] = React.useState(false);
  const [editingUser, setEditingUser] = React.useState(null);

  // Fetch users
  const fetchUsers = () => {
    const url =
`http://localhost:3001/api/users?page=${page}&limit=${limit}&search=${search}`;

    fetch(url)
      .then(res => res.json())
      .then(data => {
```

```
      setUsers(data.data);
      setTotal(data.total);
      setTotalPages(data.totalPages);
    })
    .catch(err => console.error("Fetch error:", err));
};

// useEffect: Gọi API khi page, limit, search thay đổi
React.useEffect(() => {
  fetchUsers();
}, [page, limit, search]);

return (
  <div className="container">
    <h1>Quản lý Người dùng</h1>

    <SearchBar value={search} onChange={setSearch} />

    <button onClick={() => { setEditingUser(null); setShowModal(true); }}>
      Thêm người dùng
    </button>

    <UserTable
      users={users}
      onEdit={(user) => { setEditingUser(user); setShowModal(true); }}
      onDelete={(id) => deleteUser(id)}
    />

    <Pagination
      page={page}
      totalPages={totalPages}
      limit={limit}
```

```
        onPageChange={setPage}
        onLimitChange={setLimit}
      />

      {showModal && (
        <UserModal
          user={editingUser}
          onClose={() => setShowModal(false)}
          onSave={() => { fetchUsers(); setShowModal(false); }}
        />
      )}
    </div>
  );
}
```

Lưu ý quan trọng:

- **Sau mỗi CRUD phải gọi lại fetchUsers()** để UI cập nhật
- **Không được update state users trực tiếp**, phải fetch lại từ API

2.4. Implement Component SearchBar

```
function SearchBar({ value, onChange }) {
  return (
    <div className="search-bar">
      <input
        type="text"
        placeholder="Tìm theo tên, email, địa chỉ..."
        value={value}
        onChange={(e) => onChange(e.target.value)}
        style={{ width: '100%', padding: '10px', fontSize: '16px' }}
      />
    </div>
  );
}
```

Bonus: Debounce search (tránh gọi API liên tục)

```
const [searchInput, setSearchInput] = React.useState("");

React.useEffect(() => {
  const timer = setTimeout(() => {
    onChange(searchInput);
  }, 200); // Đợi 200ms sau khi user ngừng gõ

  return () => clearTimeout(timer);
}, [searchInput]);
```

Và đổi

onChange={(e) => onChange(e.target.value)} thành

onChange={(e) => setSearchInput(e.target.value)}

2.5. Implement Component UserTable

```
function UserTable({ users, onEdit, onDelete }) {
  const handleDelete = (id, name) => {
    if (window.confirm(`Bạn có chắc muốn xóa "${name}"?`)) {
      onDelete(id);
    }
  };

  return (
    <table>
      <thead>
        <tr>
          <th>STT</th>
          <th>Họ tên</th>
          <th>Tuổi</th>
          <th>Email</th>
          <th>Địa chỉ</th>
```

```
          <th>Thao tác</th>
        </tr>
      </thead>
      <tbody>
        {users.length === 0 ? (
          <tr>
            <td colSpan="6" style={{ textAlign: 'center' }}>
              Không có dữ liệu
            </td>
          </tr>
        ) : (
          users.map((user, index) => (
            <tr key={user._id}>
              <td>{index + 1}</td>
              <td>{user.name}</td>
              <td>{user.age}</td>
              <td>{user.email}</td>
              <td>{user.address || "-"}</td>
              <td>
                <button onClick={() => onEdit(user)}> Sửa</button>
                <button onClick={() => handleDelete(user._id, user.name)}>Xóa
                </button>
              </td>
            </tr>
          ))
        )}
      </tbody>
    </table>
  );
}
```

2.6. Implement Component Pagination

```
function Pagination({ page, totalPages, limit, onPageChange, onLimitChange }) {
  return (
    <div className="pagination">
      <div>
        Hiển thị:
        <select value={limit} onChange={(e) =>
onLimitChange(Number(e.target.value))}>
          <option value="3">3</option>
          <option value="5">5</option>
          <option value="10">10</option>
        </select>
        dòng/trang
      </div>

      <div>
        <button
          onClick={() => onPageChange(page - 1)}
          disabled={page === 1}
        >
          Prev
        </button>

        <span style={{ margin: '0 15px' }}>
          Trang {page}/{totalPages || 1}
        </span>

        <button
          onClick={() => onPageChange(page + 1)}
          disabled={page >= totalPages}
        >
          Next
```

```
        </button>
      </div>
    </div>
  );
}
```

2.7. Implement Component UserModal

```
function UserModal({ user, onClose, onSave }) {
  const [formData, setFormData] = React.useState(
    user || { name: "", age: "", email: "", address: "" }
  );
  const [errors, setErrors] = React.useState({});

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

  const validate = () => {
    const newErrors = {};

    if (!formData.name || formData.name.trim().length < 2) {
      newErrors.name = "Tên phải có ít nhất 2 ký tự";
    }

    if (!formData.age || formData.age < 0) {
      newErrors.age = "Tuổi phải >= 0";
    }

    if (!formData.email || !formData.email.includes("@") ||
!formData.email.includes(".")) {
      newErrors.email = "Email không hợp lệ";
    }

    setErrors(newErrors);
```

```
  return Object.keys(newErrors).length === 0;
};

const handleSubmit = () => {
  if (!validate()) return;

  const url = user
    ? `http://localhost:3001/api/users/${user._id}`
    : `http://localhost:3001/api/users`;

  const method = user ? "PUT" : "POST";

  fetch(url, {
    method,
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(formData)
  })
    .then(res => res.json())
    .then(data => {
      if (data.error) {
        alert("Lỗi: " + data.error);
      } else {
        alert(data.message);
        onSave(); // Gọi lại fetchUsers() và đóng modal
      }
    })
    .catch(err => alert("Lỗi: " + err.message));
};

return (
  <div className="modal-overlay" onClick={onClose}>
    <div className="modal-content" onClick={(e) => e.stopPropagation()}>
```

```
<h3>{user ? "Sửa người dùng" : "Thêm người dùng"}</h3>

<label>Họ tên *</label>
<input
  name="name"
  value={formData.name}
  onChange={handleChange}
/>
{errors.name && <p className="error">{errors.name}</p>}

<label>Tuổi *</label>
<input
  name="age"
  type="number"
  value={formData.age}
  onChange={handleChange}
/>
{errors.age && <p className="error">{errors.age}</p>}

<label>Email *</label>
<input
  name="email"
  type="email"
  value={formData.email}
  onChange={handleChange}
/>
{errors.email && <p className="error">{errors.email}</p>}

<label>Địa chỉ</label>
<input
  name="address"
  value={formData.address}
```

```
          onChange={handleChange}
        />

        <div className="modal-actions">
          <button onClick={handleSubmit}>Lưu</button>
          <button onClick={onClose}>Hủy</button>
        </div>
      </div>
    </div>
  );
}
```

2.8. Implement DELETE function

```
// Trong component App
const deleteUser = (id) => {
  fetch(`http://localhost:3001/api/users/${id}`, {
    method: "DELETE"
  })
    .then(res => res.json())
    .then(data => {
      alert(data.message);
      fetchUsers(); // Phải gọi lại để cập nhật UI
    })
    .catch(err => alert("Lỗi: " + err.message));
};
```

2.9. App Render

Trong script, thực hiện render Component App

```
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<App />);
```

2.10. CSS Styling

```
* {
```

```
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  padding: 20px;
  background: #f5f5f5;
}

.container {
  max-width: 1200px;
  margin: 0 auto;
  background: white;
  padding: 30px;
  border-radius: 8px;
  box-shadow: 0 2px 10px rgba(0,0,0,0.1);
}

h1 {
  margin-bottom: 20px;
  color: #333;
}

/* Search Bar */
.search-bar {
  margin-bottom: 15px;
}

.search-bar input {
  width: 100%;
```

```
  padding: 12px;
  font-size: 16px;
  border: 2px solid #ddd;
  border-radius: 4px;
}

/* Table */
table {
  width: 100%;
  border-collapse: collapse;
  margin: 20px 0;
}

th, td {
  padding: 12px;
  text-align: left;
  border: 1px solid #ddd;
}

th {
  background: #4CAF50;
  color: white;
  font-weight: bold;
}

tr:nth-child(even) {
  background: #f9f9f9;
}

tr:hover {
  background: #f5f5f5;
}
```

```
/* Buttons */
button {
  padding: 8px 16px;
  margin: 0 4px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  font-size: 14px;
}

/* Modal */
.modal-overlay {
  position: fixed;
  top: 0;
  left: 0;
…
}
```

2.11. Yêu cầu bổ sung

- Số thứ tự tăng theo page, thay vì bắt đầu lại ở mỗi page
- Chuẩn hóa dữ liệu đầu vào trước khi gửi Backend (loại bỏ khoảng trắng)
- Reset page về 1 khi limit thay đổi
- Thêm xử lý lỗi trả về chi tiết hơn 400, 500,…