FPT UNIVERSITY

ON-JOB TRAINING PROJECT

Sinh viên ngành Kỹ thuật phần mềm

Report 2 – Tuần 3 và 4

Sinh viên thực hiện: Trần Tấn Phát
Mã số sinh viên: SE183281
Giảng viên hướng dẫn: Ngô Đăng Hà An

– Hồ Chí Minh, Tháng 10 2025 –

Tổng quan lý thuyết

1.Front-end

Trạng thái (State) là gì?

Hãy tưởng tượng trạng thái giống như kho dữ liệu tạm thời của ứng dụng.

Ví dụ: Bạn mở ứng dụng bán hàng, trạng thái có thể chứa danh sách sản phẩm, thông tin người dùng đã đăng nhập, hoặc trạng thái "Đã bấm nút Thêm vào giỏ" chưa.

Tại sao cần quản lý trạng thái?

Ứng dụng càng lớn thì càng có nhiều dữ liệu cần lưu và chia sẻ giữa các phần khác nhau.

Nếu không quản lý tốt, dữ liệu sẽ bị lộn xộn và ứng dụng dễ bị lỗi.

React Hooks - Công cụ cơ bản để quản lý trạng thái

React Hooks là một tính năng được giới thiệu từ React 16.8 để giúp bạn làm việc với state (trạng thái) và lifecycle methods (các phương thức vòng đời) trong functional components (thành phần hàm). Trước đó, bạn chỉ có thể làm điều này với class components.

1. **useState – Quản lý trạng thái đơn giản**
   useState là một Hook dùng để quản lý trạng thái cục bộ (local state) trong một functional component.
   Trạng thái cục bộ là dữ liệu chỉ tồn tại trong thành phần hiện tại và không ảnh hưởng đến các thành phần khác.
   Ví dụ :

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

function Counter() {
  const [count, setCount] = useState(0); // Khai báo một state
"count" với giá trị khởi đầu là 0

  return (
    <div>
      <p>Bạn đã bấm {count} lần</p>
```

```
      <button onClick={() => setCount(count + 1)}>Bấm tôi</button>
    </div>
  );
}
```

2. useEffect – Xử lý logic phụ thuộc vào trạng thái hoặc vòng đời

là Hook dùng để thực hiện các side effects (tác dụng phụ) trong thành phần, ví dụ:

- Gọi API.
- Cập nhật tiêu đề trang.
- Đăng ký sự kiện (event listeners).

Ví dụ :

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

function Timer() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log('Component đã được render');
    document.title = `Bạn đã bấm ${count} lần`;

    // Cleanup function (chạy khi component bị hủy hoặc trước khi
useEffect chạy lại)
    return () => {
      console.log('Cleanup...');
    };
  }, [count]); // Chỉ chạy khi "count" thay đổi

  return (
    <div>
      <p>Bạn đã bấm {count} lần</p>
      <button onClick={() => setCount(count + 1)}>Bấm tôi</button>
    </div>
  );
}
```

3. useContext – Truyền dữ liệu giữa các thành phần

là Hook giúp sử dụng dữ liệu từ một context (ngữ cảnh) mà không cần truyền qua nhiều cấp props.
Ví dụ:

```
import React, { createContext, useContext } from 'react';

const ThemeContext = createContext('light'); // Tạo context với giá
trị mặc định là 'light'

function App() {
  return (
    <ThemeContext.Provider value="dark"> {/* Truyền giá trị 'dark'
*/}
      <Toolbar />
    </ThemeContext.Provider>
  );
}

function Toolbar() {
  return <ThemedButton />;
}

function ThemedButton() {
  const theme = useContext(ThemeContext); // Lấy giá trị từ context
  return <button style={{ background: theme === 'dark' ? '#333' :
'#FFF' }}>Click me</button>;
}
```

4. useReducer – Quản lý trạng thái phức tạp

là một Hook dùng để quản lý trạng thái phức tạp (có nhiều hành động hoặc logic cập nhật phức tạp).
Ví dụ:

```
import React, { useReducer } from 'react';

function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    case 'decrement':
      return { count: state.count - 1 };
    default:
      throw new Error();
  }
}
```

```
        function Counter() {
          const [state, dispatch] = useReducer(reducer, { count: 0 }); // Khai báo
        state và hàm dispatch

          return (
            <div>
              <p>Count: {state.count}</p>
              <button onClick={() => dispatch({ type: 'increment' })}>+</button>
              <button onClick={() => dispatch({ type: 'decrement' })}>-</button>
            </div>
          );
}
```

5. useRef – Truy cập trực tiếp DOM hoặc lưu giá trị không thay đổi

useRef là Hook dùng để:
Truy cập trực tiếp đến phần tử DOM.
Lưu trữ giá trị không bị mất sau mỗi lần render.
Ví dụ:

```jsx
import React, { useRef } from 'react';

function TextInput() {
  const inputRef = useRef(null); // Tạo một ref

  const focusInput = () => {
    inputRef.current.focus(); // Truy cập trực tiếp DOM để focus
  };

  return (
    <div>
      <input ref={inputRef} type="text" />
      <button onClick={focusInput}>Focus Input</button>
    </div>
  );
}
```

6. useMemo – Tối ưu hiệu năng

là Hook dùng để ghi nhớ kết quả tính toán và chỉ tính lại khi giá trị phụ thuộc thay đổi.
Ví dụ:

```jsx
import React, { useState, useMemo } from 'react';

function App() {
  const [count, setCount] = useState(0);
```

```
  const expensiveCalculation = useMemo(() => {
    console.log('Tính toán phức tạp...');
    return count * 2;
  }, [count]); // Chỉ tính lại khi "count" thay đổi

  return (
    <div>
      <p>Kết quả: {expensiveCalculation}</p>
      <button onClick={() => setCount(count + 1)}>Tăng</button>
    </div>
  );
}
```

7. **useCallback – Ghi nhớ hàm**

Cũng giống useMemo, nhưng dùng để ghi nhớ hàm thay vì giá trị.
Ví dụ:

```
import React, { useState, useCallback } from 'react';

function App() {
  const [count, setCount] = useState(0);

  const increment = useCallback(() => {
    console.log('Tăng số...');
    setCount(count + 1);
  }, [count]); // Ghi nhớ hàm, chỉ thay đổi khi "count" thay đổi

  return (
    <div>
      <button onClick={increment}>Tăng</button>
    </div>
  );
}
```

Redux - Công cụ nâng cao để quản lý trạng thái toàn cục

Redux lưu trạng thái ở một nơi duy nhất gọi là store.

Phù hợp khi ứng dụng lớn, nhiều thành phần (components) cần chia sẻ dữ liệu.

Ví dụ: Khi người dùng đăng nhập, trạng thái (user info) sẽ được lưu ở Redux Store để các trang khác (như Trang Hồ sơ hoặc Trang Giỏ hàng) cũng có thể truy cập.

Giao tiếp với API bằng Axios hoặc Fetch

API là gì?

API (Application Programming Interface) là cầu nối giúp Frontend (giao diện người dùng) và Backend (máy chủ) giao tiếp với nhau.

Khi bạn mở một ứng dụng web, API sẽ giúp lấy hoặc gửi dữ liệu từ/đến máy chủ, ví dụ:

Lấy danh sách sản phẩm.

Gửi thông tin đăng nhập.

Fetch API:

Fetch là công cụ có sẵn trong trình duyệt để gửi yêu cầu HTTP.

Ví dụ:

```
fetch('https://api.example.com/products')
  .then(response => response.json()) // Chuyển dữ liệu trả về thành JSON
  .then(data => console.log(data))   // Hiển thị dữ liệu
  .catch(error => console.error(error)); // Xử lý lỗi nếu có
```

Axios:

Axios là thư viện mạnh mẽ hơn Fetch, giúp gửi yêu cầu HTTP dễ dùng hơn.

Ví dụ:

```
import axios from 'axios';
axios.get('https://api.example.com/products')
  .then(response => console.log(response.data)) // Hiển thị dữ liệu
  .catch(error => console.error(error));        // Xử lý lỗi nếu có
```

So sánh giữa Fetch và Axios:

Đặc điểm	Fetch	Axios
Cung cấp sẵn	Có (tích hợp trình duyệt)	Thư viện bên ngoài cần cài đặt
Cú pháp	Phức tạp hơn nếu xử lý lỗi	Dễ dùng, tự động xử lý lỗi
Tính năng nâng cao	Ít tính năng hơn	Có (timeout, cấu hình trước)

Thiết kế giao diện người dùng với Material-UI/Bootstrap

Thiết kế giao diện là gì?

Giao diện người dùng (UI – User Interface) là phần mà người dùng nhìn thấy và tương tác trực tiếp.

Giao diện đẹp, dễ sử dụng sẽ làm người dùng cảm thấy thoải mái khi sử dụng ứng dụng.

Material-UI (MUI):

Material-UI là một thư viện giao diện được thiết kế theo phong cách Material Design của Google.

Đặc điểm:

Thiết kế phẳng, hiện đại.

Cung cấp sẵn các thành phần giao diện như nút bấm, hộp thoại, bảng biểu...

Dễ dàng tùy chỉnh màu sắc, kích thước để phù hợp với nhu cầu.

Bootstrap:

Bootstrap là thư viện CSS phổ biến nhất, giúp tạo giao diện nhanh chóng và dễ dàng.

Đặc điểm:

Cung cấp sẵn các thành phần như nút bấm, lưới (grid system) để tạo bố cục, biểu mẫu...

Hỗ trợ responsive design (hiển thị tốt trên mọi thiết bị: điện thoại, tablet, máy tính).

So sánh Material-UI và Bootstrap:

Đặc điểm	Material-UI	Bootstrap
Phong cách thiết kế	Material Design (Google)	Cổ điển, phổ biến
Linh hoạt	Tích hợp tốt với React	Phù hợp với mọi framework
Tùy chỉnh giao diện	Dễ tùy chỉnh	Dùng nhanh, cần viết thêm CSS nếu tùy chỉnh

Khi xây dựng giao diện, bạn có thể kết hợp cả hai:

Sử dụng Bootstrap để tạo bố cục nhanh.

Sử dụng Material-UI để thiết kế các thành phần với phong cách hiện đại.

2.Back-end

1.Tổng quan về .NET MVC:

.NET MVC (Model-View-Controller) là một kiến trúc phát triển ứng dụng web trong đó:

Model: Quản lý dữ liệu và logic xử lý dữ liệu.

View: Hiển thị dữ liệu (giao diện người dùng).

Controller: Xử lý các yêu cầu từ phía người dùng, tương tác với Model, và trả về View.

Ví dụ: Khi người dùng nhấn nút "Đăng nhập", Controller sẽ nhận yêu cầu, kiểm tra thông tin đăng nhập trong Model, và hiển thị kết quả trên View.

1.REST API – Khái niệm chính

REST (Representational State Transfer):

Là một kiểu kiến trúc thiết kế API, sử dụng giao thức HTTP để giao tiếp giữa client và server.

Nguyên tắc cơ bản:

Stateless (Không trạng thái):

Mỗi yêu cầu độc lập, không lưu trạng thái giữa các lần gọi.

Client-Server:

Client xử lý giao diện, server xử lý dữ liệu.

Cacheable:

Cho phép lưu trữ dữ liệu tạm thời để tăng hiệu suất.

HTTP Methods (Phương thức HTTP):

GET: Lấy dữ liệu (vd: danh sách người dùng).

POST: Tạo mới một tài nguyên.

PUT/PATCH: Cập nhật tài nguyên (toàn bộ hoặc một phần).

DELETE: Xóa tài nguyên.

So sánh ngắn gọn giữa PUT và PATCH:

Đặc điểm	PUT	PATCH
Mục đích	Ghi đè toàn bộ tài nguyên.	Cập nhật một phần tài nguyên.
Cần toàn bộ dữ liệu	Có, tất cả các trường.	Không, chỉ các trường cần thay đổi.
Tác động	Ghi đè hoặc xóa các trường không được gửi.	Giữ nguyên các trường không được gửi.
Hiệu suất	Tốn nhiều băng thông hơn.	Tối ưu hơn khi cập nhật nhỏ.

HTTP Status Codes:

2xx: Thành công (vd: 200 OK, 201 Created).

4xx: Lỗi phía client (vd: 400 Bad Request, 404 Not Found).

5xx: Lỗi phía server (vd: 500 Internal Server Error).

URI (Uniform Resource Identifier):

Mỗi tài nguyên được định danh bằng một URL duy nhất (vd: /api/users).

Biểu diễn tài nguyên:

Dữ liệu thường được trả về dưới dạng JSON hoặc XML.

2. Entity Framework (EF) là gì?

Entity Framework (EF) là một công cụ (framework) của .NET giúp lập trình viên làm việc với cơ sở dữ liệu dễ dàng hơn.

Thay vì phải viết SQL (các câu lệnh để truy vấn và thao tác với cơ sở dữ liệu), bạn chỉ cần làm việc với các đối tượng (objects) trong C#, và EF sẽ tự chuyển đổi chúng thành các câu lệnh SQL.

Hay nói cách khác, EF giúp bạn làm việc với cơ sở dữ liệu mà không cần hiểu sâu về SQL.

3. Migrations (Di chuyển cơ sở dữ liệu)

Migrations là gì?

Migrations trong Entity Framework giúp bạn theo dõi và quản lý các thay đổi trong cấu trúc cơ sở dữ liệu (ví dụ: thêm cột, sửa bảng, v.v.).

Khi bạn thay đổi mô hình dữ liệu trong code (C# Classes), EF sẽ đồng bộ các thay đổi này tới cơ sở dữ liệu thông qua Migration.

Tại sao cần Migrations?

Khi ứng dụng phát triển, sẽ thường xuyên thay đổi cấu trúc cơ sở dữ liệu:

- Thêm bảng mới.
- Thêm/sửa/xóa cột trong bảng.
- Thay đổi mối quan hệ giữa các bảng.

Migrations giúp quản lý những thay đổi này một cách tự động và ghi lại lịch sử các thay đổi.

3. Query LINQ

LINQ là gì?

LINQ (Language Integrated Query) là một cú pháp truy vấn dữ liệu có sẵn trong C#.

LINQ cho phép lấy dữ liệu từ nhiều nguồn khác nhau (cơ sở dữ liệu, danh sách, mảng, v.v.) bằng cú pháp đơn giản và dễ đọc.

LINQ trong Entity Framework:

Khi sử dụng Entity Framework, bạn có thể dùng LINQ để truy vấn dữ liệu từ cơ sở dữ liệu thông qua các lớp DbSet (là đại diện cho các bảng trong cơ sở dữ liệu).

4. Transactions

Transactions là gì?

Transaction là một nhóm các thao tác xử lý dữ liệu (thêm, sửa, xóa) được gộp lại và đảm bảo rằng:

Tất cả các thao tác phải thành công, nếu một bước thất bại thì toàn bộ thay đổi sẽ bị hủy.

Điều này giúp đảm bảo tính nhất quán của cơ sở dữ liệu.

Tại sao cần Transactions?

Trong thực tế, có nhiều trường hợp cần thực hiện nhiều thay đổi liên quan đến nhau. Nếu một thay đổi thất bại, bạn không muốn các thay đổi khác được lưu vào cơ sở dữ liệu.

Ví dụ: Khi xử lý đơn hàng:

Bước 1: Trừ số lượng tồn kho của sản phẩm.

Bước 2: Ghi lại đơn hàng vào bảng Orders.

Bước 3: Trừ tiền trong tài khoản người dùng.

Nếu một trong các bước trên thất bại, tất cả thay đổi phải bị hủy.

Tổng quan bài tập

1.Giới thiệu tổng quan

1.1 Mục tiêu MOOC2

MOOC2 nhằm thực hành kỹ năng phát triển ứng dụng full-stack với 3 thành phần chính:

Frontend (ReactJS): State management, API integration, UI design

Backend (.NET MVC/API): REST API, Entity Framework, CRUD operations

Database (PostgreSQL): Database design, SQL queries

1.2 Hệ thống đã xây dựng

Library Management System - Hệ thống quản lý thư viện với các chức năng:

Quản lý người dùng (Admin/User roles)

Quản lý sách và danh mục

Hệ thống mượn/trả sách

Authentication với JWT

1.3 Công nghệ sử dụng

Frontend: React 19 + Axios + CSS3

Backend: ASP.NET Core 8 + Entity Framework Core + JWT

Database: PostgreSQL

Tools: VS Code + Git + GitHub

1.4 Kiến trúc hệ thống

React App (3000) ←→ ASP.NET Core API (5053) ←→ PostgreSQL Database

1.5 Kết quả đạt được

- ✅ Hoàn thành 100% yêu cầu MOOC2
- ✅ Ứng dụng full-stack hoạt động ổn định
- ✅ 4 tables, 20+ API endpoints, 6 UI pages

2.Database

2.1 Thiết kế và tối ưu cơ sở dữ liệu

Database Schema Design:

Users ──(1:N)──→ Borrows ──(N:1)──→ Books ──(N:1)──→ Categories4 Tables chính:

+Users Table: Quản lý thông tin người dùng hệ thống

+Categories Table: Phân loại sách theo thể loại

+Books Table: Quản lý kho sách trong thư viện

+Borrows Table: Theo dõi việc mượn/trả sách

Tên bảng	Thuộc tính	Mục đích của thuộc tính
users	id	ID duy nhất của người dùng
	username	Tên đăng nhập hệ thống
	email	Email đăng nhập và liên lạc
	password	Mật khẩu đã mã hóa BCrypt
	fullname	Họ tên đầy đủ của người dùng
	role	Phân quyền: admin hoặc user
	isactive	Trạng thái kích hoạt tài khoản

categories	id	ID duy nhất của danh mục
	name	Tên danh mục sách
	description	Mô tả chi tiết về danh mục
books	id	ID duy nhất của sách
	title	Tên sách
	author	Tác giả của sách
	isbn	Mã số sách quốc tế
	publisher	Nhà xuất bản
	totalcopies	Tổng số bản sách có trong thư viện
	availablecopies	Số bản sách còn có thể mượn
	categoryid	Liên kết đến danh mục sách
borrows	id	ID duy nhất của phiếu mượn
	userid	Liên kết đến người mượn sách
	bookid	Liên kết đến sách được mượn
	borrowdate	Ngày mượn sách
	duedate	Hạn trả sách
	returndate	Ngày trả sách thực tế
	status	Trạng thái mượn: borrowed/returned
	notes	Ghi chú về việc mượn sách

2.2 Thực hành các truy vấn SQL

Truy vấn cơ bản (CRUD):

```
-- Thêm người dùng mới
INSERT INTO users (username, email, password, fullname, role)
VALUES ('newuser', 'user@example.com', 'hashedpassword', 'Nguyen Van A', 'user');

-- Lấy danh sách sách còn có thể mượn
SELECT title, author, availablecopies
FROM books WHERE availablecopies > 0;

-- Cập nhật số sách khi mượn
UPDATE books SET availablecopies = availablecopies - 1 WHERE id = 1;
```

Truy vấn kết hợp (JOIN):

```
-- Lấy sách kèm tên danh mục
SELECT b.title, b.author, c.name as category_name
FROM books b
JOIN categories c ON b.categoryid = c.id;

-- Lịch sử mượn sách của người dùng
```

```
SELECT u.fullname, b.title, br.borrowdate, br.status
FROM borrows br
JOIN users u ON br.userid = u.id
JOIN books b ON br.bookid = b.id;
```

Truy vấn thống kê:

```
-- Số sách theo danh mục
SELECT c.name, COUNT(b.id) as so_luong
FROM categories c
LEFT JOIN books b ON c.id = b.categoryid
GROUP BY c.name;

-- Tìm sách quá hạn
SELECT u.fullname, b.title, br.duedate
FROM borrows br
JOIN users u ON br.userid = u.id
JOIN books b ON br.bookid = b.id
WHERE br.status = 'borrowed' AND br.duedate < CURRENT_TIMESTAMP;
```

Back-end

3.1 Xây dựng API với ASP.NET Core

Cấu hình hệ thống: Backend được xây dựng với ASP.NET Core 8.0, sử dụng Entity Framework Core kết nối PostgreSQL. CORS được cấu hình cho phép frontend localhost:3000 truy cập. Swagger được tích hợp để test API.

5 Controllers đã triển khai:

AuthController - Xử lý đăng nhập

UsersController - Quản lý người dùng

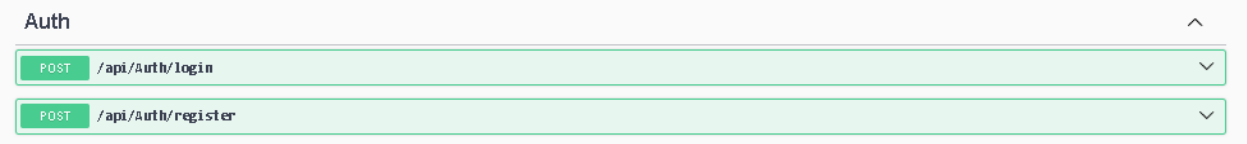
CategoryController - Quản lý danh mục sách

BookController - Quản lý thông tin sách

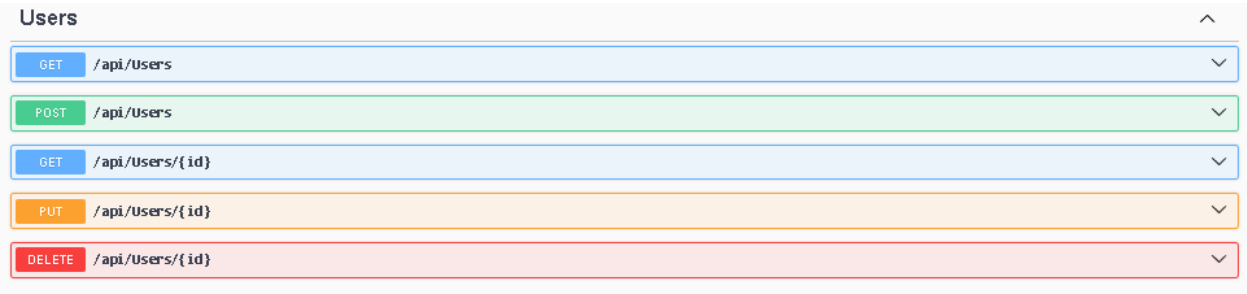
BorrowController - Quản lý mượn trả

API Endpoints đã hoàn thành:

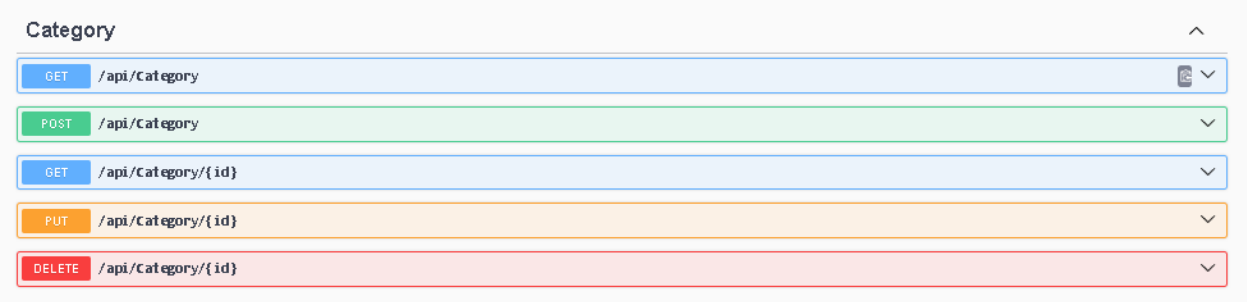
Auth: POST /api/Auth/login



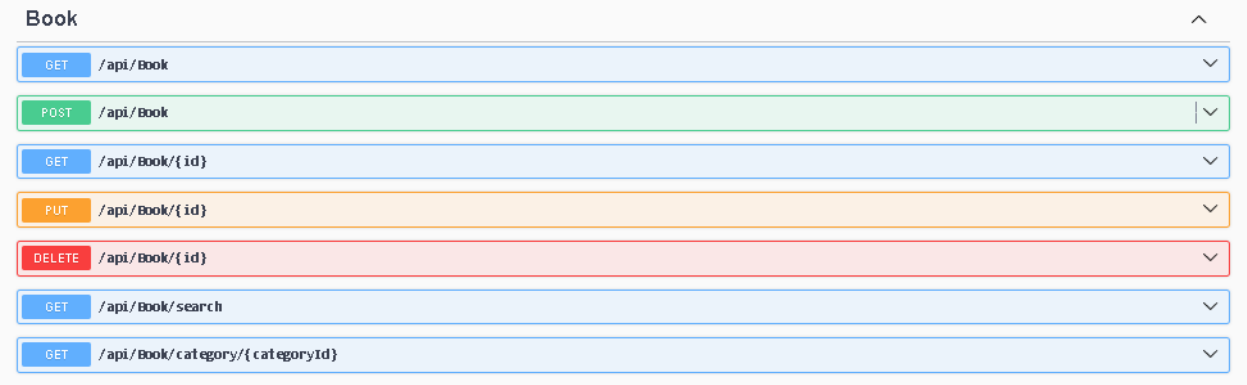
Users: GET, GET/{id}, POST, PUT/{id}, DELETE/{id}



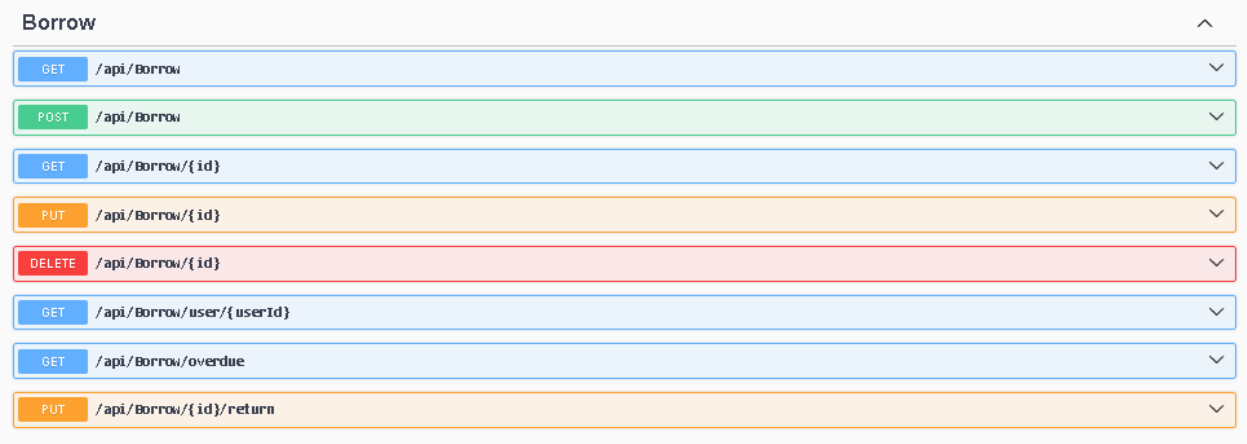
Category: GET, GET/{id}, POST, PUT/{id}, DELETE/{id}



Book: GET, GET/{id}, POST, PUT/{id}, DELETE/{id}



Borrow: GET, GET/{id}, POST, PUT/{id}, DELETE/{id}, PUT/{id}/return



3.2 Entity Framework Core

Database Context: OjtDbContext quản lý 4 DbSet: Users, Categories, Books, Borrows với các navigation properties được cấu hình trong OnModelCreating.

LINQ Queries thực tế:

```
// Lấy categories kèm books
var categories = await _context.Categories
    .Include(c => c.Books)
    .OrderBy(c => c.Name)
    .ToListAsync();

// Lấy books kèm category
var books = await _context.Books
    .Include(b => b.Category)
    .Include(b => b.Borrows)
    .ToListAsync();
```

Migrations: Database được tạo từ migration AddLibraryTables với proper foreign keys và constraints.

3.3 Authentication & Security

BCrypt Password Hashing:

```
// Hash password khi tạo user
string hashedPassword = BCrypt.Net.BCrypt.HashPassword(user.Password);

// Verify password khi login
bool isValid = BCrypt.Net.BCrypt.Verify(request.Password, user.Password);
```

Input Validation: Mỗi controller có validation cho null checks, required fields và business logic validation.

3.4 CRUD Operations thực tế

Users CRUD:

GET /api/Users - Lấy tất cả users

GET /api/Users/{id} - Lấy user theo ID

POST /api/Users - Tạo user mới với BCrypt hashing

PUT /api/Users/{id} - Update user information

DELETE /api/Users/{id} - Xóa user

Error Handling:

400 cho bad request và validation errors

401 cho unauthorized login

404 cho resource not found

200/201 cho successful operations

Front-end

Quản lý trạng thái ứng dụng (State Management với React Hooks)

React Hooks được sử dụng:

useState: Quản lý dữ liệu form đăng nhập, trạng thái loading, và thông báo lỗi trong từng component

useEffect: Tự động khôi phục phiên đăng nhập từ localStorage khi mở lại trang và tải dữ liệu dashboard

useContext: Chia sẻ thông tin người dùng đã đăng nhập cho toàn bộ ứng dụng

useNavigate: Chuyển hướng trang sau khi đăng nhập thành công hoặc khi chưa có quyền truy cập

Context API: Lưu trữ thông tin đăng nhập toàn cục và cung cấp các hàm login/logout cho tất cả components.

Giao tiếp với API bằng Axios

Services Layer: Tách riêng logic gọi API thành các file service riêng biệt để dễ quản lý và tái sử dụng.

Các service đã tạo:

authService: Xử lý đăng nhập và đăng ký người dùng

userService: Quản lý thông tin người dùng (thêm, sửa, xóa, xem)

categoryService: Quản lý danh mục sách

bookService: Quản lý thông tin sách và liên kết với danh mục

borrowService: Xử lý mượn sách và trả sách

Xử lý lỗi: Hiển thị thông báo lỗi bằng tiếng Việt khi API trả về lỗi hoặc mất kết nối.

Thiết kế giao diện người dùng với CSS3

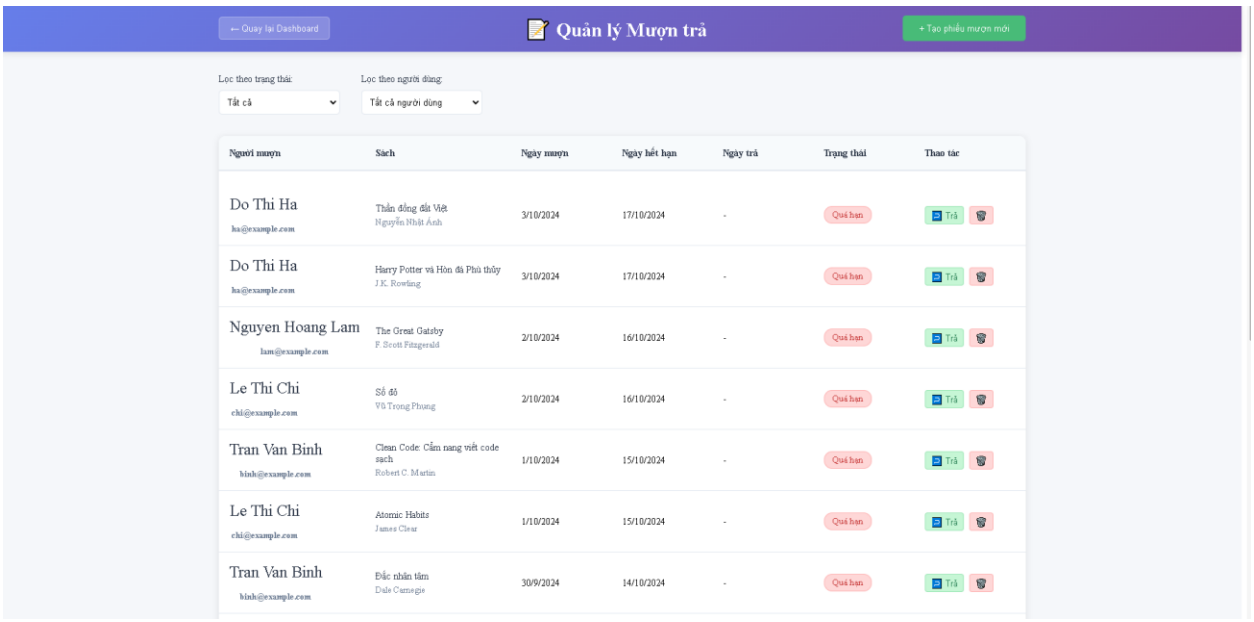
CSS tùy chỉnh: Sử dụng CSS3 thay vì framework để có thiết kế độc đáo và tải trang nhanh hơn.

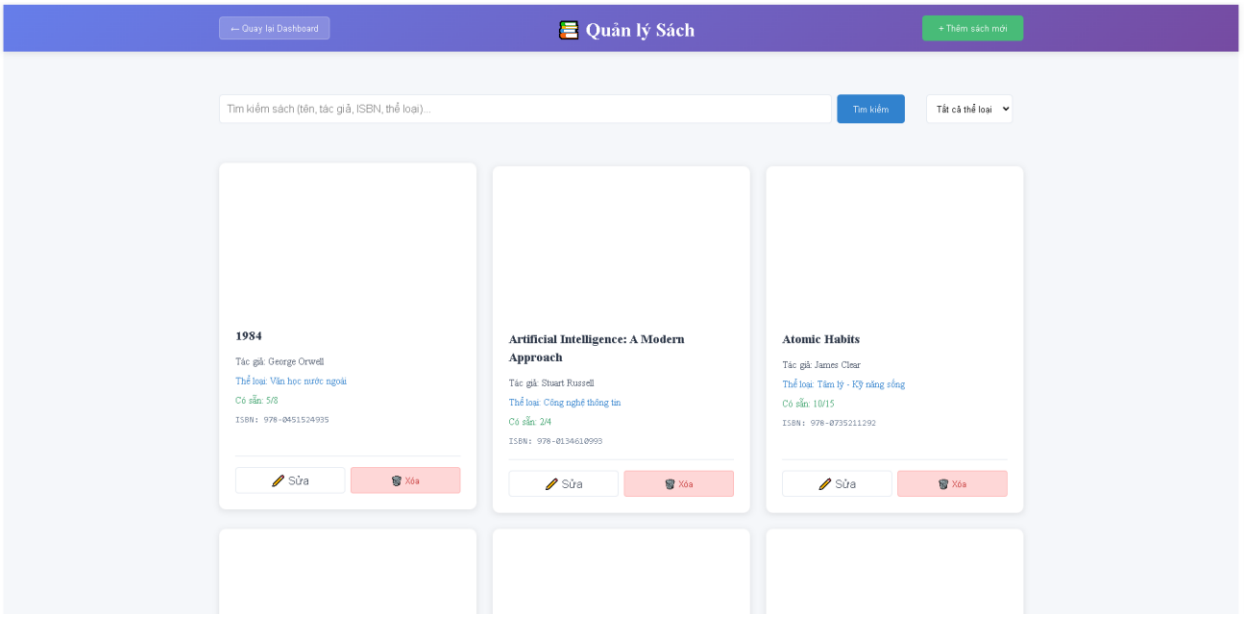
Tính năng giao diện:

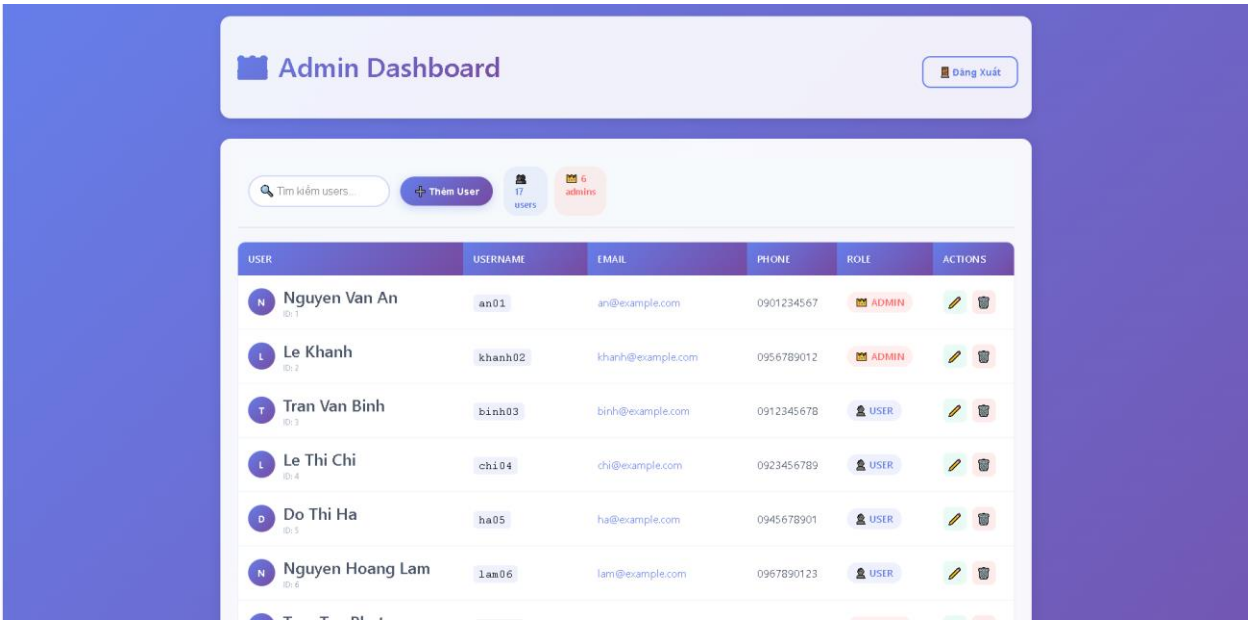
Form validation: Kiểm tra và hiển thị lỗi khi người dùng nhập sai thông tin

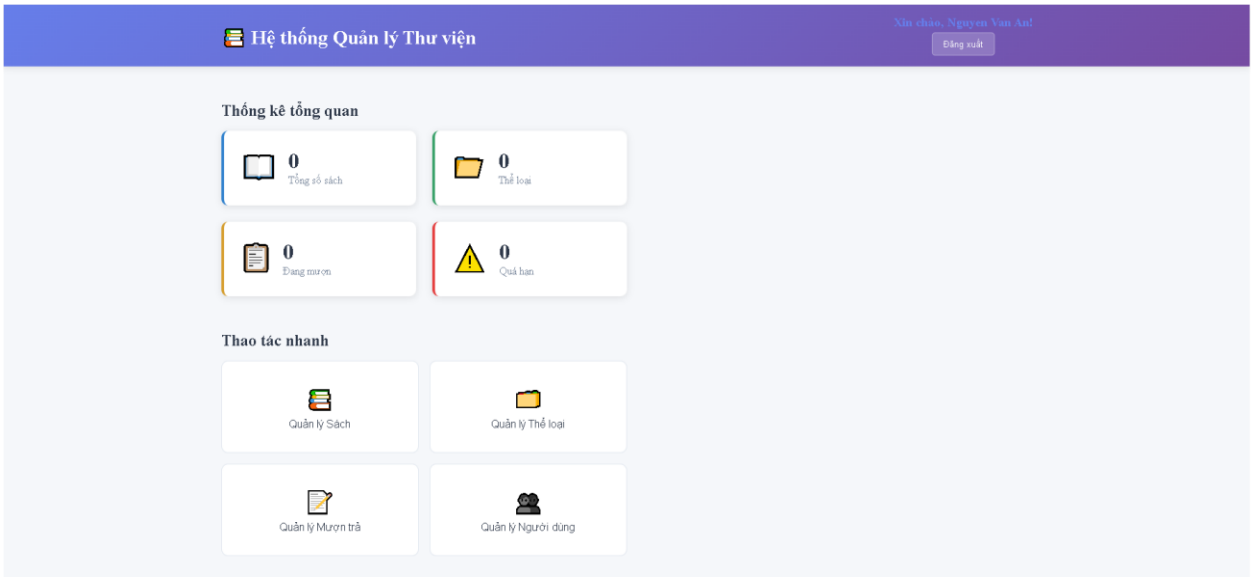
Professional styling: Thiết kế giao diện chuyên nghiệp với màu sắc và bố cục nhất quán

Interactive elements: Các nút và form có hiệu ứng khi hover và focus




← Quay lại Dashboard
Quản lý Sách
+ Thêm sách mới
Tìm kiếm sách (tên, tác giả, ISBN, thể loại)...
Tìm kiếm
Tất cả thể loại
1984
Tác giả: George Orwell
Thể loại: Văn học nước ngoài
Có sẵn: 5/8
ISBN: 978-0451524935
Sửa
Xóa
Artificial Intelligence: A Modern Approach
Tác giả: Stuart Russell
Thể loại: Công nghệ thông tin
Có sẵn: 2/4
ISBN: 978-0134610993
Sửa
Xóa
Atomic Habits
Tác giả: James Clear
Thể loại: Tâm lý - Kỹ năng sống
Có sẵn: 10/15
ISBN: 978-0735211292
Sửa
Xóa


Admin Dashboard
Đăng Xuất
Tìm kiếm users...
Thêm User
17 users
6 admins
USER
USERNAME
EMAIL
PHONE
ROLE
ACTIONS
Nguyen Van An
an01
an@example.com
0901234567
ADMIN
Le Khanh
khanh02
khanh@example.com
0956789012
ADMIN
Tran Van Binh
binh03
binh@example.com
0912345678
USER
Le Thi Chi
chi04
chi@example.com
0923456789
USER
Do Thi Ha
ha05
ha@example.com
0945678901
USER
Nguyen Hoang Lam
lam06
lam@example.com
0967890123
USER


Hệ thống Quản lý Thư viện
Xin chào, Nguyen Van An!
Đăng xuất
Thống kê tổng quan
0
Tổng số sách
0
Thể loại
0
Đang mượn
0
Quá hạn
Thao tác nhanh
Quản lý Sách
Quản lý Thể loại
Quản lý Mượn trả
Quản lý Người dùng