VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

INFORMATION TECHNOLOGY PROJECT 2

CONSTRUCT STUDENT MANAGEMENT SYSTEM

Instructor: **ĐẶNG MINH THẮNG**

Student: **NGUYỄN CHÂU THẢO QUÂN – 51600072**

Class **: 16050310**

Course **: 20**

HO CHI MINH CITY, 2021

VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

INFORMATION TECHNOLOGY PROJECT 2

CONSTRUCT STUDENT MANAGEMENT SYSTEM

Instructor: **ĐẶNG MINH THẮNG**

Người thực hiện: **NGUYỄN CHÂU THẢO QUÂN – 51600072**

Class **: 16050310**

Course **: 20**

HO CHI MINH CITY, 2021

ACKNOWLEDGEMENT

I would like to express my special thanks of gratitude to Mr Đặng Minh Thắng as well as our faculty of information technology who gave me the golden opportunity to do this information technology project 2 on the topic (Construct the student management system) for graduation which also helped me in doing a lot of research and I came to know so many new thinks I am really thankful to them.

Ho Chi Minh city, day ... month ... year 20...

Authors

(signature and full name)

Nguyễn Châu Thảo Quân

GUARANTEES OF GRADUATION PROJECT

This graduation project was carried out at Ton Duc Thang University.

Advisor: .....

.....

(Title, fullname and signature)

This thesis is defended at the Undergraduate Thesis Examination Committee was hold at Ton Duc Thang University on …/…/…

Confirmation of the Chairman of the Undergraduate Thesis Examination Committee and the Dean of the faculty after receiving the modified thesis (if any).

CHAIRMAN **DEAN OF FACULTY**

....

DECLARATION OF AUTHORSHIP

I hereby declare that this thesis was carried out by myself under the guidance and supervision of Mr Đặng Minh Thắng; and that work and the results contained in it are original and have not been submitted anywhere for any previous purposes. The data and figures presented in this thesis are for analysis, comments and evaluations from various resources by my own work and have been duly acknowledged in the reference part.

In addition, other comments, reviews and data used by other authors, and organizations have been acknowledged and explicitly cited.

I will take full responsibility for any fraud detected in my thesis. Ton Duc Thang University is unrelated to any copyright infringement caused on my work (if any).

Ho Chi Minh city, day ... month ... year ……

Authors

(signature and full name)

Nguyễn Châu Thảo Quân

TEACHER'S CONFIRMATION AND EVELUATION SECTION

Instructor's confirmation section

_____

_____

_____

_____

_____

_____

_____

Ho Chi Minh city, day … month … year …
(signature and fullname)

Examiner's evaluation section

_____

_____

_____

_____

_____

_____

_____

Ho Chi Minh city, day … month … year …
(signature and fullname)

STUDENT MANAGEMENT SYSTEM
ABSTRACT

In this 21st century, all is about technology. The schools and education nowadays also already changed. Not only for all high schools and primary schools, the universities also must have their own management system.

System analysis and design is an dispensable first step in the software system development process. It is realized and assumed that information system analysis and design is the most vital stage in software development process. Otherwise, a mistake in the data design process can lead to a poor quality product or no long-term value.

The management and provision of information about the educational process is an essential part of effective management of the educational process in the institutes of higher education. Nowadays, along with the development of science and technology, the demand for the informatics applications is also increasing rapidly. The construction of management system to meet above these demands is very essential. Student management is not an exception. The birth of student management system will reduce manpower, helping to manage students and even making it more convenient and especially helpful. As a result, a system called Student Management System will be developing as a replacement of the manual management methods to solve problem that facing when was using that methods. Student Management System is a system for education that use to manage student's information, also known as student information system, student information management system. This design of this system is web-based type, so the user can use the system directly by connect to internet.

The requirements of a reliable student management system are analysed, formed a use-case diagram of student management system, designed and implemented the architecture of the application. Regarding the implementation process, modern approaches were used to develop a reliable websites written by PHP.

TABLE OF CONTENTS

ABBREVIATIONS

API	Application Programming Interface
GUI	Graphic User Interface
IoC	Inversion of Control
MVC	Model – View – Controller
ORM	Object-Relational Mapping
PHP	Hypertext Preprocessor

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 – INTRODUCTION

1.1 Objectives and the Objects.

1.1.1 Objectives

To design and develop a Student Management System for an university.

To record all the student's information for future reference and to manage student's information include personal information and the academic information.

1.1.2 Objects

The users of this system are given to three groups which are administrators and students. Administrator can manage all student's data and information easily and students can check their information after login successful.

1.2 Scopes

The main target of this system is an university, it is important because to make sure the system meet their requirements.

.

CHAPTER 2 – SYSTEM ANALYSIS AND DESIGN

2.1 System Analysis

The Student Management System can handle all the details about a student, student personal details, academic details, college details etc., The Student Manage System is an automated version of manual student management process.

2.1.1 Purpose

This System Requirements Specification contains the complete system requirements for the Student Management System and describes the design decisions, architectural design and the detailed design needed to implement the system. It provides the visibility in the design and information needed for software support.

2.1.2 System Perspective

The proposed system will be developed using MVC architecture and be compatible with any platform. The front end of the system will be developed using Laravel's blade engine and backend will be developed using Laravel.

2.1.3 Actors And Use Cases Descriptions

2.1.3.1 Diagrams

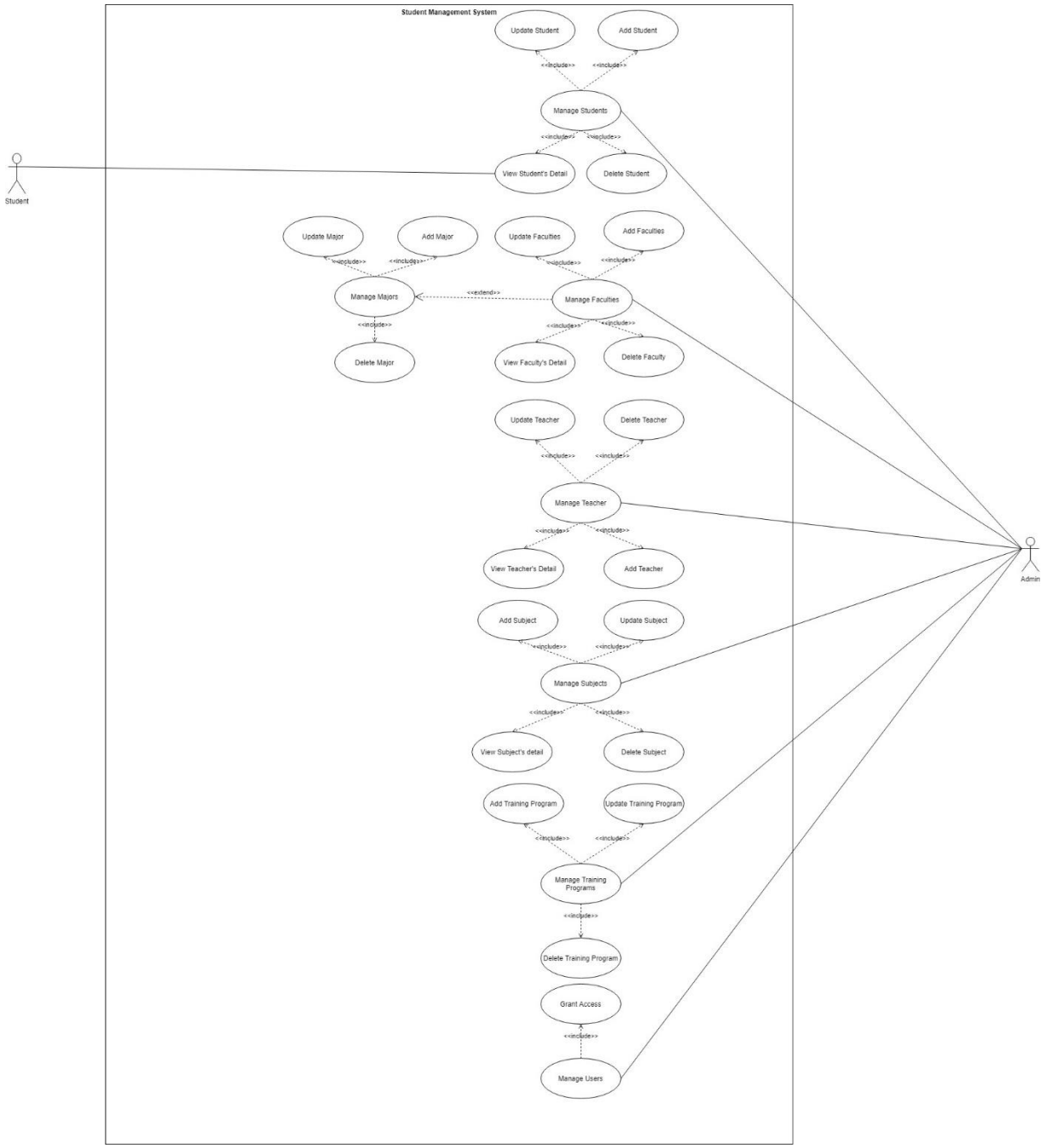

Figure 2.1: Student Information and University Management

2.1.3.2 Actors Description

#	**Actors**	**Definitions**
1	Admin	
2	Teacher	
3	Student	

Table 2.1: Actors description table

2.1.3.3 Use Case Description

Use Case ID:	UC01
Use Case Name:	Add Student
Description:	This use case allow user to add the new informations of student.
Actor(s):	Admin
Pre-conditions:	User is permitted to access to the dashboard.
Post-conditions:	A new student is enlisted.

Main Flow:	1. User clicks on button “Quản Lý Sinh Viên” on the left sidebar. 2. System displays a list of student. 3. User clicks on button “+” on the top of the list. 4. System displays a form which allows user to add a new student. 5. User enters student’s information and submits. 6. System validates these information. 7. System displays an added student to the list. 8. Use Case ends.
Alternative Flow:	6a. Missing the required information or duplicating the information. 1. Error messages display under these fields. 2. Use case resumes at main flow step 5.
Exception Flow:	

Table 2.2: Use Case “Add Student”.

Use Case ID:	UC02
Use Case Name:	Update Student
Description:	This use case allow user to update student’s informations.
Actor(s):	Admin
Pre-conditions:	User is permitted to access to the dashboard.

Post-conditions:	Any student's information is updated and a list of student displays.
Main Flow:	1. User click the button "Quản Lý Sinh Viên" on the left sidebar. 2. System displays a list of students. 3. User click on button "✍" on the right of a specific student on the list. 4. System displays a form which allows user to update student's information. 5. User enters new informations which are needed to update and submits. 6. System displays student's profile with updated informations. 7. Use case ends.
Alternative Flow:	
Exception Flow:	

Table 2.3: Use Case "Update Student".

Use Case ID:	UC03
Use Case Name:	View Student's Detail
Description:	This use case allow user to view a detailed information of specific student including their personal information, the information at university.
Actor(s):	Admin

Pre-conditions:	User is permitted to access to the dashboard.
Post-conditions:	A student’s profile displays.
Main Flow:	1. User clicks the button “Quản Lý Sinh Viên” on the left sidebar. 2. System displays a list of student. 3. User clicks on student’s name. 4. System displays a student’s profile. 5. Use case ends.
Alternative Flow:	
Exception Flow:	

Table 2.4: Use case “View Student's Detail”.

Use Case ID:	UC04
Use Case Name:	Delete Student
Description:	This use case allow user to delete a specific student from a list.
Actor(s):	Admin
Pre-conditions:	User is permitted to access to the dashboard.
Post-conditions:	A deleted student is removed from list.

Main Flow:	1. User click the button “Quản Lý Sinh Viên” on the left sidebar. 2. System displays a list of student. 3. User click on button “–” on the right of a specific student in the list. 4. System refreshes a list and removes a deleted student from a list. 5. User can click on 🗑 to view all deleted students. 6. System will display a list of deleted students. 7. Use case ends.
Alternative Flow:	
Exception Flow:	

Table 2.5: Use Case “Delete Student”.

Use Case ID:	UC05
Use Case Name:	Add Teacher
Description:	This use case allow user to add the new informations of a new teacher.
Actor(s):	Admin
Pre-conditions:	User is permitted to access to the dashboard.
Post-conditions:	A new teacher is enlisted.

Main Flow:	1. User click on button “Quản Lý Giảng Viên” on the left sidebar. 2. System displays a list of teacher. 3. User click on button “+” on the top of the list. 4. System displays a form which allows user to add a new teacher. 5. User enters all teacher’s information and submits. 6. System validates these information. 7. System displays an added teacher into a list. 8. Use Case ends.
Alternative Flow:	6a. Missing the required information or duplicating information. 1. Error messages display under these fields. 2. Use case resumes at main flow step 5.
Exception Flow:	

Table 2.6: Use Case “Add Teacher”.

Use Case ID:	UC06
Use Case Name:	Update Teacher
Description:	This use case allow user to update a new teacher’s informations including their personal information, the information at university.
Actor(s):	Admin

Pre-conditions:	User is permitted to access to the dashboard.
Post-conditions:	Any teacher's information is updated and a list of teacher displays.
Main Flow:	1. User click on button "Quản Lý Giảng Viên" on the left sidebar. 2. System displays a list of teacher. 3. User click on button "✍" on the right of a specific teacher in the list. 4. System displays a form which allows user to update teacher's information. 5. User enters the informations which are needed to update and submits these information. 6. System displays teacher's profile with updated informations. 7. Use Case ends.
Alternative Flow:	
Exception Flow:	

Table 2.7: Use Case "Update Teacher".

Use Case ID:	UC07
Use Case Name:	View Teacher's Detail
Description:	This use case allow user to view a detailed information of specific teacher including their personal information, the information at university.

Actor(s):	Admin
Pre-conditions:	User is permitted to access to the dashboard.
Post-conditions:	Teacher's profile displays.
Main Flow:	1. User click the button "Quản Lý Giảng Viên" on the left sidebar. 2. System displays a list of teacher. 3. User click on teacher's name. 4. System displays teacher's profile. 5. Use case ends.
Alternative Flow:	
Exception Flow:	

Table 2.8: Use Case "View Teacher's Detail".

Use Case ID:	UC08
Use Case Name:	Delete Teacher
Description:	This use case allow user to delete a specific teacher from a list.
Actor(s):	Admin
Pre-conditions:	User is permitted to access to the dashboard.
Post-conditions:	A deleted teacher is removed from list.

Main Flow:	1. User click the button “Quản Lý Giảng Viên” on the left sidebar. 2. System displays a list of teacher. 3. User click on button “–” on the right of a specific teacher in the list. 4. System refreshes a list. 5. User can click on 🗑 to view all deleted teachers. 6. System will display a list of deleted teachers. 7. Use case ends.
Alternative Flow:	
Exception Flow:	

Table 2.9: Use Case “Delete Teacher”.

Use Case ID:	UC09
Use Case Name:	Add Faculty
Description:	This use case allow user to add a specific faculty into a list.
Actor(s):	Admin
Pre-conditions:	User is permitted to access to the dashboard.
Post-conditions:	A new faculty is added to the list.

Main Flow:	1. User click the button “Quản Lý Khoa” on the left sidebar. 2. System displays a list of faculty. 3. User click on button “+” on top of the list. 4. System displays a form which allows user to add a new faculty. 5. User enter faculty’s information and submits. 6. System validates these information. 7. Use Case ends.
Alternative Flow:	6a. Missing id, name or dupplicating. 1. Error messages display under these fields. 2. Use case resumes at main flow step 5.
Exception Flow:	

Table 2.10: Use Case “Add Faculty”.

Use Case ID:	UC10
Use Case Name:	Update Faculty
Description:	This use case allow user to update a specific faculty into a list.
Actor(s):	Admin
Pre-conditions:	User is permitted to access to the dashboard.
Post-conditions:	The updated information of faculty displays into a list.

Main Flow:	1. User click the button “Quản Lý Khoa” on the left sidebar. 2. System displays a list of faculty. 3. User click on button “✍” on the right of a specific faculty on the list. 4. System displays a form which allows user to add a new faculty. 5. User enter faculty’s information and submits. 6. System displays a list of faculty with updated informations. 7. Use Case ends.
Alternative Flow:	
Exception Flow:	

Table 2.11: Use Case “Update Faculty”.

Use Case ID:	UC11
Use Case Name:	View Faculty
Description:	This use case allow user to view faculty’s detailed information.
Actor(s):	Admin
Pre-conditions:	User is permitted to access to the dashboard.
Post-conditions:	Faculty’s detail displays.

Main Flow:	1. User click the button “Quản Lý Khoa” on the left sidebar. 2. System displays a list of faculty. 3. User click on a specific faculty’s name. 4. System displays faculty’s detailed information. 5. Use Case ends.
Alternative Flow:	
Exception Flow	

Table 2.12: Use Case “View Faculty's Detail”.

Use Case ID:	UC12
Use Case Name:	Delete Faculty
Description:	This use case allow user to delete a specific faculty with its information.
Actor(s):	Admin
Pre-conditions:	User is permitted to access to the dashboard.
Post-conditions:	Faculty are removed from list and so are its related information.

Main Flow:	1. User click the button “Quản Lý Khoa” on the left sidebar. 2. System displays a list of faculty. 3. User click button “–” on the right of a specific faculty. 4. System refreshes a page. 5. User can click on 🗑 to view all deleted faculties. 6. System will display a list of deleted faculties. 7. Use case ends.
Alternative Flow:	
Exception Flow:	

Table 2.13: Use Case “Delete Faculty”.

Use Case ID:	UC13
Use Case Name:	Add Subject
Description:	This use case allow user to add a specific subject into a list.
Actor(s):	Admin
Pre-conditions:	User is permitted to access to the dashboard.
Post-conditions:	A new subject is enlisted.

Main Flow:	1. User click the button “Quản Lý Môn Học” on the left sidebar. 2. System displays a list of subjects. 3. User click on button “+” on top of the list. 4. System displays a form which allows user to add a new subject. 5. User enter subject’s information and submits. 6. System validates these information. 7. Use Case ends.
Alternative Flow:	6a. Missing required information or dupplicating. 1. Error messages display under these fields. 2. Use case resumes at main flow step 5.
Exception Flow:	

Table 2.14: Use Case “Add Subject”.

Use Case ID:	UC14
Use Case Name:	Update Subject
Description:	This use case allow user to update a specific subject from a list.
Actor(s):	Admin
Pre-conditions:	User is permitted to access to the dashboard.
Post-conditions:	Updated informations of subject display.

Main Flow:	1. User click the button “Quản Lý Môn Học” on the left sidebar. 2. System displays a list of subject. 3. User click on button “✍” on the right of a specific subject on the list. 4. System displays a form which allows user to update a new subject. 5. User enter faculty’s information which are needed to updated and submits. 6. System displays subject’s detail with updated informations. 7. Use Case ends.
Alternative Flow:	

Table 2.15: Use Case “Update Subject”.

Use Case ID:	UC15
Use Case Name:	View Subject
Description:	This use case allow user to view subject’s detailed information.
Actor(s):	Admin
Pre-conditions:	User is permitted to access to the dashboard.
Post-conditions:	Subject’s detailed information displays.

Main Flow:	1. User click the button “Quản Lý Môn Học” on the left sidebar. 2. System displays a list of subject. 3. User click on a specific subject’s name. 4. System displays subject’s detail. 5. Use Case ends.
Alternative Flow:	

Table 2.16: Use Case “View Subject's Detail”.

Use Case ID:	UC16
Use Case Name:	Delete Subject
Description:	This use case allow user to delete a specific subject.
Actor(s):	Admin
Pre-conditions:	User is permitted to access to the dashboard.
Post-conditions:	Subject are removed from list.

Main Flow:	1. User click the button “Quản Lý Môn Học” on the left sidebar. 2. System displays a list of subject. 3. User click button “–” on the right of a specific subject. 4. System refreshes a page and removes a deleted subject. 5. User can click on 🗑 to view all deleted subjects. 6. System will display a list of deleted subjects. 7. Use case ends.
Alternative Flow:	
Exception Flow:	

Table 2.17: Use Case “Delete Subject”.

Use Case ID:	UC17.
Use Case Name:	Add Training Program.
Description:	This use case allow user to add a new training program.
Actor(s):	Admin
Pre-conditions:	User is permitted to access the dashboard.
Post-conditions:	A new training program is enlisted

Main Flow:	1. User click the button “Quản Lý Chương Trình Đào Tạo” on the left sidebar. 2. System displays a list of training programs. 3. User click button “+” on top of the list. 4. System displays a form which allows user to add a new training program. 5. User enters the information and submits. 6. System validates these information. 7. Use Case ends.
Alternative Flow:	6a. Missing the required informations 1. Error messages display under these required fields. 2. Use Case resumes in main flow step 5
Exception Flow:	

Table 2.18: Use Case “Add Training Program”.

Use Case ID:	UC18.
Use Case Name:	Update Training Program.
Description:	This use case allow user to update the information of training program.
Actor(s):	Admin
Pre-conditions:	User is permitted to access the dashboard.
Post-conditions:	A new information of training program is updated

Main Flow:	1. User clicks the button “Quản Lý Chương Trình Đào Tạo” on the left sidebar. 2. System displays a list of training programs. 3. User clicks button “✍” on top of the list. 4. System displays a form which allows user to update a new information of training program. 5. User enters the information and submits. 6. System validates these information. 7. Use Case ends.
Alternative Flow:	
Exception Flow:	

Table 2.19: Use Case “Update Training Program”.

Use Case ID:	UC19.
Use Case Name:	Delete Training Program.
Description:	This use case allow user to delete a specific training program.
Actor(s):	Admin
Pre-conditions:	User is permitted to access the dashboard.
Post-conditions:	A training program is removed from list.

Main Flow:	1. User clicks the button “Quản Lý Chương Trình Đào Tạo” on the left sidebar. 2. System displays a list of training programs. 3. User clicks button “–” on top of the list. 4. System refreshes and removes a deleted training program from list. 5. User can click on 🗑 to view all deleted training programs. 6. System will display a list of deleted training programs. 7. Use case ends.
Alternative Flow:	
Exception Flow:	

Table 2.20: Use Case “Delete Training Program”.

Use Case ID:	UC20.
Use Case Name:	Grant Access.
Description:	This use case allow user to authorize a specific user with roles.
Actor(s):	Admin
Pre-conditions:	User is permitted to access the dashboard.
Post-conditions:	A user is granted a role.

Main Flow:	1. User clicks the button “Quản Lý Tài Khoản” on the left sidebar. 2. System displays a list of accounts. 3. User clicks button 🡻 on right of an account to the list and clicks on “Phân Quyền”. 4. System refreshes a lists. 5. User can click on 🡻 to view all roles of an account with the checked checkbox. 6. Use case ends.
Alternative Flow:	
Exception Flow:	

Table 2.21: Use Case “Grant Access”

Use Case ID:	UC21.
Use Case Name:	Add Major.
Description:	This use case allow user to add a new major.
Actor(s):	Admin
Pre-conditions:	1. User is permitted to access the dashboard. 2. A new faculty is added. 3. A profile of a faculty displays.
Post-conditions:	A new major is enlisted

Main Flow:	1. User click the button “Quản Lý Khoa” on the left sidebar. 2. System displays a list of faculties. 3. User click faculty’s name. 4. System displays faculty’s profile. 5. User clicks on button “+” on top of the list of majors. 6. System displays a form which allows user to add a new major. 7. User enters the informations and submits 8. System validates these information. 9. Use Case ends.
Alternative Flow:	8a. Missing the required informations 1. Error messages display under these required fields. 2. Use Case resumes in main flow step 5
Exception Flow:	

Table 2.22: Use Case “Add Major”.

Use Case ID:	UC22.
Use Case Name:	Update Major.
Description:	This use case allow user to update a specific major.
Actor(s):	Admin

Pre-conditions:	1. User is permitted to access the dashboard. 2. A new faculty is added. 3. A profile of a faculty displays.
Post-conditions:	The informations of a major is updated.
Main Flow:	1. User click the button “Quản Lý Khoa” on the left sidebar. 2. System displays a list of faculties. 3. User click faculty’s name. 4. System displays faculty’s profile. 5. User clicks on button “+” on top of the list of majors. 6. System displays a form which allows user to update a new information. 7. User enters the informations and submits. 8. Use Case ends.
Alternative Flow:	
Exception Flow:	

Table 2.23: Use Case “Update Major”.

Use Case ID:	UC23.
Use Case Name:	Delete Major.
Description:	This use case allow user to delete a specific major.
Actor(s):	Admin

Pre-conditions:	1. User is permitted to access the dashboard. 2. A new faculty is added. 3. A profile of a faculty displays.
Post-conditions:	A major is deleted.
Main Flow:	1. User click the button “Quản Lý Khoa” on the left sidebar. 2. System displays a list of faculties. 3. User click faculty’s name. 4. System displays faculty’s profile. 5. User clicks on button “–” on right of the list of majors. 7. System refreshes a list and a deleted major is removed from list. 8. Use Case ends.
Alternative Flow:	8a. Missing the required informations 1. Error messages display under these required fields. 2. Use Case resumes in main flow step 5
Exception Flow:	

Table 2.24: Use Case “Delete Major”.

2.2 System Design

2.2.1 Enhanced Entity Relationship Diagram

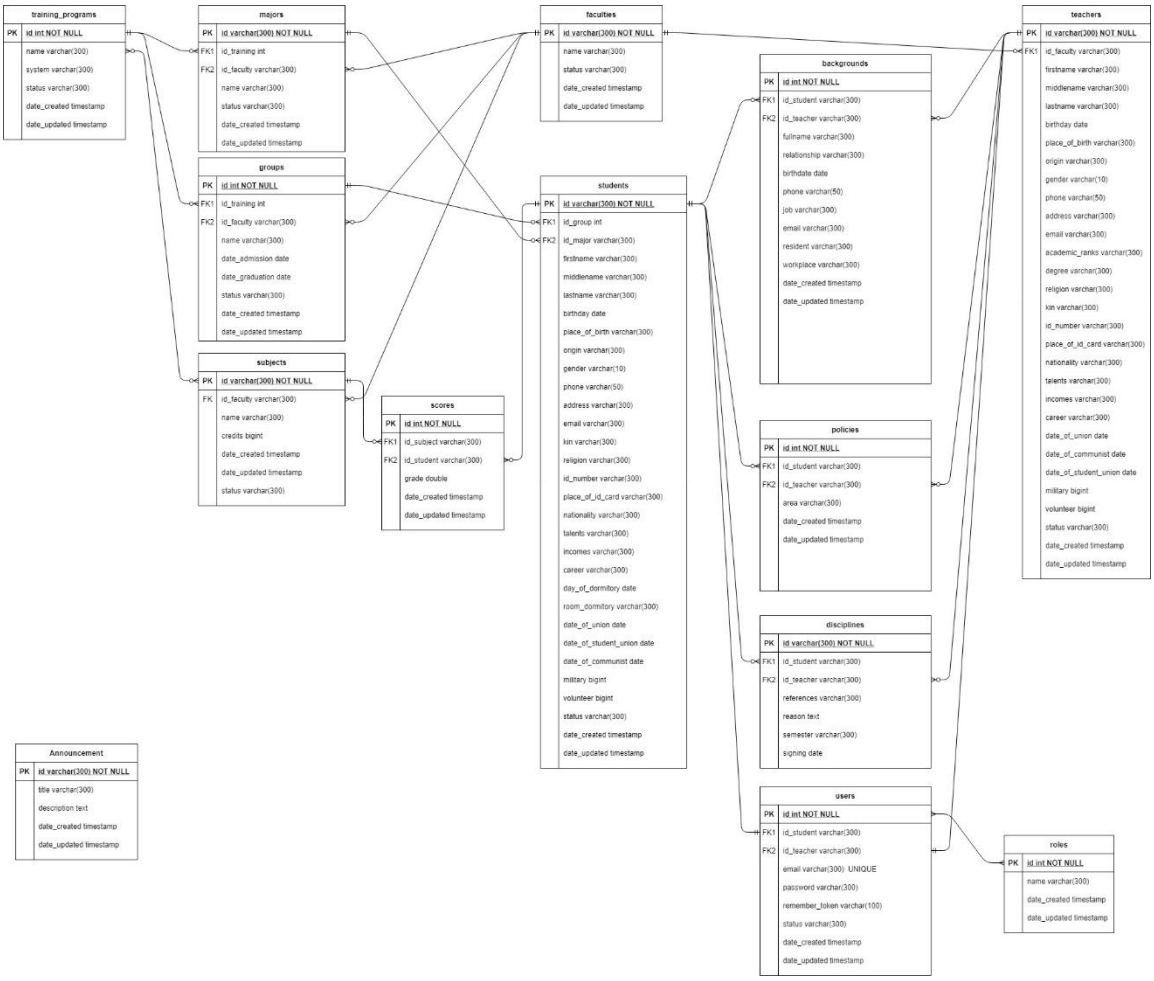

Figure 2.2: Entity Relationship Diagram.

Columns	Type	Constrains	Descriptions
id	int	NOT NULL	Primary key
name	varchar(300)		
system	varchar(300)		
status	varchar(300)		Default value: Đang Mở
date_created	timestamp		
date_updated	timestamp		

Table 2.25: Table training_programs

Columns	Type	Constrains	Descriptions
id	varchar(300)	NOT NULL	Primary key
name	varchar(300)		
status	varchar(300)		Default value: Đang Mở
date_created	timestamp		
date_updated	timestamp		

Table 2.26: Table faculties.

Columns	Type	Constrains	Descriptions
id	varchar(300)	NOT NULL	Primary key
id_training	int	Foreign key references to training_programs	
id_faculty	varchar(300)	Foreign key references to faculties	
name	varchar(300)		
status	varchar(300)		Default value: Đang Mở
date_created	timestamp		
date_updated	timestamp		

Table 2.27: Table majors.

Columns	Type	Constrains	Descriptions
id	int	NOT NULL	Primary key
id_training	int	Foreign key references to training_programs	
id_faculty	varchar(300)	Foreign key references to faculties	
name	varchar(300)		
status	varchar(300)		Default value: Đang Mở
date_admission	date		
date_graduation	date		
date_created	timestamp		
date_updated	timestamp		

Table 2.28: Table groups.

Columns	Type	Constrains	Descriptions
id	varchar(300)	NOT NULL	Primary key
id_faculty	varchar(300)	Foreign key references to faculties	
name	varchar(300)		
credits	bigint		
status	varchar(300)		Default value: Đang Mở
date_admission	date		
date_graduation	date		
date_created	timestamp		
date_updated	timestamp		

Table 2.29: Table subjects.

Columns	Type	Constrains	Descriptions
id_training	int	Foreign key references to training_programs	Show that the relationship between training_programs and subjects is many to many
id_subject	varchar(300)	Foreign key references to subjects	Show that the relationship between training_programs and subjects is many to many

Table 2.30: Table programs_subjects.

Columns	Type	Constrains	Descriptions
id	int	NOT NULL	Primary key
id_group	int	Foreign key references to groups	Show that the relationship between groups and students is one to many
id_major	varchar(300)	Foreign key references to majors	Show that the relationship between majors and students is one to many
firstname	varchar(300)		
middlename	varchar(300)		
lastname	varchar(300)		
birthday	date		
place_of_birth	varchar(300)		
origin	varchar(300)		
gender	varchar(10)		
phone	varchar(50)		
address	varchar(300)		
email	varchar(300)		
kin	varchar(300)		
religion	varchar(300)		

id_number	varchar(300)		
place_of_id_number	varchar(300)		
nationality	varchar(300)		
talents	varchar(300)		
incomes	varchar(300)		
career	varchar(300)		
date_of_dormitory	date		
room_dormitory	varchar(300)		
date_of_union	date		
date_of_student_union	date		
date_of_communist	date		
military	int		
volunteer	int		
status	varchar(300)		Default value: Đi Học
date_created	timestamp		
date_updated	timestamp		

Table 2.31: Table students.

Columns	Type	Constrains	Descriptions
id	int	NOT NULL	Primary key
id_faculty	varchar(300)	Foreign key references to faculties	Show that relationship between faculties and teachers is one to many
firstname	varchar(300)		
middlename	varchar(300)		
lastname	varchar(300)		
birthday	date		
place_of_birth	varchar(300)		
origin	varchar(300)		
gender	varchar(10)		
phone	varchar(50)		
address	varchar(300)		
email	varchar(300)		
kin	varchar(300)		
religion	varchar(300)		
id_number	varchar(300)		
place_of_id_number	varchar(300)		

nationality	varchar(300)		
talents	varchar(300)		
incomes	varchar(300)		
career	varchar(300)		
academic_rank	varchar(300)		
degree	varchar(300)		
date_of_union	date		
date_of_student_union	date		
date_of_communist	date		
military	int		
volunteer	int		
status	varchar(300)		Default value: Đang Công Tác
date_created	timestamp		
date_updated	timestamp		

Table 2.32: Table teachers

Columns	Type	Constrains	Descriptions
id	int	NOT NULL	Primary key
id_subject	varchar(300)	Foreign key references to subjects	Show that relationship between subjects and scores is one to many
id_student	varchar(300)	Foreign key references to students	Show that relationship between students and scores is one to many
grade	double		
date_created	timestamp		
date_updated	timestamp		

Table 2.33: Table scores.

Columns	Type	Constrains	Descriptions
id	int	NOT NULL	Primary key
id_student	varchar(300)		Show that relationship between students and backgrounds is one to many
id_teacher	varchar(300)		Show that relationship between teachers and backgrounds is one to many
fullname	varchar(300)		
relationship	varchar(300)		
birthday	date		
phone	varchar(50)		
job	varchar(300)		
email	varchar(300)		
resident	varchar(300)		
workplace	varchar(300)		
date_created	timestamp		
date_updated	timestamp		

Table 2.34: Table backgrounds.

Columns	Type	Constrains	Descriptions
id	int	NOT NULL	Primary key
id_student	varchar(300)		Show that relationship between students and policies is one to many
id_teacher	varchar(300)		Show that relationship between teachers and policies is one to many
area	varchar(300)		
date_created	timestamp		
date_updated	timestamp		

Table 2.35: Table policies.

Columns	Type	Constrains	Descriptions
id	varchar(300)	NOT NULL	Primary key
id_student	varchar(300)		Show that relationship between students and disciplines is one to many
id_teacher	varchar(300)		Show that relationship between teachers and disciplines is one to many
references	varchar(300)		
reason	varchar(300)		
semester	varchar(300)		
signing	date		
date_created	timestamp		
date_updated	timestamp		

Table 2.36: Table disciplines.

Columns	Type	Constrains	Descriptions
id	int	NOT NULL	Primary key
id_student	varchar(300)	UNIQUE	Show that relationship between students and users is one to many
id_teacher	varchar(300)	UNIQUE	Show that relationship between teachers and users is one to many
email	varchar(300)	UNIQUE	
password	varchar(300)		
remember_token	varchar(300)		
status	varchar(300)		
date_created	timestamp		
date_updated	timestamp		

Table 2.37: Table users

Columns	Type	Constrains	Descriptions
id	int	NOT NULL	Primary key
name	varchar(300)		
date_created	timestamp		
date_updated	timestamp		

Table 2.38: Table roles.

Columns	Type	Constrains	Descriptions
id_user	int	Foreign key references to users	Show that the relationship between users and roles is many to many.
id_role	int	Foreign key references to roles	Show that the relationship between users and roles is many to many.

Table 2.39: Table users_roles

CHAPTER 3 – INTRODUCTION TO LARAVEL

3.1 Concept of MVC architecture

MVC is an abbreviation of Model – View – Controller, an architecture or a software design pattern used in software engineering that makes creating huge applications easy. It does not belong to any programming language or framework, but it is a concept that can be used in creating any kind of application or software in any programming language.

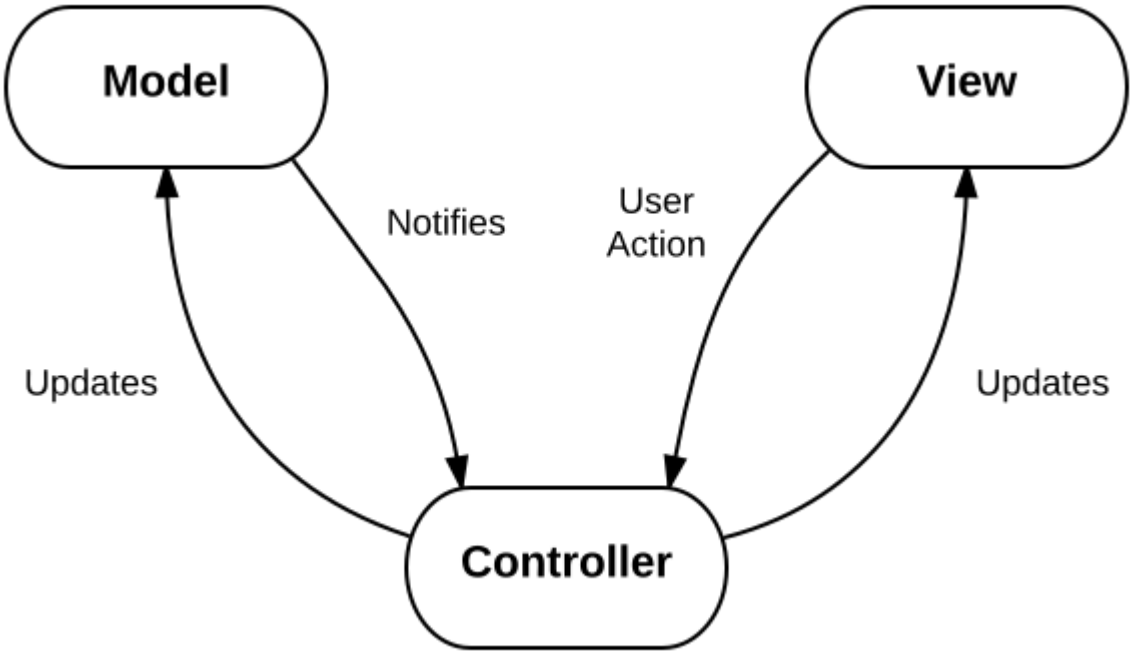

Figure 3.1: MVC architecture.

(Source: https://www.interserver.net/tips/kb/mvc-advantages-disadvantages-mvc/)

3.1.1 Model

Model works directly with the database by fetching, inserting, updating or deleting from it. It does not have to deal with user GUI or data processing.

3.1.2 View

View is the GUI on which users perform some action such as showing, search data to the user on GUI and to respond to the events. In other words, it is used for displaying data and sending the events to the respective controller.

3.1.3 Controller

Controller is the part in which we process the data after getting a request from View and before changing anything in database with Model. it contains the functions that we can program however we want.

3.1.4 Advantages and disadvantages

3.1.4.1 Advantages

Reducing time exponentially because of this architecture seperation.

The development process of the application becomes fast.

Easy for a development team to collaborate and work together.

3.1.4.2 Disadvantages

Hard to develope the small applications.

3.2 Concept of Laravel

Laravel is a powerful and flexible PHP framework provides a structure and starting point for creating a web application. It has a thriving community and a wide ecosystem of tools, and as a result it’s growing in appeal and reach.

3.2.1 Why should we use Laravel ?

Unlike other frameworks, it is easy to see why it’s beneficial to use the individual components or packages that are available to all developers. With packages, someone else is responsible for developing and maintaining an isolated piece of code that has a well – defined job.

It prepackage a collection of third – party components together with custom framework like configuration files, service providers, prescribed directory structures.

The benefit of using a framework in general is that someone has made decisions not just about the individual components, but also about the connectivity of those components together

3.2.1.1 Consistency and Flexibility

Unlike other frameworks, they address this issue by providing a carefully considered answer to the question “Which component should we use ?” and ensuring that the particular components chosen work well together. Additionally, they provide conventions that reduce the amount of code a developer new to the project.

3.2.2 History of Laravel

Laravel is created by Taylor Otwell as an alternative solution for CodeIgniter, provide many more vital features such as authentication and authorization.

Laravel 1 was released in June 2011 and was written completely from scratch. It featured a custom ORM (Eloquent); closure – based routing; a module system for extension; and helpers for forms, validation, authentication,…

Both Laravel 2 and 3 were released in November 2011 and February 2012 respectively. They introduced controllers, unit testing, a command – line tool, an IoC container, Eloquent Relationships and migrations.

Taylor Otwell rewrote the entire framework from the ground up and developed a set of components under the code name Illuminate and released Laravel 4 in May 2013 with an entirely new structure. Instead of bundling the majority of its code as a download, Laravel now pulled the majority of its components from Symfony and the Illuminate components through composer. It also introduced queues, mailing, facades and database seeding and this is the reason why Laravel was now relying on Symfony components.

Laravel 5 was released in February 2015, as a result of significant changes to the end of the Laravel upgrade lifecycle to 4.3. In addition to the current array of new features and enhancements, Laravel 5 also introduces an internal directory tree structure for new application development.

In March 2015, programmers has voted for the most popular PHP framework, Laravel took the first place for it in 2015. It became the most popular PHP projec and the most followed on Github.

3.2.3 Installation.

3.2.3.1 System requirements

PHP $7.1.3^{+}$ for Laravel versions 5.6 to 5.8, PHP $7.0.0^{+}$ for version 5.5, PHP $5.6.4^{+}$ for version 5.4, PHP between 5.6.4 and 7.1.* for version 5.3 or PHP $5.5.9^{+}$ for version 5.2 and 5.1.

The required PHP extensions include OpenSSL, PDO, Mbstring, Tokenizer, XML (Laravel 5.3^{+}), Ctype, JSON (Laravel 5.6^{+}), BCMath (Laravel 5.7^{+}).

Composer.

Apache, MySQL.

3.2.3.2 Install Laravel with composer's create-project feature

We can create a Laravel project directly with the following steps:

1. composer create-project laravel/laravel <project_directory>
2. cd <project_directory>
3. php artisan serve

3.2.3.3 Install Laravel with Laravel installer tool

We can also create a Laravel project from Laravel installer with the following steps:

1. composer global require laravel/installer
2. laravel new <project_directory>
3. cd <project_directory>
4. php artisan serve

3.2.4 Basic components

3.2.4.1 Router

In Laravel application, all web routes must be defined in routes/web.php and routes/api.php is for API routes. Web routes are the routes that will be visited by users. The most basic routes accept a URI and a closure, providing a very simple and expressive method of defining routes and behavior without complicated routing configuration files.

```
Route::get( uri: '/', function () {
    return view( view: 'login');
});
```

Figure 3.2: Example of route definition.

```
Route::get( uri: '/{id}', function () {
    return view( view: 'login');
});
```

Figure 3.3: Example of route definition with parameter.

The router allows to register routes that respond to any HTTP method:

1. Route::get($uri, $callback);
2. Route::post($uri, $callback);
3. Route::put($uri, $callback);
4. Route::patch($uri, $callback);
5. Route::delete($uri, $callback);

Any routes with POST, PUT, PATCH or DELETE method should include a csrf token field:

```
<form method="POST" action="/profile">
    @csrf
    ...
</form>
```

We can group several routes together and apply any shared configuration settings once to the entire group in order to reduce this duplication. We can use this operation for prefixing path, restricting.

```
Route::group(function () {
    Route::get('hello', function () {
        return 'Hello';
    });
    Route::get('world', function () {
        return 'World';
    });
});
```

Figure 3.4: Basic example of grouping routes.

```
Route::prefix('dashboard')->group(function () {
    Route::get('/', function () {
        // Handles the path /dashboard
    });
    Route::get('users', function () {
        // Handles the path /dashboard/users
    });
});
```

Figure 3.5: Path prefixing example

```
Route::middleware('auth')->group(function() {
    Route::get('dashboard', function () {
        return view('dashboard');
    });
    Route::get('account', function () {
        return view('account');
    });
});
```

Figure 3.6: Restricting routes

3.2.4.2 Controller

Create a controller: php artisan make:controller <ControllerName>. For example of creating TaskController: php artisan make:controller TaskController.

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class TasksController extends Controller
{
    //
}
```

Figure 3.7: Default generated controller.

In order to assign controller's actions to the route, we need to define a public method to this controller.

```php
namespace App\Http\Controllers;

class TasksController extends Controller
{
    public function index()
    {
        return 'Hello, World!';
```

Figure 3.8: Simple definition method of controller.

```php
Route::get( uri: '/', [TaskController::class, 'index']);
```

Figure 3.9: Route for Controller.

This controller's method loads the resources/views/announcement/view and passes it variables named id, announcement which contains the result of Eloquent method and route parameter.

```php
public function viewAnnouncement($id) {
    $announcement = Announcement::find($id);
    return view( view: 'announcement.view', [
        'id' => $id,
        'announcement' => $announcement
    ]);
}
```

Figure 3.10: Load a view with controller.

3.2.4.3 Blade template

Blade is a Laravel's template engine which inspired by Microsoft's Razor engine. It's boasts a concise syntax, a shallow learning curve, a powerful and intuitive inheritance model and easy extensibility.

Blade uses curly braces for its echo and introduces a convention in which its custom tags called "directives", are prefixed with an @. These directives are an alternative solutions to the original PHP syntax, can compiles it to the embedded PHP.

#	**PHP syntax**	**Blade's directives**
01	<?php echo $a ?>	{{$a}}
Conditional structure		

02	<?php
	if(condition)
	…
	else if (condition)
	…
	else
	…
	?>
		@if (condition)
		….
		
		@elseif (condition)
		…..
		@else
		……
		@endif

03	<?php switch($var) case 1: break; case 2: break; ?>	@switch($var) @case(1): @break @case(2) @break @endswitch
Loop structure		

04	<?php
	for ($i = 0; $i <= n; $i++)
	….
	if (condition)
	break;
	if (condition)
	continue;
	?>
		@for ($i = 0; $i <= n; $i++)
		…
		@break (condition)
		@continue (condition)
		@endfor

05	<?php while ($i <= n) …. ?>	@while ($i <= n) … @endwhile
06	<?php foreach ($array as $a) …. ?>	@foreach ($array as $a) … @endforeach
Page customization		
07		@extends()
08		@section()
09	<?php include(‘’) ?>	@include()

Table 3.1: Difference of writing styles between PHP and Blades.

```
<h5>{{$student->id}}</h5>
```

Figure 3.11: Example of echoing variable.

```
@if (count($talks) === 1)
    There is one talk at this time period.
@elseif (count($talks) === 0)
    There are no talks at this time period.
@else
    There are {{ count($talks) }} talks at this time period.
@endif
```

Figure 3.12: Basic example of conditional directives.

```
@for ($i = 0; $i < $talk->slotsCount(); $i++)
    The number is {{ $i }}<br>
@endfor
```

Figure 3.13: Example of loop directive.

3.2.4.4 Eloquent

Eloquent is an ORM that interacts with the database. When using Eloquent, each database table has a corresponding Model that used to interact with a table from a database and managing the relationships between 2 tables.

Eloquent models allow to insert, update and delete records from a table.

To create a model, use artisan command php artisan make:model <model_name> to generate a new model or to generate it with a database migration php artisan make:model <model_name> --m, this new model will be created at app/Models.

Specify the model's information by defining these property below on the model.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Flight extends Model
{
    /**
     * The table associated with the model.
     *
     * @var string
     */
    protected $table = 'my_flights';
}
```

Figure 3.14: table's name in database with table property.

(Source: https://laravel.com/docs/8.x/eloquent)

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Flight extends Model
{
    /**
     * The primary key associated with the table.
     *
     * @var string
     */
    protected $primaryKey = 'flight_id';
}
```

Figure 3.15: primary key of table with primaryKey property.

(Source: https://laravel.com/docs/8.x/eloquent)

```php
<?php

class Flight extends Model
{
    /**
     * Indicates if the model's ID is auto-incrementing.
     *
     * @var bool
     */
    public $incrementing = false;
}
```

Figure 3.16: id incremental with incrementing property.

(Source: https://laravel.com/docs/8.x/eloquent)

We can query a model using one of these steps below:

- Flight::all(): get all records from model “Flight”.
- Flight::find($id): get a record with a corresponding id from model “Flight”.
- Flight::where(‘id, $id)->get(): get all records from model “Flight”.
- Flight::where(‘id, $id)->first(): get a records from model “Flight”, it is similar with find.

We can insert into table using one of these steps below:

- Declare a model and assign:

```
$flight = new Flight();
$flight->name = “Vietjet”;
$flight->save()
```

- Use array as an argument of model:

```
$flight = new Flight([‘name’ => ‘Vietjet’]);
```

- Use create method: Flight::create([‘name’ => ‘Vietjet’]);

Update a record:

- Flight::update([‘name’ => ‘Jetstar’]);
- Find a model and assign:

```
$flight = Flight::find($id);
$flight->name = “Vietjet”;
$flight->save()
```

- Update with condition:

```
Flight::where('active', 1)
    ->where('destination', 'San Diego')
    ->update(['delayed' => 1]);
```

One to one and one to many relationships:

Figure 3.17: Example diagrams with one to one.

(Source: http://laravel.at.jeffsbox.eu/laravel-5-eloquent-relationship-types-one-to-one)

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * Get the phone associated with the user.
     */
    public function phone()
    {
        return $this->hasOne(Phone::class);
    }
}
```

Figure 3.18: Parent model definition for one to one relationship.

(Source: https://laravel.com/docs/8.x/eloquent-relationships#one-to-one)

```php
return $this->hasOne(Phone::class, 'foreign_key', 'local_key');
```

Figure 3.19: Parent model definition with parameters for one to one relationship.

(Source: https://laravel.com/docs/8.x/eloquent-relationships#one-to-one)

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Phone extends Model
{
    /**
     * Get the user that owns the phone.
     */
    public function user()
    {
        return $this->belongsTo(User::class);
    }
}
```

Figure 3.20: Child model definition for one to one relationship.

(Source: https://laravel.com/docs/8.x/eloquent-relationships#one-to-one)

```php
/**
 * Get the user that owns the phone.
 */
public function user()
{
    return $this->belongsTo(User::class, 'foreign_key');
}
```

Figure 3.21: Child model definition with a parameter for one to one relationship.

(Source: https://laravel.com/docs/8.x/eloquent-relationships#one-to-one)

Figure 3.22: Example of one to many definition.

(http://laravel.at.jeffsbox.eu/laravel-5-eloquent-relationship-types-one-to-many-2)

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    /**
     * Get the comments for the blog post.
     */
    public function comments()
    {
        return $this->hasMany(Comment::class);
    }
}
```

Figure 3.23: Parent model definition for one to many relationship

(Source: https://laravel.com/docs/8.x/eloquent-relationships#one-to-many)

```php
return $this->hasMany(Comment::class, 'foreign_key', 'local_key');
```

Figure 3.24: Parent model definition with parameters for one to many relationship

(Source: https://laravel.com/docs/8.x/eloquent-relationships#one-to-many)

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Comment extends Model
{
    /**
     * Get the post that owns the comment.
     */
    public function post()
    {
        return $this->belongsTo(Post::class);
    }
}
```

Figure 3.25: Child model definition for one to many relationship.

(Source: https://laravel.com/docs/8.x/eloquent-relationships#one-to-many)

```php
/**
 * Get the post that owns the comment.
 */
public function post()
{
    return $this->belongsTo(Post::class, 'foreign_key');
}
```

Figure 3.26: Child model definition with parameter for one to many relationship.

(Source: https://laravel.com/docs/8.x/eloquent-relationships#one-to-many)

Eloquent provides convenient methods for adding new models to relationships. Instead of manually to setting the attribute with forein key constrain, we can use save method to insert the child model with an example belows:

```
use App\Models\Comment;
use App\Models\Post;

$comment = new Comment(['message' => 'A new comment.']);

$post = Post::find(1);

$post->comments()->save($comment);
```

Figure 3.27: Insert one to one or one to many relationship using save method.

(Source: https://laravel.com/docs/8.x/eloquent-relationships#the-save-method)

To insert multiple relation, we may use the saveMany method with an example belows:

```
$post = Post::find(1);

$post->comments()->saveMany([
    new Comment(['message' => 'A new comment.']),
    new Comment(['message' => 'Another new comment.']),
]);
```

Figure 3.28: Insert multiple one to one or one to many relationship using saveMany method.

(Source: https://laravel.com/docs/8.x/eloquent-relationships#the-save-method)

In addition to the save and saveMany method, we can also use create or createMany, its procedure is as same as save and saveMany but the difference between

them is the parameters: save/saveMany accepts a full instance of Eloquent model while create and createMany accepts an array.

```
use App\Models\Post;

$post = Post::find(1);

$comment = $post->comments()->create([
    'message' => 'A new comment.',
]);
```

Figure 3.29: Insert one to one or one to many relationship using create method.

(Source: https://laravel.com/docs/8.x/eloquent-relationships#the-create-method)

```
$post = Post::find(1);

$post->comments()->createMany([
    ['message' => 'A new comment.'],
    ['message' => 'Another new comment.'],
]);
```

Figure 3.30: Insert multiple one to one or one to many relation with createMany.

(Source: https://laravel.com/docs/8.x/eloquent-relationships#the-create-method)

associate will set the forein key on the child model and allow to assign a child model to a new parent model.

```php
use App\Models\Account;

$account = Account::find(10);

$user->account()->associate($account);

$user->save();
```

Figure 3.31: Insert into one to one or one to many relationship from child model.

(Source: https://laravel.com/docs/8.x/eloquent-relationships#updating-belongs-to-relationships)

dissociate() will set the relationship's forein key to null and allow to remove relationship from parent to child,

```php
$user->account()->dissociate();

$user->save();
```

Figure 3.32: Remove relationship from parent to child (one to one or one to many)

(Source: https://laravel.com/docs/8.x/eloquent-relationships#updating-belongs-to-relationships)

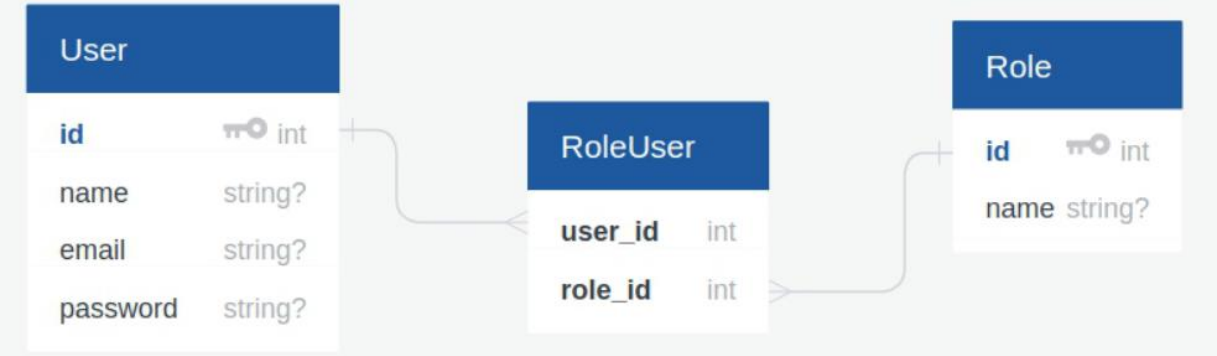

Figure 3.33: Example of many to many relationship.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * The roles that belong to the user.
     */
    public function roles()
    {
        return $this->belongsToMany(Role::class);
    }
}
```

Figure 3.34: Parent model definition of many to many relationship.

(Source: https://laravel.com/docs/8.x/eloquent-relationships#many-to-many)

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Role extends Model
{
    /**
     * The users that belong to the role.
     */
    public function users()
    {
        return $this->belongsToMany(User::class);
    }
}
```

Figure 3.35: Child model for many to many relationship.

(Source: https://laravel.com/docs/8.x/eloquent-relationships#many-to-many)

```php
return $this->belongsToMany(Role::class, 'role_user', 'user_id', 'role_id');
```

Figure 3.36: Parameters definition for many to many relationship.

(Source: https://laravel.com/docs/8.x/eloquent-relationships#many-to-many)

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Relations\Pivot;

class RoleUser extends Pivot
{
    //
}
```

Figure 3.37: Pivot model definition.

(Source: https://laravel.com/docs/8.x/eloquent-relationships#defining-custom-intermediate-table-models)

Insert into relation by using 3 ways: attach method is to add a relationship, sync is to add the specific records from an array, if any elements that are not in the given array, they will be removed from the intermediate table (if we don't want to delete these elements, we can use syncWithOutDetaching), toggle is to add the elements that are given, if it is currently detached, it will be attached.

```php
use App\Models\User;

$user = User::find(1);

$user->roles()->attach($roleId);
```

Figure 3.38: simple insertion in many to many using attach.

(Source: https://laravel.com/docs/8.x/eloquent-relationships#updating-many-to-many-relationships)

```php
$user->roles()->attach($roleId, ['expires' => $expires]);
```

Figure 3.39: insert into multiple columns of pivot table in many to many using attach.

(Source: https://laravel.com/docs/8.x/eloquent-relationships#updating-many-to-many-relationships)

```php
$user->roles()->sync([1, 2, 3]);
```

Figure 3.40: Insert relationship with sync.

```php
// Detach a single role from the user...
$user->roles()->detach($roleId);

// Detach all roles from the user...
$user->roles()->detach();
```

Figure 3.41: Delete relationship using detach.

(Source: https://laravel.com/docs/8.x/eloquent-relationships#updating-many-to-many-relationships)

```php
$user = User::find(1);

$user->roles()->updateExistingPivot($roleId, [
    'active' => false,
]);
```

Figure 3.42: Update a record on the pivot table.

(Source: https://laravel.com/docs/8.x/eloquent-relationships#updating-many-to-many-relationships)

3.3 Product

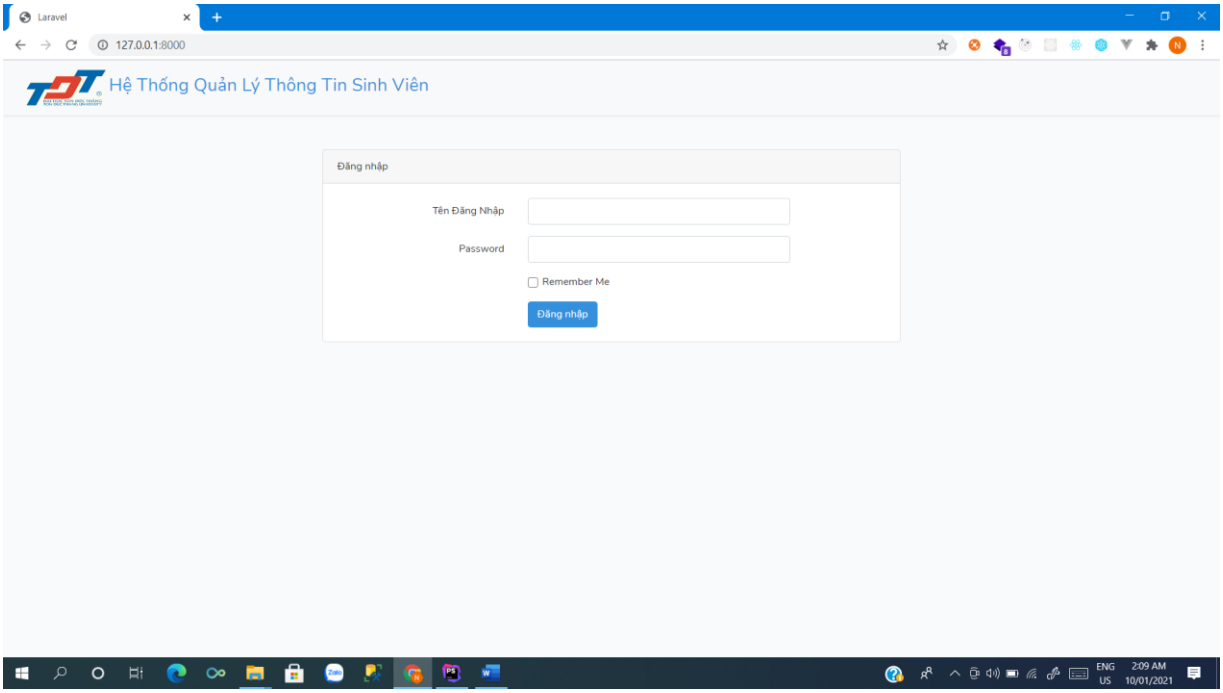

Figure 3.43: Home page with login form.

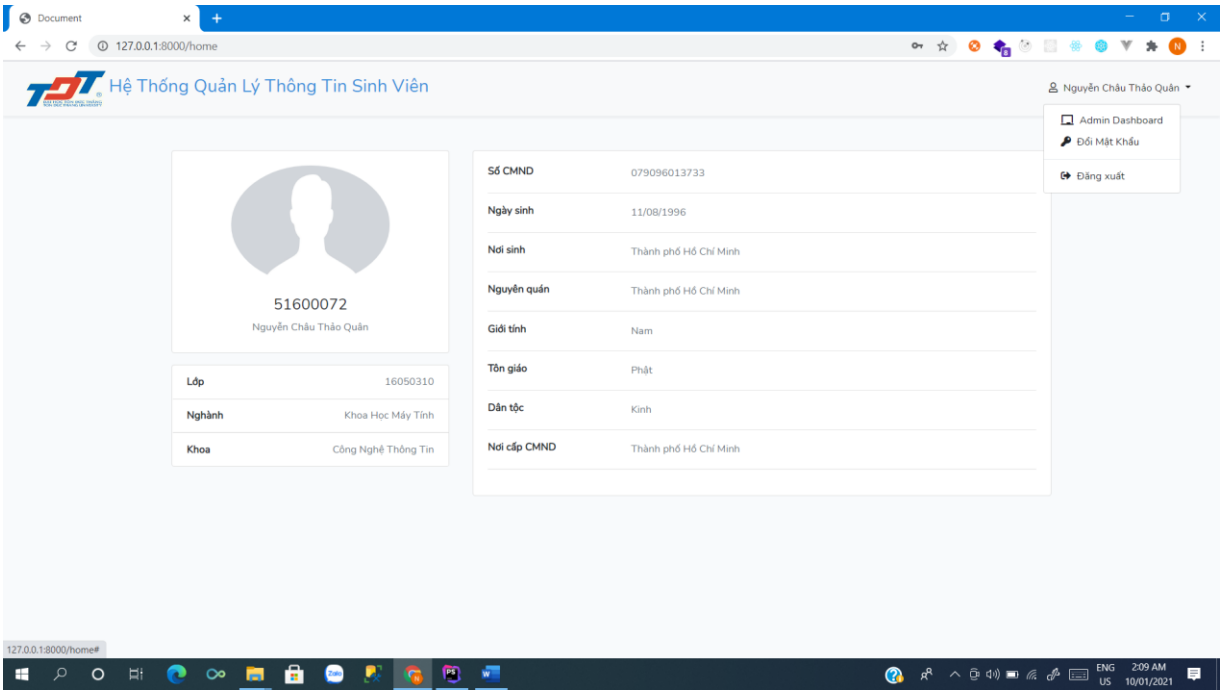

Figure 3.44: Profile page after login successfully.

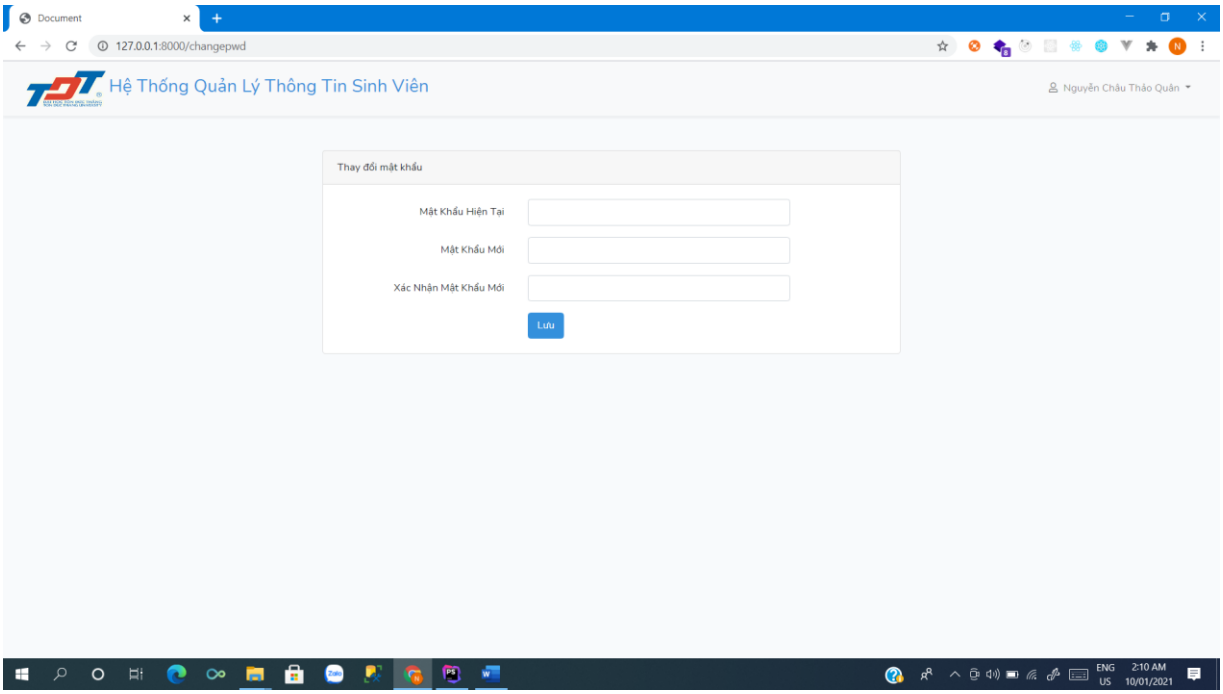

Figure 3.45: Form to change password.

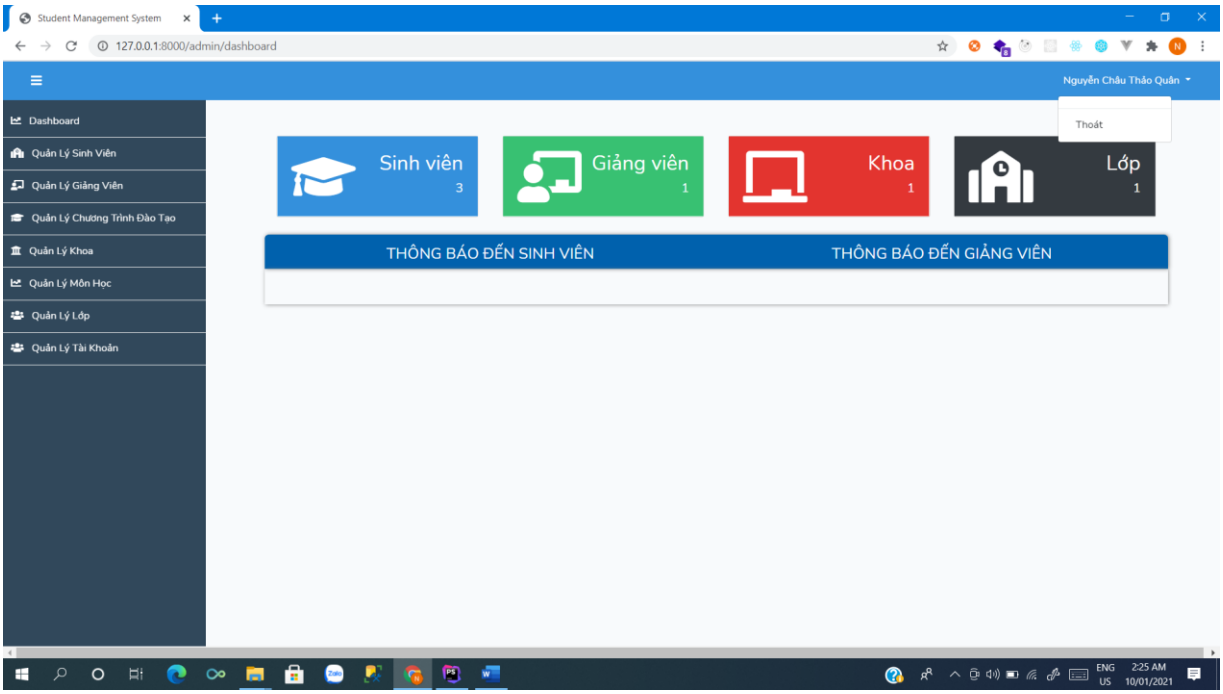

Figure 3.46: Admin home page.

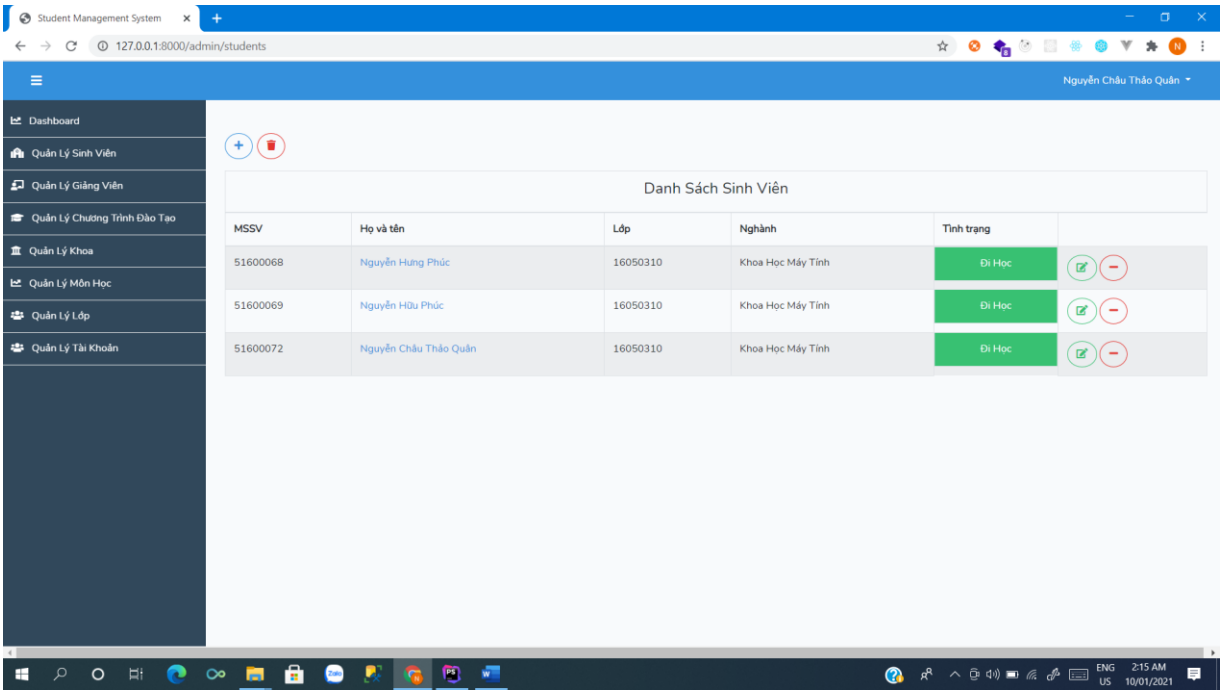

Figure 3.47: List of students.

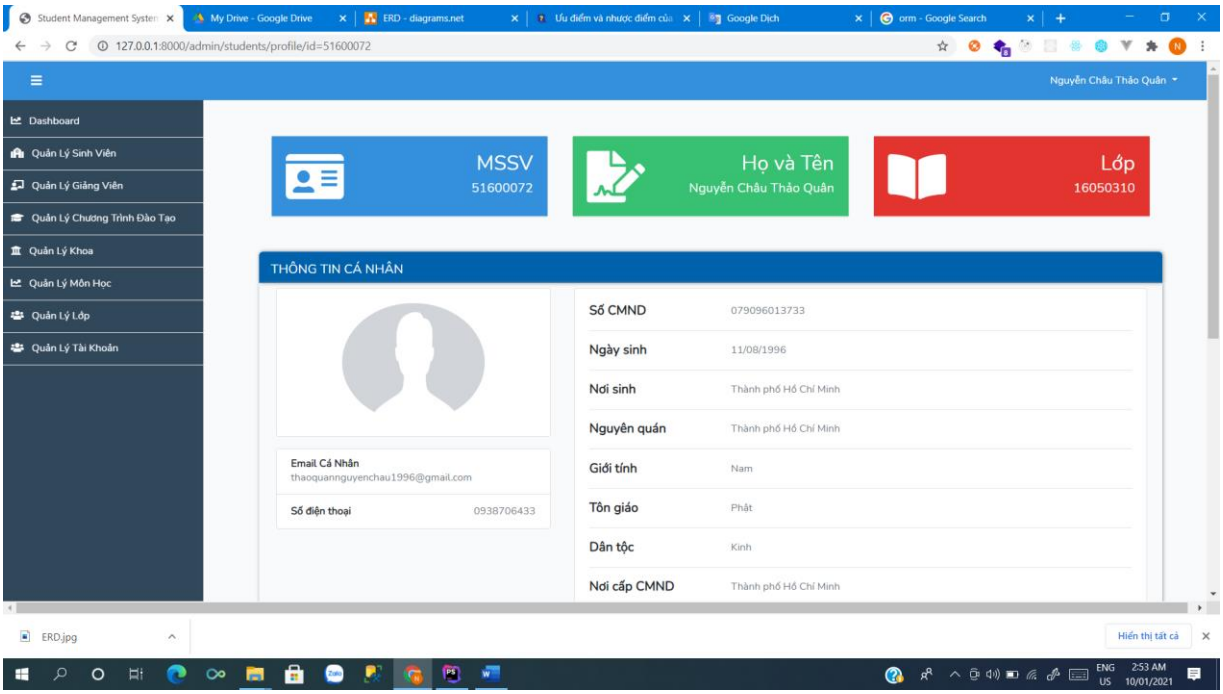

Figure 3.48: Student’s profile

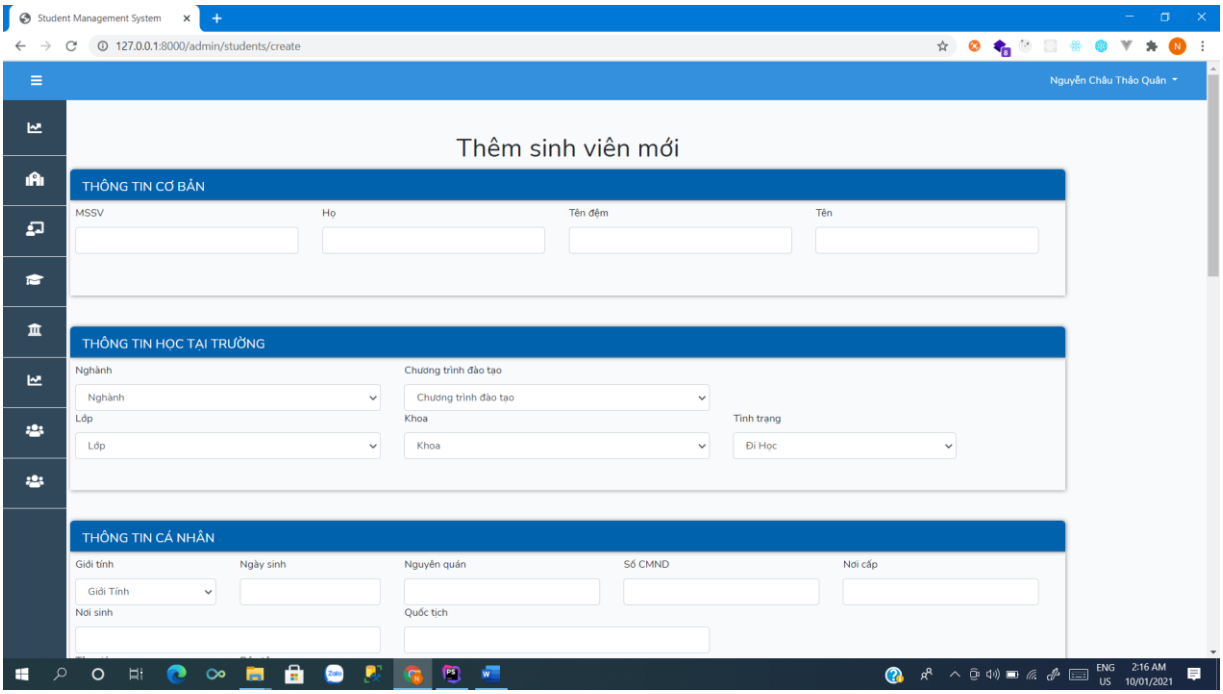

Figure 3.49: Form to add new student's informations.

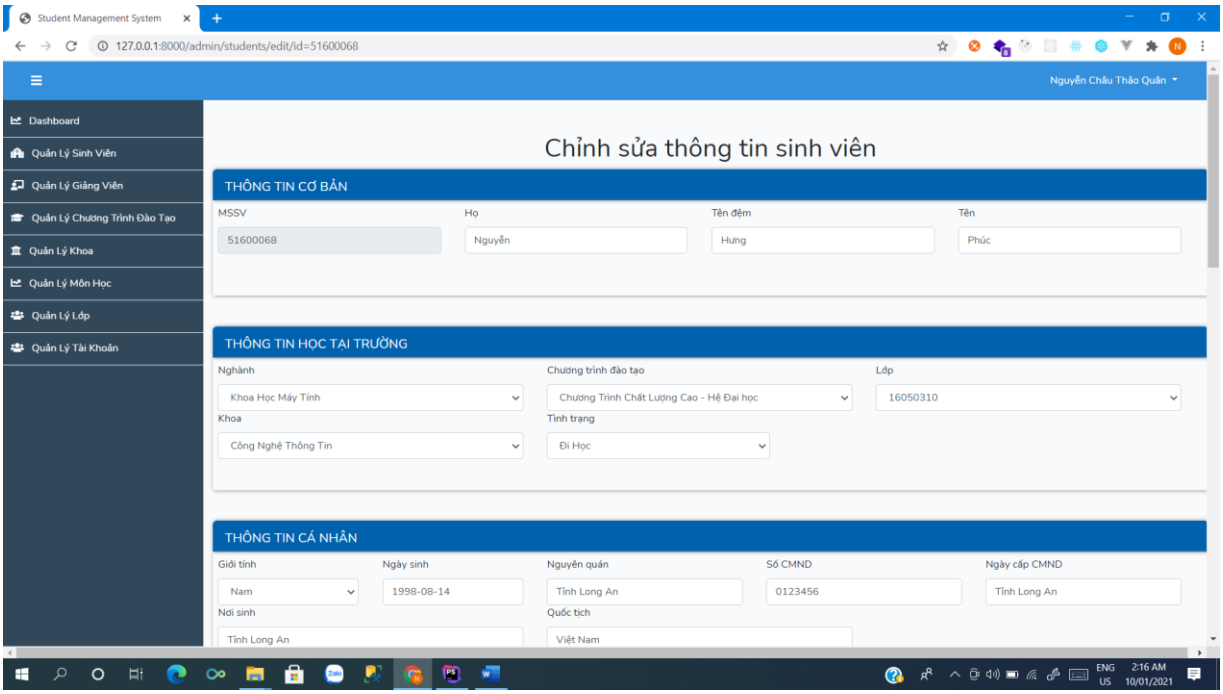

Figure 3.50: Form to update student's informations.

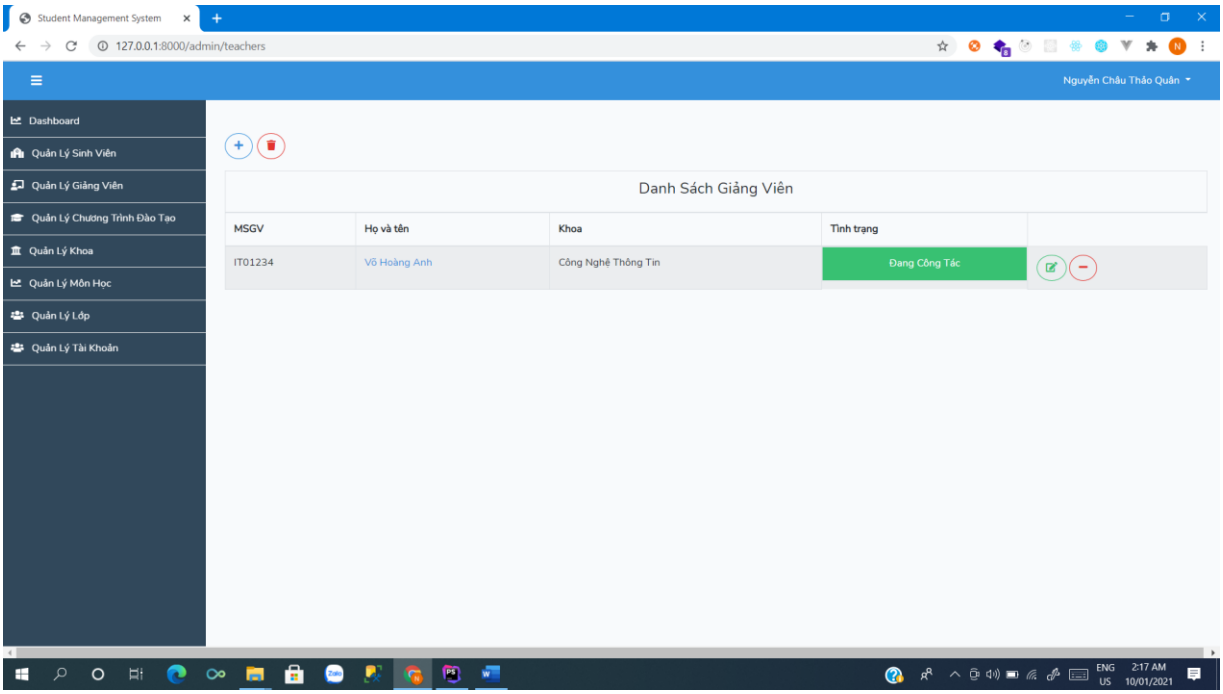

Figure 3.51: List of teachers.

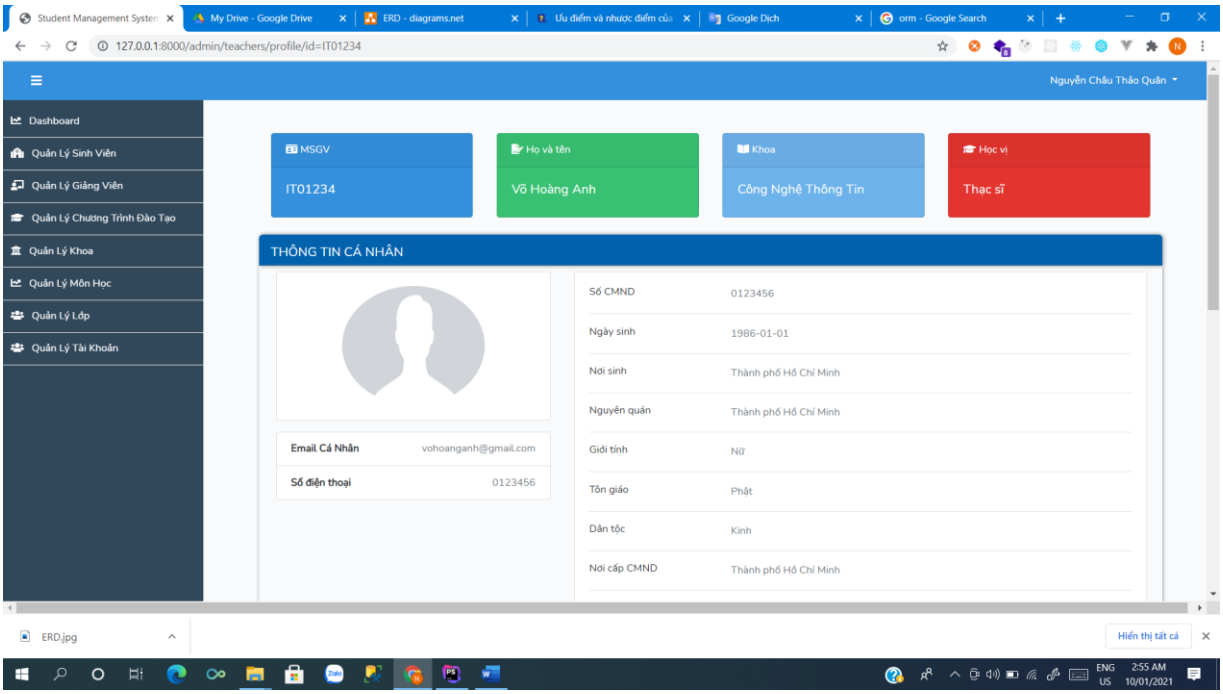

Figure 3.52: Teacher’s profile

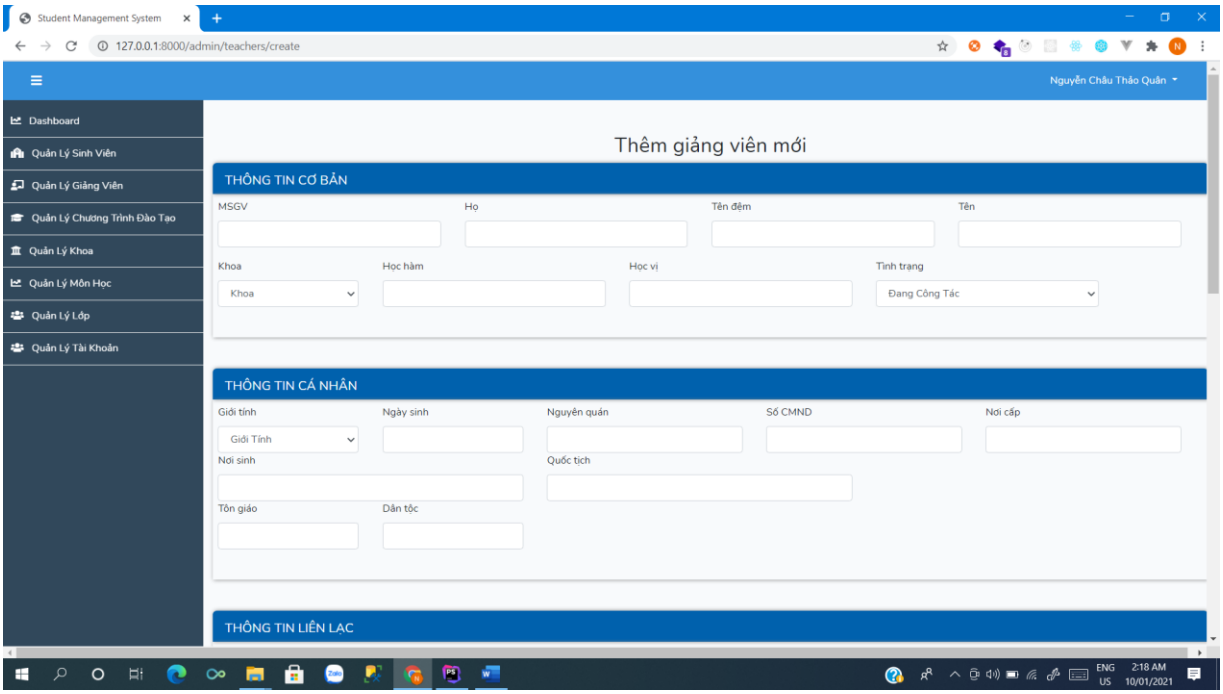

Figure 3.53: Form to add new teacher's informations.

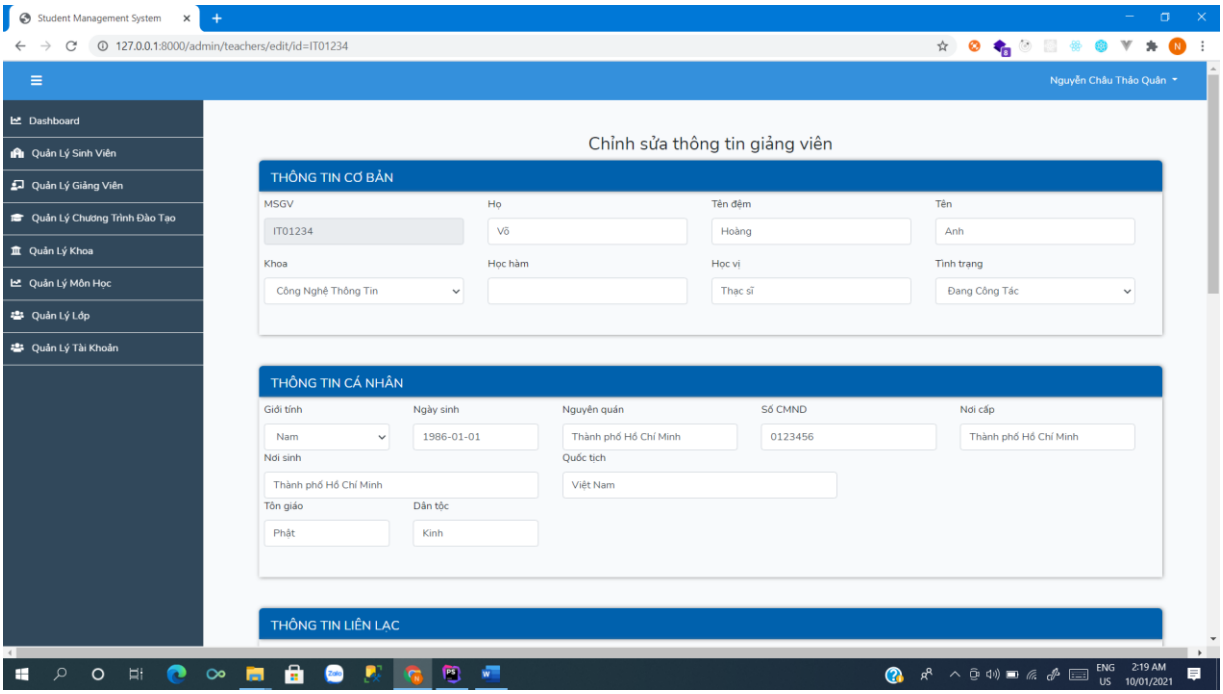

Figure 3.54: Form to update teacher's information.

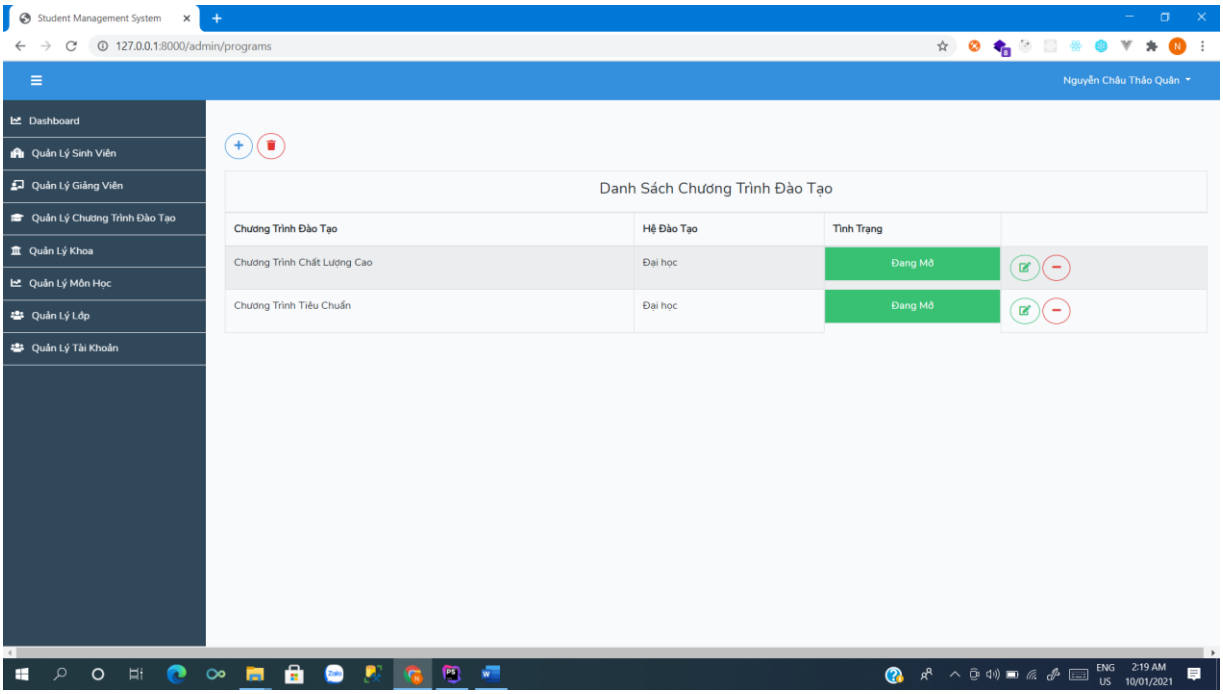

Figure 3.55: List of education programs.

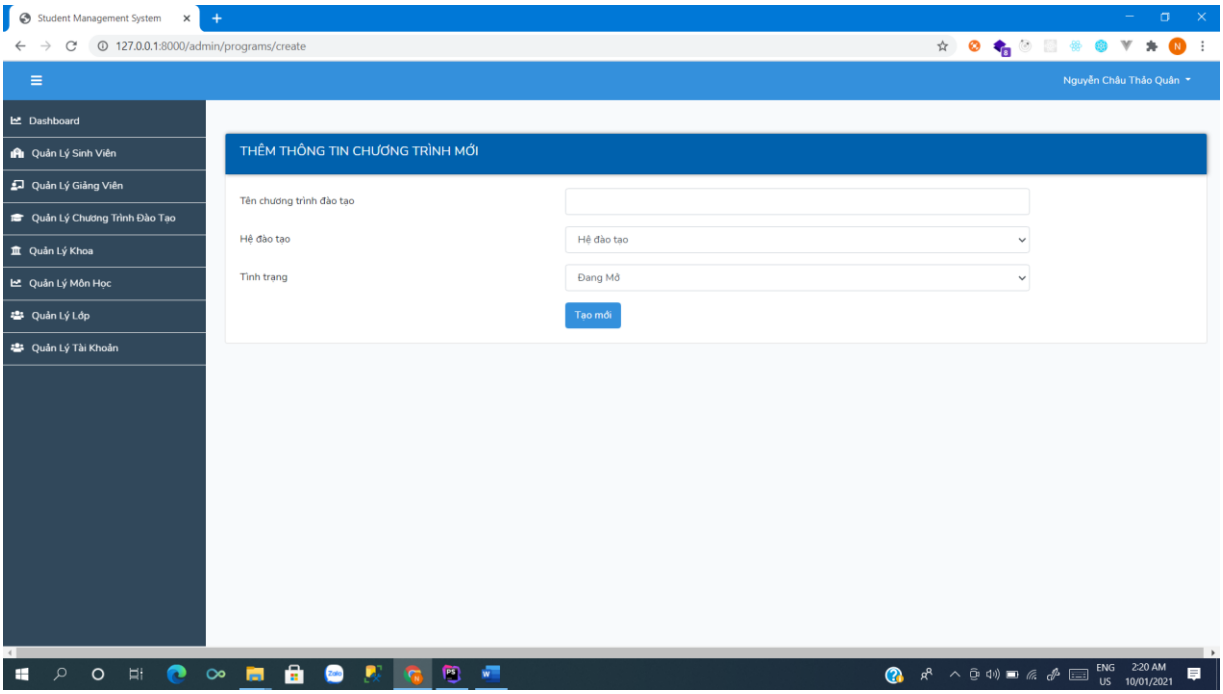

Figure 3.56: Form to add new education program informations.

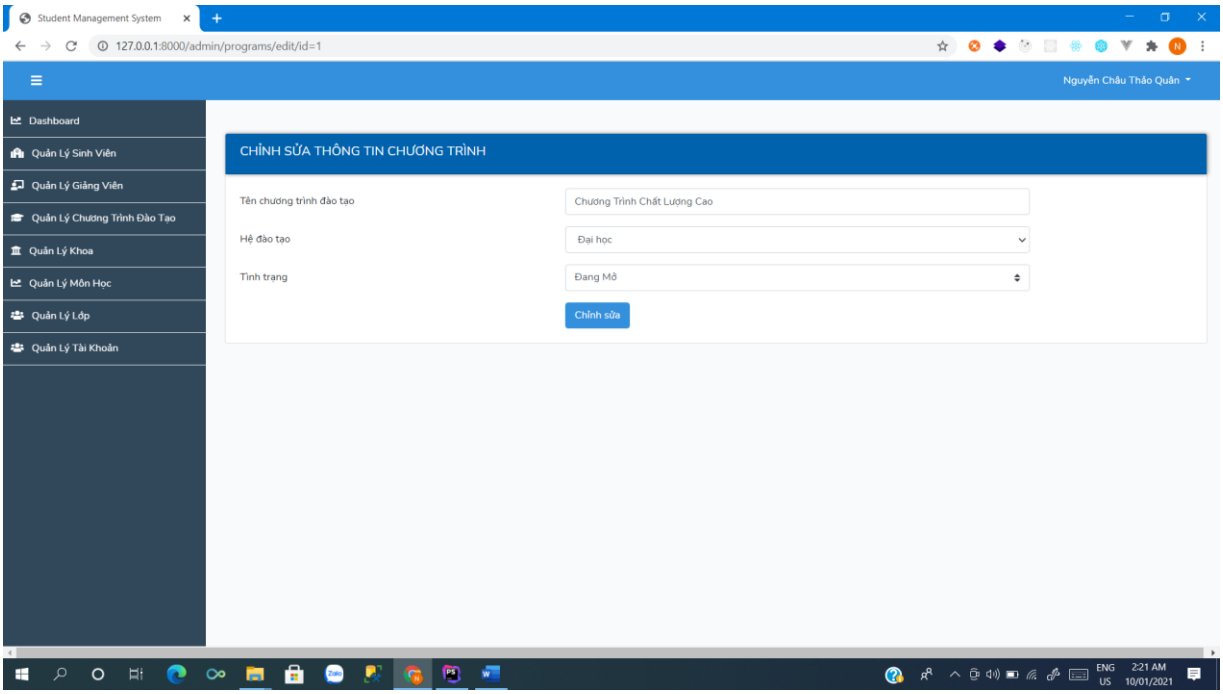

Figure 3.57: Form to update new program's informations.

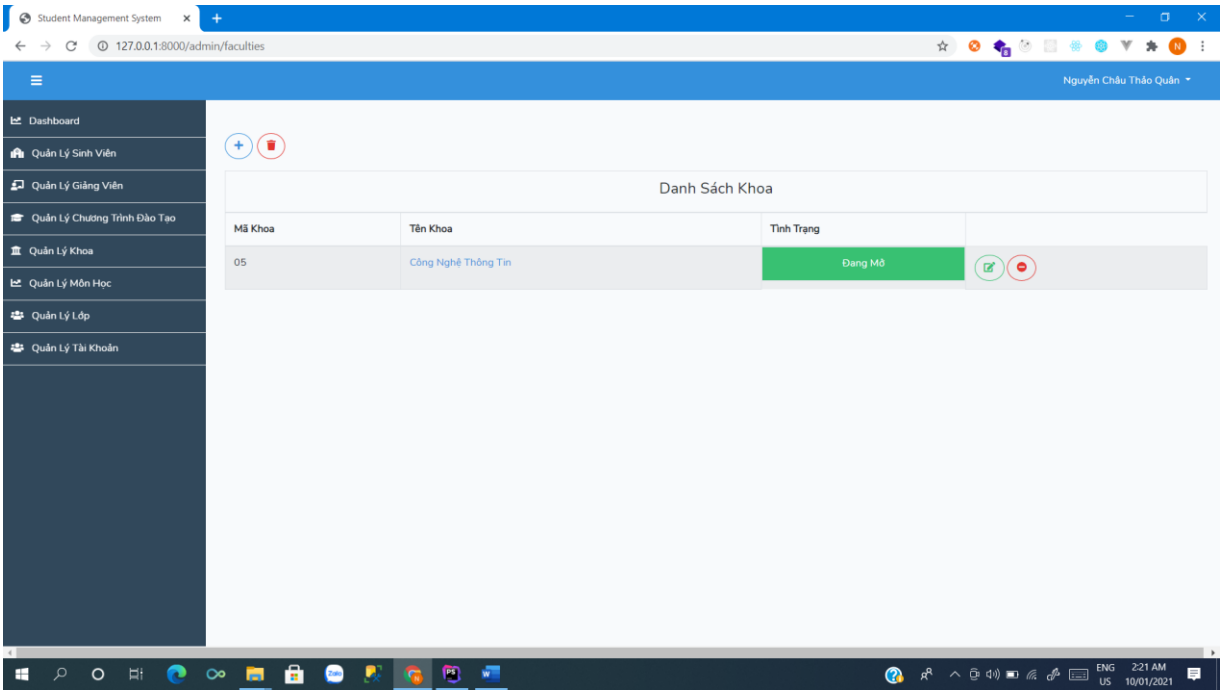

Figure 3.58: List of faculties.

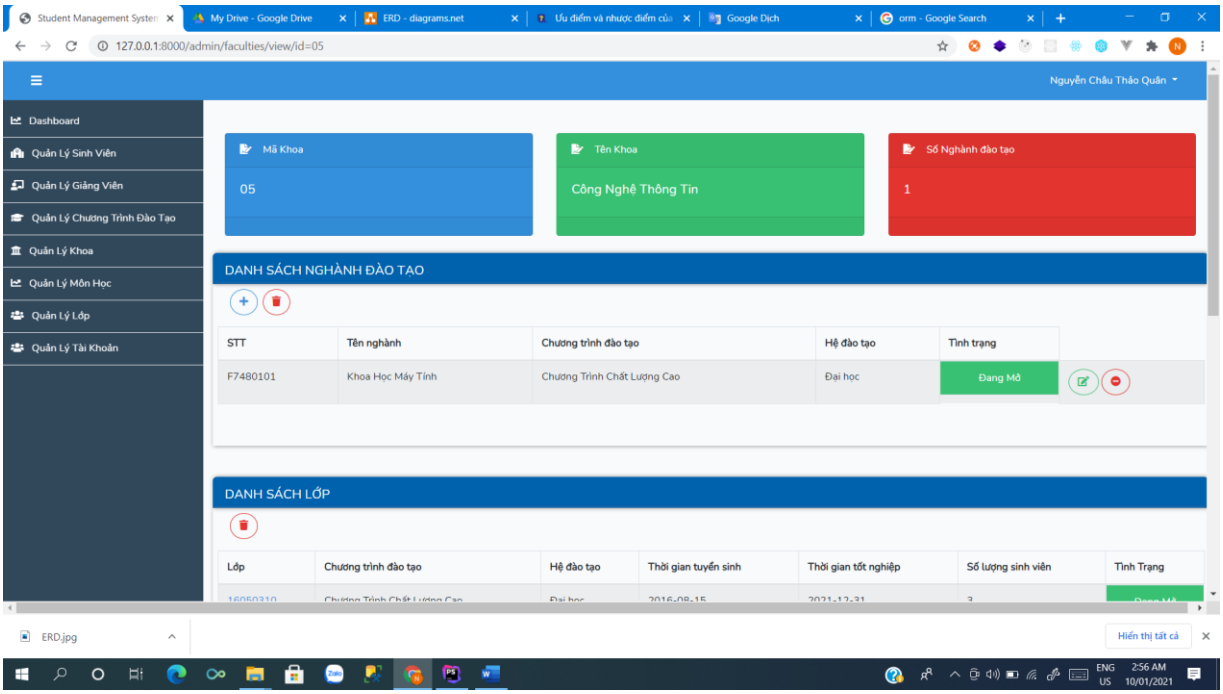

Figure 3.59: Faculty's detail

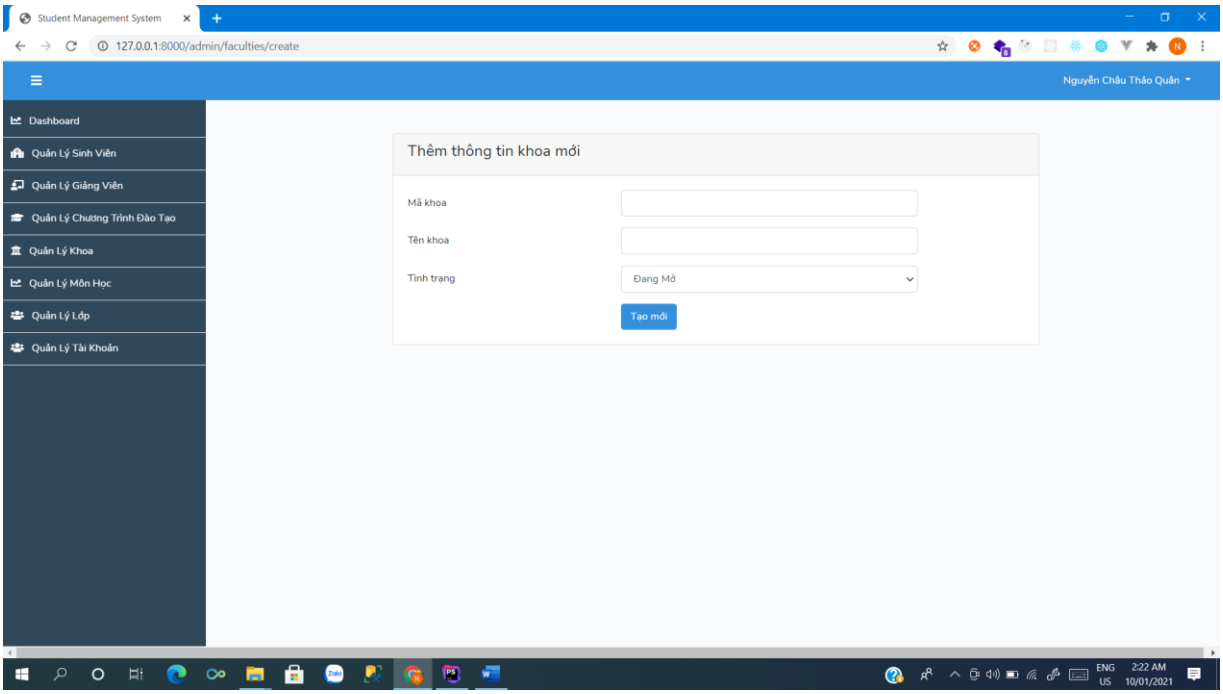

Figure 3.60: Form to add a new faculty.

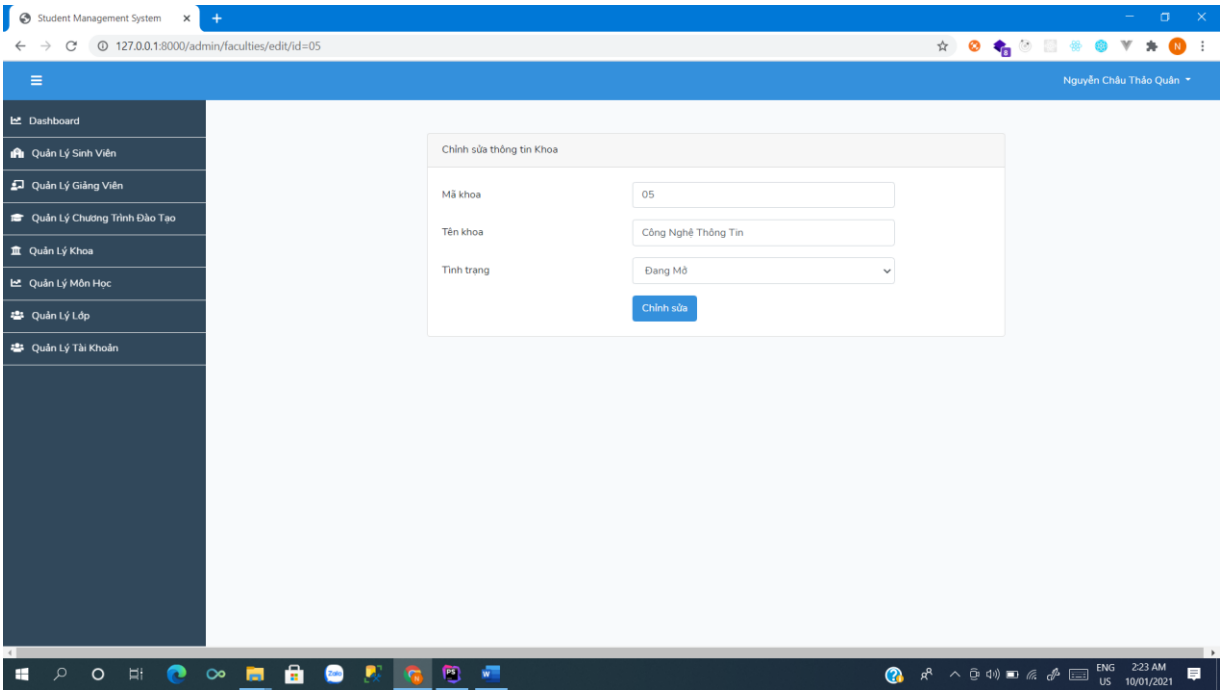

Figure 3.61: Form to update faculty's information.

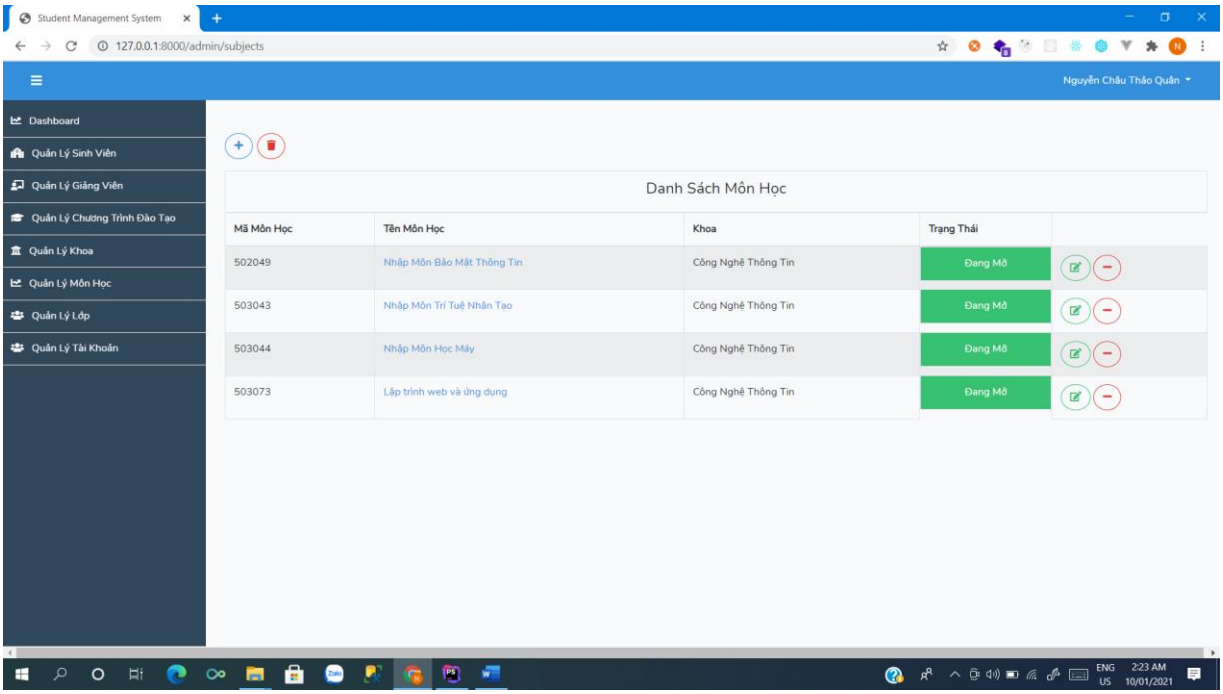

Figure 3.62: List of subjects.

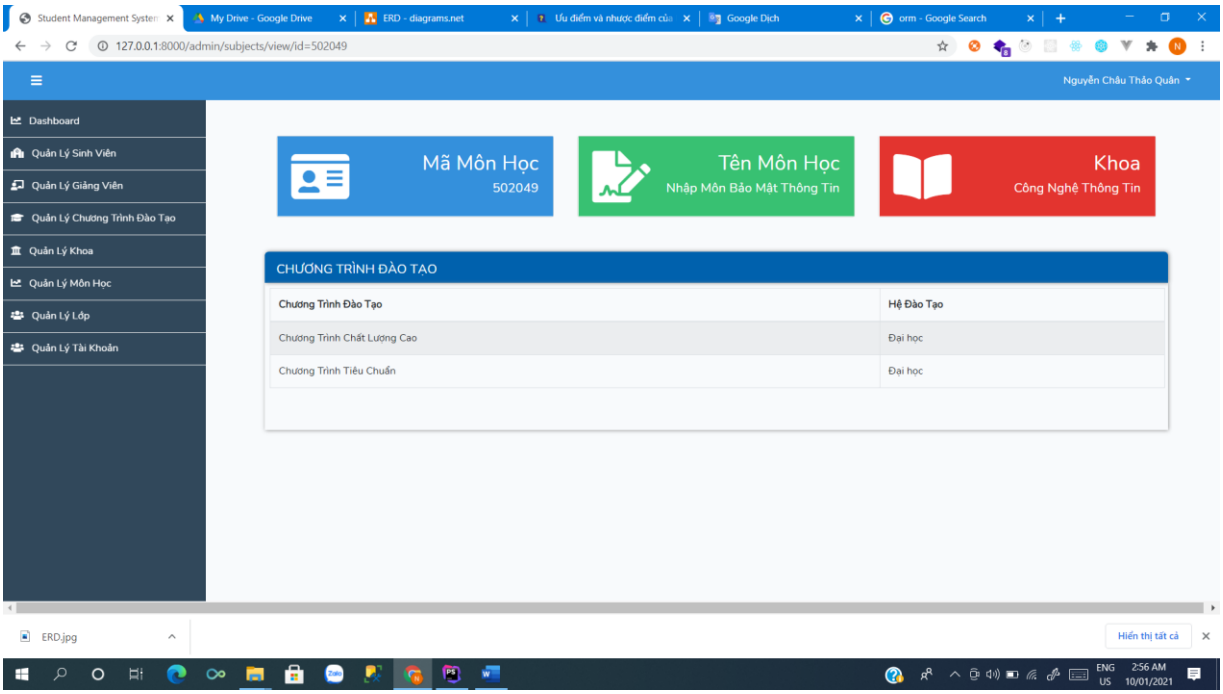

Figure 3.63: Subject's detail.

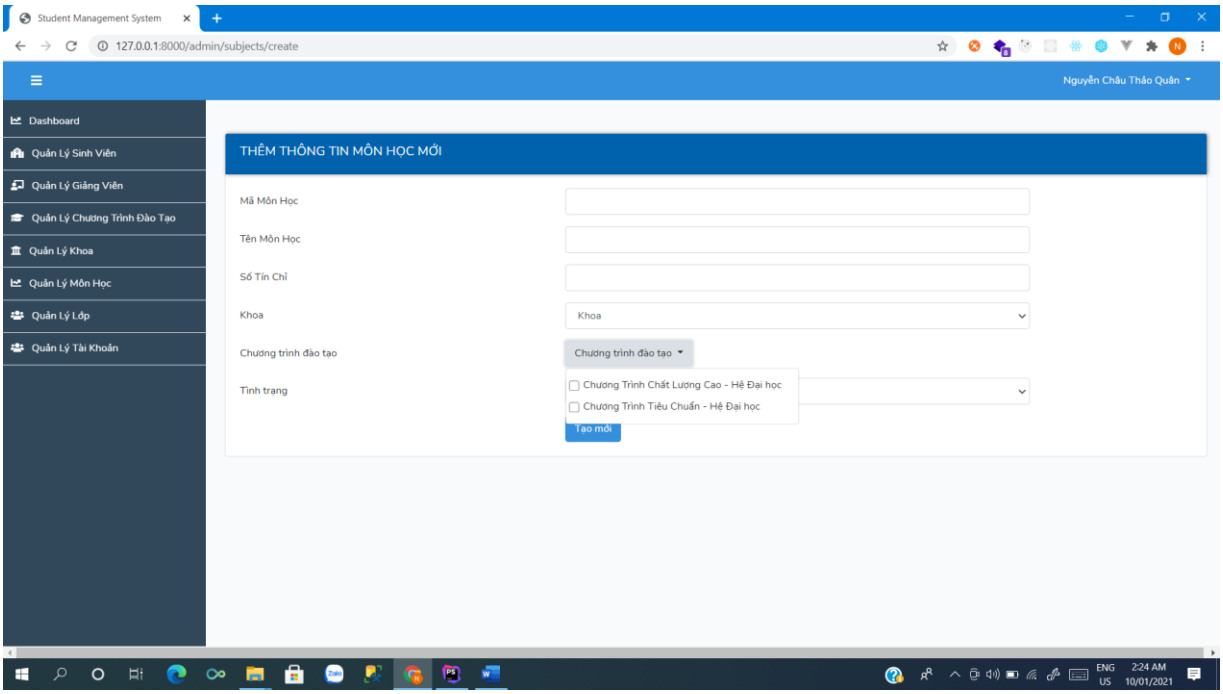

Figure 3.64: Form to add new subjects.

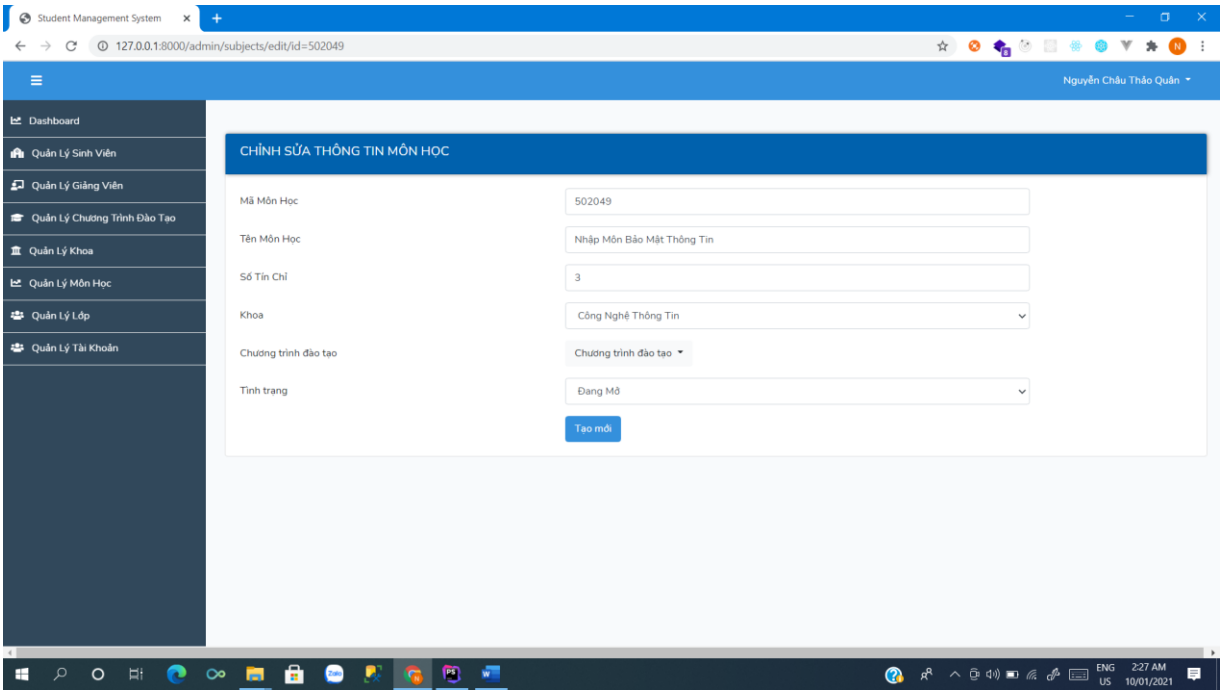

Figure 3.65: Form to update subject.

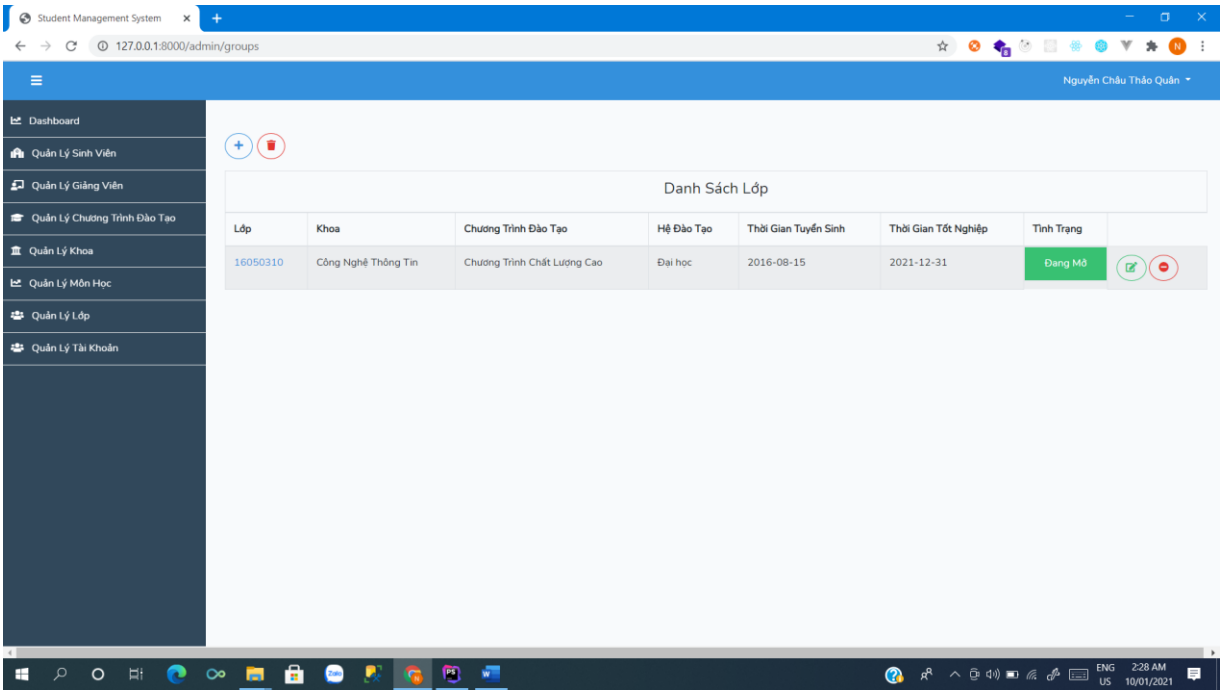

Figure 3.66: List of groups.

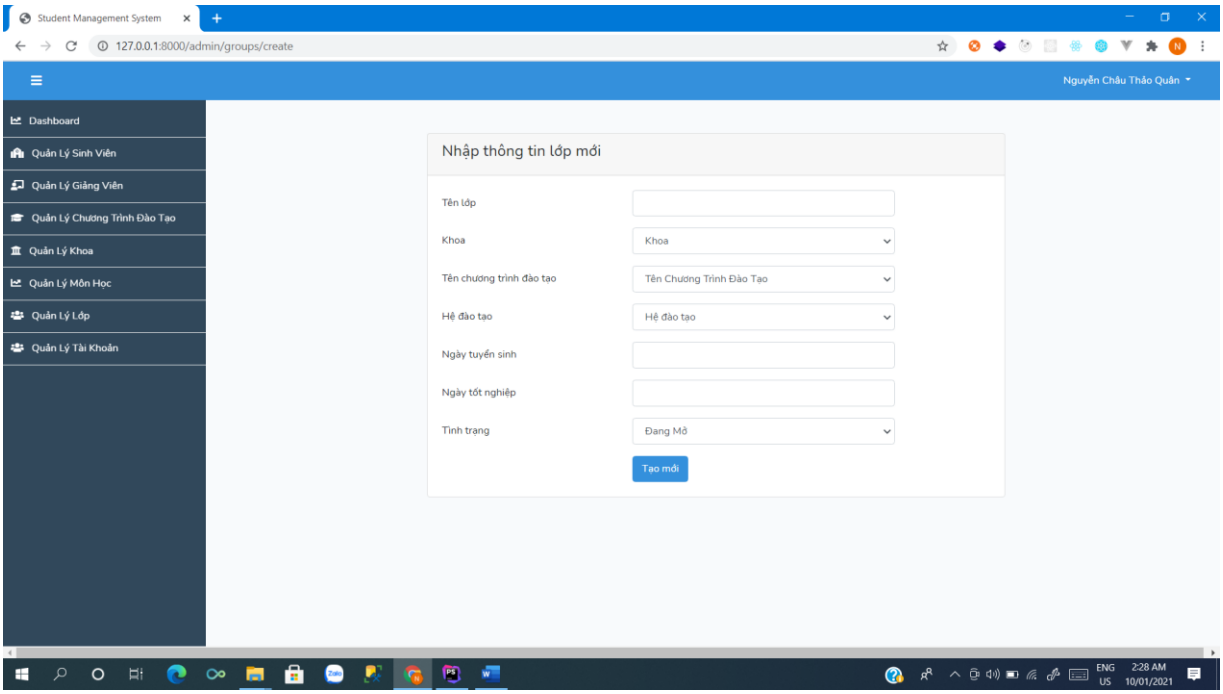

Figure 3.67: Form to add a new group.

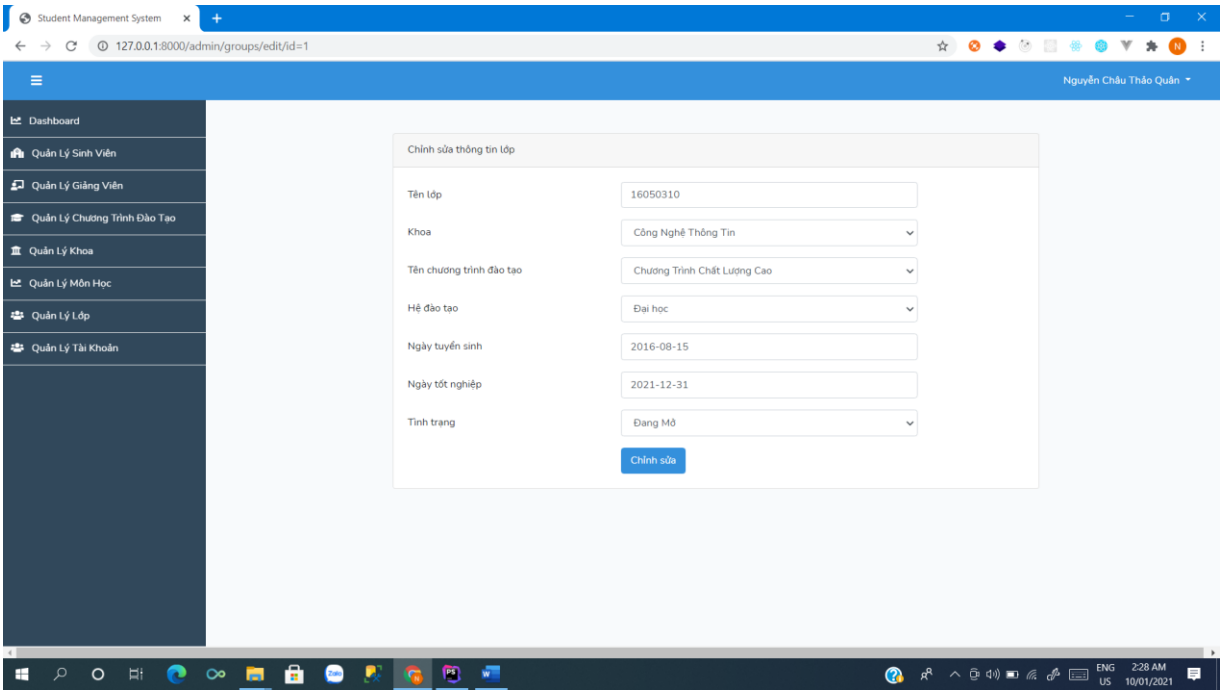

Figure 3.68: Form to update group.

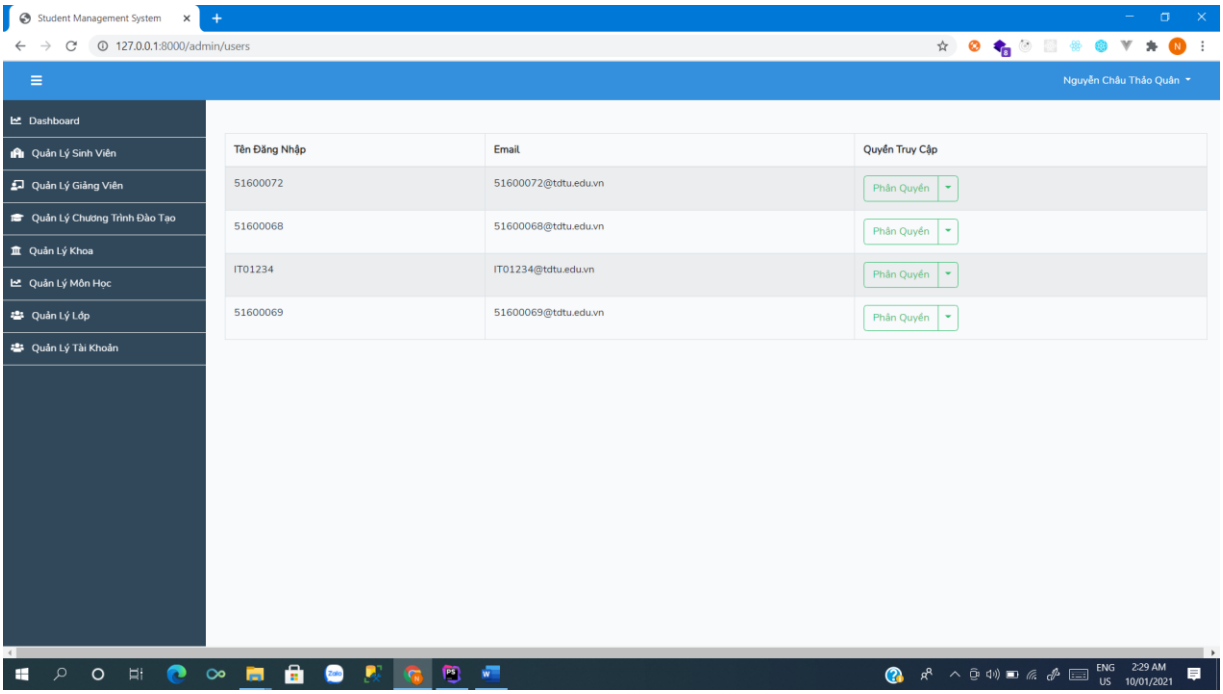

Figure 3.69: List of accounts.

CHAPTER 4 – CONCLUSION

Laravel is a progressor framework because of its incredible features. Apart from this, extensive community support is provided for its developers which makes it approachable and understandable by all its developers. It is scalable and helps in software delivery in a fast and cost-effective manner.

4.1 Advantages of laravel.

4.1.1 Easy to approach the latest features.

Developers have an opportunity to approach the latest feature that PHP provided, for example Namespaces, OOP, Overloading, Anonymous functions and shorter array syntax.

4.1.2 Has a large resources.

Laravel's resourses are friendly with all PHP developers with a variety of documents. Each version of Laravel have the right document for its application.

4.1.3 Mail service integration.

With SwiftMailer's apis, we can send an email through services based on cloud computing or local.

4.1.4 High security.

Laravel uses PDO to ignore the SQL injection and a hidden token field to ignore the attacks with CSRF type.

4.2 Disadvantages of laravel.

Its performance will be reduced when running mobile platform.

Missing the alignment between its version.

REFERENCES

Vietnamese

1. Khoa Nguyễn (2015), Mô hình MVC là gì ?, <https://khoanguyen.me/tim-hieu-mo-hinh-mvc-la-gi/ />, viewed on November 4th, 2020.
2. Hoàng Tựa (2015). Laravel là gì? Giới thiệu Laravel Framework, <http://webfaver.com/php-coding/laravel-5/tut-laravel-5-0-laravel-la-gi-gioi-thieu-laravel-framework.html/>, viewed on November 5th, 2020.

English

3. Jithin (2016), < https://www.interserver.net/tips/kb/mvc-advantages-disadvantages-mvc/ />, viewed on November 4th , 2020.
4. <https://laravel.com/docs/8.x />, viewed on November 4th, 2020.
5. Matt Stauffer (2019), *Laravel Up & Running: A Framework for Building Modern PHP Apps*, O'Reilly Media, United States of America.