UNIVERSITY OF Southampton

Web Development Coursework

COMP6205 - Web Development

2016/17

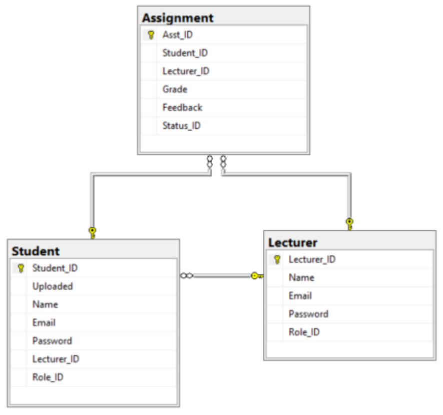
Instructor: Dr. Andrew Gravell

Assignment by 28907787 and 28886461

[For evaluation purposes: we have 2 users m@g.com and n@g.com with passwords "mary" and "nath" respectively who are assigned to the user j@g.com with password "johnny" by default.]

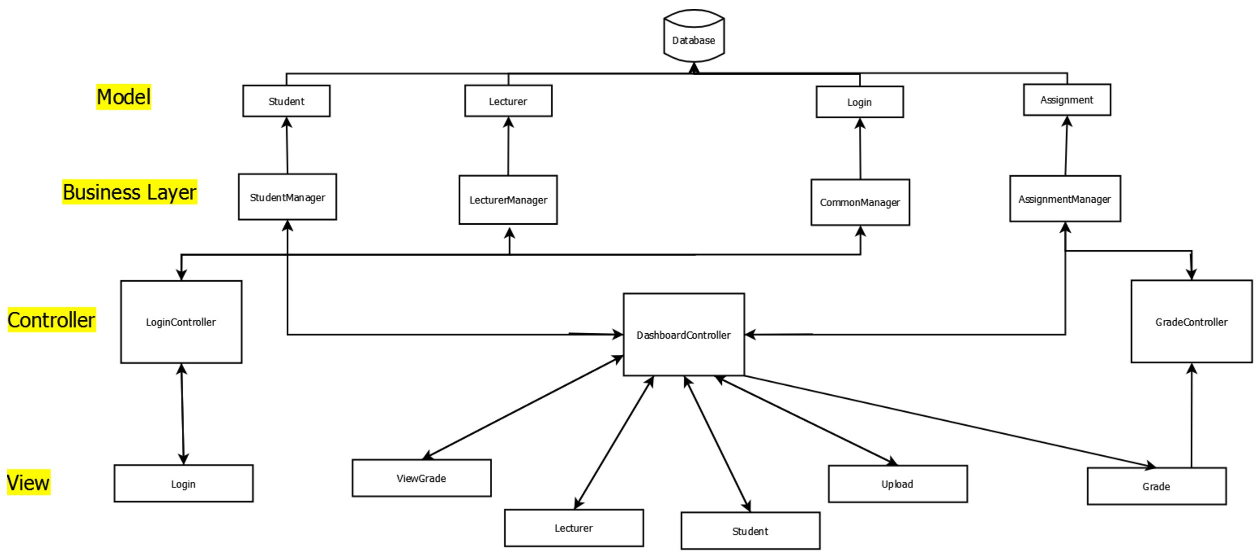
1. Design Diagrams

1.1. Structure of Database



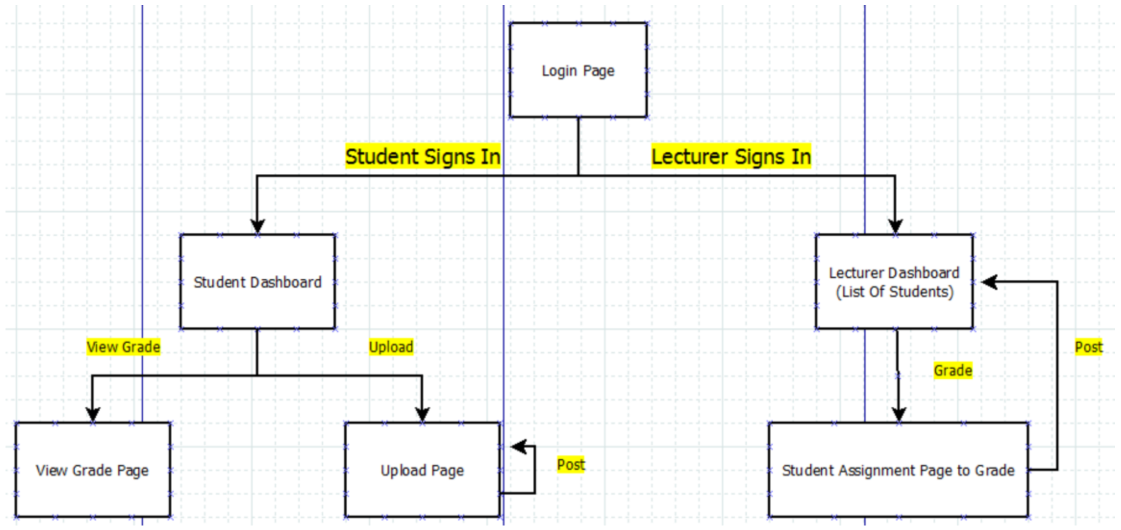
Since we did not use Individual User Accounts, there are three tables for the two sets of users, *Student* and *Lecturer*, and one for the *Assignments*. A foreign key *Lecturer_ID* is present in the student table since every student is assigned to a certain lecturer. The *Uploaded* indicates if a student has uploaded his assignment. *Role_ID* is a bit field to distinguish the different roles. *Assignment* is the table that represents the students uploaded assignment, it has its own ID, and includes two foreign keys that reference the ID of the student that has uploaded it, and the ID of the lecturer responsible for assessing it. *Assignment* also has two fields which are *Grade* and *Feedback* that are updated when the Lecturer makes his assessment. The *Status_ID* helps us know if the assignment has been assessed.

1.2. Structure of Code



This corresponds to the normal MVC model but to prevent code repetition, we implemented a *business layer* which is responsible for querying the database. This helps us keep the code more organised and allows code reutilisation. These methods are called from the *Controllers* which either update the model objects, or retrieve model objects to be returned to the *Controller*.

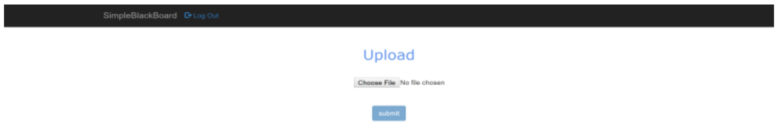
1.3. Structure of Web Site



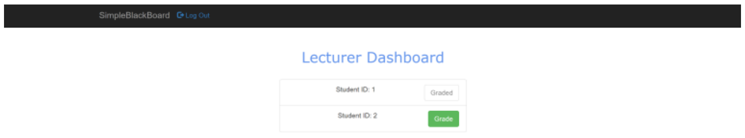
The landing page is the Login. From here you sign in, being attributed a Cookie. Depending on the Role of the user, a different Dashboard will be displayed allowing the user to perform the desired action. The pages are GET requests except when the user wishes to change upload information to the server, which must be made through a POST request.

2. Screenshots of Implemented Features

Students can upload an HTML file so it can be viewed. [/Dashboard/Upload]

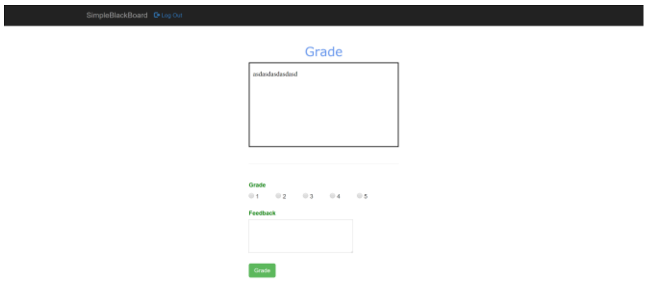


Lecturers can view each student's uploaded work so they can mark it [/Dashboard]



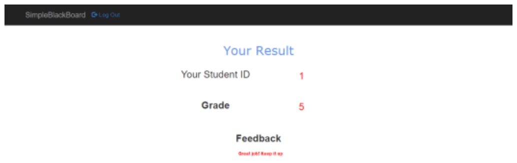
Student ID: 1	Graded
Student ID: 2	Grade

Lecturers can select a grade and provide feedback on the work of each student [/Grade/{int}]



asdasdasdasdasd

Students can view their grade and feedback so they can understand their grade and improve. [/Dashboard/View]



Your Result

Your Student ID 1

Grade 5

Feedback

Great job! Keep it up

3 Samples of Code and Explanations

3.1 Entity Framework and Models (Code First)

All models were created to achieve a code-first implementation. This *Student* class represents table in our Database named Student. We annotate this with the [Table(“Student”)] data annotation. Some other relevant annotations and reasons for their use are:

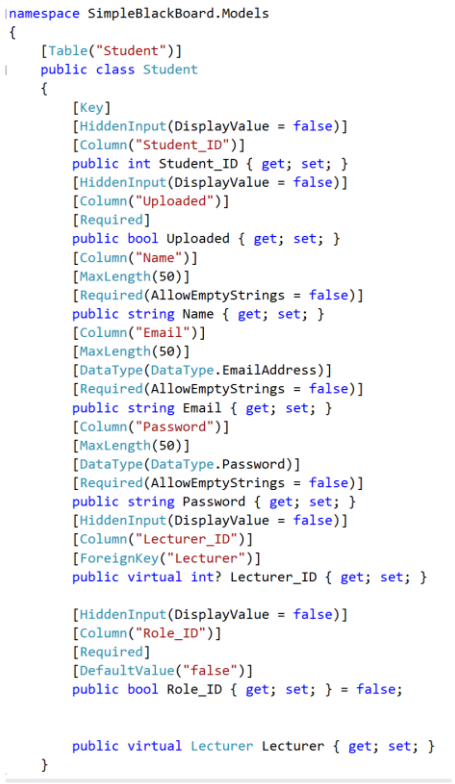
- [Key]: Student_ID is the identity of the student, and represents the primary key of the table increments automatically.

- [HiddenInput]: In case we need to display the model, certain parameters should not be displayed or editable in anyway.

- [Required]: Some attributes must exist to have a valid user and this annotation is used to enforce that those attributes have a value.

- [DataType(Datatype.EmailAddress)]: When the model is displayed, it will provide automatic front-end verification that the string is has a valid email format.

- [ForeignKey]: Since we are using the Lecturer_ID as a foreign key, we need to include a Lecturer field of type *Lecturer* which provides a way to navigate an association between the two entity types.

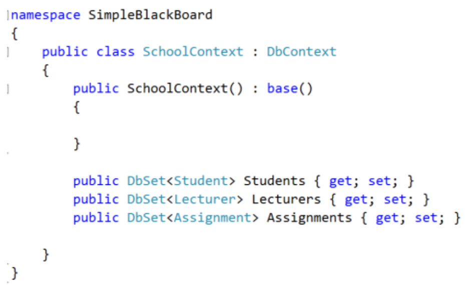


```
namespace SimpleBlackBoard.Models
{
    [Table("Student")]
    public class Student
    {
        [Key]
        [HiddenInput(DisplayValue = false)]
        [Column("Student_ID")]
        public int Student_ID { get; set; }
        [HiddenInput(DisplayValue = false)]
        [Column("Uploaded")]
        [Required]
        public bool Uploaded { get; set; }
        [Column("Name")]
        [MaxLength(50)]
        [Required(AllowEmptyStrings = false)]
        public string Name { get; set; }
        [Column("Email")]
        [MaxLength(50)]
        [DataType(DataType.EmailAddress)]
        [Required(AllowEmptyStrings = false)]
        public string Email { get; set; }
        [Column("Password")]
        [MaxLength(50)]
        [DataType(DataType.Password)]
        [Required(AllowEmptyStrings = false)]
        public string Password { get; set; }
        [HiddenInput(DisplayValue = false)]
        [Column("Lecturer_ID")]
        [ForeignKey("Lecturer")]
        public virtual int? Lecturer_ID { get; set; }

        [HiddenInput(DisplayValue = false)]
        [Column("Role_ID")]
        [Required]
        [DefaultValue("false")]
        public bool Role_ID { get; set; } = false;


        public virtual Lecturer Lecturer { get; set; }
    }
```

~SimpleBlackBoard\Models\Student.cs

3.2 Entity Framework and DBContext

The *DbContext* class is called upon the startup of the application in the *Global.asax* file. If the database doesn’t exist, a database called *SimpleBlackboard.SchoolContext* is created, having 3 tables which are Student, Lecturer, and Assignment. The *DbSets* in the class, represent collections of the specified entities in the context.



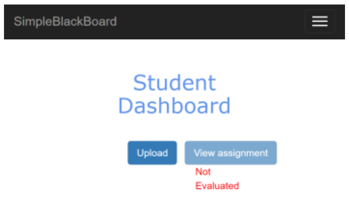
```
namespace SimpleBlackBoard
{
    public class SchoolContext : DbContext
    {
        public SchoolContext() : base()
        {

        }

        public DbSet<Student> Students { get; set; }
        public DbSet<Lecturer> Lecturers { get; set; }
        public DbSet<Assignment> Assignments { get; set; }

    }
}
```

~SimpleBlackBoard\SchoolContext.cs

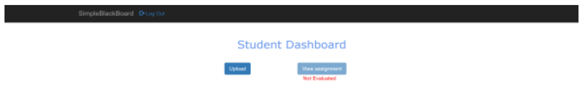
3.3 CSS and responsive design

To achieve a responsive design, we decided to use Bootstrap, which is an external library that helps achieve responsive design without the concern of different screen sizes.

Below, we can see the student dashboard on a desktop or laptop. Meanwhile, on the right, we can see how the page would look like if opened on a mobile phone, we realize that the design has not been compromised and it is usable on both types of screen sizes.



Mobile layout



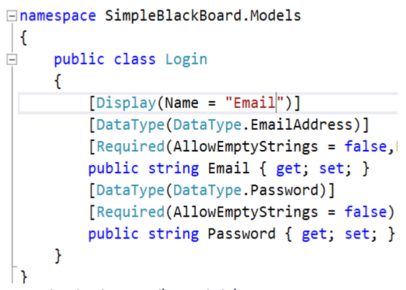
Desktop or laptop layout

3.4 Authentication and Authorization

Since we started the assignment before being introduced to Individual User Accounts, we implemented our own Authentication and Authorization system using database and Sessions.

When logging in, we can concatenate the two tables through the *Role_ID* column and distinguish the user based on whether he's a student or a lecturer.

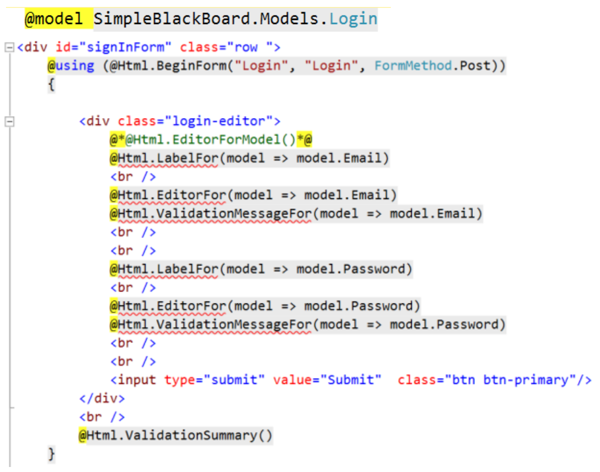
To better utilise the available features of ASP.NET, we created a login model which only has an email and password attributes allowing the Login View to post the input which will be matched to the Login object through model binding.



```
namespace SimpleBlackBoard.Models
{
    public class Login
    {
        [Display(Name = "Email")]
        [DataType(DataType.EmailAddress)]
        [Required(AllowEmptyStrings = false,
        public string Email { get; set; }
        [DataType(DataType.Password)]
        [Required(AllowEmptyStrings = false)
        public string Password { get; set; }
    }
}
```

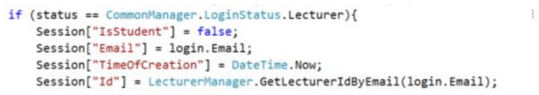
~SimpleBlackBoard\Models\Login.cs

The result from the Login method is a *LoginStatus* which indicates the result of the operation. Based on the result, we can either redirect the user to the Dashboard of back to the Login view. In the snippet below, the user is a lecturer so we put the corresponding information in the Session object to be used later. This allows the application to always know the user that is logged in so we only need a conditional clause to display different views depending on the type of user. A good thing about this is that the application can display different views while keeping the URL the same.



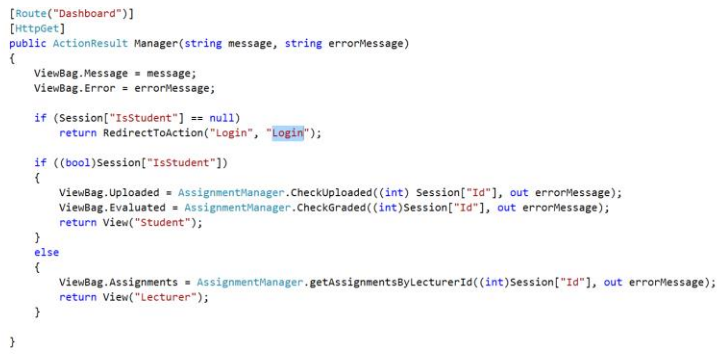
```
@model SimpleBlackBoard.Models.Login
<div id="signInForm" class="row ">
    @using (@Html.BeginForm("Login", "Login", FormMethod.Post))
    {

        <div class="login-editor">
            @*@Html.EditorForModel()*@
            @Html.LabelFor(model => model.Email)
            <br />
            @Html.EditorFor(model => model.Email)
            @Html.ValidationMessageFor(model => model.Email)
            <br />
            <br />
            @Html.LabelFor(model => model.Password)
            <br />
            @Html.EditorFor(model => model.Password)
            @Html.ValidationMessageFor(model => model.Password)
            <br />
            <br />
            <input type="submit" value="Submit"  class="btn btn-primary"/>
        </div>
        <br />
        @Html.ValidationSummary()
    }
```

~SimpleBlackBoard\Views\Login\Login.cshtml



```
if (status == CommonManager.LoginStatus.Lecturer){
    Session["IsStudent"] = false;
    Session["Email"] = login.Email;
    Session["TimeOfCreation"] = DateTime.Now;
    Session["Id"] = LecturerManager.GetLecturerIdByEmail(login.Email);
```

~SimpleBlackBoard\Controllers\LoginController.cs

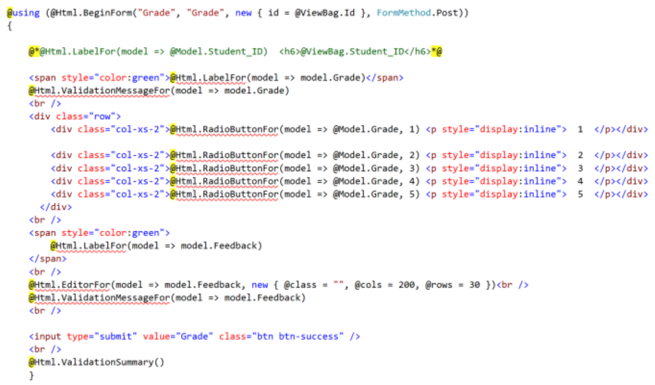


```
[Route("Dashboard")]
[HttpGet]
public ActionResult Manager(string message, string errorMessage)
{
    ViewBag.Message = message;
    ViewBag.Error = errorMessage;

    if (Session["IsStudent"] == null)
        return RedirectToAction("Login", "Login");

    if ((bool)Session["IsStudent"])
    {
        ViewBag.Uploaded = AssignmentManager.CheckUploaded((int) Session["Id"], out errorMessage);
        ViewBag.Evaluated = AssignmentManager.CheckGraded((int)Session["Id"], out errorMessage);
        return View("Student");
    }
    else
    {
        ViewBag.Assignments = AssignmentManager.getAssignmentsByLecturerId((int)Session["Id"], out errorMessage);
        return View("Lecturer");
    }

}
```

~SimpleBlackBoard\Controllers\DashboardController.cs

3.5 Validation on client and server sides[i]

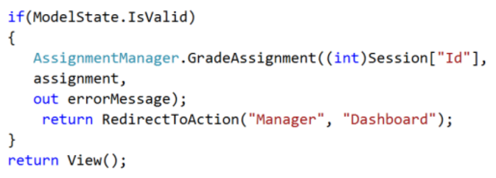
Regarding front end validation, we take advantage of the built in unobtrusive JavaScript and validate jQuery scripts. MVC integrates these libraries for front end validation. For front end validation we use *@Html.ValidationSummary()* at the bottom of the form along with *@Html.ValidationMessageFor(model => model.field)*. The validation summary provides a summary of all the validation errors at the end, while the validation message for, provides the error regarding the certain field.



```
@using (@Html.BeginForm("Grade", "Grade", new { id = @ViewBag.Id }, FormMethod.Post))
{

    @*@Html.LabelFor(model => @Model.Student_ID)  <h6>@ViewBag.Student_ID</h6>*@

    <span style="color:green">@Html.LabelFor(model => model.Grade)</span>
    @Html.ValidationMessageFor(model => model.Grade)
    <br />
    <div class="row">
        <div class="col-xs-2">@Html.RadioButtonFor(model => @Model.Grade, 1) <p style="display:inline">  1  </p></div>

        <div class="col-xs-2">@Html.RadioButtonFor(model => @Model.Grade, 2) <p style="display:inline">  2  </p></div>
        <div class="col-xs-2">@Html.RadioButtonFor(model => @Model.Grade, 3) <p style="display:inline">  3  </p></div>
        <div class="col-xs-2">@Html.RadioButtonFor(model => @Model.Grade, 4) <p style="display:inline">  4  </p></div>
        <div class="col-xs-2">@Html.RadioButtonFor(model => @Model.Grade, 5) <p style="display:inline">  5  </p></div>
      </div>
    <br />
    <span style="color:green">
        @Html.LabelFor(model => model.Feedback)
    </span>
    <br />
    @Html.EditorFor(model => model.Feedback, new { @class = "", @cols = 200, @rows = 30 })<br />
    @Html.ValidationMessageFor(model => model.Feedback)
    <br />

    <input type="submit" value="Grade" class="btn btn-success" />
    <br />
    @Html.ValidationSummary()
    }
```

~SimpleBlackBoard\Views \Dashboard\Grade.cshtml

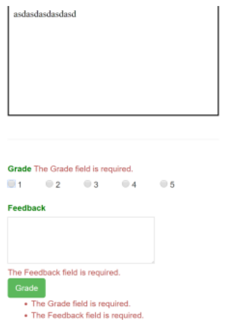
If the model state is valid we redirect to the manager which takes us back to the landing page or lecturer dashboard. Otherwise we return the same view where the errors then appear.



```
if(ModelState.IsValid)
{
   AssignmentManager.GradeAssignment((int)Session["Id"],
   assignment,
   out errorMessage);
    return RedirectToAction("Manager", "Dashboard");
}
return View();
```

~SimpleBlackBoard\Controllers \DashboardController.cs

On the grade page of the lecturer, if there are any missing fields, the submit will not work due to the validation and unobtrusive JavaScript validation that doesn't allow the post to work.



asdasdasdasdasd

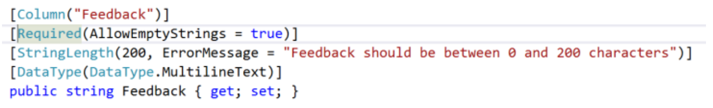
Grade The Grade field is required.

1 2 3 4 5

Feedback

The Feedback field is required.

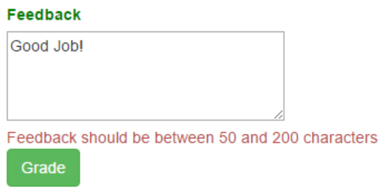
- The Grade field is required.
- The Feedback field is required.



```
[Column("Feedback")]
[Required(AllowEmptyStrings = true)]
[StringLength(200, ErrorMessage = "Feedback should be between 0 and 200 characters")]
[DataType(DataType.MultilineText)]
public string Feedback { get; set; }
```

~SimpleBlackBoard\Models\Assignment.cs

The feedback length validation is done by the [StringLSength] annotation, which is utilised in the HTML Helper method *LabelFor*() where the Assignment model is used.

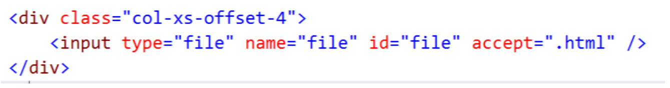


Feedback

Good Job!

Feedback should be between 50 and 200 characters

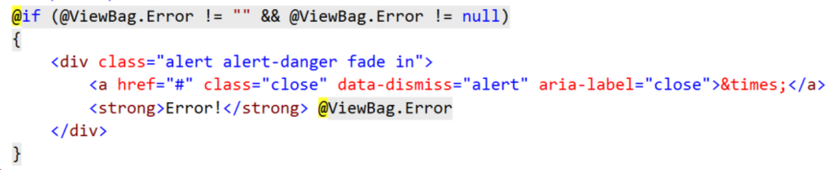
In addition, we provide a simple front end validation for the upload of an assignment to only accept html files.



```
<div class="col-xs-offset-4">
    <input type="file" name="file" id="file" accept=".html" />
</div>
```

~SimpleBlackBoard\Views\Dashboard \Upload.cshtml

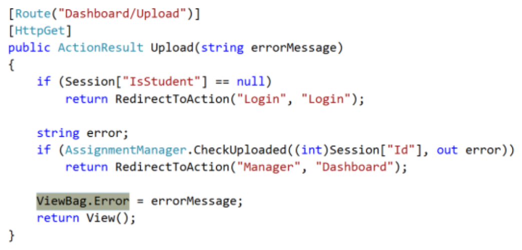
Regarding Server side validation, we provide the Data Annotations that we talked about previously. But to convey these server side errors to the user, we do the following:



```
@if (@ViewBag.Error != "" && @ViewBag.Error != null)
{
    <div class="alert alert-danger fade in">
        <a href="#" class="close" data-dismiss="alert" aria-label="close">&times;</a>
        <strong>Error!</strong> @ViewBag.Error
    </div>
}
```

SimpleBlackBoard\Views\Dashboard \Upload.cshtml

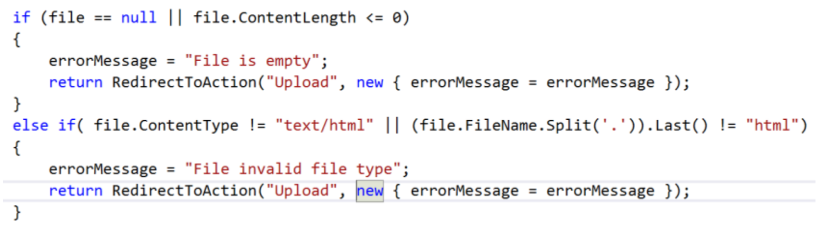
This division is added to the bottom of the page, where it uses razor syntax to check the ViewBag.Error which is assigned in controllers.



```
[Route("Dashboard/Upload")]
[HttpGet]
public ActionResult Upload(string errorMessage)
{
    if (Session["IsStudent"] == null)
        return RedirectToAction("Login", "Login");

    string error;
    if (AssignmentManager.CheckUploaded((int)Session["Id"], out error))
        return RedirectToAction("Manager", "Dashboard");

    ViewBag.Error = errorMessage;
    return View();
}
```

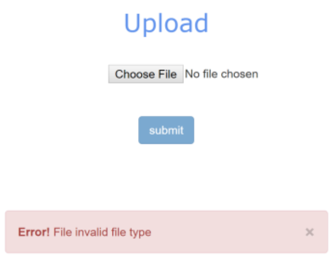
~SimpleBlackBoard\Controllers\DashboardController.cs

The ViewBag.Error is always assigned a variable called errorMessage. This is an *out string* that is assigned in the managers upon performing certain queries, and if a certain error occurs in the database, we assign to the appropriate message. However, if there was another issue we assign it in the controller.

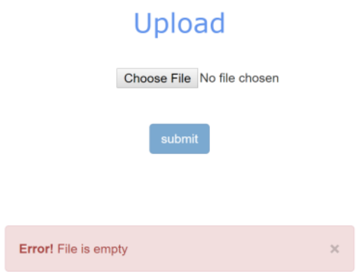


```
if (file == null || file.ContentLength <= 0)
{
    errorMessage = "File is empty";
    return RedirectToAction("Upload", new { errorMessage = errorMessage });
}
else if( file.ContentType != "text/html" || (file.FileName.Split('.')).Last() != "html")
{
    errorMessage = "File invalid file type";
    return RedirectToAction("Upload", new { errorMessage = errorMessage });
}
```

~SimpleBlackBoard\Controllers\DashboardController.cs



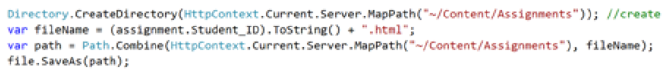
Error! File invalid file type ×



Error! File is empty ×

3.6 Extras (File System, Password Hashing, Assigning of Lecturers, JQuery on upload)

We decided to store the assignments in the file system, these assignments are in ~/Content/Assignments/. When the student uploads, we update the Uploaded field in his record to 1 thus confirming he has successfully uploaded his assignment.

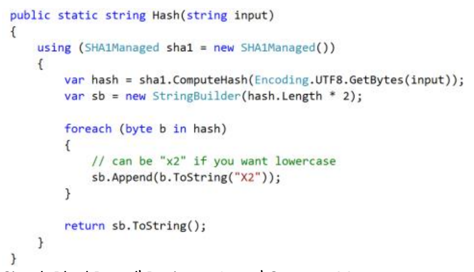


```
Directory.CreateDirectory(HttpContext.Current.Server.MapPath("~/Content/Assignments")); //create
var fileName = (assignment.Student_ID).ToString() + ".html";
var path = Path.Combine(HttpContext.Current.Server.MapPath("~/Content/Assignments"), fileName);
file.SaveAs(path);
```

~SimpleBlackBoard\Business_Layer\AssignmentManager.cs

To facilitate testing for the professor, we assigned each user with a specific lecturer. In a real application, when a student registers he would be assigned a random lecturer using a helper method available in the application.

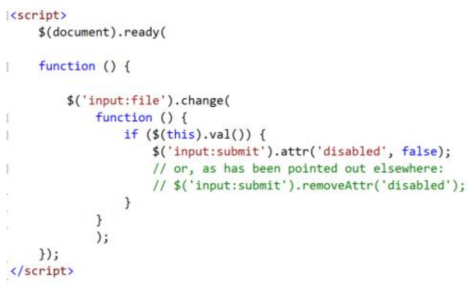
Upon adding students or lecturers, we check if the user already exists and then the password is hashed and stored in the database. We also hash the password using SHA1[ii] when logging in to match it to the one in the database.



```
public static string Hash(string input)
{
    using (SHA1Managed sha1 = new SHA1Managed())
    {
        var hash = sha1.ComputeHash(Encoding.UTF8.GetBytes(input));
        var sb = new StringBuilder(hash.Length * 2);

        foreach (byte b in hash)
        {
            // can be "x2" if you want lowercase
            sb.Append(b.ToString("X2"));
        }

        return sb.ToString();
    }
}
```

~SimpleBlackBoard\Business_Layer\CommonManager.cs

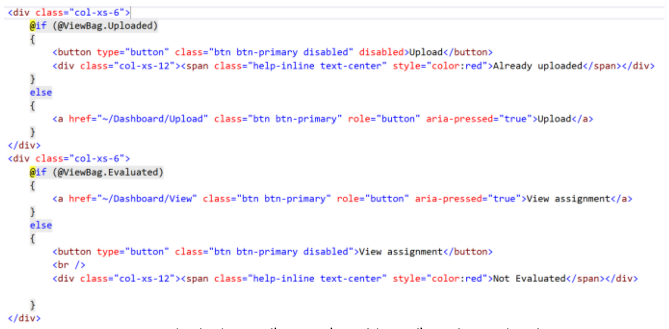
A user cannot press the upload button if no file has been chosen to upload, so we added a little bit of JQuery that controls that by disabling the button if no file is uploaded.



```
<script>
    $(document).ready(

    function () {

        $('input:file').change(
            function () {
                if ($(this).val()) {
                    $('input:submit').attr('disabled', false);
                    // or, as has been pointed out elsewhere:
                    // $('input:submit').removeAttr('disabled');
                }
            }
            );
    });
</script>
```

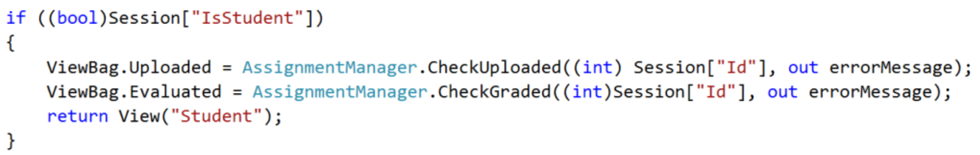
~SimpleBlackBoard\Views \Dashboard\Upload.cshtml

In the student landing page, the student cannot upload if his uploaded bit is 1 thus use helper methods that help us enable and disable these buttons in an alternating way. This information is passed to the View using the ViewBag object.



```
<div class="col-xs-6">
    @if (@ViewBag.Uploaded)
    {
        <button type="button" class="btn btn-primary disabled" disabled>Upload</button>
        <div class="col-xs-12"><span class="help-inline text-center" style="color:red">Already uploaded</span></div>
    }
    else
    {
        <a href="~/Dashboard/Upload" class="btn btn-primary" role="button" aria-pressed="true">Upload</a>
    }
</div>
<div class="col-xs-6">
    @if (@ViewBag.Evaluated)
    {
        <a href="~/Dashboard/View" class="btn btn-primary" role="button" aria-pressed="true">View assignment</a>
    }
    else
    {
        <button type="button" class="btn btn-primary disabled">View assignment</button>
        <br />
        <div class="col-xs-12"><span class="help-inline text-center" style="color:red">Not Evaluated</span></div>

    }
</div>
```

~SimpleBlackBoard\Views \Dashboard\Student.cshtml



```
if ((bool)Session["IsStudent"])
{
    ViewBag.Uploaded = AssignmentManager.CheckUploaded((int) Session["Id"], out errorMessage);
    ViewBag.Evaluated = AssignmentManager.CheckGraded((int)Session["Id"], out errorMessage);
    return View("Student");
}
```

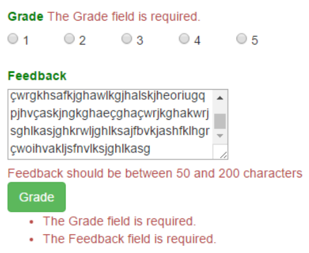
~SimpleBlackBoard\Controllers \DashboardController.cs

4 Testing

Since the assignment is not too complex, it is possible to test part of it manually and as it will not scale in the future, automated testing does not bring many advantages. Nevertheless, the following sections will go into the details of the way we tested the system to make sure it was functioning as intended.

4.1 Front End Testing

Front-end testing was done manually due to the small size of the application. We create a list of possible problems such as empty inputs or invalid types in forms. Most of this validation is done by the framework through the annotations. The error messages that are displayed to the user are automatically generated making it easy to convey the error messages to the user.



Grade The Grade field is required.

1 2 3 4 5

Feedback

çwrgkhsafkjghawlkgjhalskjheoriugq
pjhvçaskjngkghaeçghaçwrjkghakwrj
sghlkasjghkrwljghlksajfbvkjashfklhgr
çwoihvakljsfnvlksjghlkasg

Feedback should be between 50 and 200 characters

- The Grade field is required.
- The Feedback field is required.

4.2 Portability Testing

Portability testing was also done manually, and all three of the biggest browsers functioned as expected. This stems from the limited use of JavaScript and the compatibility of HTML5.

4.3 Business Logic Testing

For the business logic, we implemented a separate project named "UnitTesting" which is a Unit Test Project. The tests use the NUnit framework and use the Assert class to verify the results. Once again, since these were not a main focus on the assignment, they utilise the same database as the application and are meant for informal testing.

5 Additional and Advanced Technologies

Using chrome developer tools we were able to measure the loading times and sizes of the pages. Due to this being a fairly simple project the loading times are reasonably fast, not to mention the rare use of Javascript and the small size of our database making the backend quicker. Moreover, the use of LINQ helps increase querying time and the MVC framework leads to better web performance**.**

Average Page size:

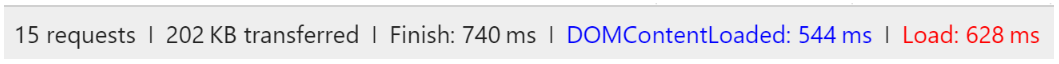
(203 + 202 + 202+ 0.494 + 201 +202 +201 + 0.437) / 8

= 151.49 KB

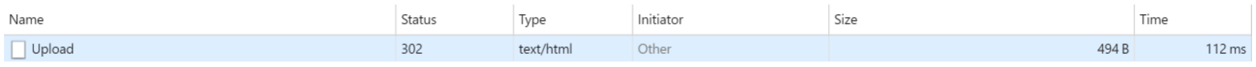
Average Load Time:

(35 + 657 + 579 + 112 + 628 +557 + 567 +455) / 8

= 448.75 ms

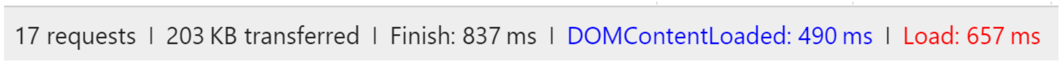

15 requests | 202 KB transferred | Finish: 740 ms | DOMContentLoaded: 544 ms | Load: 628 ms

Upload: Loading time 628ms ; Page size:202KB

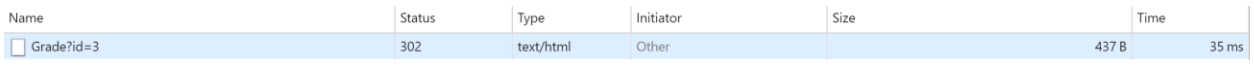


Name	Status	Type	Initiator	Size	Time
Upload	302	text/html	Other	494 B	112 ms

Uploading: Loading time 112ms ; Page size: 494B [POST]


17 requests | 203 KB transferred | Finish: 837 ms | DOMContentLoaded: 490 ms | Load: 657 ms

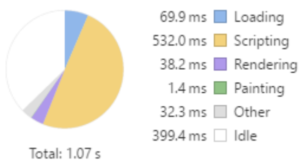
Grade Page: Loading time 657ms ; Page size 203KB



Name	Status	Type	Initiator	Size	Time
Grade?id=3	302	text/html	Other	437 B	35 ms

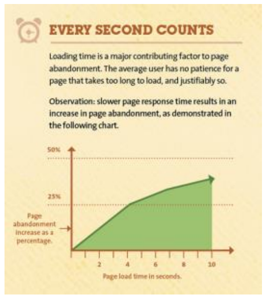
Grading: load time 35ms ; Page size of 437B [POST]

As seen on the pie chart, we can see that most of the load time comes from scripting, which comes from bootstrap and the MVC framework. Other than that, the page is very light overall



Load login summary

Nowadays, site speed is a crucial area of website development. Users expect a site to load within a second or two. It is one of the ranking factors in Google. A fast site allows for a better user experience (UX), and a satisfying UX leads to higher conversions.[iii]

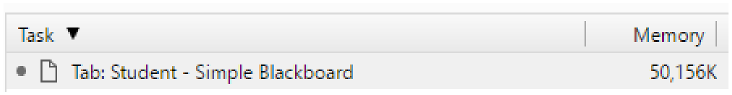


EVERY SECOND COUNTS

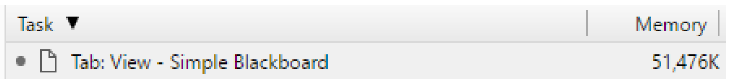
Loading time is a major contributing factor to page abandonment. The average user has no patience for a page that takes too long to load, and justifiably so.

Observation: slower page response time results in an increase in page abandonment, as demonstrated in the following chart.

In terms of memory, we have the following two examples which represent the average webpage memory usage:

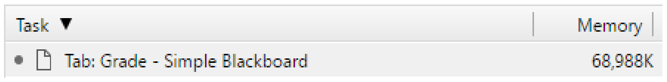


Task ▼	Memory
Tab: Student - Simple Blackboard	50,156K



Task ▼	Memory
Tab: View - Simple Blackboard	51,476K

The usual webpage is about 2MB, but due to the simple design of the service and the few existing functionalities, our implementation is significantly lighter in size. The lack of complex styling and JavaScript help make sure the webpage is as light as possible. This remains somewhat constant across the application apart from the Grade page, which contains the <*iframe*> and the content inside it causes the page to be larger in size.



Task ▼	Memory
Tab: Grade - Simple Blackboard	68,988K

6 Evaluation

6.1 Critical Evaluation

- Despite not being our first time using ASP.NET MVC, it was the first time building an application from scratch and we have found that there are a lot of existing features that facilitate some of the existing requirements when building a web application.
- The Individual User Accounts provide multiple extra features such as validation and authentication, login with external services. It provides an easy way of routing the controllers using the [Route] and manage the different HTTP requests.
- One feature we found to be particularly useful was the HTML Helper. With this, automatic HTML is generated making it easier to edit or display a model. This HTML also does validation on the inputs based on the annotations present in the model facilitation front-end validation.
- Besides the lectures, ASP.NET offers a comprehensive documentation about every aspect of the development cycle. Alongside the Microsoft documentation, there are a lot of other online resources.
- Overall, ASP.NET MVC was a very straightforward framework to use, with a very intuitive API. C# is a very familiar language to us making it easy to structure the code and get the most out of object oriented programming.

6.2 Comparative Evaluation[iv]

- ASP.NET framework is built on an OOP paradigm and OOP concepts; PHP is not.
- PHP only supports partial encapsulation (such as support for declaring methods and fields in the class) and partial polymorphism (no overloading, no abstraction).
- ASP.NET applications result in better designed code, have clear separation of content, logic, and data and thus are generally easier to support over the long term of an applications life cycle.
- PHP has drivers for most databases, while .NET Supports OLE-DB and ODBC directly, and includes native drivers for Microsoft SQL Server™ and Oracle.
- PHP does not support error trapping but has various error-handling functionality and logging, Supports structured exception handling.
- PHP is loosely typed so it's easier to make mistakes, while ASP.NET is strongly typed having type and syntax checking.
- Ability to create a database using code first approach in ASP.NET, meanwhile in PHP one must create a database manually or use any other DBMS
- ASP.NET has built in validation handling which uses bootstrap and unobtrusive JavaScript that is integrated to work. While in PHP you must have libraries or manually do validation.

7 References and Bibliography

I. Anderson, Rick. "Getting Started." *The Official Microsoft ASP.NET Site*. N.p., 12 Aug. 2014. Web. https://www.asp.net/mvc/overview/getting-started/introduction/getting-started Accessed 18 Oct. 2016

II. Shekhawat, Sandeep Singh. "ASP.NET MVC Client Side Validation." *CodeProject*. N.p., 02 Oct. 2014. Web. https://www.codeproject.com/articles/718004/asp-net-mvc-client-side-validation Accessed 22 Nov. 2016.

III. Everts, Tammy. "Page Bloat Update: The Average Web Page Is More than 2 MB in Size | SOASTA." *SOASTA*. N.p., 11 June 2015. Web. https://www.soasta.com/blog/page-bloat-average-web-page-2-mb/ Accessed 30 Nov. 2016.

IV. Dykstra, Tom. "Creating an Entity Framework Data Model." *The Official Microsoft ASP.NET Site*. N.p., 23 Apr. 2014. Web. https://www.asp.net/mvc/overview/getting-started/getting-started-with-ef-using-mvc/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application Accessed Oct. 2016.

V. Google Developers. "Improve Server Response Time". *Developers of Google*. N.P, April 8, 2015. https://developers.google.com/speed/docs/insights/Server/ Accessed Nov 20. 2016

[i] Kennedy, Michael. "Validating ASP.NET MVC Forms with DataAnnotations" https://www.youtube.com/watch?v=Nf617_tjUtc

[ii] Stack Overflow "Hashing with SHA1 Algorithm in C#" http://stackoverflow.com/questions/17292366/hashing-with-sha1-algorithm-in-c-sharp accessed Nov 2. 2016

[iii] Anderson, Shaun. "How Fast Should A Website Load?" http://www.hobo-web.co.uk/your-website-design-should-load-in-4-seconds/ accessed December 2. 2016

[iv] Microsoft Corporation "Migrating from PHP to ASP.NET" https://msdn.microsoft.com/en-us/library/aa479002.aspx accessed Nov 28. 2016