BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

Bộ môn: Thiết kế phần mềm

Developer Guide

Giảng viên lý thuyết	Trần Duy Thảo	
Sinh viên thực hiện	22120230	Phạm Tấn Nghĩa
	22120218	Lý Trường Nam
	22120268	Nguyễn Đình Phú

Table of Contents

I. Introduction

Chào mừng bạn đến với dự án "Hệ thống Quản lý Sinh viên" của nhóm TamLongHoi!

Tài liệu hướng dẫn này được biên soạn nhằm mục đích cung cấp cho các nhà phát triển một bộ quy tắc, tiêu chuẩn và quy trình làm việc thống nhất khi tham gia vào dự án. Mục tiêu của chúng tôi là xây dựng một môi trường phát triển chuyên nghiệp, nơi mọi người có thể hợp tác một cách hiệu quả để tạo ra một sản phẩm chất lượng cao.

Việc tuân thủ các chỉ dẫn trong tài liệu này không chỉ giúp đảm bảo tính nhất quán của mã nguồn mà còn góp phần duy trì sự ổn định và dễ dàng bảo trì của hệ thống trong tương lai.

Chúng tôi khuyến khích bạn dành thời gian đọc kỹ tài liệu này trước khi bắt đầu đóng góp vào dự án. Chúc bạn có những trải nghiệm làm việc hiệu quả và gặt hái nhiều thành công!

II. Getting Started

1. Overview of Architecture

Dự án được xây dựng dựa trên kiến trúc client-server hiện đại, sử dụng stack công nghệ MERN (MongoDB, Express.js, React, Node.js).

- **Frontend (Client):** Một ứng dụng trang đơn (Single-Page Application - SPA) được xây dựng bằng **React**. Nó chịu trách nhiệm về giao diện người dùng (UI) và trải nghiệm người dùng (UX).
    - Giao tiếp với backend thông qua các yêu cầu HTTP đến API RESTful.
    - Sử dụng các hook của React (useState, useEffect, useMemo) để quản lý trạng thái và vòng đời của component.
    - Triển khai đa ngôn ngữ (i18next) để hỗ trợ quốc tế hóa.
- **Backend (Server):** Một máy chủ API RESTful được xây dựng bằng **Node.js** và framework **Express.js**.
    - Cung cấp các điểm cuối (endpoints) để thực hiện các thao tác CRUD (Tạo, Đọc, Cập nhật, Xóa) trên dữ liệu.
    - Sử dụng **MongoDB** làm cơ sở dữ liệu NoSQL để lưu trữ dữ liệu.
    - **Mongoose** được sử dụng làm thư viện Mô hình hóa Dữ liệu Đối tượng (ODM) để tương tác với MongoDB, định nghĩa schema và thực hiện xác thực dữ liệu.
- **API:** Backend cung cấp một API RESTful được phiên bản hóa (ví dụ: /api/v1/...). Điều này cho phép frontend tương tác với dữ liệu một cách có cấu trúc và an toàn.

2. Source Code Organization

Cấu trúc thư mục của dự án được tổ chức rõ ràng để phân tách các mối quan tâm giữa frontend và backend.

```
├── backend/
```

```
│   ├── src/
│   │   ├── constants/      # (VD: api.constants.js)
│   │   ├── controllers/    # (VD: course.controller.js)
│   │   ├── models/         # (VD: course.model.js)
│   │   ├── responses/      # (VD: responseFactory.js)
│   │   ├── routes/         # (VD: department.route.js)
│   │   ├── services/       # (VD: translation.service.js)
│   │   └── utils/          # (VD: winston.js for logging)
│   ├── .env.example
│   └── package.json
│
├── frontend/
│   ├── src/
│   │   ├── assets/         # (Hình ảnh, fonts, ...)
│   │   ├── components/
│   │   │   ├── common/     # (VD: Button.jsx, SearchInput.jsx)
│   │   │   └── domain/     # (VD: students/StudentTable.jsx)
│   │   ├── hooks/          # (VD: useStudents.js, useCourse.js)
│   │   ├── services/       # (VD: student.service.js, BaseService.js)
│   │   ├── styles/         # (VD: pages/StudentScreen.scss)
│   │   ├── utils/          # (VD: string.util.js, factories/ValidationFactory.js)
│   │   └── views/          # (VD: RegistrationScreen.jsx)
│   └── package.json
│
├── .dockerignore
├── .gitignore
├── docker-compose.yml
├── Dockerfile
└── README.md
```

3. Getting Started with your app development

Làm theo các bước sau để thiết lập và chạy dự án trên máy của bạn.

Cách 1: Chạy với Docker (Khuyến nghị)

1. **Clone a repository:**
   Bash

   ```
   git clone https://github.com/JakePham23/TamLongHoi-Ex-TKPM.git
   ```

2. **Build and run containers:**
   - Trên macOS hoặc Linux:

     ```
     docker compose up -d --build
     ```

   - Trên Windows:

     ```
     docker-compose up -d --build
     ```

3. **Access the application:**

- Mở trình duyệt và truy cập: http://localhost:8080/

Cách 2: Chạy thủ công

Start the Backend:
Bash

```
cd backend
npm install
npm start
```

Start the Frontend:
Bash

```
cd ..
cd frontend
npm install
npm run dev
```

Access the application:

Mở trình duyệt và truy cập: http://localhost:5173/

4. Database Schema

Cơ sở dữ liệu được định nghĩa bằng Mongoose Schemas. Dưới đây là một ví dụ về schema cho collection Courses.

File:backend/src/models/course.model.js

JavaScript

```
import mongoose from 'mongoose';
const courseSchema = new mongoose.Schema({
 courseId: {
  type: String,
  required: true,
  unique: true
 },
 courseName: {
  type: String,
  required: true
 },
 credit: {
```

```
  type: Number,
  required: true,
  min: [2, 'Credit must be at least 2']
 },
 practicalSession: {
  type: Number,
  required: true,
  min: [0, 'Practical session must be at least 0']
 },
 theoreticalSession: {
  type: Number,
  required: true,
  min: [0, 'Theoretical session must be at least 0']
 },
 department: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'Departments', // Tham chiếu đến collection Departments
  required: true,
 },
 description: {
  type: String
 },
 prerequisite: { // Môn học tiên quyết
  type: String,
  validate: {
   validator: async function (value) {
    if (!value || value.trim() === "" || value === "Không có") return true;
    const course = await mongoose.model("Courses").findOne({ courseId: value });
    return !!course;
   },
   message: props => `Prerequisite course with ID "${props.value}" does not exist.`
  }
 }
}, {
 collection: 'Courses',
```

```
  timestamps: true // Tự động thêm createdAt và updatedAt
});

const Course = mongoose.model('Courses', courseSchema);
export default Course;
```

5. Updating an existing entity (How to add a new property)

Để thêm một thuộc tính mới vào một thực thể hiện có (ví dụ: thêm language vào Course), hãy làm theo các bước sau:

Bước 1: Cập nhật Model (Backend) Mở tệp model tương ứng (course.model.js) và thêm trường mới vào schema.

```
// backend/src/models/course.model.js
...
  description: {
    type: String
  },
  language: { // <-- Thuộc tính mới
    type: String,
    default: 'Vietnamese'
  },
  prerequisite: {
...
```

Bước 2: Cập nhật Controller (Backend) Trong course.controller.js, điều chỉnh các hàm addCourse và updateCourse để xử lý thuộc tính mới từ req.body.

```
// backend/src/controllers/course.controller.js
// trong hàm addCourse
const {
    courseId, courseName, credit, ..., prerequisite, language // <-- Thêm vào đây
} = req.body;

const newCourse = new courseModel({
    courseId, courseName, credit, ..., prerequisite, language // <-- Thêm vào đây
});

// trong hàm updateCourse
// Đảm bảo `updateData` từ `req.body` chứa `language`
```

```
const updatedCourse = await courseModel.findOneAndUpdate(
   { courseId: courseId },
   updateData, // updateData sẽ chứa { ..., language: 'English' }
   { new: true, runValidators: true }
);
```

Bước 3: Cập nhật Component Form (Frontend) Trong component React chịu trách nhiệm tạo/cập nhật thực thể (ví dụ CourseForm.jsx), thêm một trường nhập liệu mới để người dùng có thể cung cấp giá trị cho thuộc tính mới.

Bước 4: Cập nhật State và Service (Frontend) Đảm bảo rằng trạng thái của component quản lý giá trị của trường mới và service gửi nó đến backend khi gọi API create hoặc update.

6. Registering New Routes

Để tạo một bộ các điểm cuối API mới, hãy làm theo quy trình sau:

Bước 1: Tạo Controller Tạo một tệp controller mới trong backend/src/controllers/ để xử lý logic cho thực thể mới.

Bước 2: Tạo tệp Route Tạo một tệp route mới trong backend/src/routes/, ví dụ teacher.route.js.

Bước 3: Định nghĩa các Routes Sử dụng express.Router() để định nghĩa các điểm cuối.

```
// backend/src/routes/teacher.route.js
import express from "express";
import teacherController from "../controllers/teacher.controller.js";

const router = express.Router();

// GET /api/v1/teachers/
router.get("/", teacherController.getAllTeachers);

// POST /api/v1/teachers/add
router.post("/add", teacherController.addTeacher);

// PUT /api/v1/teachers/update/:teacherId
router.put("/update/:teacherId", teacherController.updateTeacher);

// DELETE /api/v1/teachers/delete/:teacherId
router.delete("/delete/:teacherId", teacherController.deleteTeacher);
```

```
export default router;
```

Bước 4: Đăng ký Router trong ứng dụng chính Trong tệp app.js hoặc index.js chính của backend, import và sử dụng router mới.

```
import teacherRoutes from './routes/teacher.route.js';
// ...
app.use('/api/v1/teachers', teacherRoutes);
```

7. Data Validation

Xác thực dữ liệu được thực hiện ở cả phía client và server để đảm bảo tính toàn vẹn của dữ liệu.

- **Backend (Server-Side Validation):**
    1. **Schema-Level:** Mongoose schema được sử dụng để xác thực kiểu dữ liệu, các trường bắt buộc (required), tính duy nhất (unique), và các quy tắc phức tạp hơn thông qua thuộc tính validate. (Xem course.model.js để biết ví dụ).
    2. **Controller-Level:** Các kiểm tra bổ sung được thực hiện trong các hàm controller trước khi tương tác với cơ sở dữ liệu. Ví dụ, kiểm tra sự tồn tại của các trường bắt buộc trong req.body.
- **Frontend (Client-Side Validation):**
    1. **Validation Factory:** Dự án sử dụng một ValidationFactory (frontend/src/utils/factories/ValidationFactory.js) để tạo ra các trình xác thực có thể tái sử dụng cho các thực thể khác nhau (ví dụ: studentValidator).
    2. **Form Validation:** Trước khi gửi dữ liệu đến backend, dữ liệu từ form được xác thực bằng cách sử dụng trình xác thực đã tạo. Nếu có lỗi, một thông báo sẽ được hiển thị cho người dùng và yêu cầu API sẽ không được gửi đi.

8. Web API documentation

Tài liệu API được tạo tự động bằng **Swagger**.

Cách viết tài liệu: Tài liệu được viết dưới dạng các khối bình luận JSDoc ngay phía trên mỗi định nghĩa route. Các thẻ như @swagger, @summary, @tags, @param, @requestBody, và @responses được sử dụng để mô tả điểm cuối.
Ví dụ từ department.route.js:

```
/**
* @swagger
* /api/v1/departments/add:
* post:
* summary: Thêm khoa mới
* tags: [departments]
```

```
* requestBody:
* required: true
* content:
* application/json:
* schema:
* type: object
* properties:
* departmentName:
* type: string
* responses:
* 201:
* description: Thêm khoa thành công
*/
router.post("/add", departmentController.addDepartment);
```

Cách truy cập tài liệu:

Chạy backend với lệnh đặc biệt:

```
cd backend
npm run api
```

- Mở trình duyệt và truy cập URL sau để xem giao diện Swagger UI tương tác: http://localhost:4000/api-docs

9. Unit testing

1. Cài đặt thư viện:

Bash

```
npm install --save-dev jest supertest
npm install --save-dev babel-jest @babel/preset-env
```

2. Thực hiện unit test:

- Course:

```
npx jest src/tests/courses.test.js
```

- Department:

```
npx jest src/tests/department.test.js
```

- Student:

```
npx jest src/tests/student.test.js
```

III. Conclusion

"Hướng dẫn dành cho Nhà phát triển Dự án TamLongHoi" là một tài nguyên quý giá cho tất cả các nhà phát triển làm việc trong hệ sinh thái của chúng tôi. Bằng cách tuân thủ các chỉ dẫn và những phương pháp tốt nhất được nêu trong tài liệu này, bạn sẽ góp phần vào sự thành công và ổn định của các dự án của chúng ta.

Để được hỗ trợ thêm hoặc có bất kỳ câu hỏi nào, vui lòng liên hệ với Trưởng dự án (Phạm Tấn Nghĩa) qua email jakeptnn@gmail.com.

Chúng tôi rất mong chờ những đóng góp của bạn và chúc bạn thành công trên hành trình phát triển của mình!