ĐẠI HỌC QUỐC GIA TP HCM
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

DEVELOPER GUIDE

Nhóm thực hiện: niceScore
Thành viên: 22120062 Nguyễn Đăng Điền
22120186 Huỳnh Tấn Lộc
22120199 Trần Lượng
Lớp: Thiết kế phần mềm (22_3)

Tp HCM – Tháng 6/2025

Mục lục

I. Coding Standards

- Ngôn ngữ: TypeScript/JavaScript (Node.js, Express).
- Quy tắc đặt tên:
    - Biến và hàm: camelCase *(ví dụ:* studentName, getStudentById).
    - Class và interface: PascalCase (*ví dụ*: StudentModel, IStudentService).
    - Hằng số: UPPER_CASE (*ví dụ*: MAX_STUDENTS_PER_CLASS).
- Cấu trúc file:
    - Mỗi thực thể (model, controller, service, router) được đặt trong file riêng, tên file phản ánh chức năng (ví dụ: student.model.ts, student.controller.ts).
    - Tên file nên ngắn gọn, rõ ràng, tránh dư thừa (ví dụ: không dùng StudentModel.model.ts).
- Comment:
    - Sử dụng JSDoc cho tất cả function, class, và interface để hỗ trợ IntelliSense và tài liệu hóa.
    - Thêm chú thích ngắn gọn cho các đoạn logic phức tạp, giải thích ý nghĩa và mục đích.

- Linting:
    - Sử dụng ESLint với cấu hình chuẩn Airbnb, tùy chỉnh trong .eslintrc.json.
    - Đảm bảo chạy npm run lint trước khi commit để kiểm tra lỗi định dạng.

II. Overview of Architecture

- Kiến trúc: MVC (Model-View-Controller) để tách biệt logic nghiệp vụ, giao diện, và dữ liệu.
- Backend:
    - Framework: Node.js + Express.
    - ORM: Sequelize để tương tác với database.
- Frontend:
    - Template engine: Handlebars cho các view.
    - Assets (CSS, JS, images) được tổ chức trong src/public/assets.
- Database: Hỗ trợ PostgreSQL hoặc MySQL, cấu hình qua file .env.
- I18n: Sử dụng module i18n để hỗ trợ đa ngôn ngữ, lưu các chuỗi tĩnh trong src/i18n.
- Logging: Sử dụng Winston để ghi log, lưu vào src/logs với định dạng rõ ràng (bao gồm timestamp, level, message).

III. Source code organization

/niceScore-Ex-Sat

```
├── .vscode
├── node_modules
├── dist                    # Output biên dịch TS
├── report                  # Danh sách báo cáo cho các tuần
├── screenshot              # Snapshot cho một số kết quả của đồ án
├── src
│   ├── /config             # Cài đặt database, handlebar, …
│   ├── /controllers        # Cài đặt controllers
│   ├── /i18n               # Cài đặt i18n cho đa ngôn ngữ tĩnh
│   ├── /logs               # Các file log
│   ├── /models             # Định nghĩa mô hình
│   ├── /public/assets
│   │   ├── /scripts        # Danh sách các file JS cho front-end
│   ├── /routes             # Cài đặt routes
│   ├── /services           # Cài đặt các hàm tương tác với database
│   ├── /templates          # Các template dán
│   ├── /views              # Danh sách trang html/hbs
├── tests                   # Các bài kiểm thử
├── index.ts                # File khởi động
├── .gitignore
├── .env                    # File môi trường, thông tin database
├── package.json, pacake-lock.json, tsconfig,json        # File cài đặt
└── README.md
```

IV. Getting Started with your app development

1. Clone source code: git clone https://github.com/htloc0610/niceScore-Ex-Sat.git
2. Cài đặt dependencies: npm install
3. Cấu hình database: Thêm file .env

```
PORT=3000
DB_NAME=student_management_d13z
DB_USER=student_management_d13z_user
DB_PASSWORD=E6eBP0SYivWihc9tqhciVwAVmTY48y30
DB_HOST=dpg-cveie6dumphs73br78lg-a.oregon-postgres.render.com
DB_PORT=5432
DB_DIALECT=postgres
VERCEL=false
```

4. Chạy ứng dụng: npm start
5. Truy cập: Mở trình duyệt tại http://localhost:3000

V. Database Schema

- Các bảng chính:
    - students: Thông tin sinh viên.
    - faculties: Thông tin khoa.
    - modules: Thông tin về môn học (*vd:* Môn Thiết kế phần mềm, …).
    - classes: Thông tin về lớp học cụ thể (*vd:* CQ2022/3).
    - class_registrations: Thông tin sinh viên đăng ký lớp học.
    - status: Danh sách các tình trạng của sinh viên (đang học, tạm ngưng, …).
    - configurations: Cặp giá trị (key, value) lưu thông tin ràng buộc (đuôi email, đầu số điện thoại).
    - transcripts: Thông tin điểm số của sinh viên.
    - module_translations: Thông tin chuyển trạng thái, các trạng thái có thể chuyển tiếp từ trạng thái này.
    - address: địa chỉ (số nhà, đường, huyện, …) của sinh viên.
    - identification: thông tin giấy tờ (CCCD, CMND, hộ chiếu) của sinh viên.
    - course: thông tin khóa học/chuyên ngành của sinh viên.
    - registration_cancellations: thông tin về các khóa học bị hủy của sinh viên cùng lý do.
    - module_translations: lưu tên và mô tả của môn học bằng nhiều ngôn ngữ khác nhau.

- Quan hệ:
    - students liên kết với faculties, courses, status, address, identification.
    - modules liên kết với faculties, prerequisite (self-reference).
    - classes liên kết với modules.
    - class_registrations liên kết với students, classes.
    - transcripts liên kết với students, classes.
    - module_translations liên kết với modules.
    - status_transitions liên kết với status.

- Ví dụ bảng modules (môn học):

Các trường	**module_id**	module_code	credits	faculty_id	prerequisite_id	is_active
Kiểu dữ liệu	Số nguyên	Chuỗi (50)	Số nguyên	Số nguyên	Số nguyên	Boolean
Ý nghĩa	Khóa chính	Mã môn học để phân biệt	Số tín chỉ	Liên kết đến mã khoa tương ứng	Liên kết đến mã môn học (module_id) của môn học bắt buộc tương ứng	Trạng thái (0: đã ngưng, 1: đang hoạt động)
Ví dụ	**10**	CSC13010	4	12	1	1

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

1. Thêm thuộc tính vào model: Sửa file trong src/models/, ví dụ thêm trường avatar vào student.model.ts.
2. Tạo migration: Sử dụng Sequelize CLI để tạo migration thêm cột.
3. Cập nhật service/controller nếu cần.
4. Cập nhật giao diện (views) nếu hiển thị thuộc tính mới.
5. Viết unit test cho thuộc tính mới.

VII. Registering New Routes

- Thêm file router mới trong src/routes/, ví dụ book.router.ts.

```
import { Router } from "express";
import bookController from "../controllers/book.controller";

const bookRouter = Router();

// [GET] /api/course
courseRouter.get("/", bookController.getAllBooks);

// [POST] /api/course
courseRouter.post("/", bookController.addBook);

// [PUT] /api/course
courseRouter.put("/", bookController.getAllBook);

export default bookRouter;
```

- Định nghĩa endpoint với Express Router.
- Thêm vào index.router.ts: *app.use('/api/book', bookRouter);*

```
  app.use("/api/book", bookRouter);
```

- Tạo controller và service tương ứng.

VIII. Data Validation

1. Mục đích

Data Validation (kiểm tra dữ liệu đầu vào) giúp đảm bảo dữ liệu mà người dùng gửi lên API hoặc nhập vào hệ thống là hợp lệ, đúng định dạng, đầy đủ và không gây lỗi cho hệ thống phía sau (database, logic nghiệp vụ).

2. Cách triển khai

Kiểm tra thủ công: Trong các controller hoặc service, kiểm tra từng trường dữ liệu (ví dụ: không được rỗng, đúng kiểu dữ liệu, giá trị hợp lệ).

Sử dụng thư viện: Có thể tích hợp các thư viện như joi, express-validator để tự động kiểm tra dữ liệu đầu vào cho các API.

3. Ví dụ kiểm tra trong controller

```
// Check if the email domain is allowed
const emailConfig = await configurationService.getConfiguration(
  "allowed_email_domain"
);
const emailRegex = new RegExp(
  `^[a-zA-Z0-9._%+-]+@${emailConfig.config_value}$`
);
if (email && !emailRegex.test(email)) {
  logger.error("Invalid email domain");
```

IX. Unit Testing

Hệ thống đã được xây dựng với chiến lược kiểm thử đơn vị (unit test) rõ ràng, đảm bảo chất lượng và độ tin cậy của từng thành phần chức năng. Các bài kiểm thử đơn vị được tổ chức trong thư mục tests, bao phủ cả service và router của các module chính như: sinh viên, lớp học, học phần, trạng thái, khoa, cấu hình, v.v.

1. Mục tiêu kiểm thử

- Đảm bảo các hàm nghiệp vụ (service) trả về đúng dữ liệu, xử lý đúng logic, bao gồm cả trường hợp thành công và thất bại.
- Đảm bảo các API endpoint (router) phản hồi đúng định dạng, mã trạng thái HTTP, và dữ liệu trả về.
- Kiểm tra các trường hợp ngoại lệ, lỗi truy vấn, và các tình huống biên.

2. Công nghệ sử dụng

- **Jest**: Framework kiểm thử chính, hỗ trợ mock, spy, và assertion mạnh mẽ.
- **Supertest**: Dùng để kiểm thử các endpoint HTTP của Express.

3. Cách tổ chức test

- Mỗi service hoặc router đều có file test riêng, ví dụ: *status.service.test.ts, student.router.test.ts, module.service.test.ts*, v.v.
- Các model, controller, logger đều được mock để kiểm soát dữ liệu đầu vào/đầu ra và cô lập phạm vi kiểm thử.
- Các hàm như *findAll, findOne, create, update*, ... của model được mock để kiểm tra logic xử lý mà không phụ thuộc vào database thật.

4. Một số ví dụ tiêu biểu

- Service Test: Kiểm tra hàm lấy danh sách trạng thái (*getAllStatuses*) trả về đúng thứ tự, đúng dữ liệu, và ghi log đúng cách.
- Router Test: Kiểm tra endpoint /api/student trả về đúng thông tin sinh viên, endpoint /api/module thêm mới, cập nhật, xóa module đúng chuẩn RESTful.
- Xử lý lỗi: Các test case đều có kiểm tra trường hợp lỗi, ví dụ khi truy vấn thất bại sẽ trả về lỗi đúng định dạng và thông báo phù hợp.
- Ví dụ về Service test cho course:

```
describe("courseService", () => {
  describe("getAllCourses", () => {
    it("should return a list of all courses", async () => {
      const mockCourses = [
        {
          dataValues: {
            course_id: 1,
            course_name: "Math 101",
          },
        },
        {
          dataValues: {
            course_id: 2,
            course_name: "Physics 101",
          },
        },
      ];

      (Course.findAll as jest.Mock).mockResolvedValue(mockCourses);

      const result = await courseService.getAllCourses();

      expect(Course.findAll).toHaveBeenCalled();
      expect(result).toEqual([
        mockCourses[0].dataValues,
        mockCourses[1].dataValues,
      ]);
    });
   … // Tests for other functions
});
```

5. Đánh giá

Việc áp dụng kiểm thử đơn vị giúp phát hiện lỗi sớm, tăng độ tin cậy khi refactor code, và đảm bảo các chức năng cốt lõi luôn hoạt động ổn định. Các bài test được viết rõ ràng, dễ bảo trì, và có thể mở rộng khi bổ sung tính năng mới.

X. Web API documentation

Hệ thống cung cấp API cho phép quản lý các cấu hình hệ thống (Settings) thông qua các endpoint RESTful. Các cấu hình này được lưu trữ trong bảng configurations và có thể được truy xuất, thêm mới hoặc cập nhật thông qua các API sau:

- **GET /api/configurations**: Lấy danh sách tất cả các cấu hình hiện tại của hệ thống.
- **POST /api/configurations**: Thêm mới một cấu hình hệ thống.
- **PUT /api/configurations**: Cập nhật giá trị của một cấu hình đã có.

Các API này giúp quản trị viên dễ dàng điều chỉnh các tham số hệ thống (ví dụ: ngôn ngữ mặc định, giới hạn truy cập, v.v.) mà không cần can thiệp trực tiếp vào mã nguồn. Việc thay đổi cấu hình sẽ được ghi log để đảm bảo tính minh bạch và dễ dàng kiểm soát.