|  |  |
| --- | --- |
|  | **Cognizant Academy**  **School Management System**  **ASP .Net MVC, Entity Framework, SQL Server Integrated Capability Test**  **Version 1.0** |
|  |
|  |

Contents

[1.0 Introduction 3](#_Toc46996494)

[1.0 Purpose of this document 3](#_Toc46996495)

[2.0 Definitions & Acronyms 3](#_Toc46996496)

[3.0 Project Overview 3](#_Toc46996497)

[4.0 Scope 5](#_Toc46996498)

[5.0 Hardware and Software Requirment 5](#_Toc46996499)

[2.0 System diagram 6](#_Toc46996500)

[3.0 Design for Displaying School Details (HomePage) 6](#_Toc46996501)

[1.0 Requirement flow 7](#_Toc46996502)

[2.0 Technical guidelines 8](#_Toc46996503)

[Component Specification – Model 10](#_Toc46996504)

[4.0 Design for Adding New School Details to the database 12](#_Toc46996505)

[1.0 Requirement flow 12](#_Toc46996506)

[2.0 Create Student 13](#_Toc46996507)

[3.0 Technical guidelines 15](#_Toc46996508)

[5.0 Design for Adding New Course Details to the database 16](#_Toc46996509)

[6.0 Standards and Guidelines 17](#_Toc46996510)

[1.0 Controller & View 17](#_Toc46996511)

[7.0 Design constraints 17](#_Toc46996512)

[8.0 Code submission Instructions 17](#_Toc46996513)

[9.0 Evaluation Areas 18](#_Toc46996514)

# Introduction

## Purpose of this document

The purpose of this document is to define the server side implementation of the School Management System application.

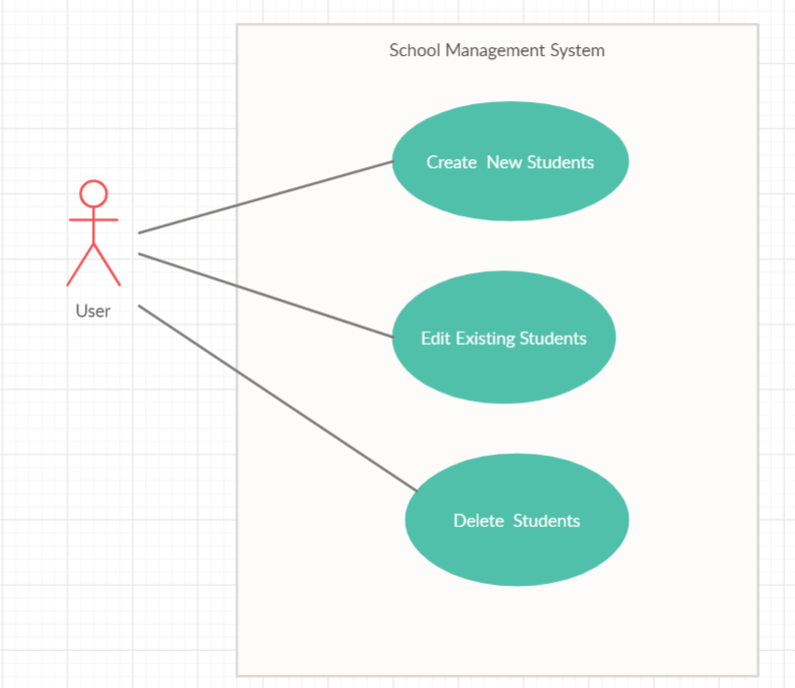
## Definitions & Acronyms

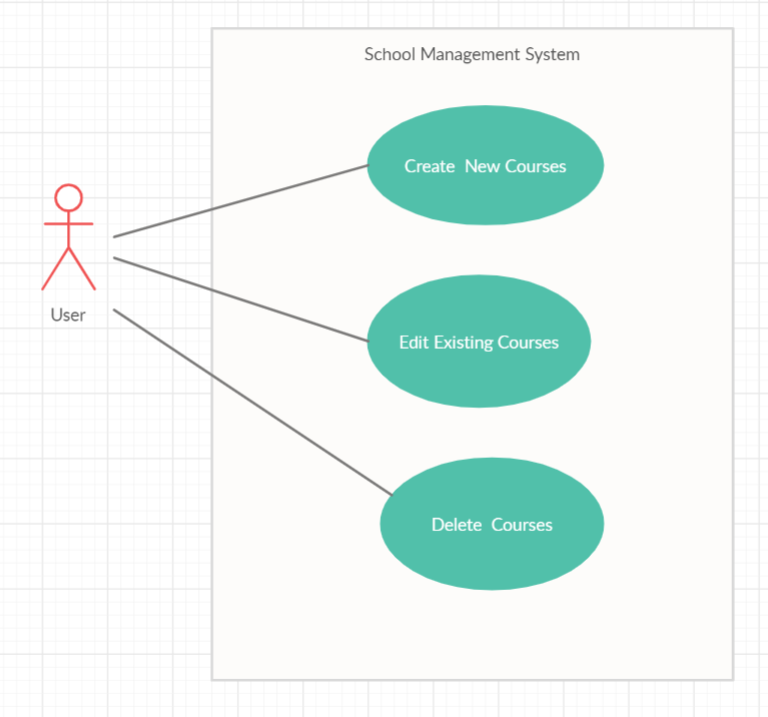
|  |  |
| --- | --- |
| Definition / Acronym | Description |
| ASP.NET MVC | ASP.Net MVC is a Web development framework built on top of ASP.Net with certain changes in the internal workings of web page rendering |

## Project Overview

The project involves creating School management system. The user can enter the School details like the name, type of food, online order acceptance, launch date of the School. This School details are stored in the database. All the School details are retrieved from the database and displayed on the screen.

Use Case Diagram





## Scope

1. Creation of ASP.Net MVC web application for School Management system application

## Hardware and Software Requirment

1. Hardware Requirement:
   1. Developer PC with 8GB RAM
2. Software Requirement
   1. IE or Chrome
   2. .Net Framework 4.7 and above
   3. Visual Studio Community Edition 2019
   4. SQL Server enterprise edition 2014

# System diagram

**Store in database**

**Validate Student details**

**Add new Student details**

**Display all Student details**

**Store in database**

**Validate Course details**

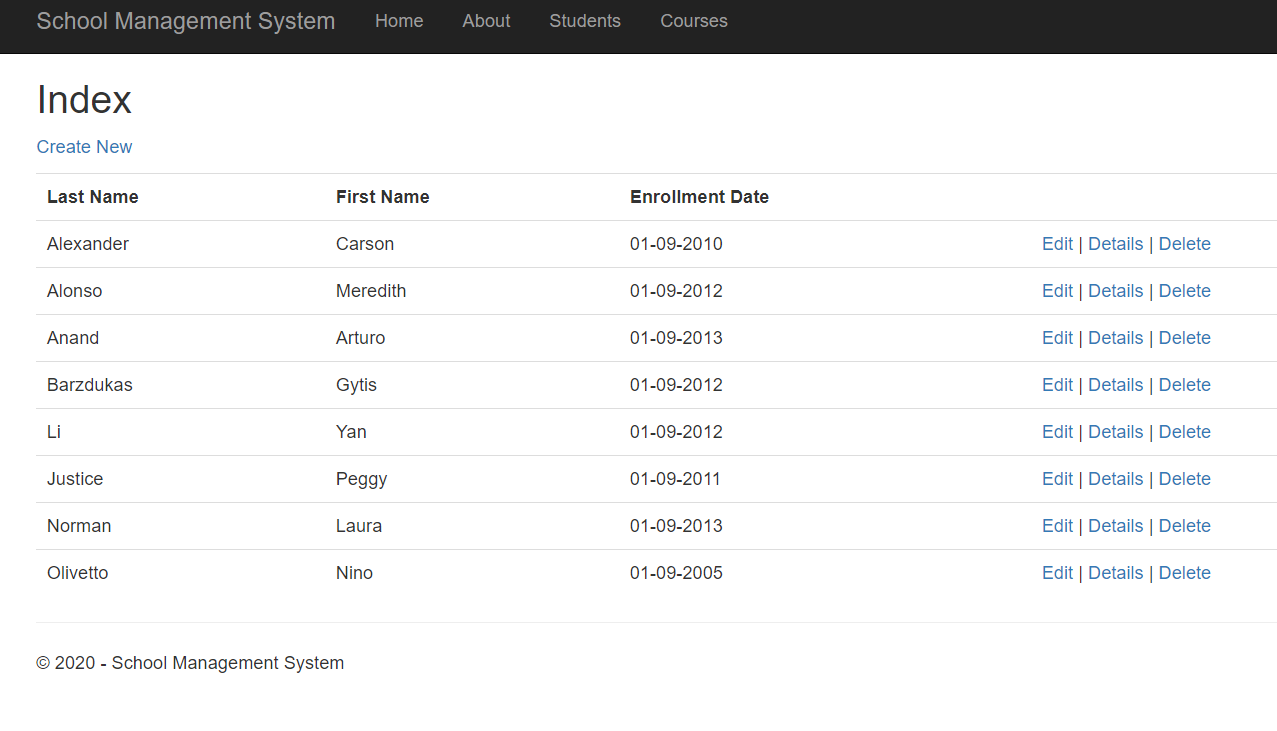
**Add new Course details**

# Design for Displaying School Details (HomePage)

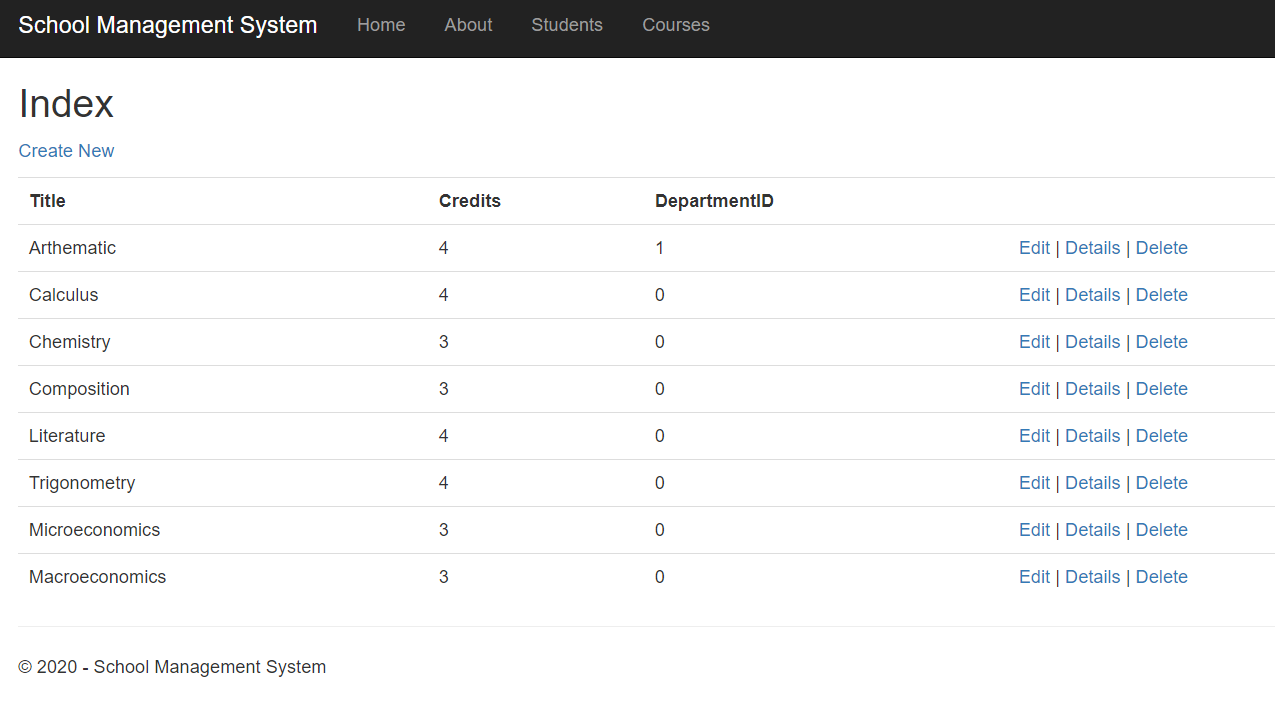
## Requirement flow

**Steps Explanation**

1. Application user launches the application.
2. He is given a Home Page – Welcome Screen
3. Clicking on the Students Tab shows the Student details retrieved from the database. The snippet of the Student Screen is shown below.



1. Clicking on the Courses Tab shows the Courses details retrieved from the database. The snippet of the Student Screen is shown below.



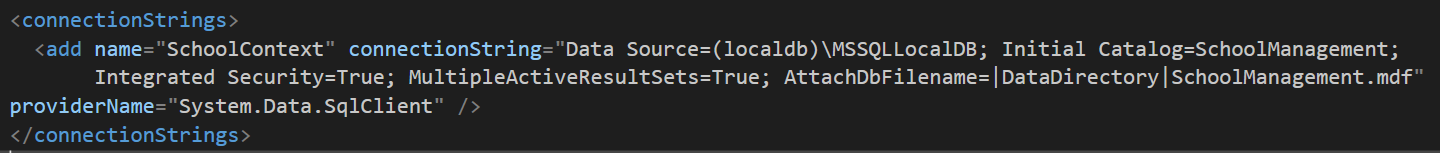
## Technical guidelines

1. Create a controller named StuentController.
2. Create an ‘Index’ action with no arguments in the‘StudentController’.

Controller Specification for Index Action with no argument

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Action Name** | **ReturnType** | **HTTP Method** |
| StudentController | Index | ActionResult | GET |

1. In School Controller, inside the ‘Index’ action, use Entity Framework to get the Student details from the database and display it in theView.
2. Create ‘SchoolContext’ class which inherits DbContext class. Include namespace“System.Data.Entity”.
3. Create a constructor of the SchoolContext. Specify the name of the database connection string element to be “RMS\_DB” (connection string name in Web.config).
   1. Use “SchoolContext” as ‘Context’ name in Entity Framework. Sample config entry is provided below.



1. Declare a property ‘Students’ of type DbSet<Student> in the SchoolContext class.
2. The Code First approach is used to implement the entityframework.
3. Retrieve all the Student details from the database.
4. Create the School model with Id, FirstName, LastName, EnrollmentDate and ICollection<Enrollment> as fields.
5. Following are the models which and their relations

|  |  |
| --- | --- |
| **Student Model** | **public class Student  {  public int ID { get; set; }  public string LastName { get; set; }  public string FirstName { get; set; }  public DateTime EnrollmentDate { get; set; }  public virtual ICollection<Enrollment> Enrollments { get; set; }  }** |
| **Course Model** | **public class Course  {   public int CourseID { get; set; }  public string Title { get; set; }  public int Credits { get; set; }  public int DepartmentID { get; set; }  public virtual ICollection<Enrollment> Enrollments { get; set; }  }** |
| **Enrollment Model** | **public class Enrollment  {  public int EnrollmentID { get; set; }  public int CourseID { get; set; }  public int StudentID { get; set; }  public Grade? Grade { get; set; }  public virtual Course Course { get; set; }  public virtual Student Student { get; set; }  }  public enum Grade { A, B, C, D, F }** |

# Component Specification – Model

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **DataType** | **Property** | **Data Annotation** |
| Student | int | Id | Key |
| string | FirstName | [Required]  [StringLength(50, ErrorMessage = "First Name must be less than 50 Characters")] [Column("FirstName")] [Display(Name = "First Name")] |
| string | LastName | [Required]  [StringLength(50)] [Display(Name = "Last Name")] |
| DateTime | EnrollmentDate | [DataType(DataType.Date)]  [DisplayFormat(DataFormatString = "{0:dd-MM-yyyy}", ApplyFormatInEditMode = true)]  [Display(Name = "Enrollment Date")] |
| Enrollment  Custom Class | ICollection<Enrollment> | Check the model skeleton from above |

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **DataType** | **Property** | **Data Annotation** |
| Course | int | CourseID | Key |
| string | Title | [StringLength(50, MinimumLength = 3)] |
| int | Credits | [Range(0, 5)] |
| int | DepartmentID |  |
| Enrollment  Custom Class | ICollection<Enrollment> | Check the model skeleton from above |

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **DataType** | **Property** | **Data Annotation** |
| Enrollment | int | EnrollmentID |  |
| int | CourseID |  |
| int | StudentID |  |
| Enum | Grade | [DisplayFormat(NullDisplayText = "No Grade")]  Check the Enum skeleton from above |
| **Course**  Custom Class | **Course** | Check the model skeleton from above |
|  | **Student**  Custom Class | **Student** | Check the model skeleton from above |

1. Modify the **\_Layout** file to point to the Home pagelink.
   1. A sample \_Layout.cshtml is given as reference. Create a link ‘**School**. On clicking this link the page must navigate to the homepage.
   2. Map the ‘Student controller and ‘Index’ action to the‘Student.
   3. A sample \_Layout.cshtml is given as reference. Create a link ‘**Course**. On clicking this link the page must navigate to the homepage.
   4. Map the ‘Course controller and ‘Index’ action to the ‘Course’.
2. The “Create New” link in this Student details list page should be used to Create a new Student detail by the user *(refer section3.0)*.
3. Assign the details retrieved from database to the ‘list’ in the Studentcontroller.
4. Return the view.
5. These Student details from the ‘list’ must be displayed on thepage.
6. The “Create New” link in this Course details list page should be used to Create a new Course detail by the user *(refer section3.0)*.
7. Assign the details retrieved from database to the ‘list’ in the CourseController.
8. Return the view.
9. These Course details from the ‘list’ must be displayed on thepage.

Note:

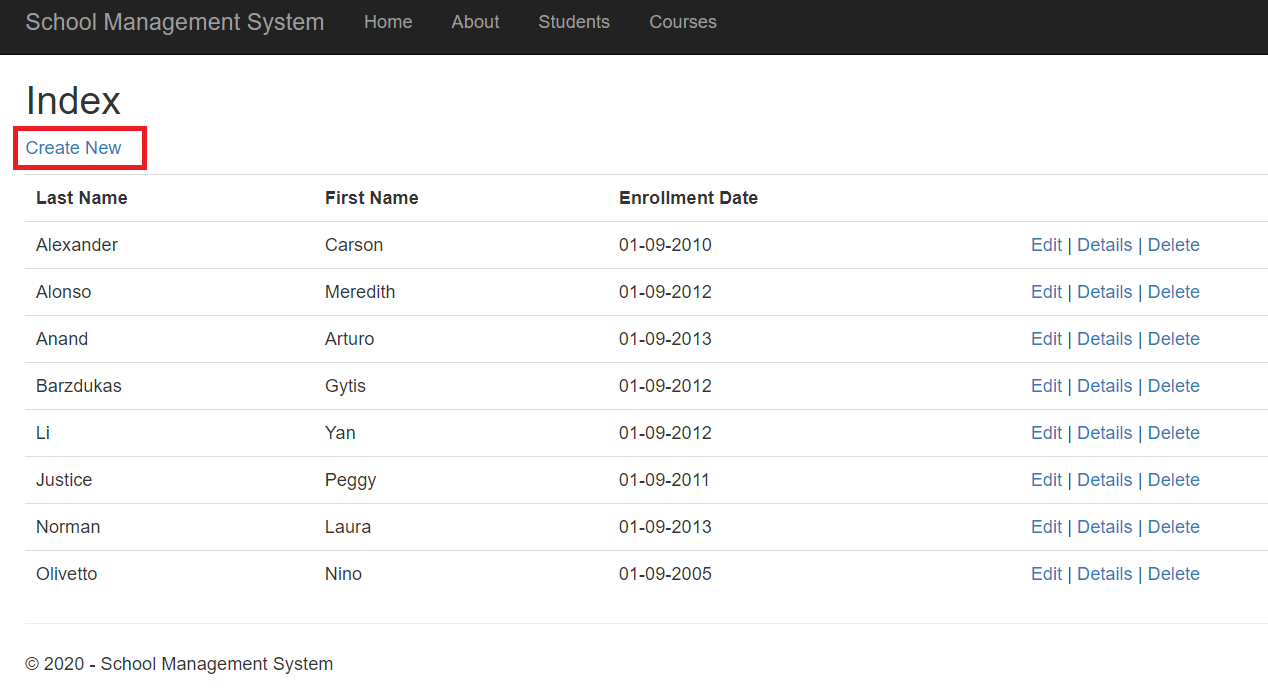
* + Use ‘**HTMLHelper’** to list Studentdetails.

# Design for Adding New School Details to the database

## Requirement flow

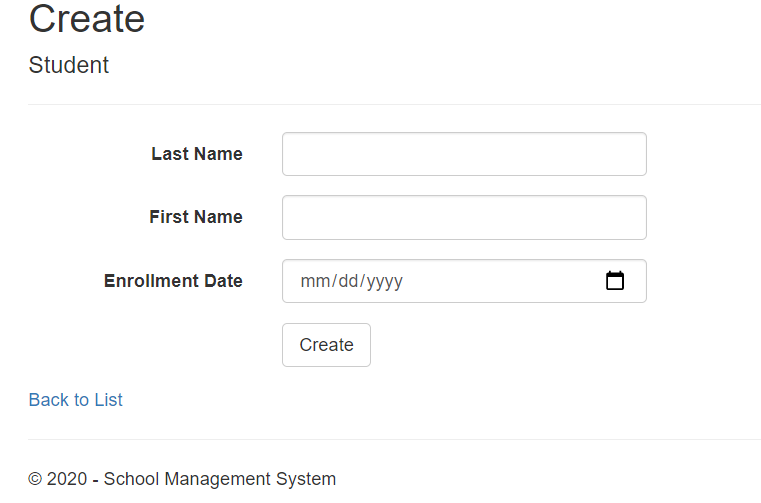
**Steps Explanation**

1. Create a link with text “Create New” in the homepage.



1. On clicking this ink, the page should navigate to the add Student page.
2. The user must be able to enter the Student details in the displayed form.

## Create Student

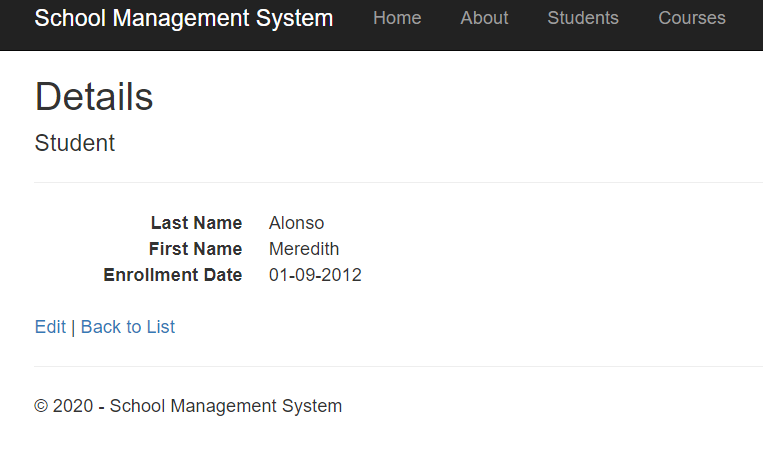


UI Controls for Student Addition

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Element** | **Control type** | **Default Values** | **Editable Field** | **Mandatory/Not Mandatory** |
| FirstName | Textbox | No | Yes | Mandatory |
| LastName | Textbox | No | Yes | Mandatory |
| Enrollment Date | Calender, Textbox | No | Yes | Mandatory |

Validate the form to ensure there are no empty fields.

1. On Submit, the Student details must be stored to database. Display a message, “Student added successfully!” and display Student details in “Details” view as shown below.



## Technical guidelines

* Steps for displaying Add Student Form – Points 1 to 3
* Steps for storing the details to database - Points 2.0

1. Create an **‘AddStudent’** action with no arguments. This ‘AddStudent’ action return type must be‘ActionResult’

Controller Specification for **AddStudent Action** with no argument

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Action Name** | **ReturnType** | **HTTP**  **Method** |
| StudentController | AddStudent | ActionResult | GET |

1. The ‘Create New’ link is mapped to this action. This action must return ‘AddStudent’ View containing the form to enter the Studentdetails.

Note:

* Use ‘**HTMLHelper’** to create aform.
* Using HTMLHelper will automatically create ‘id’ for all the formelement like the input box, drop-down. These ‘id’ is required for auto-evaluation.

Like,

id="Name" id="Type of food" id="Submit"

id of ‘submit’ button must be ‘Submit’

1. Create an **‘AddStudent ’** action (HttpPost) with **“School”** model as arguments in the ‘SchoolController’. This ‘AddSchool’ action return type must be ‘ActionResult’.

Controller Specification for **AddSchool Action** with argument

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Action Name** | **ReturnType** | **HTTP**  **Method** |
| StudentController | AddStudent | ActionResult | POST |

1. This ‘AddStudent’ action is implemented as POST to get the values of Student details posted from the form.
2. Use **Entity framework** to store the details to the database.
3. Use the ‘**StudentContext**’ to connect to the database**.**
4. Inside the ‘AddStudent’action,
   1. Create an instance of ‘StudentContext’.
   2. Add the data in Student object to itsentity.

# Design for Adding New Course Details to the database

Follow the similar steps as above to Create, Update and Delete Course Details.

# Standards and Guidelines

## Controller & View

1. Action methods should have a meaningful name
2. Remove unused Action methods
3. There should not be any hard coded values in code. It has to be referenced from Web.config file
4. Database connection string should be set in the ConnectionStrings section of Web.config and NOT in the AppSettings
5. Meaningful names should be given to the controls created in View

# Design constraints

Required packages are already supplied with ‘Using’ statement. So do not try to add packages using Nugetpackages.

# Code submission Instructions

* + 1. Do not change the code skeleton given, as your code will be auto-evaluated.
    2. Your last submitted solution will be considered for detailed evaluation.
    3. Make sure to submit the solution before the specified time limit. You will not be allowed to submit the solution once the mention time for the assessment is over.

# Evaluation Areas

|  |  |
| --- | --- |
| 1 | Launched application lands in the Home page |
| 2 | Home Page UI contains the welcome Text |
| 3 | Clicking on Student Tab takes the user to Student Landing Page |
| 4 | Link ‘Create New’ navigates to Add Student page |
| 5 | Add Student Page contains required form elements |
| 6 | Submit the Student details |
| 7 | Display all the Student details |
| 8 | Clicking on Course Tab takes the user to Course Landing Page |
| 9 | Link ‘Create New’ navigates to Add Course page |
| 10 | Add Course Page contains required form elements |
| 11 | Submit the Course details |
| 12 | Display all the Course details |
| 14 | Form validation for empty fields |
| 15 | Implementation of StudentContext |