** MINISTRY OF EDUCATION AND TRAINING**

**FPT UNIVERSITY**

Capstone Project Document

Student Finance Management

|  |  |
| --- | --- |
| **Group 2** | |
| **Group members** | Tran Khac Vy – Team Leader – SE60964  Ho Doan Trung – Team Member – SE60882  Tran Quang Phuc – Team Member - SE60993  Vu Nhat Anh Khoa – Team Member – SE60817  Le Tuan Anh – Team Member – SE60685 |
| **Supervisor** | Mr. Kieu Trong Khanh |
| **Ext. Supervisor** | N/A |
| **Capstone Project code** | SFM |

Ho Chi Minh City, 07/09/2015

*This page is intentionally left blank*

***ACKNOWLEDGEMENTS***

We wish to thank various people for their contribution to this project: Our teachers for their advice and participation in the final review, our friends for their valuable technical support.

Special thanks should be given to Mr.Kiều Trọng Khánh, our research supervisor for his professional guidance and the useful, constructive recommendations throughout the course of this project.

# Table of Contents

[Table of Contents 4](#_Toc430635611)

[List of Tables 5](#_Toc430635612)

[List of Figures 6](#_Toc430635613)

[Definitions, Acronyms, and Abbreviations 6](#_Toc430635614)

[Report No.2 Software Project Management Plan 7](#_Toc430635615)

[1. Problem Definition 7](#_Toc430635616)

[1.1 Name of this Capstone Project 7](#_Toc430635617)

[1.2 Problem Abstract 7](#_Toc430635618)

[1.3 Project Overview 7](#_Toc430635619)

[2. Project organization 9](#_Toc430635620)

[2.1 Software Process Model 9](#_Toc430635621)

[2.2 Roles and responsibilities 10](#_Toc430635622)

[2.3 Tools and Techniques 11](#_Toc430635623)

[3. Project Management Plan 11](#_Toc430635624)

[3.1 Software Development Life Cycle 11](#_Toc430635625)

[3.2 Phase Detail 14](#_Toc430635626)

[3.3 All Meeting Minutes 16](#_Toc430635627)

[4. Coding Convention 16](#_Toc430635628)

# List of Tables

[Table 1: Hardware Requirement for Server 8](#_Toc377250806)

[Table 2: Hardware Requirement for Mobile App 8](#_Toc377250807)

[Table 3: Roles and Responsibilities Details 10](#_Toc377250808)

[Table 4: Iteration 12](#_Toc377250809)

[Table 5: Phase 1: Preliminary Investigation or Analysis 12](#_Toc377250810)

[Table 6: Phase 2: Data Management 12](#_Toc377250811)

[Table 8: Phase 4: User Related Functions 13](#_Toc377250812)

[Table 7: Phase 3: Suggestion Algorithm 13](#_Toc377250813)

[Table 9: Phase 5: Market Management 13](#_Toc377250814)

# List of Figures

[Figure 1: Waterfall Development Model 9](#_Toc377233927)

# Definitions, Acronyms, and Abbreviations

|  |  |
| --- | --- |
| 1. Learning outcome | Student study result |

# Introduction

## Project Information

* Project name: **Student Finance Management**
* Project Code: **SFM**
* Product Type: **Website**
* Start Date: **September 7th, 2015**
* End Date: **November 1st, 2015**

## Introduction

Currently at the FPT University, the financial management of students is conducted by the helpful of the Excel files. The using Excel offers many inconveniences to the accountant in input data processing, searching or monitoring the financial situation of the student. Besides, although FPT had to send mail notification before each semester about tuition must be paid, but these messages often very general unclear and inappropriate with each students.

Realizing the difficulties, we have built a system "Student Finance Management". By accessing to the system, the accountant can easily enter, manage, search and render reports on the financial condition of each student. Students can use the system to keep track of your financial. Periodically time, the system will automatically send notifications to students about the fees which must pay for school.

## Current Situation

For now, student records are managed and stored in Excel spreadsheets by Student Assistance’s Office and Registrar’s Office. The school’s accountants will collect those spreadsheets, re-organize and re-enter data into other spreadsheets. The tuition that each student must pay and their payment status are automatically calculated by using pre-existent formulas and money transmission records from the bank.

## Problem Definition

The current financial procedure costs a lot of time since the Accounting Department has to wait for Student Assistance’s Office and Registrar’s Office to complete storing student data, and the bank to send the transmission records. In addition, the process of calculating and re-organizing all of the data is lengthy and error-prone. Therefore, it becomes greatly complicated to manage student financial records. As all the data is stored only in Excel spreadsheets, the Accounting Department faces the risk of losing data due to computer’s failure.

## Proposed Solution

To resolve these difficulties, we build system which called "Student Finance Management". The system provides the convenience for accounting staff in data entry, searching, tracking and reporting output. Based on the data entered, the system will use reasonable and announced plans to charge fees for each student then will periodically send mail to each student to notice the fees pay for the new semester.

The Student Finance Management will have following functions:

### Feature functions

* The system allows school staff can easily to enter information as well as the financial situation of students by using the form on the screen or imported from excel file available. Besides, staff can manage search and track financial status of each student.
* Students can log in the system to keep track of their financial situation.
* The system will automatically send periodically mail to students as well as families of students know about the tuition and fees for the semester/term coming next.

### Advantages and disadvantages

The advantages and disadvantages of the proposed solution:

* Advantages:
* Easy to enter data, searching, management and follow financial situation of students
* Automatically scheduled charge fees for each student.
* Automatically send mail notification about course fees for students with the most relevant contents.
* Disadvantages:
* Currently, it just applies for FPT University.

## Functional Requirements

Function requirements of the system are listed as below:

### Manager Component

* Created initial data for the system: the type of student, fees each semester, the student's status ...
* Search, monitoring financial status of students.

### Staff Component

* Entering data of students.
* Tracking financial situation of students.

### Student Component

* Tracking the financial situation of himself/herself.

## Role and Responsibility

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Full Name | Role | Position | Contact |
| 1 | Kieu Trong Khanh | Project Manager | Instructor | khanhkt@fpt.edu.vn |
| 2 | Tran Khac Vy | Developer | Leader | vytkse60964@gmail.com |
| 3 | Ho Doan Trung | Developer | Member | trunghdse60882@fpt.edu.vn |
| 4 | Tran Quang Phuc | Developer | Member | phuctqse60993@gmail.com |
| 5 | Le Tuan Anh | Developer | Member | anhltse60685@gmail.com |
| 6 | Vu Nhat Anh Khoa | Developer | Member | Khoavnase60817@gmail.com |

Table 1: Roles and Responsibilities

# Software Project Management Plan

## Problem Definition

### Name of this Capstone Project

* Official name : Student Finance Management
* Vietnamese name : Quản Lý Tài Chính Của Sinh Viên
* Abbreviation : SFM

### Problem Abstract

Currently at FPT University, the record and the financial information of each student are stored separately in different Excel spreadsheets. Therefore, the process of information retrieval of each student record or listing registered students in each trimester is extremely complicated. Besides, the school’s staff faces difficulties in managing student records and notifying required tuition and fees to each specific student.

In order to reduce the cost of those processes, SFM system was created to simplify the process of organizing records. The system will support the information retrieval of student records, automatically sends notification mail to students, and optimizes the monitoring and the financial management procedure.

### Project Overview

#### Current Situation

FPT University currently manages student records manually through Excel spreadsheets. However:

* Spreadsheets only support analyzing and storing data.
* Since it is difficult to retrieve and email to each specific student their record and tuition and fees information for the next trimester, the school only sends students notification of deadline for registering to study in the next trimester.

The system will solve those problems by supporting the school’s staff to:

* Import all data from Excel spreadsheets to the system’s storage
* Retrieve the information of a specific student easily
* Calculate plan for each student.
* Automatically email each student notification for their specific situation (study record, required tuition and fees)

The system also let students keep track on their records

#### The Proposed System

While handling the requirements of the system we found out some difficulties as follows:

* Import many excel files without closely associated. Validate data processing facing many difficulties.
* Processing and storage data related to the school's financial facing many difficulties due to lack of understanding of the financial regulations of the university.
* How to solve the prerequisite courses or courses that the school canceled

In order to solve these problems, our group has agreed to make the corresponding solutions:

* Using the open source Apache POI to process input data
* Thanks to the support from the University's finance department
* Thanks to the Education department of the university to provide information of courses and specific relationship of each course

##### System

The Student Finance Management system is intended to help managing student financial records and data. It requires web browser and Internet connection to operate. System can match existing records with newly imported data, calculate financial plan for students, notify and send email to students according to the plan. The system will have following functions:

* **For admin:**
* Admins can manage accounts, manage and configure the system.
* **For students:**
* Students can request to view their own records and financial status.
* **For staff:**
* Staff can import excel file or manually enter new student record data.
* **For manager:**
* Managers can use the system to search information based on student id, name and payment status.

#### Boundaries of the System

* The system can be used by every people with a laptop/computer.
* The system is **not intended** for managing these aspects:
* Managing learning outcome (1) detail.
* Managing exam’s information.
* Managing student’s schedule.
* The language of the system is English.
* The complete product includes:
* The website, for staff and user.
* The entire process document involved.

#### Future plans

The current system can only be applied to FPT University. We design the system can be expanded and applied to many universities across our country.

#### Development Environment

##### Hardware requirements

**For server**

|  |  |  |
| --- | --- | --- |
| Windows | Minimum Requirements | Recommended |
| Internet Connection | Cable, Wifi (4 Mbps) | Cable, Wifi (6 Mbps) |
| Operating System | Window Server 2008 | Window Server 2008 |
| Computer Processor | Intel® Xeon ® 1.4GHz | Intel(R) Core(TM) i5-2430M CPU @ 2.40GHz |
| Computer Memory | 1GB RAM | 4GB or more |

Table 1: Hardware Requirement for Server

##### Software requirements

|  |  |  |
| --- | --- | --- |
| Software | Name/ Version | Description |
| Environment | Java EE 7 | Specification for developing web application |
| IDE | Intellij IDEA 14.1 | Programming tools |
| DBMS | MySQL 5.6 | Used to create & manage the database for system |
| Source control | TortoiseSVN 1.8.11 | Used for source control |
| Web browser | Chrome 42 or above | Testing browser |

## Project organization

### Software Process Model

This project is developed under waterfall model. We applied customized waterfall model to capable with current situation in our team. We choose this model because following reasons:

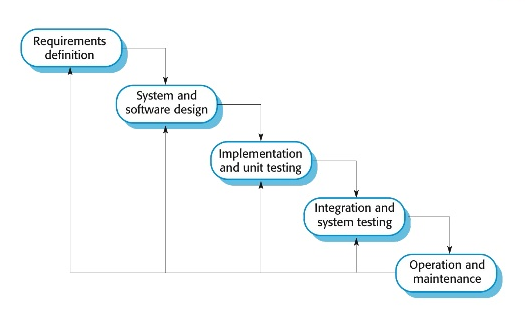
* Based researches and clarify FPT Student Finance Management, the requirements of this project are stable, clear, fixed and well understood by all team members.
* This project use Java technology
* FPT University has provided all necessary information for capstone project to students

Figure 1: Waterfall model

Reference: Page 30, chapter 2, Software process model, SOFTWARE ENGINEERING 9th Edition, by Ian Sommerville.

### Roles and responsibilities

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Full name** | **Role in Group** | **Responsibilities** |
| **1** | Kieu Trong Khanh | Project manager | * Specify user requirement * Control the development process * Give out technique and business analysis support |
| **2** | Tran Khac Vy | Team Leader, BA, DEV, Tester | * Managing process * Designing database * Clarifying requirements * Prepare documents * GUI Design * Create test plan * Coding * Testing |
| **3** | Ho Doan Trung | Team Member, BA, DEV, Tester | * Designing database * Clarifying requirements * Prepare documents * GUI Design * Create test plan * Coding * Testing |
| **4** | Le Tuan Anh | Team Member, BA, DEV, Tester | * Designing database * Clarifying requirements * Prepare documents * GUI Design * Create test plan * Coding * Testing |
| **5** | Vu Nhat Anh Khoa | Team Member, BA, DEV, Tester | * Designing database * Clarifying requirements * Prepare documents * GUI Design * Create test plan * Coding * Testing |
| **5** | Tran Quan Phuc | Team Member, BA, DEV, Tester | * Designing database * Clarifying requirements * Prepare documents * GUI Design * Create test plan * Coding * Testing |

Table 3: Roles and Responsibilities Details

### Tools and Techniques

|  |  |
| --- | --- |
| Tool/ Technique | Name/ Version |
| Frontend | HTML, CSS, JavaScript, jQuery, Bootstrap |
| Backend | Spring framework, Servlet, Thymeleaf, Hibernate framework |

## Project Management Plan

### Software Development Life Cycle

Below are all the major tasks that need to be performed sequentially during the development of the system.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Phase** | **Description** | **Deliverables** | **Resource needed** | **Dependencies and Constrains** | **Risk** |
| Requirements Definition | * Identify and clarify system requirements. | * Report No.1 Introduction. | 20 man-days | * N/A | * Missing requirement. * Project’s scope can be unclear. Lack of member share and understand. |
| System and Software Design | * Identify hardware and software requirements. * Decide software architect and clarify software detail design. * Design database. | * Report No.2 Software Project Management Plan * Report No. 3 Software Requirement Specification * Report No. 4 Software Design Description. | 60 man-days | * Depend on Requirements Definition. | * Misunderstood or unclear system’s requirement. * Lack of practical experience leading to unreasonable design. |
| Implementation and Unit Testing | * Implements all functions of system. * Create test plan. * Perform Unit testing. | * Software package. | 120 man-days | * Base on Software Requirement Specification and Software Design Description. * Coding try to follow coding convention. | * Member does not performs unit test. * Lack of practical experience. |
| Integration and System Testing | * Perform integration test and system test. | * Report No. 5 System Implementation & Test | 35 man-days | * Implementation and Unit Testing are finished. | * Lack of testing experience leading to lack of test cases. * Not enough time for performing test. |
| Operation and Maintenance | * Deploy the system * Create the user’s manuals. * Do routine maintenance activities. | * Report No.6 Software User’s Manual | 15 man-days | * Integration and System Testing are finished. | * User’s manual may be difficult for user to understand and confuse. |

### Phase Detail

|  |  |  |  |
| --- | --- | --- | --- |
| **Phase** | **Task** | **Description** | **Author** |
| Requirements Definition | * Identify and clarify system requirements. | * Research current systems to collect requirements. * Define main and needed functions the system must include. | Tran Khac Vy  Ho Doan Trung  Le Tuan Anh  Vu Nhat Anh Khoa  Tran Quang Phuc |
| System and Software Design | * Decide software architect and clarify software detail design. * Decide software architect and clarify software detail design. * Design database. | * Find out the suitable hardware and software for the system, as well as its minimum and recommended requirements. * Define the major software components and interfaces. * Draw core flow diagram, use case diagram, prototype … * Group meeting to review and modify. * Design database for the system. | Tran Khac Vy  Ho Doan Trung  Le Tuan Anh  Vu Nhat Anh Khoa  Tran Quang Phuc |
| Implementation and Unit Testing | * Implements all functions of system. * Create test plan. * Perform Unit testing. | * Coding all the components. * Planning for testing. * Write Unit test cases. * Implement Unit tests. | Tran Khac Vy  Ho Doan Trung  Le Tuan Anh  Vu Nhat Anh Khoa  Tran Quang Phuc |
| Integration and System Testing | * Perform integration test and system test. | * Test groups of modules and test whole the system. | Tran Khac Vy  Ho Doan Trung  Le Tuan Anh  Vu Nhat Anh Khoa  Tran Quang Phuc |
| Operation and Maintenance | * Deploy the system * Create the user’s manuals. * Do routine maintenance activities. | * Deploy the system in client environment. * Create a guideline to instruct users using system. * Do routine maintenance activities for client system. | Tran Khac Vy  Ho Doan Trung  Le Tuan Anh  Vu Nhat Anh Khoa  Tran Quang Phuc |

### All Meeting Minutes

Refer to Meeting Minutes folder.

[Meeting Minutes Folder](https://github.com/khacvy/SFM_Project/tree/master/document/Meeting)

## Coding Convention

Java: Using to develop Website.

Summary:

* Naming Convention.
  + Use camel case for both variable and function name.
  + Use pascal case for class name.
* Indentation.
  + Four spaces should be used as the unit of indentation. The exact construction of the indentation (spaces vs. tabs) is unspeciﬁed. Tabs must be set exactly every 8 spaces (not 4).
  + Avoid lines longer than 80 characters, since they’re not handled well by many terminals and tools.
* Declaration.
  + One declaration per line is recommended since it encourages commenting.
  + In absolutely no case should variables and functions be declared on the same line.
  + Do not put different types on the same line.
* Code Examples

Follow “Code Conventions for the Java TM Programming Language, by Sun Microsystems, rev April 20, 1999”.

<http://www.oracle.com/technetwork/java/codeconventions-150003.pdf>

# Software Requirement Specification

## User Requirement Specification

### Student Requirement

* Student is a user who utilizes the system for viewing their profile, study progress, or financial details. There are two users that inherit from Student
* CreditStudent
* InvestmentStudent
* Student can use all of the following functions:
* View Study Progress:
  + View all past semester’s details
  + View incomplete courses (failed courses and courses that require failed courses to be completed)
* View financial status: including tuition payments history, retake courses’ payments history, and the tuition plan
* View and edit profile’s information
* Change account’s password
* CreditStudent can use all Student’s available functions. In addition, CreditStudent can use the following function:
* View Credit Debt
* InvestmentStudent can use all Student’s available functions. In addition, InvestmentStudent can use the following function:
* View Returned Money Status

### Staff Requirement

* Staff manages students’ recorded such as study’s record and financial record. There are three types of Staff:
* Student Service Staff
* Student Service Staff is allowed to add, update and import Student’s information such as name, address, phone number, etc.
* Registrar Staff
* Registrar Staff is allowed to add, update and import Student’s study’s record such as test result, course’s status, etc.
* Financial Staff
* Financial Staff is allowed to add, update and import Student’s financial details such as student loans, payed, etc.

### Manager Requirement

* Manager manages financial details regarding students’ school’s progression. In addition, Manager also manages student’s suspension. Manager can use all of the following functions:
* View and export Financial Information
* Add, update, delete the suspension of a student
* Update and delete Exchange Rate

### Admin Requirement

* Admin is the person who manages the system. Admin can use these functions:
* Configure system
* Manage Course
* Manage user

## System Requirement Specification

### External Interface Requirement

#### User Interface

* General requirement for graphics user interface is the GUI should be simple, clear, intuitive, and reminiscent.
* The interface design is an iterate process includes: design, sketching, prototyping, user assessment.
* Some design principles will be taken into consideration:
  + UI for business web applications - Janko Jovanovic [Ref:http://www.smashingmagazine.com/2010/02/25/designing-user-interfaces-forbusiness-web-applications/]
  + Ten principles of effective web design – Vitaly Friedman [Ref:http://www.smashingmagazine.com/2008/01/31/10-principles-of-effective-webdesign/]

#### Hardware Interface

**N/A**

#### Software Interface

* Web application: works with Firefox (v30 or above), Chromes (v14 or above), Internet Explorer (v10 or above) browse.

#### Communication Protocol

* Use HTTP protocol 1.1 for communication between the web browser and the web server.

### System Overview Use Case

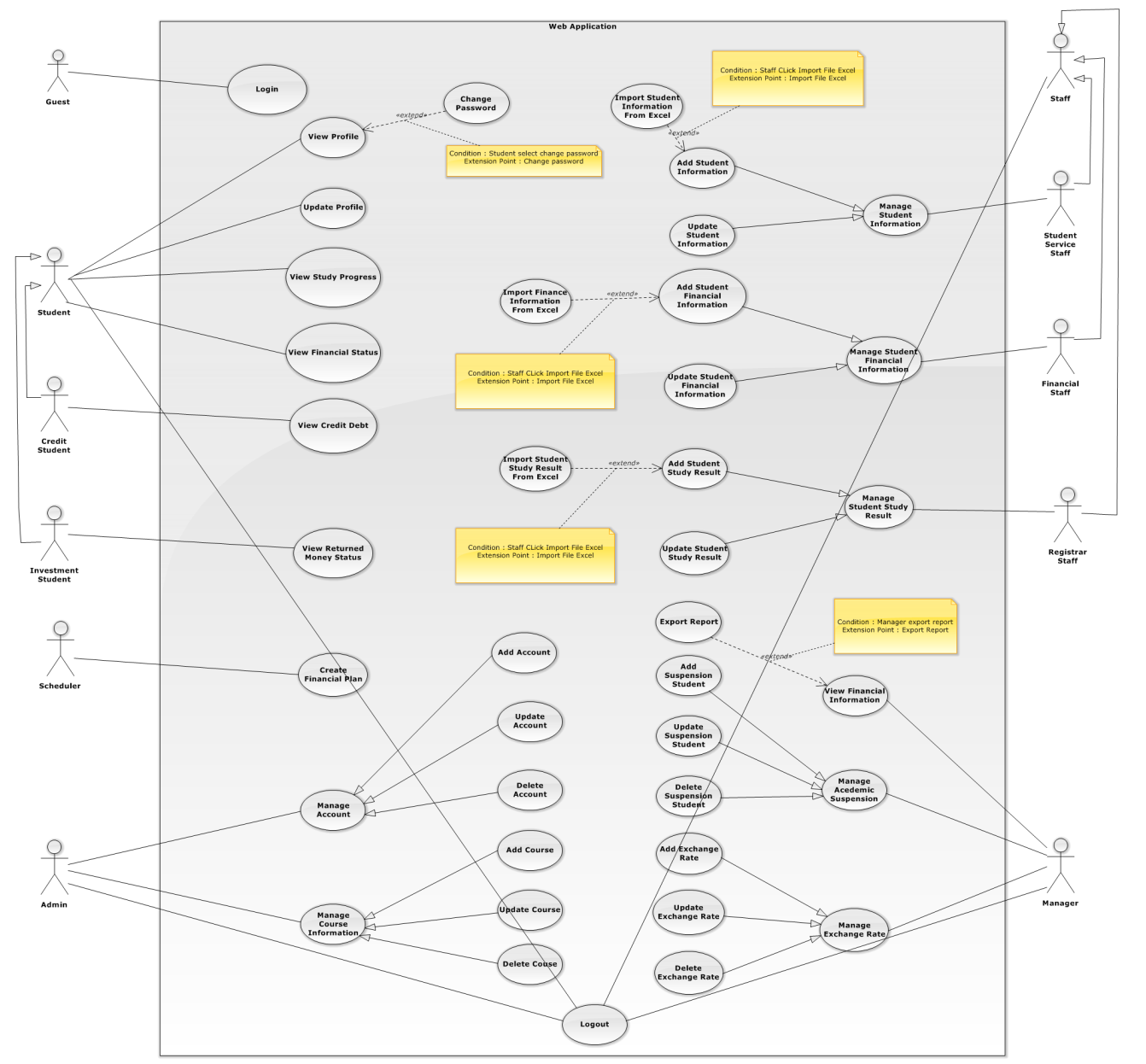
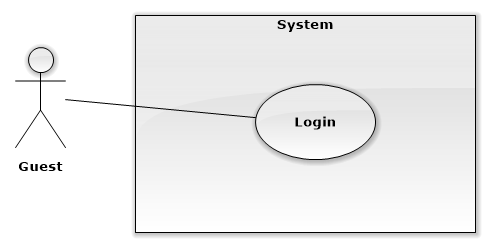


Figure 1: System Overview Use Case

### List of Use Case

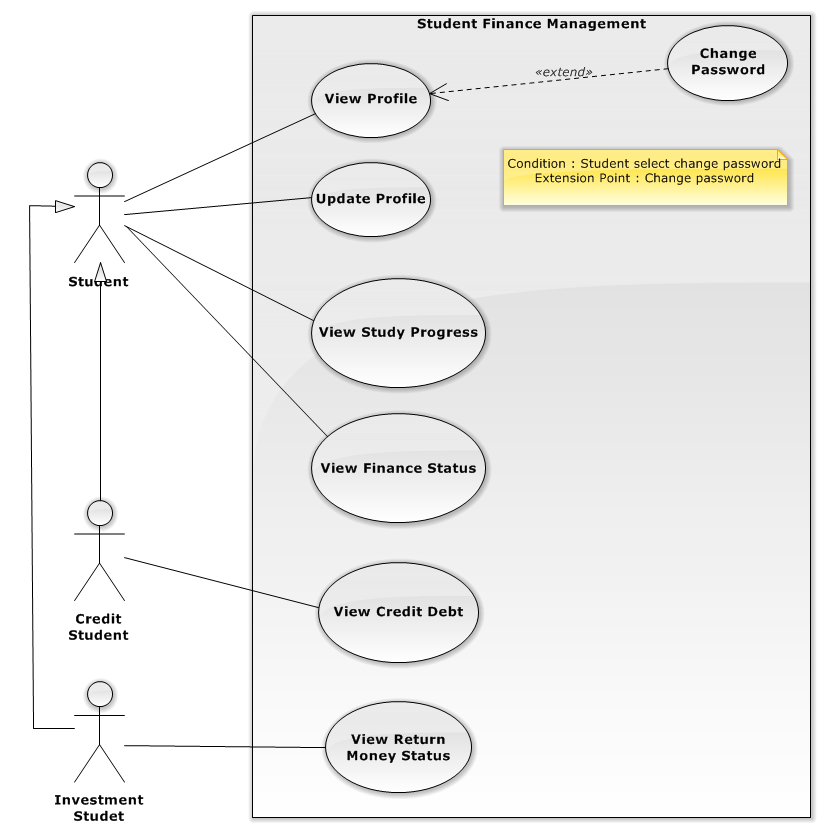
#### <Guest> Overview Use Case

##### <Guest> Log in

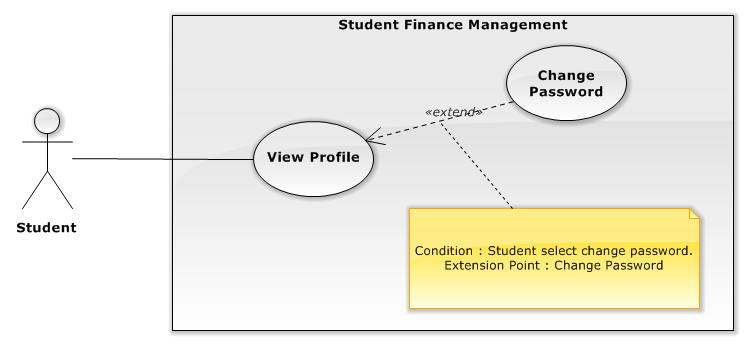


|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – Guest01** | | | |
| **Use Case No.** | WG01 | **Use Case Version** | 1.0 |
| **Use Case Name** | Login | | |
| **Author** | TrungHD | | |
| **Date** | 06/10/2015 | **Priority** | Normal |
| **Actor**:   * Guest   **Summary**:   * This use case allows guest to log in the system.   **Goal**:   * Guest can log in the system. Therefore, they can use some additional functions.   **Triggers**:   * Guest sends the login command.   **Preconditions**:   * N/A   **Post** **Conditions**:   * **Success**: Guest login the system. * **Fail**: Show error message.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | **Step** | **Actor Action** | **System Response** | | 1 | Guest goes to login view. | System requires identity information form Guest:   * Username: Free text input, required, length 6 - 250 * Password: Password input, required, length 8 - 32 | | 2 | Guest input information. |  | | 3 | Guest sends command to login to system. | Guest will login system with their specific role.  [Alternative 1]  [Exception 1] |   **Alternative Scenario[1]:**   |  |  |  | | --- | --- | --- | | **Step** | **Actor Action** | **System Response** | | 1 | Guest enters wrong identity information. | System shows error message. |   **Exceptions[1]:**   |  |  |  | | --- | --- | --- | | **Step** | **Actor Action** | **System Response** | | 1 | Guest enters wrong identity information. | System shows error message. |   **Relationships**: N/A  **Business** **Rules**:   * Password is encrypted before being sent to server. * After login to system, guest will be redirected to specific view based on their role on the system: student, staff, manager or admin. * If role is “Student”, the system will display to Student view. * If role is “Staff”, the system will display to Staff view. * If role is “Manager”, the system will display to Manager View. | | | |

#### <Student> Overview Use case

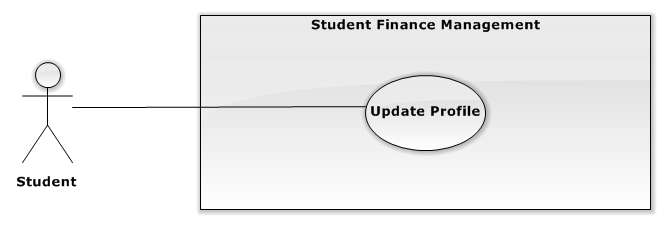


##### <Student> View Profile



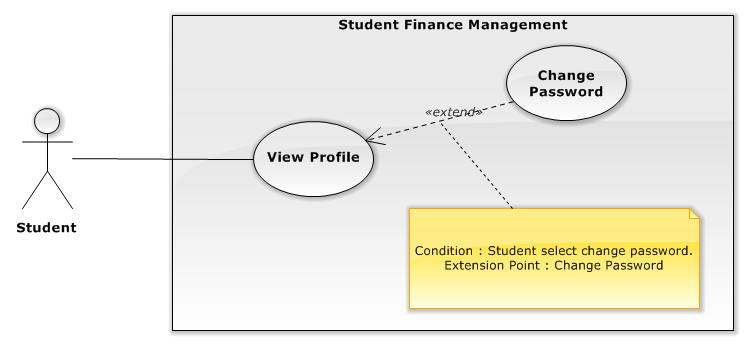
|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – STU001** | | | |
| **Use Case No.** | STU001 | **Use Case Version** | 1.0 |
| **Use Case Name** | View Profile | | |
| **Author** | Vu Nhat Anh Khoa | | |
| **Date** | 26/09/2015 | **Priority** | Normal |
| **Actor:**   * Student   **Summary:**   * This use case allows Student to view their own profile.   **Goal:**   * The system displays Student’s information such as name, student’s code, phone number, etc. This function help user view their profile’s details to see if there is wrong information in the database.   **Triggers:**   * Student sends command to view profile.   **Preconditions:**   * User must log into the system with role as Student.   **Post Conditions:**   * **Success**: Show Student their profile information * **Fail:** Show error message.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | Student goes to view their profile | Display new view that shows Student’s profile details including:   * + Student’s name   + Student’s code   + Student’s identification number   + Student’s gender   + Student’s date of birth   + Student’s address   + Student’s phone number   + Student’s email address   + Phone number of Student’s parent   + Email address of Student’s parent   + Student’s tuition payments type   + Student’s class year   + Student’s entry English level   + Student’s beginning semester |   **Alternative Scenario:** N/A  **Exceptions:**  N/A  **Relationships:** N/A  **Business Rules:**   * Student’s information is loaded from the system’s storage * Only “Student’s address”, “Student’s email address”, “Phone number of Student’s parent”, “Email address of Student’s parent” are available to modify directly by Student (see use case STU002). Otherwise, student will have to contact the school’s Office of Registrar to correct other information. | | | |

##### <Student> Update Profile



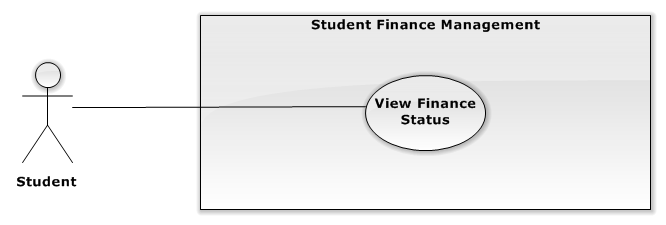
|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – STU002** | | | |
| **Use Case No.** | STU002 | **Use Case Version** | 1.0 |
| **Use Case Name** | Update Profile | | |
| **Author** | Vu Nhat Anh Khoa | | |
| **Date** | 25/09/2015 | **Priority** | Normal |
| **Actor:**   * Student   **Summary:**   * This use case allows Student to update their profile information.   **Goal:**   * The system provides Student a conventional mean to edit phone number, address, parent’s phone number and parent’s email address.   **Triggers:**   * Student sends command to update profile.   **Preconditions:**   * User must log into the system with role as Student.   **Post Conditions:**   * **Success**: Update new profile information to storage. * **Fail:** Show error message.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | Student goes to Update Profile view | Display new view that allows Student to input new information:   * + Student’s phone number: number input, required, length 8 - 15   + Student’s address: free text input, required, length 3 -250   + Parent’s phone number: number input, not required, length 8 - 15   + Parent’s email address: free text input, not required, length 3 -250 | | 2 | Student alters the information in the textboxes |  | | 3 | Student sends command to update profile information | System validates fields’ values, request confirmation from Student  [Exception 1, 2, 3, 4] | | 4 | Student sends confirm command | * System updates new information to storage if the data is valid * Show success message |   **Alternative Scenario:** N/A  **Exceptions:**   |  |  |  | | --- | --- | --- | | No | Cause | System Response | | 1 | Any empty required fields | System shows message to notify user to which fields are empty | | 2 | Length of field’s value is out of specified range. | System shows message to notify user which field’s value is out of range | | 3 | Email address’s format is incorrect | System shows message to notify user inputted email is invalid | | 4 | Any characters other than number in fields with number input requirement | System shows message to notify user to input only number as the fields’ requirements | | 5 | Length of field’s value is out of specified range. | System shows message to notify user which field’s value is out of range |   **Relationships:** N/A  **Business Rules:**   * Required fields must not be empty * An email address must be validated by this regular expression:   */^([a-z0-9\_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/*   * Each textbox has pre-inputted text showing current profile information * In successful scenario, new information will be updated to the system | | | |

##### <Student> Change Password



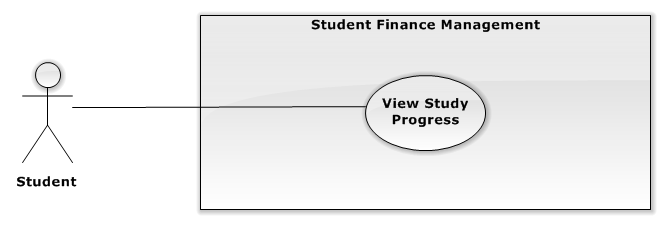
|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – STU003** | | | |
| **Use Case No.** | STU003 | **Use Case Version** | 1.0 |
| **Use Case Name** | Change Password | | |
| **Author** | Vu Nhat Anh Khoa | | |
| **Date** | 25/09/2015 | **Priority** | Normal |
| **Actor:**   * Student   **Summary:**   * This use case allows Student to change their account’s password.   **Goal:**   * The system provides Student a conventional mean to change their password. User can use this function to change their password.   **Triggers:**   * Student sends command to change password.   **Preconditions:**   * User must log into the system with role as Student   **Post Conditions:**   * **Success**: Update new password to storage. * **Fail:** The system shows an error message.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | Student goes to change password view | Display new view containing textboxes for user to input their current and new password:   * + Current password: free text input, required, length 6-32   + New password: free text input, required, length 6-32   + Confirm new password: free text input, required, length 6-32 | | 2 | User fills in the textboxes |  | | 3 | Student sends command to change password | System validates fields’ values, request confirmation from Student  [Exception 1, 2, 3, 4, 5] | | 4 | Student sends confirm command | * System updates new password to storage if the data is valid * Show success message |   **Alternative Scenario:** N/A  **Exceptions:**   |  |  |  | | --- | --- | --- | | No | Cause | System Response | | 1 | Student inputs the wrong “Current password” | System shows message to notify user that the current password is wrong. | | 2 | If the “New password” and the “Confirm new password” are not matched. | System shows message to notify user that the confirm password is not matched with the new password. | | 3 | If the “New password” and the “Current password” are matched | System shows message to notify user that the new password is the same as the current password | | 4 | Any empty required fields | System shows message to notify user to which fields are empty | | 5 | Length of field’s value is out of specified range. | System shows message to notify user which field’s value is out of range |   **Relationships:** N/A  **Business Rules:**   * Required fields must not be empty * In successful scenario, new password will be updated to the system * Password is encrypted before sending to the server * Password is encrypted before saving into storage * Textboxes will display \* character for all inputted characters * New password must be different from the current password | | | |

##### <Student> View Financial Status



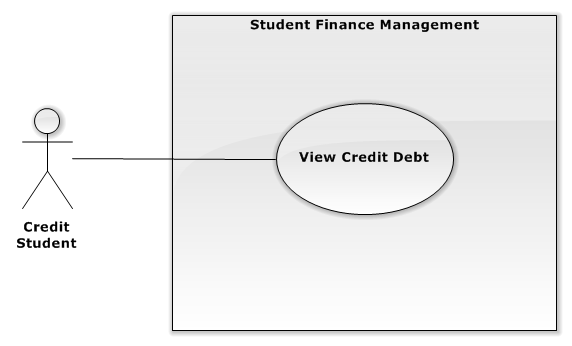
|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – STU004** | | | |
| **Use Case No.** | STU004 | **Use Case Version** | 1.0 |
| **Use Case Name** | View Financial Status | | |
| **Author** | Vu Nhat Anh Khoa | | |
| **Date** | 25/09/2015 | **Priority** | Normal |
| **Actor:**   * Student   **Summary:**   * This use case allows Student to view tuition payments history, retake courses’ payments history, and the tuition payments plan.   **Goal:**   * The system provides Student a conventional mean to view their financial details such as payment history and tuition payments plan in order to predict and plan out future tuition payments.   **Triggers:**   * Student sends command to view their financial status information   **Preconditions:**   * User must log into the system with role as Student   **Post Conditions:**   * **Success**: Show financial status information to Student * **Fail:** Show error message.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | Student goes to Financial status view | Display new view showing following information:   * Tuition payments history: link to view details of past tuition payments * Retake Courses’ payments history: link to view details of past retake courses’ payments * Tuition plan: link to view details of tuition for each academic advancement | | 2 | Student goes to “Tuition payments history” view  [Alternative 1, 2] | Display new view showing Tuition payments history in tabular form including:   * + Ordering number   + Semester’s information   + Student’s payment status   + Transmission date   + Transmission deadline   + A link to transmission details | | 3 | Student sends command to view transmission details | Display new view showing transmission details including:   * + Student’s name   + Student’s code   + Student’s gender   + Student’s identification number   + Student’s tuition payments type   + Payment detailed description   + Initial charge   + Deductions   + Actual charge   + Payed amount   + Balance   + Tuition payment status   + The transmission date   + Bank’s name |   **Alternative Scenario:**  *[Alternative 1]*   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | Student sends command to view retake courses’ payments history | Display new view showing Retake courses’ payments history in tabular form including:   * + Ordering number   + Retake course’s code   + Retake course’s name   + Student’s payment status   + Transmission date   + Transmission deadline   + A link to transmission details | | 2 | Student sends command to view transmission details | Display new view showing transmission details including:   * + Student’s name   + Student’s code   + Student’s gender   + Student’s identification number   + Student’s tuition payments type   + Payment detailed description   + Initial charge   + Deductions   + Actual charge   + Payed amount   + Balance   + Tuition payment status   + The transmission date   + Bank’s name |   *[Alternative 2]*   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | Student sends command to view tuition payments plan | Display new view showing Tuition payments plan in tabular form including:   * + Ordering number   + Semester’s information   + Tuition   + Actual charge   + A link to transmission details |   **Exceptions:** N/A  **Relationships:** N/A  **Business Rules:**   * All information is loaded from the system’s storage * “Semester’s information” is display in this format: *[term] [year] ([start date] - [end date])*. A school year has three [term]: Spring, Summer, Fall | | | |

##### <Student> View Academic Progress



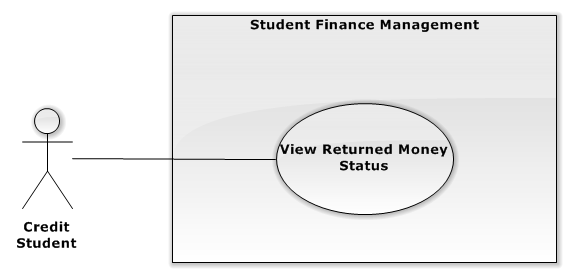
|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – STU005** | | | |
| **Use Case No.** | STU005 | **Use Case Version** | 1.0 |
| **Use Case Name** | View Academic Progress | | |
| **Author** | Vu Nhat Anh Khoa | | |
| **Date** | 25/09/2015 | **Priority** | Normal |
| **Actor:**   * Student   **Summary:**   * This use case allows Student to view their semesters and courses history, and their incomplete courses.   **Goal:**   * The system provides Student a conventional mean to view their study progress. The system will display student’s grade and status for each course from specific semester. Student can also view their incomplete courses for future plans.   **Triggers:**   * Student sends command to view their study progress   **Preconditions:**   * User must log into the system with role as Student.   **Post Conditions:**   * **Success**: Show study progress information to student. * **Fail:** Show an error message.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | Student goes to Academic Progress view | Display new view showing Study Progress of the latest semester including:  ***Semester’s information***  ***Course’s information*** (displayed in tabular form)   * + Ordering number   + Course’s code   + Course’s name   + Numbers of credits   + Course’s grade   + Course’s status   + Additional note | | 2 | Student sends command to view different semester  [Alternative 1] | Display new view showing selected semester’s details as above |   **Alternative Scenario:**  *[Alternative 1]*   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | Student sends command to view Incomplete Courses | Display new view showing Incomplete courses in tabular form including:   * + Ordering number   + Course’s code   + Course’s name   + Numbers of credits   + Semester’s information   + Reason of incompleteness   + Additional note |   **Exceptions:** N/A  **Relationships:** N/A  **Business Rules:**   * All information is loaded from the system’s storage * The latest semester is determined by Semester’s start date and end date * “Semester’s information” is display in this format: *[term] [year] ([start date] - [end date])*. A school year has three [term]: Spring, Summer, Fall * Only finished semesters are available for viewing | | | |

##### <CreditStudent> View Credit Debt



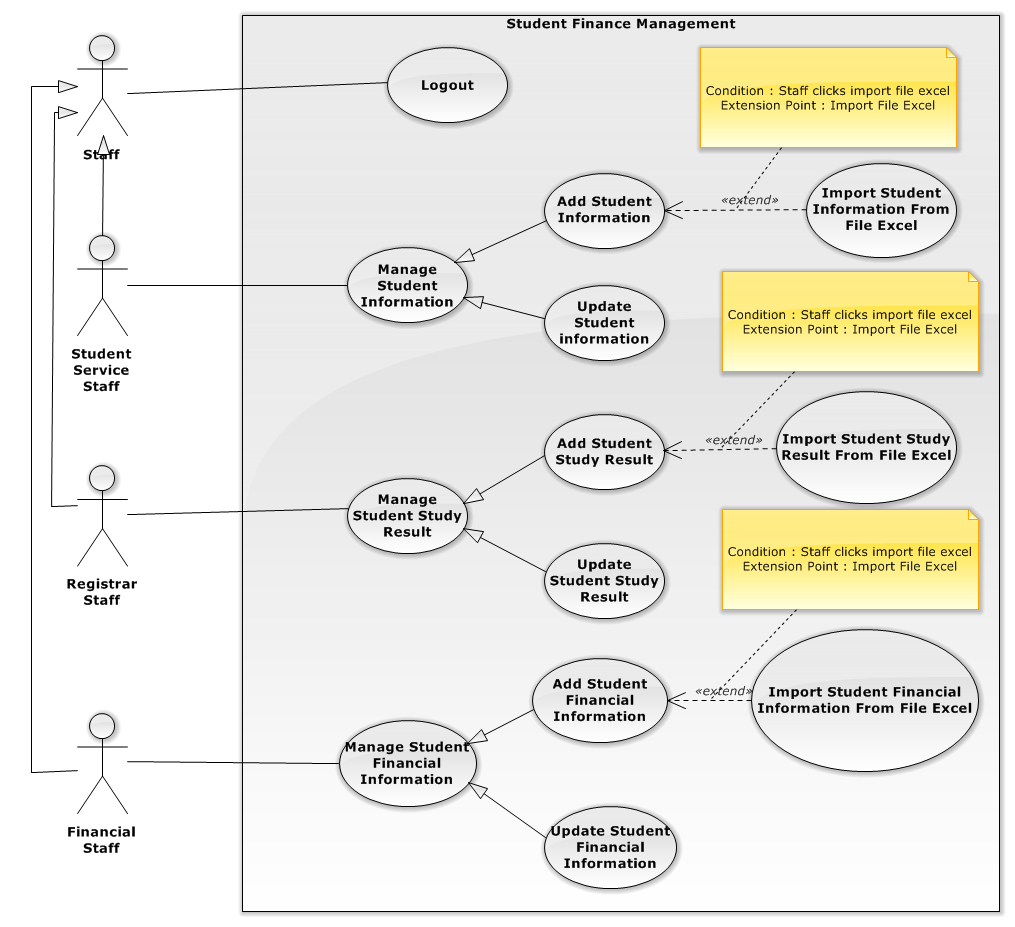
|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – STU006** | | | |
| **Use Case No.** | STU006 | **Use Case Version** | 1.0 |
| **Use Case Name** | View Credit Debt | | |
| **Author** | Vu Nhat Anh Khoa | | |
| **Date** | 26/09/2015 | **Priority** | Normal |
| **Actor:**   * CreditStudent   **Summary:**   * This use case allows CreditStudent to view their student loans.   **Goal:**   * The system provides user a conventional mean to view their student loans. User uses this function to plan for future payment, to be reminded the repayment date and check if anything wrong with provided information.   **Triggers:**   * CreditStudent sends command to view their student loans   **Preconditions:**   * User must log into the system with role as CreditStudent   **Post Conditions:**   * **Success**: Show user’s debt information * **Fail:** Show an error message.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | CreditStudent goes to Debt view | If CreditStudent is undergraduate, display new view showing Debt information including:  ***CreditStudent information:***   * CreditStudent’s name * CreditStudent’s code   ***Debt information:*** (displayed in tabular form)   * + Date   + Loan’s description   + Tuition   + Payed amount   + Loan   + Exchange rate of dollar to Vietnamese dong   + Additional note   [Alternative 1] |   **Alternative Scenario:**  *[Alternative 1]*   |  |  |  | | --- | --- | --- | | No | Actor Action | System Response | | 1 | CreditStudent goes to Debt view | If CreditStudent is graduate. display new view showing Debt information including:  ***CreditStudent information:***   * + CreditStudent’s name   + CreditStudent’s code   + CreditStudent’s date of graduation   ***Debt information:*** (displayed in tabular form)   * + Debt   + Interest   + Payed amount   + Remaining debt   + Next repayment date   + Interest rate |   **Exceptions:** N/A  **Relationships:** N/A  **Business Rules:**   * All information is loaded from the system’s storage. * “Remaining debt” is calculated by “Payed amount”, “Interest” and “Debt” * “Interest” is calculated by "Remaining debt” and “Interest rate”. * Repayment dates is exactly one month apart, started from the date of graduation. If graduation date is at the end of the month, then repayment dates is also at the end of each month   (e.g. If “Student’s date of graduation” is February 1st 2015, “Next repayment date” will be March 1st 2015, and the following “Next repayment date” will be April 1st 2015) | | | |

##### <InvestmentStudent> View Returned Money Status



|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – STU007** | | | |
| **Use Case No.** | STU007 | **Use Case Version** | 1.0 |
| **Use Case Name** | View Return Money Status | | |
| **Author** | Vu Nhat Anh Khoa | | |
| **Date** | 26/09/2015 | **Priority** | Normal |
| **Actor:**   * InvestmentStudent   **Summary:**   * This use case allows Student to view their own student loans.   **Goal:**   * The system provides user a conventional mean to view their student loans. User uses this function to plan for future payment, to be reminded the repayment date and check if anything wrong with provided information.   **Triggers:**   * Student sends command to view their return money status.   **Preconditions:**   * User must log into the system with role as InvestmentStudent.   **Post Conditions:**   * **Success**: Show user’s return money status information * **Fail:** Show error message.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | InvestmentStudent goes to Return money status view | Display new view showing Return money status information including:  ***InvestmentStudent’s information:***   * + Student’s name   + Student’s code   + Student’s date of graduation   ***Debt information:***   * + Remaining debt.   + Payed amount   + Interest rate   + Next repayment date |   **Alternative Scenario:** N/A  **Exceptions:** N/A  **Relationships:** N/A  **Business Rules:**   * All information is loaded from the system’s storage * Repayment dates is exactly one month apart, started from the date of graduation. If graduation date is at the end of the month, then repayment dates is also at the end of each month   (e.g. If “Student’s date of graduation” is February 1st 2015, “Next repayment date” will be March 1st 2015, and the following “Next repayment date” will be April 1st 2015) | | | |

#### <Staff> Overview Use Case



##### <Student Service Staff> Add Student’s Information

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – STA001** | | | |
| **Use Case No.** | STA001 | **Use Case Version** | 1.0 |
| **Use Case Name** | Add Student’s Information | | |
| **Author** | Ho Doan Trung | | |
| **Date** | 25/09/2015 | **Priority** | High |
| **Actor:**   * Student Service Staff   **Summary:**   * This use case allows user to add students’ details into the system   **Goal:**   * The system provides user a conventional mean to add newly enrolled students and their information such as name, address, phone number, etc. New student’s details will be added into system’s storage. User can also import those data from Excel spreadsheets   **Triggers:**   * User sends command to enter student information   **Preconditions:**   * User must log into the system with role as the Student Service Staff   **Post Conditions:**   * **Success:** The system adds new student’s information into storage. System notifies user that the data was successfully added * **Fail:** There is no information added and the system will display error messages to user.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | **Step** | **Actor Action** | **System Response** | | 1 | User goes to add student’s information view  [Alternative 1] | The system displays view that allows user to input information for a new student:   * Full name: free text input, required, length 3-250 * Sex: select one of these options: (male, female), required * Student ID: free text input, required, length 8 * Date of birth: date, required * Address: free text input, required, length 3-250 * Email: free text input, required, length 3-250 * Admission date: date, required * Start from Semester: date, required * Entry English level: select one of these options : (1, 2, 3, 4, 5, none), required * Parent’s phone number: number input, length 3-10, not required * Parent’s email: free text input, length 3-250, not required * Student’s school status: select from these options: (Studying, Graduated, Suspended), required | | 2 | User enters new information, then sends add command to the system | The system validates fields’ value, sends confirmation request to user  [Exception 1, 2, 3, 4] | | 3 | User sends confirm command  [Alternative 2] | * The system stores new information to storage if the data is valid * Show success message * Display “Students’ list” view |   **Alternative Scenario:**  *[Alternative 1]*   |  |  |  | | --- | --- | --- | | **Step** | **Actor Action** | **System Response** | | 1 | User sends command to import excel spreadsheet’s data into storage. | The system will display view allowing user to import excel spreadsheet file to the system | | 2 | User selects an Excel file to import and then sends import command to the system. | The system validates the file, sends confirmation request to user  [Exception 5] | | 3 | User sends command to confirm importing the file | * The system adds new information into storage if the data is valid * Show success message * Display “Students’ list” view |   **Exceptions:**   |  |  |  | | --- | --- | --- | | **No** | **Cause** | **System Response** | | 1 | Length of field’s value is out of specified range. | System shows message to notify user which field’s value is out of range | | 2 | Any empty required fields | System shows message to notify user specific empty fields | | 3 | Wrong email format. | Show error message: “”Email” must be formatted as abc@example.fpt.edu.vn!” | | 4 | Any characters other than number in fields with number input requirement | System shows message to notify user to input only number as the fields’ requirements | | 5 | Import wrong Excel spreadsheet’s format | System shows message to notify user that the format of the imported spreadsheet is wrong |   **Relationships:** N/A  **Business Rules:**   * Required fields must not be empty * An email address must be validated by this regular expression:   */^([a-z0-9\_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/*   * “Sex” has two options: Male and Female * “Entry English level” has 6 options: *1, 2, 3, 4, 5, none*. The “none” option means students skip English course and go straight to study their majors * If the student is in “Suspension list” (see use case MAN002), “Student’s school status” will be set as “Suspended” and user won’t be able to change the field’s value * In successful scenario, new information will be added to the system * “Students’ list” view contains following information in tabular form:   + Full name   + Gender   + Student ID   + Date of birth   + Address   + Email   + Admission date   + Start from Semester   + Entry English level   + Supported   + Parent’s phone number   + Parent’s email * Excel spreadsheet’s format: | | | |

##### <Student Service Staff> Update Student’s Information

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – STA002** | | | |
| **Use Case No.** | STA002 | **Use Case Version** | 1.0 |
| **Use Case Name** | Update Student’s Information | | |
| **Author** | Ho Doan Trung | | |
| **Date** | 25/09/2015 | **Priority** | High |
| **Actor:**   * Student Service Staff   **Summary:**   * This use case allows user to update student’s information.   **Goal:**   * The system provides user a conventional mean to update student’s information. Using this function, Student’s information will be modified and updated to the system’s storage when needed.   **Triggers:**   * The staff will select a row in list student and send command “Update”.   **Preconditions:**   * Staff login to the system with the rights of the Student Service Staff   **Post Conditions:**   * **Success:** The system updates student’s information into storage. System notifies user that the data was updated success full * **Fail:** There is no student’s information updated and the system will display error messages to the staff   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | **Step** | **Actor Action** | **System Response** | | 1 | User goes to “Students’ list” view, selects a student then sends command to update information to the system. | The system displays view that allows user to edit information:   * Full name: free text input, required, length 3-250 * Sex: select one of these options: (male, female), required * Student ID: free text input, required, length 8 * Date of birth: date, required * Address: free text input, required, length 3-250 * Email: free text input, required, length 3-250 * Admission date: date, required * Start from Semester: date, required * Entry English level: select one of these options : (1, 2, 3, 4, 5, none), required * Parent’s phone number: number input, length 3-10, not required * Parent’s email: free text input, length 3-250, not required * Student’s school status: select from these options: (Studying, Graduated, Suspended), required | | 2 | User modifies information, then send command to update information to the system | The system validates fields’ values, then sends confirmation request to user  [Exception 1, 2, 3, 4] | | 3 | User sends confirm command | * The system updates new information into storage if the data is valid * Show success message * Display “Students’ list” view |   **Alternative Scenario:** N/A  **Exceptions:**   |  |  |  | | --- | --- | --- | | **No** | **Cause** | **System Response** | | 1 | Length of field’s value is out of specified range. | System shows message to notify user which field’s value is out of range | | 2 | Any empty required fields | System shows message to notify user specific empty fields | | 3 | Wrong email format. | Show error message: “”Email” must be formatted as abc@example.fpt.edu.vn!” | | 4 | Any characters other than number in fields with number input requirement | System shows message to notify user to input only number as the fields’ requirements |   **Relationships:** N/A  **Business Rules:**   * Required fields must not be empty * Following fields are unavailable to be edited:   + Email   + Student’s code   + Beginning semester   + Entry English level   + Admission date * An email address must be validated by this regular expression:   */^([a-z0-9\_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/*   * “Sex” has two options: Male and Female * “Entry English level” has 6 options: *1, 2, 3, 4, 5, none*. The “none” option means students skip English course and go straight to study their majors. * In successful scenario, new information will be updated to the system * “Students’ list” view contains following information in tabular form:   + Full name   + Gender   + Student ID   + Date of birth   + Address   + Email   + Admission date   + Start from Semester   + Entry English level   + Supported   + Parent’s phone number   + Parent’s email | | | |

##### <Financial Staff> Add Financial Information

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – STA003** | | | |
| **Use Case No.** | STA003 | **Use Case Version** | 1.0 |
| **Use Case Name** | Add Financial Information | | |
| **Author** | Ho Doan Trung | | |
| **Date** | 25/09/2015 | **Priority** | High |
| **Actor:**   * Financial Staff   **Summary:**   * This use case allows Financial Staff to add information about a student’s financial details   **Goal:**   * The system provides user a conventional mean to add student’s financial information such as tuition, retake course fees, fine level, etc.   **Triggers:**   * The staff will select function to enter financial information.   **Preconditions:**   * Staff login to the system with the rights of the Financial Staff   **Post Conditions:**   * **Success:** Financial information will be added to the database. System notifies user that the data was added successfully * **Fail:** There is no information added and the system will display error messages to the staff.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | **Step** | **Actor Action** | **System Response** | | 1 | User goes to “Students’ list” view and select a student, then sends command to add student’s financial information to the system.  [Alternative 1] | The system displays view that allows user to edit information:   * Tuition: number input, required, length 1-250 * Retake course fees: number input, required, length 1-250 * Fine level: Number input, required, length 1-250 * Payed amount: Number input, required, length 1-250 | | 2 | User input text into fields, then send command to add information to the system | The system validates fields’ values, then sends confirmation request to user  [Exception 1, 2, 3] | | 3 | User sends confirm command | * The system stores new information into storage if the data is valid * Show success message * Display “Students’ list” view |   **Alternative Scenario:**  *[Alternative 1]*   |  |  |  | | --- | --- | --- | | **Step** | **Actor Action** | **System Response** | | 1 | User sends command to import excel spreadsheet’s data into storage. | The system will display view allowing user to import excel spreadsheet file to the system | | 2 | User selects an Excel file to import and then sends import command to the system. | The system validates the file, sends confirmation request to user  [Exception 4] | | 3 | User sends command to confirm importing the file  [Alternative 2] | * The system adds new information into storage if the data is valid * Show success message * Display “Students’ list” view |   **Exceptions:**   |  |  |  | | --- | --- | --- | | **No** | **Cause** | **System Response** | | 1 | Any empty required fields | System shows message to notify user specific empty fields | | 2 | Any characters other than number in fields with number input requirement | System shows message to notify user to input only number as the fields’ requirements | | 3 | Length of field’s value is out of specified range. | System shows message to notify user which field’s value is out of range | | 4 | Import wrong Excel spreadsheet’s format | System shows message to notify user that the format of the imported spreadsheet is wrong |   **Relationships:** N/A  **Business Rules:** N/A   * Required fields must not be empty * In successful scenario, new information will be added to the system * “Students’ list” view contains following information in tabular form:   + Full name   + Student ID   + Tuition   + Retake course’s fees   + Fine level   + Payed amount * Excel spreadsheet’s format: | | | |

##### <Financial Staff> Update Financial Information

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – STA004** | | | |
| **Use Case No.** | STA004 | **Use Case Version** | 1.0 |
| **Use Case Name** | Update Financial Information | | |
| **Author** | Ho Doan Trung | | |
| **Date** | 25/09/2015 | **Priority** | High |
| **Actor:**   * Financial Staff   **Summary:**   * This use case allows Financial Staff to update information of a student’s finance   **Goal:**   * The system provides user a conventional mean to modify student’s financial information such as tuition, retake course fees , fine level, etc. when needed   **Triggers:**   * The staff will select function to enter financial information.   **Preconditions:**   * Staff login to the system with the rights of the Financial Staff   **Post Conditions:**   * **Success:** Financial information will be updated to the database. System notifies user that the data was added successfully * **Fail:** There is no information updated and the system will display error messages to the staff.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | **Step** | **Actor Action** | **System Response** | | 1 | User goes to “Students’ list” view and select a student, then sends command to update financial information to the system. | The system displays view that allows user to edit information:   * Tuition: number input, required, length 1-250 * Retake course fees: number input, required, length 1-250 * Fine level: Number input, required, length 1-250 * Payed amount: Number input, required, length 1-250 | | 2 | User alters fields’ values, then send command to add information to the system | The system validates fields’ values, then sends confirmation request to user  [Exception 1, 2, 3] | | 3 | User sends confirm command | * The system updates new information into storage if the data is valid * Show success message * Display “Students’ list” view |   **Alternative Scenario:** N/A  **Exceptions:**   |  |  |  | | --- | --- | --- | | **No** | **Cause** | **System Response** | | 1 | Any empty required fields | System shows message to notify user specific empty fields | | 2 | Any characters other than number in fields with number input requirement | System shows message to notify user to input only number as the fields’ requirements | | 3 | Length of field’s value is out of specified range. | System shows message to notify user which field’s value is out of range |   **Relationships:** N/A  **Business Rules:** N/A   * Required fields must not be empty * In successful scenario, new information will be updated to the system * “Students’ list” view contains following information in tabular form:   + Full name   + Student ID   + Tuition   + Retake course’s fees   + Fine level   + Payed amount | | | |

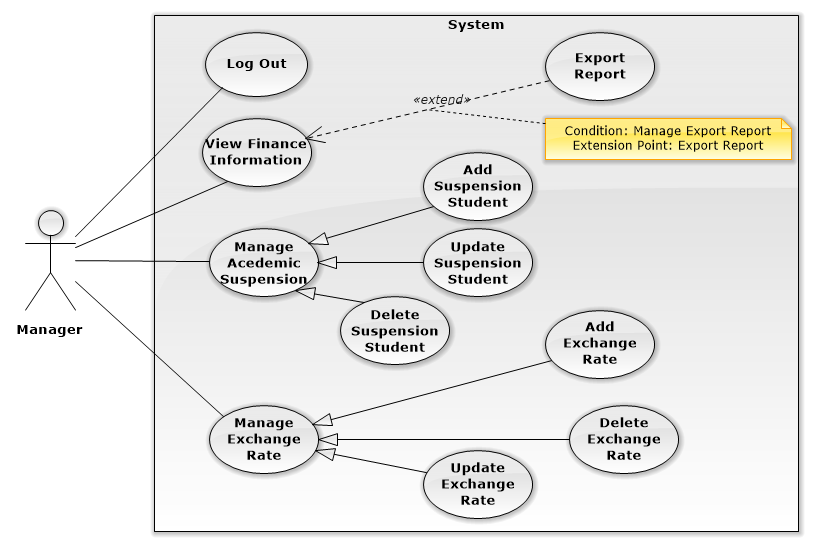
##### <Registrar Staff> Add Student’s Academic Record

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – STA005** | | | |
| **Use Case No.** | STA005 | **Use Case Version** | 1.0 |
| **Use Case Name** | Add Student’s Academic Record | | |
| **Author** | Ho Doan Trung | | |
| **Date** | 25/09/2015 | **Priority** | High |
| **Actor:**   * Student Service Staff   **Summary:**   * This use case allows user to add student’s study results of each semester into storage   **Goal:**   * The system provides user a conventional mean to add student’s study results of each semester such as grade and course’s status. New record will be added into system’s storage. User can also import those data from Excel spreadsheets   **Triggers:**   * User sends command to enter student information   **Preconditions:**   * User must log into the system with role as the Student Service Staff   **Post Conditions:**   * **Success:** The system adds new student’s study results into storage, then notified user that the data was successfully added * **Fail:** There is no information added and the system will display error messages to user.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | **Step** | **Actor Action** | **System Response** | | 1 | User goes to “Students’ list” view and select a student, then sends command to edit study record  [Alternative 1] | The system displays view that allows user to add student’s study results:   * Semester’s term: select one of these options: (Spring, Summer, Fall), required * Year: select one of the options, required * Semester’s start date: date, required * Semester’s end date: date, required * Academic progress: select one of these options: (ENG1, ENG2, ENG3, ENG4, ENG5, ENG6, CN1, CN2, CN3, CN4, CN5, CN6, CN7, CN8, CN9, none) * Course: select one of the options, required * Grade: number input, required, length 1-3 * Status: radio buttons, required | | 2 | User alters fields’ values, then sends add command to the system | The system validates fields’ value, sends confirmation request to user  [Exception 1, 2, 3] | | 3 | User sends confirm command | * The system adds new information into storage if the data is valid * Show success message * Display “Students’ list” view |   **Alternative Scenario:**  *[Alternative 1]*   |  |  |  | | --- | --- | --- | | **Step** | **Actor Action** | **System Response** | | 1 | User sends command to import excel spreadsheet’s data into storage. | The system will display view allowing user to import excel spreadsheet file to the system | | 2 | User selects an Excel file to import and then sends import command to the system. | The system validates the file, sends confirmation request to user  [Exception 5] | | 3 | User sends command to confirm importing the file | * The system adds new information into storage if the data is valid * Show success message * Display “Students’ list” view |   **Exceptions:**   |  |  |  | | --- | --- | --- | | **No** | **Cause** | **System Response** | | 1 | Any empty required fields | System shows message to notify user specific empty fields | | 2 | Any characters other than number in fields with number input requirement | System shows message to notify user to input only number as the fields’ requirements | | 3 | Length of field’s value is out of specified range. | System shows message to notify user which field’s value is out of range | | 4 | Import wrong Excel spreadsheet’s format | System shows message to notify user that the format of the imported spreadsheet is wrong | |   **Relationships:** N/A  **Business Rules:**   * In successful scenario, new information will be added to the system * “Status” has two options: Passed and Failed * “Course” options is loaded from the system’s storage. “Course” can be managed by Admin * “Grade” must be smaller than 10 * “Year” options is from 2000 to the current year. The system will add the next year option to storage at the end of the current year * When select “Semester’s term”   + If “Semester’s term” is Spring, the start date must be in January and the end date must be in April   + If “Semester’s term” is Summer, the start date must be in May and the end date must be in August   + If “Semester’s term” is Spring, the start date must be in September and the end date must be in December * “Students’ list” view contains following information in tabular form:   + Full name   + Gender   + Student ID   + Date of birth   + Address   + Email   + Admission date   + Start from Semester   + Entry English level   + Supported   + Parent’s phone number   + Parent’s email * Excel spreadsheet’s format: | | | |

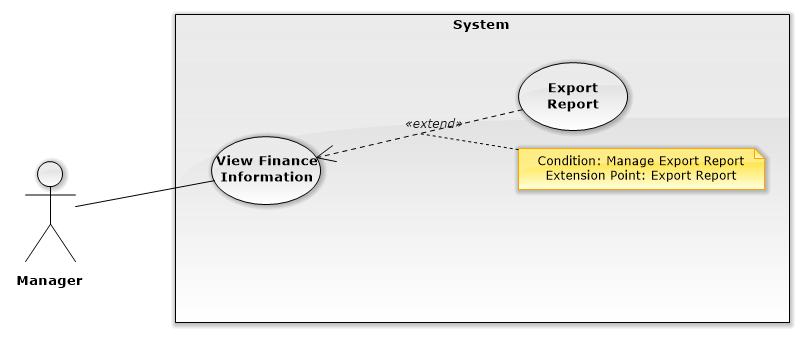
##### <Registrar Staff> Update Student’s Academic Record

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE – STA006** | | | |
| **Use Case No.** | STA006 | **Use Case Version** | 1.0 |
| **Use Case Name** | Update Student’s Academic Record | | |
| **Author** | Ho Doan Trung | | |
| **Date** | 25/09/2015 | **Priority** | High |
| **Actor:**   * Student Service Staff   **Summary:**   * This use case allows user to update student’s study results of each semester into storage   **Goal:**   * The system provides user a conventional mean to modify student’s study results of each semester such as grade and course’s status. New record will be added into system’s storage. User can also import those data from Excel spreadsheets   **Triggers:**   * User sends command to enter student information   **Preconditions:**   * User must log into the system with role as the Student Service Staff   **Post Conditions:**   * **Success:** The system updates new student’s study results into storage, then notified user that the data was successfully added * **Fail:** There is no information updated and the system will display error messages to user.   **Main Success Scenario:**   |  |  |  | | --- | --- | --- | | **Step** | **Actor Action** | **System Response** | | 1 | User goes to “Students’ list” view and select a student, then sends command to update new study record | The system displays view that allows user to edit new student’s study results:   * Semester’s term: select one of these options: (Spring, Summer, Fall), required * Year: select one of the options, required * Semester’s start date: date, required * Semester’s end date: date, required * Academic progress: select one of these options: (ENG1, ENG2, ENG3, ENG4, ENG5, ENG6, CN1, CN2, CN3, CN4, CN5, CN6, CN7, CN8, CN9, none) * Course: select one of the options, required * Grade: number input, required, length 1-3 * Status: select one of these options (Passed, Failed), required | | 2 | User enters new information, then sends update command to the system | The system validates fields’ value, sends confirmation request to user  [Exception 1, 2, 3] | | 3 | User sends command to confirm update new information to storage | * The system updates new information into storage if the data is valid * Show success message * Display “Students’ list” view |   **Alternative Scenario:** N/A  **Exceptions:**   |  |  |  | | --- | --- | --- | | **No** | **Cause** | **System Response** | | 1 | Any empty required fields | System shows message to notify user specific empty fields | | 2 | Any characters other than number in fields with number input requirement | System shows message to notify user to input only number as the fields’ requirements | | 3 | Length of field’s value is out of specified range. | System shows message to notify user which field’s value is out of range |   **Relationships:** N/A  **Business Rules:**   * Required fields must not be empty * In successful scenario, new information will be updated to the system * “Status” has two options: Passed and Failed * “Course” options is loaded from the system’s storage. “Course” can be managed by Admin * “Year” options is from 2000 to the current year. The system will add the next year option to storage at the end of the current year * When select “Semester’s term”   + If “Semester’s term” is Spring, the start date must be in January and the end date must be in April   + If “Semester’s term” is Summer, the start date must be in May and the end date must be in August   + If “Semester’s term” is Spring, the start date must be in September and the end date must be in December * “Grade” must be smaller than 10 * “Students’ list” view contains following information in tabular form:   + Full name   + Gender   + Student ID   + Date of birth   + Address   + Email   + Admission date   + Start from Semester   + Entry English level   + Supported   + Parent’s phone number   + Parent’s email | | | |

#### <Manager> Overview Use Case



##### <Manager> View Financial Information



|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE – MAN001 | | | |
| Use Case No. | MAN001 | **Use Case Version** | 1.0 |
| Use Case Name | View Financial Information | | |
| Author | Le Tuan Anh | | |
| Date | 27/09/2015 | **Priority** | Medium |
| Actor:   * Manager   Summary:   * This use case allows Manager to view students’ financial details.   Goal:   * The system displays students’ financial information such as loans, Payed amount, etc. This function help user keep track of student’s debt record.   Triggers:   * User sends command to view financial information.   Preconditions:   * User must login into the system with role manager.   Post Conditions:   * Success: Show a list of students and their financial information of student * Fail: Show error message.   Main Success Scenario:   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | User goes to students’ financial information list view | Display new view showing a list of students and their financial information in tabular form including:   * Ordering number * Student’s name   + Student’s tuition payments type * Financial details: link to view details of a student’s financial status | | 2 | User sends command to view “Financial details” | Display a chosen student’s financial details containing following information:   * Student’s name * Student’s code * Student’s school status * Student’s tuition payments type * Semesters list: select one of the options * A chosen semester’s financial details including following information:   + Semester’s information   + Date   + Payment detail description   + Initial tuition charge   + Deduction fees   + Actual tuition charge   + Payed amount   + Balance   + Tuition payment status   + The transmission date   + Bank’s name   + Additional note * Financial details for Retake course’s payment in selected semester including following information:   + Semester’s information   + Date   + Payment detail description   + Initial tuition charge   + Deduction fees   + Actual tuition charge   + Payed amount   + Balance   + Tuition payment status   + The transmission date   + Bank’s name   + Additional note   [Alternative 1, 2, 3] |   Alternative Scenario 1:  *[Alternative 1]*   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | User sends command to view “Financial details” | If “Student’s tuition payments type“ is CreditStudent and “Student’s school status” is Studying, display Student’s view as the Main flow, plus the following:   * Debt information:(displayed in tabular form)   + Date   + Loan’s description   + Tuition   + Payed amount   + Loan   + Exchange rate of dollar to Vietnamese dong   + Additional note |   *[Alternative 2]*   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | User sends command to view “Financial details” | If “Student’s tuition payments type“ is CreditStudent and “Student’s school status” is Graduated, display Student’s view as the Main flow, plus the following:   * Debt information:(displayed in tabular form)   + Debt   + Interest   + Payed amount   + Remaining debt   + Next repayment date   + Interest rate Additional note |   *[Alternative 3]*   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | User sends command to view “Financial details” | If “Student’s tuition payments type“ is InvestmentStudent, display Student’s view as the Main flow, plus the following:   * Debt information:   + Remaining debt   + Payed amount   + Interest rate   + Next repayment date |   Exceptions: N/A  Relationships: N/A  Business Rules:   * The information is loaded from the system’s storage * Semesters’ list includes all past semesters and the next semester * After each semester, Semesters’ list will added the next semester and calculate the tuition of that semester * “Remaining debt” is calculated by “Payed amount”, “Interest” and “Debt” * “Interest” is calculated by "Remaining debt” and “Interest rate”. * Repayment dates is exactly one month apart, started from the date of graduation. If graduation date is at the end of the month, then repayment dates is also at the end of each month   (e.g. If “Student’s date of graduation” is February 1st 2015, “Next repayment date” will be March 1st 2015, and the following “Next repayment date” will be April 1st 2015) | | | |

##### <Manager> Add Suspension Student

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE – MAN003 | | | |
| Use Case No. | MAN003 | **Use Case Version** | 1.0 |
| Use Case Name | Add Suspension Student | | |
| Author | AnhLT | | |
| Date | 27/09/2015 | **Priority** | Medium |
| Actor:   * Manager   Summary:   * This use case allows Manager to add students to suspension list.   Goal:   * The system provides Manager a conventional mean to add any students to suspension list and the reason for their suspension.   Triggers:   * User sends command to add new student to suspension list.   Preconditions:   * User must login into the system with role Manager.   Post Conditions:   * Success: The system stores new information into storage. System notifies user that the data was added successfully; show the list of suspended students. * Fail: Show error message.   Main Success Scenario:   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | User goes to “Suspension list” view and sends command to add new student | The system displays view that allows user to input information for a new suspended student:   * Student’s code: free text input, required, length 8 * Student’s name: free text input, required, 3-250 * Suspension period: date time input, required * Reason for the suspension: free text input, required, 3-250 | | 2 | User enters new information, then sends add command to the system | The system validates fields’ value, sends confirmation request to user  [Exception 1,2] | | 3 | User sends confirm command | * The system adds new information into storage if the data is valid * Show success message * Display “Suspension list” view |   Alternative Scenario: N/A  Exceptions:   |  |  |  | | --- | --- | --- | | **No** | **Cause** | **System Response** | | 1 | Any empty required fields | System shows message to notify user specific empty fields | | 2 | Length of field’s value is out of specified range. | System shows message to notify user which field’s value is out of range |   Relationships: N/A  Business Rules:   * Required fields must not be empty * In successful scenario, new information will be added to the system * The information is loaded from the system’s storage. * “Suspension list” view contains following information in tabular form:   + Student’s code   + Student’s name   + Suspension period   + Reason for the suspension | | | |

##### <Manager>Update Suspension List

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE – MAN004 | | | |
| Use Case No. | MAN004 | **Use Case Version** | 1.0 |
| Use Case Name | Update Suspension List | | |
| Author | Le Tuan Anh | | |
| Date | 27/09/2015 | **Priority** | Medium |
| Actor:   * Manager   Summary:   * This use case allows Manager to add students to suspension list.   Goal:   * The system provides Manager a conventional mean to update any students in suspension list and the reason for their suspension when needed.   Triggers:   * User sends command to modify a suspended student’s information.   Preconditions:   * User must login into the system with role Manager.   Post Conditions:   * Success: The system updates new information into storage. System notifies user that the data was updated successfully; show the list of suspended students. * Fail: Show error message.   Main Success Scenario:   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | User goes to “Suspension list” view and sends command to update a suspended student | The system displays view that allows user to alter information of a suspended student:   * Student’s code: free text input, required, length 8 * Student’s name: free text input, required, 3-250 * Suspension period: date time input, required * Reason for the suspension: free text input, required, 3-250 | | 2 | User alters existing information, then sends update command to the system | The system validates fields’ value, sends confirmation request to user  [Exception 1,2] | | 3 | User sends confirm command | * The system adds new information into storage if the data is valid * Show success message * Display “Suspension list” view |   Alternative Scenario: N/A  Exceptions:   |  |  |  | | --- | --- | --- | | **No** | **Cause** | **System Response** | | 1 | Any empty required fields | System shows message to notify user specific empty fields | | 2 | Length of field’s value is out of specified range. | System shows message to notify user which field’s value is out of range |   Relationships: N/A  Business Rules:   * Required fields must not be empty * In successful scenario, new information will be updated to the system * The information is loaded from the system’ storage. * “Suspension list” view contains following information in tabular form:   + Student’s code   + Student’s name   + Suspension period   + Reason for the suspension * Each textbox has pre-inputted text showing current field’s value | | | |

##### <Manager>Delete Suspended Student

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE – MAN005 | | | |
| Use Case No. | MAN005 | **Use Case Version** | 2.5 |
| Use Case Name | Delete Suspended Student | | |
| Author | Le Tuan Anh | | |
| Date | 27/09/2015 | **Priority** | Medium |
| Actor:   * Manager   Summary:   * This use case allows Manager to delete students from suspension list.   Goal:   * The system provides Manager a conventional mean to delete any students from suspension list when needed.   Triggers:   * User sends command to delete a suspended student from the list.   Preconditions:   * User must login into the system with role Manager.   Post Conditions:   * Success: The system deletes a student from suspension list and removes the data from database. System notifies user that the data was removed successfully; show the list of suspended students. * Fail: Show error message.   Main Success Scenario:   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | User goes to “Suspension list” view, select a student and sends command to delete a suspended student | Send confirmation request to user | | 2 | User sends command to confirm remove a student from suspension list | Delete a student from suspension list and remove the data from database, then notifies user that the data was successfully removed.  Display refreshed “Suspension list” view |   Alternative Scenario: N/A  Exceptions: N/A  Relationships: N/A  Business Rules:   * In successful scenario, existing information will be removed from the system * The information is loaded from the system’s storage. * “Suspension list” view contains following information in tabular form:   + Student’s code   + Student’s name   + Suspension period   + Reason for the suspension | | | |

##### <Manager> Add Exchange Rate

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE – MAN006 | | | |
| Use Case No. | MAN006 | **Use Case Version** | 1.0 |
| Use Case Name | Add Financial Rates | | |
| Author | Le Tuan Anh | | |
| Date | 27/09/2015 | **Priority** | Medium |
| Actor:   * Manager   Summary:   * This use case allows Manager to add new financial rates.   Goal:   * The system provides Manager a conventional mean to add the exchange rate of USD to VND or interest rate.   Triggers:   * User sends command to add exchange rate.   Preconditions:   * User must login into the system with role Manager.   Post Conditions:   * Success: The system stores new information into storage. System notifies user that the data was added successfully. * Fail: Show error message.   Main Success Scenario:   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | User goes to “Exchange rate” view and sends command to add new information  [Alternative 1] | The system displays view that allows user to input information:   * US Dollar: number input, required, length 1-250 * Viet Nam Dong: number input, required, length 1-250 | | 2 | User enters new information, then sends add command to the system | The system validates fields’ value, sends confirmation request to user  [Exception 1, 2] | | 3 | User sends command to confirm add new information to storage | The system stores new information to storage then notifies user that the data was successfully added. |   Alternative Scenario:  *[Alternative 1]*   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | User goes to “Interest rate” view and sends command to add new information | The system displays view that allows user to input information:   * Interest rate: number input, required, length 0-100 | | 2 | User enters new information, then sends add command to the system | The system validates fields’ value, sends confirmation request to user  [Exception 1, 2, 3] | | 3 | User sends command to confirm add new information to storage | The system stores new information to storage, then notifies user that the data was successfully added. |   Exceptions:   |  |  |  | | --- | --- | --- | | **No** | **Cause** | **System Response** | | 1 | Any empty required fields | System shows message to notify user specific empty fields | | 2 | Any characters other than number in fields with number input requirement | System shows message to notify user to input only number as the fields’ requirements | | 3 | Length of field’s value is out of specified range. | System shows message to notify user which field’s value is out of range |   Relationships: N/A  Business Rules:   * In successful scenario, new information will be added to the system * Required fields must not be empty | | | |

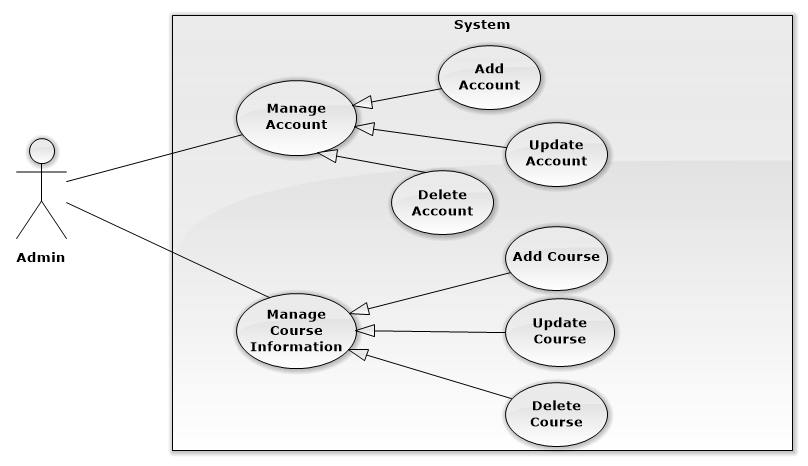
##### <Manager> Update Exchange Rate

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE – MAN007 | | | |
| Use Case No. | MAN007 | **Use Case Version** | 2.5 |
| Use Case Name | Update Financial Rates | | |
| Author | Le Tuan Anh | | |
| Date | 27/09/2015 | **Priority** | Medium |
| Actor:   * Manager   Summary:   * This use case allows Manager to update new financial rates.   Goal:   * The system provides Manager a conventional mean to edit the exchange rate of USD to VND or interest rate.   Triggers:   * User sends command to update exchange rate.   Preconditions:   * User must login into the system with role Manager.   Post Conditions:   * Success: The system updates new information into storage. System notifies user that the data was updated successfully. * Fail: Show error message.   Main Success Scenario:   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | User goes to “Exchange rate” view and sends command to modify information  [Alternative 1] | The system displays view that allows user to edit information:   * US Dollar: number input, required * Viet Nam Dong: number input, required | | 2 | User enters new information, then sends update command to the system | The system validates fields’ value, sends confirmation request to user  [Exception 1, 2] | | 3 | User sends confirm command | * The system adds new information into storage if the data is valid * Show success message * Display “Suspension list” view |   Alternative Scenario:  *[Alternative 1]*   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | User goes to “Interest rate” view and sends command to update new information | The system displays view that allows user to input information:   * Interest rate: number input, required, length 1-250 | | 2 | User enters new information, then sends update command to the system | The system validates fields’ value, sends confirmation request to user  [Exception 1, 2, 3] | | 3 | User sends command to confirm update new information to storage | The system updates new information to storage, then notifies user that the data was successfully updated. |   Exceptions:   |  |  |  | | --- | --- | --- | | **No** | **Cause** | **System Response** | | 1 | Any empty required fields | System shows message to notify user specific empty fields | | 2 | Any characters other than number in fields with number input requirement | System shows message to notify user to input only number as the fields’ requirements | | 3 | Length of field’s value is out of specified range. | System shows message to notify user which field’s value is out of range |   Relationships: N/A  Business Rules:   * In successful scenario, new information will be updated to the system * Required fields must not be empty | | | |

##### <Manager> Delete Exchange Rate

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE – MAN007 | | | |
| Use Case No. | MAN007 | **Use Case Version** | 1.0 |
| Use Case Name | Delete Exchange Rate | | |
| Author | Le Tuan Anh | | |
| Date | 27/09/2015 | **Priority** | Medium |
| Actor:   * Manager   Summary:   * This use case allows Manager to delete exchange rate’s information.   Goal:   * The system provides Manager a conventional mean to delete exchange rate’s value when needed   Triggers:   * User sends command to delete an exchange rate.   Preconditions:   * User must login into the system with role Manager.   Post Conditions:   * Success: The system removes information from the storage. Show success message. * Fail: Show error message.   Main Success Scenario:   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | User goes to “Exchange rate” view and sends command to delete an exchange rate | The system sends confirmation request to user | | 2 | User sends confirm command | * The system removes the account from the storage * Show success message |   Alternative Scenario: N/A  Exceptions: N/A  Relationships: N/A  Business Rules:   * In successful scenario, existing information will be removed from the system * The information is loaded from the system’s storage. | | | |

#### <Admin> Overview Use Case



##### <Admin> Add Account

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE – ADM001 | | | |
| Use Case No. | ADM001 | **Use Case Version** | 1.0 |
| Use Case Name | Add Account | | |
| Author | Vu Nhat Anh Khoa | | |
| Date | 27/09/2015 | **Priority** | Medium |
| Actor:   * Manager   Summary:   * This use case allows Admin to add new account.   Goal:   * The system provides Manager a conventional mean to create new account and assign role to it.   Triggers:   * User sends command to create new account.   Preconditions:   * User must login into the system with role Admin.   Post Conditions:   * Success: The system stores new information into storage. Show success message. * Fail: Show error message.   Main Success Scenario:   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | User goes to “Accounts’ list” view and sends command to add new account | The system displays view that allows user to input information for a new account:   * Account’s name: free text input, required, length 3-250 * Role: choose one of these options: (Student, CreditStudent, InvestmentStudent, Student Service Staff, Registrar Staff, Financial Staff, Manager, Scheduler), required * Password: free text input, required, length 6-32 * Confirm password: free text input, required, length 6-32 | | 2 | User enters new information, then sends create command to the system | The system validates fields’ value, sends confirmation request to user  [Exception 1,2,3] | | 3 | User sends confirm command | * The system adds new information into storage if the data is valid * Show success message * Display “Accounts’ list” view |   Alternative Scenario: N/A  Exceptions:   |  |  |  | | --- | --- | --- | | **No** | **Cause** | **System Response** | | 1 | Any empty required fields | System shows message to notify user specific empty fields | | 2 | If the “Password” and the “Confirm password” are not matched. | System shows message to notify user that the confirm password is not matched with the new password. |   Relationships: N/A  Business Rules:   * Required fields must not be empty * In successful scenario, new information will be added to the system * The information is loaded from the system’ storage. * “Role” is loaded from the system’s storage. “Role” can be manage by Admin * “Accounts’ list” view contains following information in tabular form:   + Ordering number   + Account’s name   + Role | | | |

##### <Admin> Edit Account

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE – ADM002 | | | |
| Use Case No. | ADM002 | **Use Case Version** | 1.0 |
| Use Case Name | Edit Account | | |
| Author | Vu Nhat Anh Khoa | | |
| Date | 27/09/2015 | **Priority** | Medium |
| Actor:   * Manager   Summary:   * This use case allows Admin to edit existing accounts.   Goal:   * The system provides Manager a conventional mean to edit any accounts when needed   Triggers:   * User sends command to edit an account.   Preconditions:   * User must login into the system with role Admin.   Post Conditions:   * Success: The system update new information into storage. Show success message. * Fail: Show error message.   Main Success Scenario:   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | User goes to “Accounts’ list” view and sends command to edit an account | The system displays view that allows user to input information for a new account:   * Account’s name: free text input, required, length 3-250 * Role: choose one of these options: (Student, CreditStudent, InvestmentStudent, Student Service Staff, Registrar Staff, Financial Staff, Manager, Scheduler), required * New password: free text input, required, length 6-32 * Confirm new password: free text input, required, length 6-32 | | 2 | User alters information, then sends edit command to the system | The system validates fields’ value, sends confirmation request to user  [Exception 1,2] | | 3 | User sends confirm command | * The system adds new information into storage if the data is valid * Show success message * Display “Accounts’ list” view |   Alternative Scenario: N/A  Exceptions:   |  |  |  | | --- | --- | --- | | **No** | **Cause** | **System Response** | | 1 | Any empty required fields | System shows message to notify user specific empty fields | | 2 | If the “Password” and the “Confirm password” are not matched. | System shows message to notify user that the confirm password is not matched with the new password. |   Relationships: N/A  Business Rules:   * Required fields must not be empty * In successful scenario, new information will be updated to the system * The information is loaded from the system’s storage. * “Role” is loaded from the system’s storage. “Role” can be manage by Admin * “Accounts’ list” view contains following information in tabular form:   + Ordering number   + Account’s name   + Role | | | |

##### <Admin> Delete Account

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE – ADM003 | | | |
| Use Case No. | ADM003 | **Use Case Version** | 1.0 |
| Use Case Name | Delete Account | | |
| Author | Vu Nhat Anh Khoa | | |
| Date | 27/09/2015 | **Priority** | Medium |
| Actor:   * Manager   Summary:   * This use case allows Admin to delete existing accounts.   Goal:   * The system provides Manager a conventional mean to delete any accounts when needed   Triggers:   * User sends command to delete an account.   Preconditions:   * User must login into the system with role Admin.   Post Conditions:   * Success: The system removes information from the storage. Show success message. * Fail: Show error message.   Main Success Scenario:   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | User goes to “Accounts’ list” view and sends command to delete an account | The system sends confirmation request to user | | 2 | User sends confirm command | * The system removes the account from the storage * Show success message * Display “Accounts’ list” view |   Alternative Scenario: N/A  Exceptions: N/A  Relationships: N/A  Business Rules:   * In successful scenario, existing information will be removed from the system * The information is loaded from the system’s storage. * “Accounts’ list” view contains following information in tabular form:   + Ordering number   + Account’s name   + Role | | | |

##### <Admin> Add Course

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE – ADM004 | | | |
| Use Case No. | ADM004 | **Use Case Version** | 1.0 |
| Use Case Name | Add Account | | |
| Author | Vu Nhat Anh Khoa | | |
| Date | 27/09/2015 | **Priority** | Medium |
| Actor:   * Manager   Summary:   * This use case allows Admin to add new course’s information.   Goal:   * The system provides Manager a conventional mean to add course’s information or import the data from excel spreadsheets   Triggers:   * User sends command to create new account.   Preconditions:   * User must login into the system with role Admin.   Post Conditions:   * Success: The system stores new information into storage. Show success message. * Fail: Show error message.   Main Success Scenario:   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | User goes to “Course list” view and sends command to add new course | The system displays view that allows user to input information for a new account:   * Course’s name: free text input, required, length 3-250 * Course’s code: free text input, required, length 3-6 * Numbers of credits: number input, required, length 1-3 * Prerequisite course: choose one of the options, not required | | 2 | User enters new information, then sends create command to the system | The system validates fields’ value, sends confirmation request to user  [Exception 1,2] | | 3 | User sends confirm command | * The system adds new information into storage if the data is valid * Show success message * Display “Courses’ list” view |   Alternative Scenario:  *[Alternative 1]*   |  |  |  | | --- | --- | --- | | **Step** | **Actor Action** | **System Response** | | 1 | User sends command to import excel spreadsheet’s data into storage. | The system will display view allowing user to import excel spreadsheet file to the system | | 2 | User selects an Excel file to import and then sends import command to the system. | The system validates the file, sends confirmation request to user  [Exception 5] | | 3 | User sends command to confirm importing the file | * The system adds new information into storage if the data is valid * Show success message * Display “Courses’ list” view |   Exceptions:   |  |  |  | | --- | --- | --- | | **No** | **Cause** | **System Response** | | 1 | Any empty required fields | System shows message to notify user specific empty fields | | 2 | Any characters other than number in fields with number input requirement | System shows message to notify user to input only number as the fields’ requirements | | 3 | Length of field’s value is out of specified range. | System shows message to notify user which field’s value is out of range | | 4 | Import wrong Excel spreadsheet’s format | System shows message to notify user that the format of the imported spreadsheet is wrong |   Relationships: N/A  Business Rules:   * Required fields must not be empty * In successful scenario, new information will be added to the system * The information is loaded from the system’ storage * “Prerequisite course” is loaded options is loaded from the system’s storage. “Prerequisite course” is a list of courses that was created by this use case * “Course list” view contains following information in tabular form:   + Ordering number   + Course’s name   + Course’s code   + Numbers of Credits   + Prerequisite course * Excel spreadsheet’s format | | | |

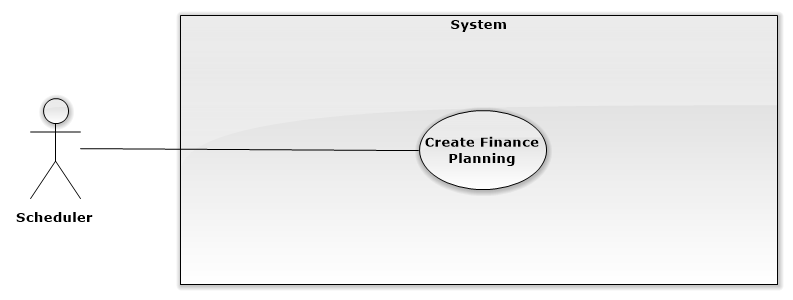
##### <Admin> Edit Course

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE – ADM005 | | | |
| Use Case No. | ADM005 | **Use Case Version** | 1.0 |
| Use Case Name | Edit Account | | |
| Author | Vu Nhat Anh Khoa | | |
| Date | 27/09/2015 | **Priority** | Medium |
| Actor:   * Manager   Summary:   * This use case allows Admin to edit existing courses.   Goal:   * The system provides Manager a conventional mean to edit any courses when needed   Triggers:   * User sends command to edit a course.   Preconditions:   * User must login into the system with role Admin.   Post Conditions:   * Success: The system update new information into storage. Show success message. * Fail: Show error message.   Main Success Scenario:   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | User goes to “Course list” view and sends command to edit a course | The system displays view that allows user to input information for a new account:   * Course’s name: free text input, required, length 3-250 * Course’s code: free text input, required, length 3-6 * Numbers of credits: number input, required, length 1-3 * Prerequisite course: choose one of the options, not required | | 2 | User enters new information, then sends edit command to the system | The system validates fields’ value, sends confirmation request to user  [Exception 1,2] | | 3 | User sends confirm command | * The system adds new information into storage if the data is valid * Show success message * Display “Courses’ list” view |   Alternative Scenario: N/A  Exceptions:   |  |  |  | | --- | --- | --- | | **No** | **Cause** | **System Response** | | 1 | Any empty required fields | System shows message to notify user specific empty fields | | 2 | Any characters other than number in fields with number input requirement | System shows message to notify user to input only number as the fields’ requirements | | 3 | Length of field’s value is out of specified range. | System shows message to notify user which field’s value is out of range |   Relationships: N/A  Business Rules:   * Required fields must not be empty * In successful scenario, new information will be updated to the system * The information is loaded from the system’s storage. * “Course list” view contains following information in tabular form:   + Ordering number   + Course’s name   + Course’s code   + Numbers of Credits   + Prerequisite course | | | |

##### <Admin> Delete Course

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE – ADM006 | | | |
| Use Case No. | ADM006 | **Use Case Version** | 1.0 |
| Use Case Name | Delete Account | | |
| Author | Vu Nhat Anh Khoa | | |
| Date | 27/09/2015 | **Priority** | Medium |
| Actor:   * Manager   Summary:   * This use case allows Admin to delete existing courses.   Goal:   * The system provides Manager a conventional mean to delete any courses when needed   Triggers:   * User sends command to delete a course.   Preconditions:   * User must login into the system with role Admin.   Post Conditions:   * Success: The system removes information from the storage. Show success message. * Fail: Show error message.   Main Success Scenario:   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | User goes to “Accounts’ list” view and sends command to delete an account | The system sends confirmation request to user | | 2 | User sends confirm command | * The system removes the account from the storage * Show success message * Display “Courses’ list” view |   Alternative Scenario: N/A  Exceptions: N/A  Relationships: N/A  Business Rules:   * In successful scenario, existing information will be removed from the system * The information is loaded from the system’s storage. * “Courses’ list” view contains following information in tabular form:   + Ordering number   + Course’s name   + Course’s code   + Numbers of Credits   + Prerequisite course | | | |

#### <Scheduler> Overview Use Case



##### <Scheduler> Create Financial Plan

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE – SCH001 | | | |
| Use Case No. | SCH001 | **Use Case Version** | 1.0 |
| Use Case Name | Create Financial Plan | | |
| Author | Vu Nhat Anh Khoa | | |
| Date | 27/09/2015 | **Priority** | Medium |
| Actor:   * Scheduler   Summary:   * This use case allows Scheduler to make new plan of tuition payments for students.   Goal:   * The system provides Scheduler a conventional mean to send mails reminding the tuition for the next semester and tuition payments deadline to all active students   Triggers:   * User sends command to create new financial plan.   Preconditions:   * User must login into the system with role Scheduler.   Post Conditions:   * Success: The system stores new information into storage, sends mail to all students. Show success message. * Fail: Show error message.   Main Success Scenario:   |  |  |  | | --- | --- | --- | | Step | Actor Action | System Response | | 1 | System checks the current time, if it hits configured time, the process starts | * The system matches each student with their respective academic information from the storage * Calculate tuition and fees for the next semester * Sends mails to all active student   [Exception 1]   * Show success message |   Alternative Scenario: N/A  Exceptions:   |  |  |  | | --- | --- | --- | | **No** | **Cause** | **System Response** | | 1 | Sent mails failed | Generate log file |   Relationships: N/A  Business Rules:   * Academic information is taken from Academic Records containing: Semester’s period, academic progress, incomplete course * Log file structure: | | | |

## Software System Attribute

#### Usability

##### Graphic User Interface

* All the texts, labels and alerts will be written in Vietnamese, except some English terms that the school is using.

##### Usability

* The system is easy to use. The school’s staffs will need less than 3 days of training to use the system effectively.
* Students don’t need any training to use the system.

##### Installation

* User can follow installation and manual guide for installation. If there are any problems, users can contact developers for help.

#### Reliability

* System notification success rate is less than 2 failed notifications per 1000 sent.
* System email notification deliver success rate is less than 2 failed deliveries per 1000 sent.
* Scheduler task run at 00:00 every day with 100% execution rate.

#### Availability

* **N/A**

#### Security

* All input data will be validated before saving into storage.
* All privacy information such as password will be encrypted to ensure security.
* The system automatically authorizes all users when they login to the system.

#### Maintainability

* The system is separated into modules.

#### Portability

* All users can use application on every OS supported web browser.

#### Performance

* Requests from web application are responded in less than 10 seconds at 8 Mbps bandwidth speed.

## Conceptual Diagram

Figure 35: ERD - Conceptual Diagram