HANU's New Educational Management System

Software Architecture Document

Version 1.1

Revision History

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 10/Nov/19 | 1.0 | Initial release | Đỗ Hải Bình |
| 25/Dec/19 | 1.1 | Final release | Đỗ Hải Bình |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Introduction 4

1.1 Purpose 4

1.2 Scope 4

1.3 Definitions, Acronyms, and Abbreviations 4

1.4 References 4

1.5 Overview 4

2. Architectural Representation 4

3. Architectural Goals and Constraints 4

4. Use-Case View 5

4.1 Use-Case Realizations 5

4.1.1 Use-Case Realization: Add Course Registration 5

4.1.2 Use-Case Realization: Create Course 6

5. Logical View 6

5.1 Overview 7

5.1.1 Views 7

5.1.2 Controllers 7

5.1.3 Models 7

5.1.4 Repositories 7

5.1.5 Toolkits 7

5.2 Architecturally Significant Design Packages 7

5.2.1 Models 7

5.2.2 Controllers 10

5.2.3 Views 14

6. Process View 15

6.1 Requesting & Routing 15

6.2 Controller Initialization & Action Execution 15

6.3 Result Execution & Response 15

7. Deployment View 16

8. Implementation View 16

9. Data View 16

10. Size and Performance 17

11. Quality 17

Software Architecture Document

# Introduction

## Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

## Scope

This Software Architecture Document applies to the Educational Management System for Hanoi University, which will be developed by Active Service Provider Network.

## Definitions, Acronyms, and Abbreviations

DBMS – Database Management System

## References

[1] ***Vision***, Do Hai Binh, November 06, 2019, Active Service Provider Network.

[2] ***Supplementary Specification***, Do Hai Binh, October 20, 2019, Active Service Provider Network.

[3] ***Use-Case Realization Specification: Add Course Registration***, Do Hai Binh, November 21, 2019, Active Service Provider Network.

[4] ***Use-Case Realization Specification: Create Course***, Nguyen Cong Quang, November 06, 2019

## Overview

Section Two denotes how the architecture is presented. The goals and constraints of the representation given in Section Two are described in Section Three. Then, the Use-case View, Logical View, Process View, Deployment View are briefly presented in relation to the architecture employed in the project. Finally, criteria concerning with size, performance and quality of the system will be proposed.

# Architectural Representation

This document presents the architecture as a series of views; use case view, logical view, process view and deployment view. There is no separate implementation view described in this document. These are views on an underlying Unified Modeling Language (UML) model developed using Visual Paradigm and Lucid Chart.

# Architectural Goals and Constraints

Below are the key requirements and system constraints on the architecture:

* The system’s data must be modelled in such a way to best integrate with existing documenting method used in the university.
* The system must be available both within the university’s intranet and on the Net outside the university.
* The system must ensure complete protection of data from unauthorized access. All accesses are subject to user identification and password control.
* The Educational Management System will be implemented as a client-server system. The client portion resides on PCs and the server portion must operate on Hanoi University’s Linux server. This will be done by using Docker images to contain .NET Core web application.
* All performance and loading requirements, as stipulated in the Vision Document [1] and the Supplementary Specification [2], must be taken into consideration as the architecture is being developed.

# Use-Case View

This system will be built upon Model-View-Controller architecture; therefore, every use-case is equally architecturally covered. Out of the list of system use-cases (refer to Use-Case folder within Requirement Definition folder), the most significant use-cases of our system are:

* Register in course (including ***Add Course Registration***, ***Modify Course Registration*** and ***Delete Course Registration***)
* Manage courses and course classes (including ***Create Course***, ***Edit Course***, ***Delete Course***, ***Create Course Class***, ***Edit Course Class*** and ***Delete Course Class***)

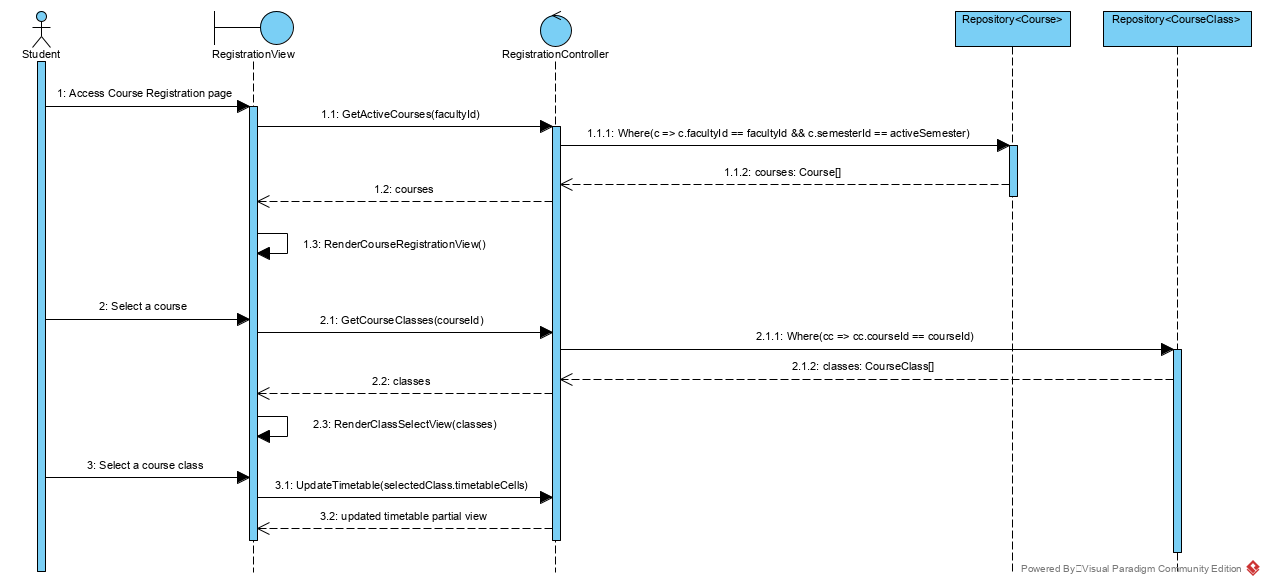
## Use-Case Realizations

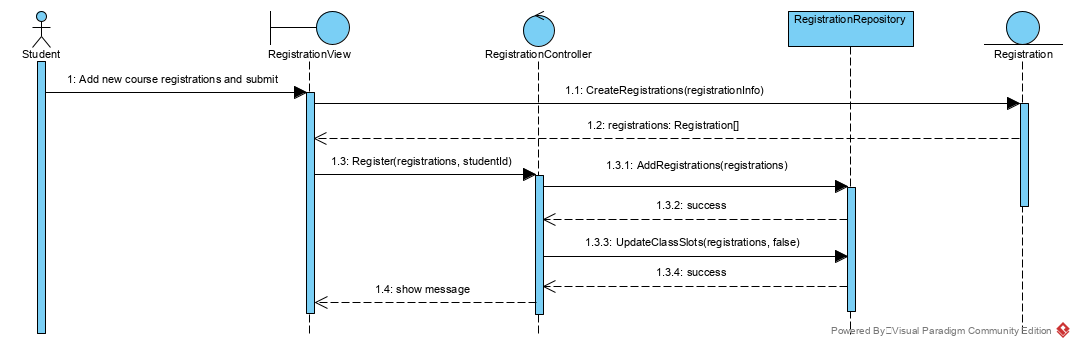
This section illustrates how the software actually works by giving a few selected use-case realizations, and explains how the various design model elements contribute to their functionality.

The complete Use-Case Realizations can be found within Use-Case Realization folder. Within this section, use-case realizations for ***Add Course Registration*** and ***Create Course*** use-cases are briefly discussed.

### Use-Case Realization: Add Course Registration

* Description: Student makes a new registration into a specific class of a course during a registration period.
* Realization Summary: This use-case involve using one view class (RegistrationView), one Controller class (RegistrationController), and four types of models (Registration, Course, CourseClass, TimetableCell).
* When a student makes a request to the server by interacting with the View, the Controller action associated with the requested route is automatically called. The sequence according to which the methods are called is describe in two sequence diagrams (shown below as an example):





* The models are passed into View through Controller. Course is used to display a course catalog on the page, while CourseClass shows a list of available classes for a specific course, which students can interact with to register into their desired course classes. Class TimetableCell represents cells on the timetable and takes part in the process of updating the timetable preview. The class Registration is used to store registration information and push the registration changes into the database through the use of RegistrationRepository.

### Use-Case Realization: Create Course

* Description: Administrator user creates a new course that will become active later.
* Realization Summary: Four types of objects take part in this use-case: CourseView, CourseController, Course and Repository which is a data access class.
* When the administrator makes a request to the server, in the same way as in section 4.1.1, the associated Controller action is invoked. The sequence is illustrated by the sequence diagram located in the Use-case Realization Specification document (refer to reference ***[4]***).
* In this use-case, administrator inputs all the required fields in the create form before submitting the form to the server. Server again calls the corresponding Controller, but this time, the model (Course) is used to bind input parameters to a proper .NET object. The Course model is also responsible for storing data onto the database through the use of the generic Repository class.

# Logical View

A description of the logical view of the architecture. Describes the most important classes, their organization in service packages and subsystems, and the organization of these subsystems into layers. Also describes the most important use-case realizations, for example, the dynamic aspects of the architecture. Class diagrams may be included to illustrate the relationships between architecturally significant classes, subsystems, packages and layers.

The logical view of the course registration system is comprised of the 3 main packages: Models, Controllers, Views. There are also Toolkits package and Repositories package to support the main packages.

The Views Package contains classes for each of the forms that the actors use to communicate with the System.

The Controllers Package consists of all controller classes that represent use case managers that make up the overall application behavior.

The Models Package includes entity classes representing university artifacts (example: course, timetable, tuition fee, …).

The system accesses the database by means of data access classes contained inside Repositories package.

There are also other toolkits to support the overall operation of the system. They are contained inside Toolkits package.

## Overview

### Views

All the front-end views are contained within the Views package. User interacts with these views, and the views in turn call the corresponding controllers for data processing. The views do not directly get the data – they only display whatever data passed to them by the controllers.

### Controllers

This package contains classes representing business logic. All the processing jobs take place within the controllers. The controllers get data from database by means of data access objects located within the Repositories package, and return correct view with data for display.

### Models

Models package include entity objects representing actual business artifacts. The model objects are employed to map data from database tables and rows onto .NET objects to be used in business logic processing. These objects are also passed to views from controllers to display data to end-user.

### Repositories

Repositories package includes classes that extend generic Repository class (which was in-house developed and located in DbTools assembly). Classes located within this package are used to perform various operations on the database, including Select, Insert and Delete.

### Toolkits

Toolkit classes are helper classes to simplify the work to be done in other packages. User validation is also included within this package.

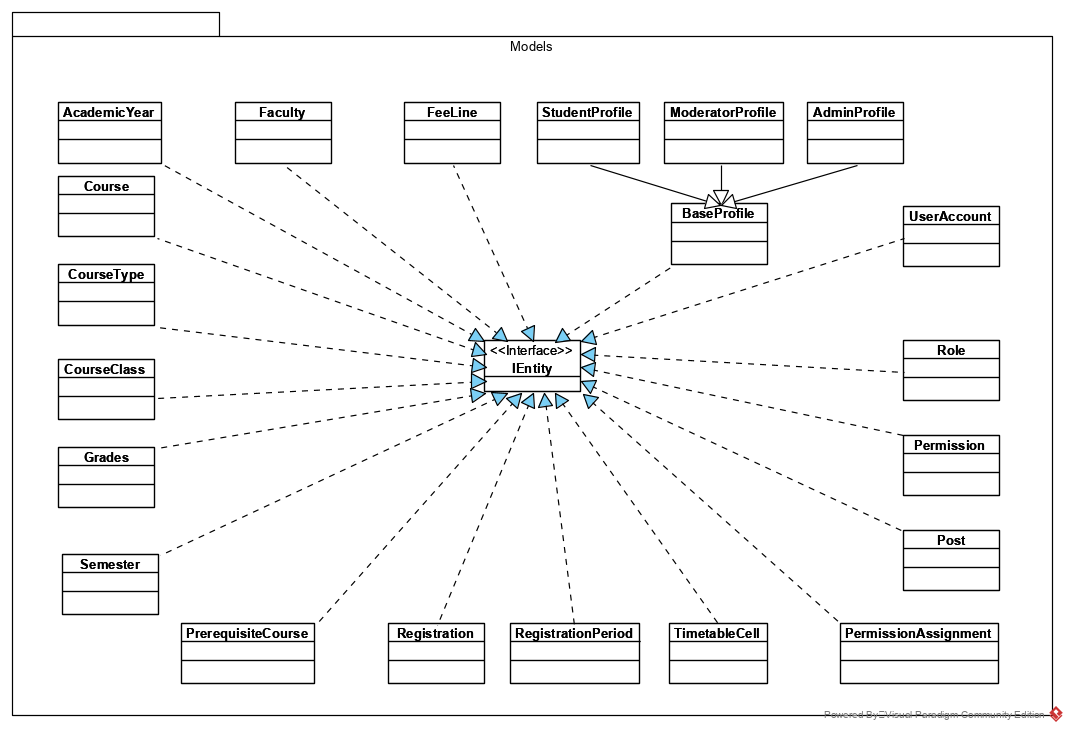
## Architecturally Significant Design Packages

As this project is built upon Model-View-Controller architecture, the most important packages are Models, Views, Controllers. This section shall provide top-level diagrams of each package. For a more detailed view of the classes within each package, refer to the Visual Paradigm illustration of Design Model located in Design Model folder.

### Models

Model classes represent domain-specific data and encapsulate business logic. In ASP.NET Core, model classes hold data in public properties, and take part in retrieving and storing data in persistent stores like databases with data access classes.

In this project, all the classes located in Models package implement IEntity interface so as to make it possible to map data pulled from database by Repositories to them directly (except BaseProfile as it is the supertype for all types of Profile in the system). Note that the IEntity interface is actually located in DbTools assembly.



In the Educational Management System, in order for core functionalities to be implemented, these classes play a vital role: UserAccount, StudentProfile, Registration, Course, CourseClass, TimetableCell, Grades, FeeLine. Following is the specifications of these models. The models do not get data from database themselves, but are used as data containers by classes located within Repository package.

**Class UserAccount:**

| **Name** | UserAccount | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Brief Description** | Data model class for user\_account entity in the database | | | | | | |
| **Attributes** | | | | | | | |
| **Name** | **Type** | **Access** | **Mutable** | **Optional** | **Length** | **Min** | **Max** |
| accountId | int | private | True | False | N/A | 1 | N/A |
| username | String | private | True | False | 15 | N/A | N/A |
| password | String | private | True | False | 64 | N/A | N/A |
| created | DateTime | private | True | False | N/A | N/A | N/A |
| lastLogin | DateTime | private | True | False | N/A | N/A | N/A |
| roleId | int | private | True | False | N/A | 1 | N/A |
| **Operations:** Only public get/set operations corresponding to the attributes, with instance scope. | | | | | | | |

**Class StudentProfile:**

| **Name** | StudentProfile | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Brief Description** | Data model class for student\_profile entity in the database | | | | | | |
| **Attributes** | | | | | | | |
| **Name** | **Type** | **Access** | **Mutable** | **Optional** | **Length** | **Min** | **Max** |
| studentId | int | private | True | False | N/A | 1 | N/A |
| profileId | int | private | True | False | N/A | 1 | N/A |
| profileName | String | private | True | False | 40 | N/A | N/A |
| dob | DateTime | private | True | False | N/A | N/A | N/A |
| gender | Gender | private | True | False | N/A | N/A | N/A |
| nationality | String | private | True | False | 25 | N/A | N/A |
| hometown | String | private | True | False | 63 | N/A | N/A |
| phoneNumber | String | private | True | False | 10 | N/A | N/A |
| email | String | private | True | False | 255 | N/A | N/A |
| passedCreditCount | int | private | True | False | N/A | 0 | N/A |
| overallMark | double | private | True | False | N/A | 0 | 10 |
| academicYearId | int | private | True | False | N/A | 1 | N/A |
| accountId | int | private | True | False | N/A | 1 | N/A |
| facultyId | int | private | True | False | N/A | 1 | N/A |
| **Operations:** Only public get/set operations corresponding to the attributes, with instance scope. | | | | | | | |

**Class Registration:**

| **Name** | Registration | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Brief Description** | Data model class for registration entity in the database | | | | | | |
| **Attributes** | | | | | | | |
| **Name** | **Type** | **Access** | **Mutable** | **Optional** | **Length** | **Min** | **Max** |
| registrationId | int | private | True | False | N/A | 1 | N/A |
| status | boolean | private | True | False | N/A | N/A | N/A |
| courseId | int | private | True | False | N/A | 1 | N/A |
| studentId | int | private | True | False | N/A | 1 | N/A |
| courseClassId | int | private | True | False | N/A | 1 | N/A |
| semesterId | int | private | True | False | N/A | 1 | N/A |
| **Operations:** Only public get/set operations corresponding to the attributes, with instance scope. | | | | | | | |

**Class Course:**

| **Name** | Course | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Brief Description** | Data model class for course entity in the database | | | | | | |
| **Attributes** | | | | | | | |
| **Name** | **Type** | **Access** | **Mutable** | **Optional** | **Length** | **Min** | **Max** |
| courseId | int | private | True | False | N/A | 1 | N/A |
| courseName | String | private | True | False | 15 | N/A | N/A |
| creditCount | String | private | True | False | 64 | N/A | N/A |
| difficultyLevel | int | private | True | False | N/A | 1 | 6 |
| required | boolean | private | True | False | N/A | N/A | N/A |
| startDate | DateTime | private | True | False | N/A | N/A | N/A |
| endDate | DateTime | private | True | False | N/A | N/A | N/A |
| facultyId | int | private | True | False | N/A | 1 | N/A |
| courseTypeId | int | private | True | False | N/A | 1 | N/A |
| semesterId | int | private | True | False | N/A | 1 | N/A |
| **Operations:** Only public get/set operations corresponding to the attributes, with instance scope. | | | | | | | |

**Class CourseClass:**

| **Name** | CourseClass | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Brief Description** | Data model class for course\_class entity in the database | | | | | | |
| **Attributes** | | | | | | | |
| **Name** | **Type** | **Access** | **Mutable** | **Optional** | **Length** | **Min** | **Max** |
| courseClassId | int | private | True | False | N/A | 1 | N/A |
| courseClassCode | String | private | True | False | N/A | N/A | N/A |
| remainingSlots | int | private | True | False | N/A | N/A | N/A |
| maxSlots | int | private | True | False | N/A | N/A | N/A |
| courseId | int | private | True | False | N/A | 1 | N/A |
| **Operations:** Only public get/set operations corresponding to the attributes, with instance scope. | | | | | | | |

**Class TimetableCell:**

| **Name** | TimatableCell | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Brief Description** | Data model class for timetable\_cell entity in the database | | | | | | |
| **Attributes** | | | | | | | |
| **Name** | **Type** | **Access** | **Mutable** | **Optional** | **Length** | **Min** | **Max** |
| id | int | private | True | False | N/A | 1 | N/A |
| day | int | private | True | False | N/A | N/A | N/A |
| sessionNo | int | private | True | False | N/A | N/A | N/A |
| venue | String | private | True | False | 15 | N/A | N/A |
| instructorName | String | private | True | False | 35 | N/A | N/A |
| isLecture | boolean | private | True | False | N/A | N/A | N/A |
| courseClassId | int | private | True | False | N/A | 1 | N/A |
| **Operations:** Only public get/set operations corresponding to the attributes, with instance scope. | | | | | | | |

**Class Grades:**

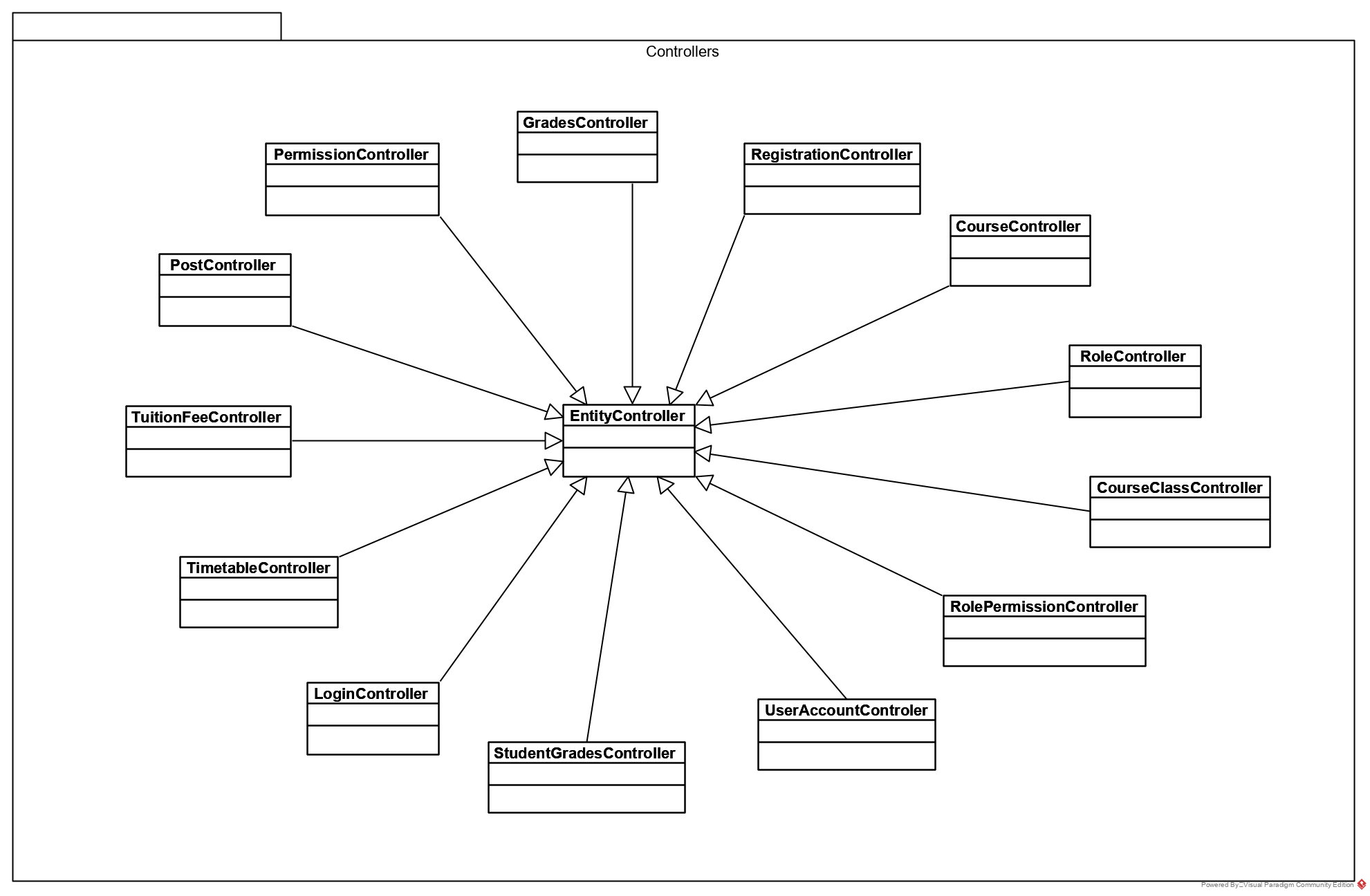
| **Name** | Grades | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Brief Description** | Data model class for grades entity in the database | | | | | | |
| **Attributes** | | | | | | | |
| **Name** | **Type** | **Access** | **Mutable** | **Optional** | **Length** | **Min** | **Max** |
| id | int | private | True | False | N/A | 1 | N/A |
| attendanceMark | float | private | True | False | N/A | N/A | N/A |
| midtermMark | float | private | True | False | N/A | N/A | N/A |
| finalMark | float | private | True | False | N/A | N/A | N/A |
| overallMark | float | private | True | False | N/A | N/A | N/A |
| courseId | int | private | True | False | N/A | 1 | N/A |
| studentId | int | private | True | False | N/A | 1 | N/A |
| **Operations:** Only public get/set operations corresponding to the attributes, with instance scope. | | | | | | | |

**Class FeeLine:**

| **Name** | FeeLine | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Brief Description** | Data model class for fee\_line entity in the database | | | | | | |
| **Attributes** | | | | | | | |
| **Name** | **Type** | **Access** | **Mutable** | **Optional** | **Length** | **Min** | **Max** |
| lineId | int | private | True | False | N/A | 1 | N/A |
| status | boolean | private | True | False | N/A | N/A | N/A |
| lineSum | float | private | True | False | N/A | N/A | N/A |
| registrationId | int | private | True | False | N/A | 1 | N/A |
| **Operations:** Only public get/set operations corresponding to the attributes, with instance scope. | | | | | | | |

### Controllers

Every controller extends from the base class EntityController, which in turn is a subtype of the default MVC Controller of ASP.NET Core.



The most significant Controller classes are: LoginController, UserAccountController, RegistrationController, GradesController, StudentsGradesController, TimetableController, TuitionFeeController.

**Class LoginController:**

| **Name** | LoginController | | | |
| --- | --- | --- | --- | --- |
| **Brief Description** | Controller to handle login to the webpage | | | |
| **Attributes:** No special attribute. | | | | | |
| **Operations:** | | | | | |
| **Header** | **Return Type** | **Access Modifier** | **Scope** | **Specification** |
| Validate(username: String, password: String) | boolean | private | Instance | Validates whether the pair username and password matches an existing user\_account row in the persistent storage. |
| Login(user: ClaimsPrincipal) | void | private | Instance | Logs a validated user into the system. |
| Login(username: String, password: String) | IActionResult | public | Instance | Route processing for logging a user into the system. Includes two sub-tasks: validating user and logging the user into the system. Redirects the user to the correct homepage if login successful. |
| Logout() | IActionResult | public | Instance | Logs an authenticated user out of the system. Redirects the user to the login page. |

**Class UserAccountController:**

| **Name** | UserAccountController | | | |
| --- | --- | --- | --- | --- |
| **Brief Description** | Controller to handle CRUD operations (Create, Retrieve, Update, Delete) on user accounts on this site. | | | |
| **Attributes:** No special attribute. | | | | | |
| **Operations:** | | | | | |
| **Header** | **Return Type** | **Access Modifier** | **Scope** | **Specification** |
| RenderCreateView() | IActionResult | public | Instance | Renders the view to create a new user on the system. |
| RenderProfileCreatePartial(type: int) | IActionResult | public | Instance | Renders the partial view for role-specific user information. |
| RenderListView() | IActionResult | public | Instance | Renders the list view of all registered users on the system. |
| RenderDetailsView(id: int) | IActionResult | public | Instance | Renders the details view of one specific user account. |
| Create(details: dynamic) | IActionResult | public | Instance | Try to create a new user based on inputs and return a message showing whether the operation succeeded. |
| Delete(id: int) | IActionResult | public | Instance | Try to delete an existing user from persistent storage and return a message showing whether the operation succeeded. |
| Edit(details: dynamic) | IActionResult | public | Instance | Try to update data of an existing user based on inputs and return a message showing whether the operation succeeded. |

**Class RegistrationController:**

| **Name** | RegistrationController | | | |
| --- | --- | --- | --- | --- |
| **Brief Description** | Controller to handle course registration in the system. | | | |
| **Attributes:** No special attribute. | | | | | |
| **Operations:** | | | | | |
| **Header** | **Return Type** | **Access Modifier** | **Scope** | **Specification** |
| GetActiveCourses(facultyId: int, semesterId: int) | Course[] | private | Instance | Validates whether the pair username and password matches an existing user\_account row in the persistent storage. |
| Login(user: ClaimsPrincipal) | void | private | Instance | Logs a validated user into the system. |
| Login(username: String, password: String) | IActionResult | public | Instance | Route processing for logging a user into the system. Includes two sub-tasks: validating user and logging the user into the system. Redirects the user to the correct homepage if login successful. |
| Logout() | IActionResult | public | Instance | Logs an authenticated user out of the system. Redirects the user to the login page. |

**Class GradesController:**

| **Name** | GradesController | | | |
| --- | --- | --- | --- | --- |
| **Brief Description** | Controller to handle uploading and updating grades, along with updating the credit count of students’ profiles. | | | |
| **Attributes:** No special attribute. | | | | | |
| **Operations:** | | | | | |
| **Header** | **Return Type** | **Access Modifier** | **Scope** | **Specification** |
| Upload() | IActionResult | public | Instance | Route processing method, renders the view for uploading grades. |
| Upload(details: dynamic) | IActionResult | public | Instance | Route processing method, for pushing the uploaded grades report lines to persistent storage. |

**Class StudentGradesController:**

| **Name** | StudentGradesController | | | |
| --- | --- | --- | --- | --- |
| **Brief Description** | Controller to handle displaying the grades report to students. | | | |
| **Attributes:** No special attribute. | | | | |
| **Operations:** | | | | |
| **Header** | **Return Type** | **Access Modifier** | **Scope** | **Specification** |
| RenderGradesView() | IActionResult | public | Instance | Route processing method to render the grades view for the active student. |
| GetStudentGrades(int studentId, int semesterId) | Grades[] | private | Instance | Retrieves the grades report for the student for a specified semester. |
| RenderGradesView(Grades[]) | IActionResult | public | Instance | Render the grades report view for the student for a specified semester. |

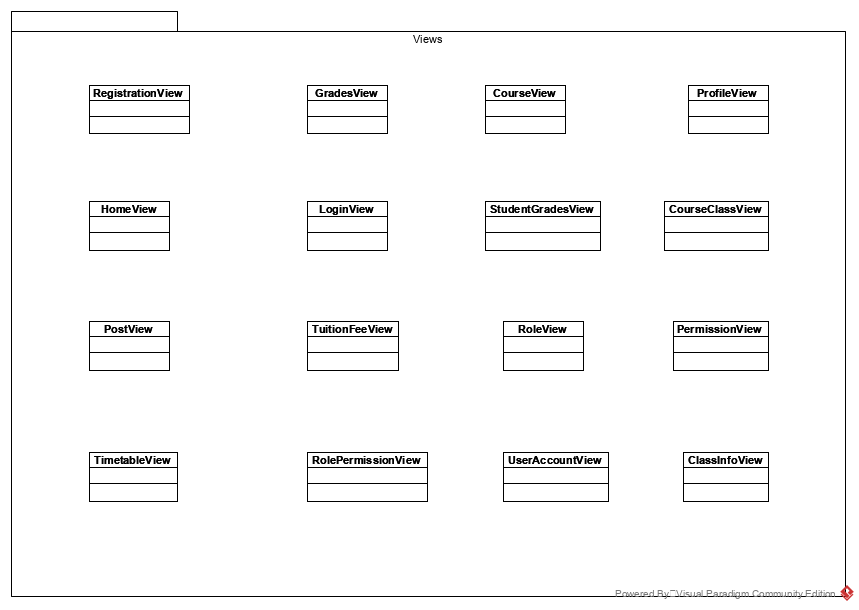
**Class TimetableController:**

| **Name** | TimetableController | | | |
| --- | --- | --- | --- | --- |
| **Brief Description** | Controller to handle displaying the class schedule for students. | | | |
| **Attributes:** No special attribute. | | | | |
| **Operations:** | | | | |
| **Header** | **Return Type** | **Access Modifier** | **Scope** | **Specification** |
| RenderTimetableView() | IActionResult | public | Instance | Route processing method to render the timetable for the user when requested. |
| RenderTimetableView(weekIndex: int) | IActionResult | public | Instance | Retrieves and renders the timetable view for a specified week within a semester. |
| GetTimeTableByWeek(week: Week) | HashSet<TimetableCell> | private | Instance | Retrieves the list of timetable cells that are occupied for a specified week for the active student. |

**Class TuitionFeeController:**

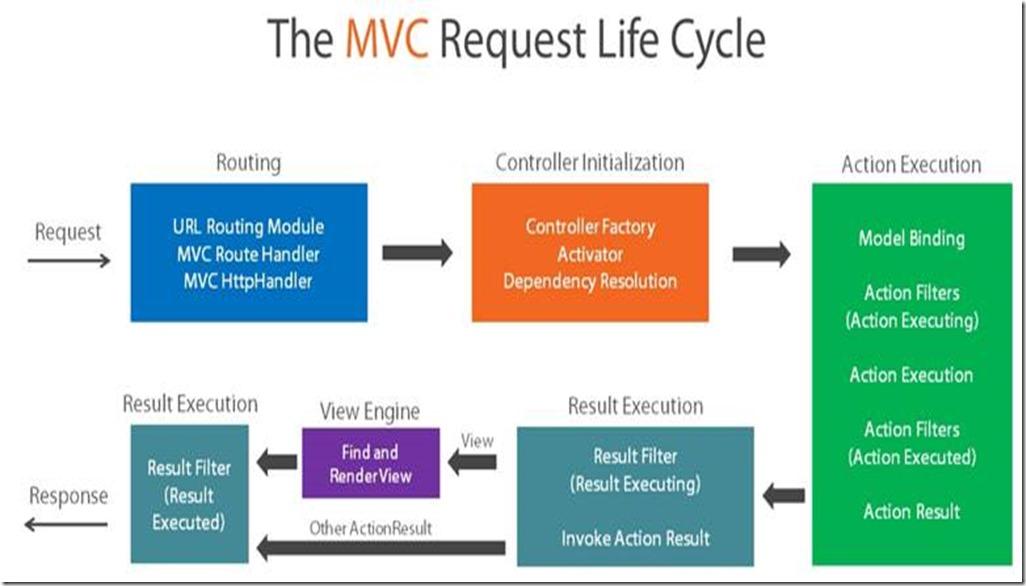
| **Name** | TuitionFeeController | | | |
| --- | --- | --- | --- | --- |
| **Brief Description** | Controller to handle computing and displaying tuition fee for students. | | | |
| **Attributes:** No special attribute. | | | | |
| **Operations:** | | | | |
| **Header** | **Return Type** | **Access Modifier** | **Scope** | **Specification** |
| RenderTuitionFeeView(studentId: int) | IActionResult | public | Instance | Route processing method for displaying the tuition fee view to the student. |
| GetTuitionFee(studentId: int) | FeeLine[] | private | Instance | Gets the list of tuition fee lines in the current semester’s tuition fee report for the specified student. |
| RenderTuitionFeeView(feeLines: FeeLine[]) | IActionResult | private | Instance | Renders the tuition fee report view based on the fee lines fetched from persistent storage. |

### Views



# Process View

The system will be built based on ASP.NET Core MVC Framework, therefore, the processes taking part in our system are the same as the core processes in the framework. Following is a brief description of the main process groups.



## Requesting & Routing

Requesting is done by the user making interaction with the View. This is also the means by which the View communicates with Controllers. Whenever a client makes a request to the server, whether by interacting with the view or by requesting an URL, the server will try to map the requested URL to the correct controller action (having the same naming convention as the URL if using MVC with default routing or the action with the URL in the attribute that matches). Then, the corresponding Controller is initialized, and the matching action method is invoked.

## Controller Initialization & Action Execution

ASP.NET Core MVC controllers are designed based on the Factory pattern, with the center component being Controller Factory. When the Controller is route-matched, it is instantiated by the DefaultControllerFactory method called CreateController. As .NET Core supports dependency injection, when the controller is being instantiated, the global dependencies passed into the controller constructor are also resolved. Then, the controller method is executed normally. Note that a valid controller action in .NET Core must return an ActionResult object.

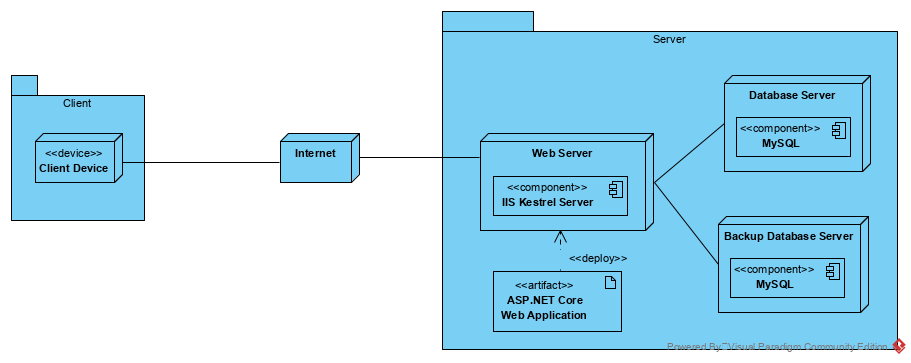
During action execution, the Controller may communicate with, or instantiate the Models directly, or may fetch data from database by means of Data Access objects.

## Result Execution & Response

Action Execution yields an ActionResult. An ActionResult may be a redirection, a data portion, or a ViewResult. If it is a ViewResult, the action result is then rendered to HTML before being passed to the client’s browser, during which the placeholders inside the View are evaluated to plain text values. Finally, the action result is passed into client browser within HTTP response.

# Deployment View

The system will be hosted on a single server cluster. This includes a Web server running IIS Server software, in which our web application will be hosted, and two MySQL database servers – one for actual operation and another for backup. Clients have access to the system by means of Internet connection. Provided is a simplified view of the Educational Management System deployment.

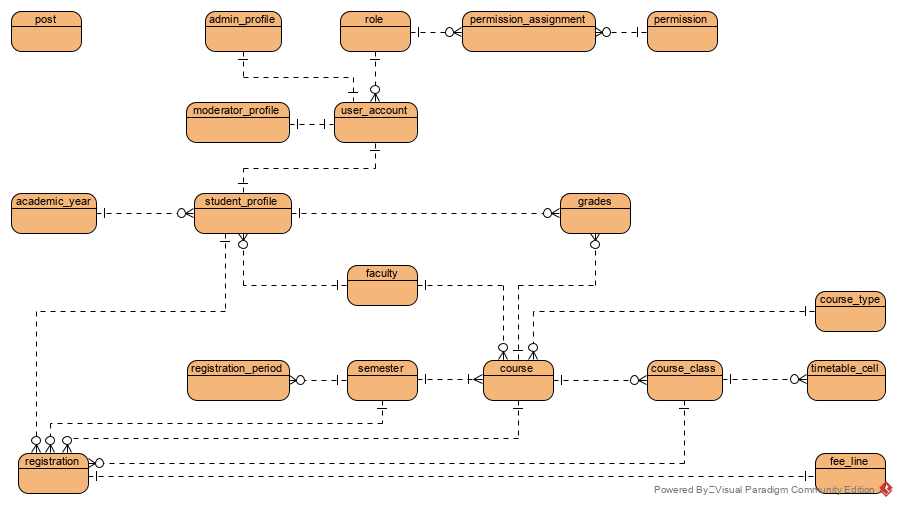


# Implementation View

The implementation of the system is strictly derived from the design; therefore, the implementation view will not be considered in this document.

# Data View

All data within the system is contained within a single schema. Below is a top-level view of the Data Model, revealing the tables and relationships between them. For the full Data Model view with complete list of entity attributes, refer to the Visual Paradigm graphical representation located within Data Model folder.



# Size and Performance

The chosen software architecture supports the key sizing and timing requirements, as specified in the ***Supplementary Specification [2]***:

1. The system shall support up to 1000 simultaneous users against the central database at any given time, and up to 500 simultaneous users against the local servers at any one time.
2. The system must be able to complete 80% of all transactions within 2 minutes.
3. The system must accurately save all information transactions.

The selected architecture supports the sizing and timing requirements through the implementation of a client-server architecture.

# Quality

The software architecture supports the quality requirements, as specified in the ***Supplementary Specification [2]***:

1. The user-interface shall be responsive.
2. The user interface of the system shall be designed for ease-of-use and shall be appropriate for a computer-literate user community with no additional training on the system.
3. Each feature of the Educational Management System shall have built-in online help for the user. Online Help shall include step by step instructions on using the System. Online Help shall include definitions for terms and acronyms.
4. The system shall be available 24 hours a day, 7 days a week. There shall be no more than 4% down time.
5. Mean Time Between Failures shall exceed 4 months.