Buổi 4 – Thực hành nhóm: Node.js + React + MongoDB + GitHub

Hoạt động 1: Chuẩn bị môi trường & khởi tạo dự án

Mục tiêu:

- Tạo môi trường làm việc chung và chuẩn bị các công cụ cần thiết.
- Thiết lập repository GitHub để nhóm quản lý code.

Vai trò: Cả nhóm phối hợp

Các bước thực hiện:

1. Cài đặt VS Code, Node.js (LTS), Git.
2. Trưởng nhóm tạo repository GitHub: `group<number>-project`.
3. Trưởng nhóm mời các thành viên vào repository đã tạo.
4. Clone repo về máy: `git clone <URL>`

4. Phân công vai trò:

- Sinh viên 1: Backend (Node.js + Express)
- Sinh viên 2: Frontend (React)
- Sinh viên 3: Database (MongoDB)

5. Tạo branch riêng cho từng thành viên và push:

```
git checkout -b backend
git push origin backend
```

(Tương tự cho frontend & database)

Sản phẩm nộp:

- Ảnh màn hình VS Code
- README_<tên>.md mô tả vai trò từng thành viên

Hoạt động 2: Cài đặt Node.js & cấu trúc backend

Mục tiêu: Thiết lập cấu trúc backend chuẩn, sẵn sàng phát triển API

Vai trò: Sinh viên 1

Các bước thực hiện:

1. Tạo thư mục backend.
2. Khởi tạo Node.js:

```
npm init -y
```

3. Cài đặt thư viện:

```
npm install express nodemon dotenv
```

4. Tạo cấu trúc thư mục:

```
backend/
  server.js
  routes/
  controllers/
  models/
```

5. Viết server.js cơ bản:

```
const express = require('express');
const app = express();
app.use(express.json());
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

Sản phẩm nộp:

- File server.js
- Ảnh chụp cấu trúc thư mục backend

Hoạt động 3: Tạo REST API GET/POST User

Mục tiêu: Tạo API cơ bản quản lý người dùng

Vai trò: Sinh viên 1

Các bước thực hiện:

1. Tạo routes và controller: `routes/user.js`, `controllers/userController.js`.
2. Tạo mảng tạm `users`.
3. Viết API:

```
// GET /users
// POST /users
```

4. Test API bằng Postman/Insomnia.
5. Commit + push branch backend → tạo Pull Request trên GitHub

Sản phẩm nộp:

- File controller + route
- Ảnh test API

- Link PR

Hoạt động 4: Khởi tạo frontend + kết nối API GET/POST

Mục tiêu: Tạo giao diện React hiển thị và thêm user

Vai trò: Sinh viên 2

Các bước thực hiện:

1. Tạo thư mục frontend:

```
npx create-react-app frontend
```

2. Cài axios:

```
npm install axios
```

3. Tạo component: `UserList.jsx`, `AddUser.jsx`.
4. Kết nối API backend:

```
axios.get("http://localhost:3000/users")
axios.post("http://localhost:3000/users", newUser)
```

5. Commit + push branch frontend → tạo PR

Sản phẩm nộp:

- Ảnh giao diện hiển thị danh sách user + form thêm user
- Link PR

Hoạt động 5: Tích hợp MongoDB Atlas

Mục tiêu: Lưu dữ liệu người dùng vào database thực tế

Vai trò: Sinh viên 3

Các bước thực hiện:

1. Tạo tài khoản MongoDB Atlas, cluster.
2. Tạo database `groupDB`, collection `users`.
3. Cài mongoose:

```
npm install mongoose
```

4. Tạo model `User.js` (name, email).
5. Cập nhật server.js để GET/POST dữ liệu từ MongoDB.
6. Commit + push branch database → tạo PR

Sản phẩm nộp:

- File `User.js`
- Ảnh dữ liệu trong MongoDB
- Link PR

Hoạt động 6: Kết nối frontend với MongoDB

Mục tiêu: Frontend hiển thị dữ liệu thực từ database

Vai trò: Sinh viên 2

Các bước thực hiện:

1. Frontend gọi API backend để hiển thị dữ liệu từ MongoDB.
2. Test thêm user từ giao diện React → dữ liệu xuất hiện trên danh sách.
3. Commit + push branch frontend → PR merge vào main

Sản phẩm nộp:

- Ảnh giao diện hiển thị dữ liệu từ MongoDB
- Link PR merge

Hoạt động 7: CRUD nâng cao (PUT/DELETE)

Mục tiêu:
Hoàn thiện backend và frontend với CRUD đầy đủ (GET, POST, PUT, DELETE).

Vai trò:

- Sinh viên 1: Backend (Node.js + Express)
- Sinh viên 2: Frontend (React)

Chi tiết các bước:

Backend (Node.js + Express)

1. **Cập nhật route user.js:**

```
const express = require('express');
const router = express.Router();
const userController = require('../controllers/userController');

router.get('/users', userController.getUsers);
router.post('/users', userController.createUser);
router.put('/users/:id', userController.updateUser);   // PUT
router.delete('/users/:id', userController.deleteUser); // DELETE

module.exports = router;
```

2. **Cập nhật controller userController.js:**

```
let users = []; // mảng tạm nếu chưa dùng MongoDB

// PUT: sửa user
exports.updateUser = (req, res) => {
    const { id } = req.params;
    const index = users.findIndex(u => u.id == id);
    if (index !== -1) {
        users[index] = { ...users[index], ...req.body };
        res.json(users[index]);
    } else {
        res.status(404).json({ message: "User not found" });
    }
};

// DELETE: xóa user
exports.deleteUser = (req, res) => {
    const { id } = req.params;
    users = users.filter(u => u.id != id);
    res.json({ message: "User deleted" });
};
```

3. **Commit & Push nhánh backend:**

```
git add .
git commit -m "Thêm API PUT/DELETE cho User"
git push origin backend
```

Frontend (React)

1. **Thêm nút Sửa/Xóa trong UserList.jsx:**

```
<button onClick={() => handleEdit(user)}>Sửa</button>
<button onClick={() => handleDelete(user.id)}>Xóa</button>
```

2. **Xử lý sự kiện Xóa:**

```
const handleDelete = async (id) => {
  await axios.delete(`http://localhost:3000/users/${id}`);
  setUsers(users.filter(user => user.id !== id));
};
```

3. **Xử lý sự kiện Sửa:**

- Hiển thị form với dữ liệu người dùng đã chọn.
- Gửi PUT request đến backend để cập nhật.

4. **Commit & Push nhánh frontend:**

```
git add .
git commit -m "Thêm chức năng Sửa/Xóa user trên React"
git push origin frontend
```

Sản phẩm nộp:

- Ảnh Postman test API PUT/DELETE.

- Ảnh giao diện React có nút Sửa/Xóa.
- Link PR trên GitHub.

Hoạt động 8: Quản lý state nâng cao & validation

Mục tiêu:
Cải thiện giao diện React, quản lý state tốt hơn và đảm bảo dữ liệu hợp lệ.

Vai trò:

- Sinh viên 2 (Frontend)

Chi tiết các bước:

1. **Sử dụng useState, useEffect, Context (nếu cần):**

```
const [users, setUsers] = useState([]);
useEffect(() => {
  fetchUsers();
}, []);
```

2. **Validation form AddUser.jsx:**

```
const handleSubmit = async (e) => {
    e.preventDefault();
    if (!name.trim()) {
        alert("Name không được để trống");
        return;
    }
    if (!/\S+@\S+\.\S+/.test(email)) {
        alert("Email không hợp lệ");
        return;
    }
    await axios.post("http://localhost:3000/users", { name, email });
    fetchUsers();
};
```

3. **Commit & Push nhánh frontend:**

```
git add .
git commit -m "Thêm validation form và quản lý state nâng cao"
git push origin frontend
```

Sản phẩm nộp:

- Ảnh giao diện có validation.
- Link PR trên GitHub.

Hoạt động 9: Git nâng cao – xử lý xung đột & squash commit

Mục tiêu:
Thực hành Git nâng cao, hợp tác nhóm và chuẩn hóa lịch sử commit.

Vai trò:

- Cả nhóm

Chi tiết các bước:

1. **Merge nhánh khác vào nhánh của mình để tạo xung đột:**

```
git checkout frontend
git merge backend
```

- Nếu cả frontend và backend chỉnh sửa cùng file (ví dụ: App.js), sẽ xảy ra xung đột.

2. **Xử lý xung đột:**

- Mở file báo xung đột, chỉnh sửa để giữ nội dung đúng:

```
<<<<<<< HEAD
// code frontend
=======
/* code backend */
>>>>>>> backend
```

- Sau khi chỉnh sửa:

```
git add App.js
git commit -m "Resolve xung đột merge backend vào frontend"
```

3. **Squash commit trước khi merge vào main:**

```
git rebase -i HEAD~n
# chọn "squash" để gộp commit nhỏ lại thành 1 commit
```

4. **Tạo Pull Request cuối cùng trên GitHub và merge vào main.**

Sản phẩm nộp:

- Ảnh màn hình hiển thị xung đột và cách giải quyết.
- Ảnh màn hình squash commit.
- Link PR merge cuối cùng vào main.

Hoạt động 10: Hoàn thiện dự án & tổng hợp

Mục tiêu: Kiểm tra và hoàn thiện sản phẩm cuối cùng

Vai trò: Cả nhóm

Các bước thực hiện:

1. Merge tất cả branch vào main.
2. Kiểm tra CRUD đầy đủ (GET, POST, PUT, DELETE).
3. Cập nhật README.md: mô tả dự án, công nghệ, hướng dẫn chạy, đóng góp từng thành viên.
4. Push bản cuối cùng lên GitHub

Sản phẩm nộp:

- Ảnh chụp README.md hoàn chỉnh
- Link repo GitHub đầy đủ code