SE Lab 6 – ASP.NET MVC

ASP.NET Web Applications (.NET Framework) 2

Key Features of ASP.NET Web Applications 2

Getting Started with ASP.NET Web Applications 2

1. ASP.NET – Database First approach 3

Step 1: Setting Up the Students Database 3

Step 2: Creating the ASP.NET Web Application 4

Step 3: Setting Up the Database Connection 4

Step 4: Creating the Students Controller and Views 5

Step 5: Testing the Application 5

2. Exercises 6

2.1. Exercise 1: Setting Up GitHub Repository for ASP.NET MVC (.Net Framework) 6

2.2. Exercise 2: School Management Web App with ASP.NET MVC (.Net Framework) 7

2.1. Functional Requirements 7

2.2. Non-Functional Requirements 7

2.3. Implementation Solution 8

Step 1: Setting ASP.NET MVC with GitHub 8

Step 2: Modify the Menu in _Layout.cshtml 9

Step 3: Setting Up the Database 10

Step 4: Setting Up the Database Connection 11

Step 5: Scaffolding Controllers and Views 12

Step 6: Testing the Application 12

2.3. Exercise 3: Challenge to get credit points 13

Step 1: Add a tblUser Table and Its Data 13

Step 2: Make a Login Form for Exercise 2 School App 13

Step 3: Set the Login Form as the First Screen 13

2.4. Exercise 4: Report in ASP.NET MVC 13

3. Submit on eLearning: Summary Report 20

FYI: MVC with Department, Dependent, and Employee: https://youtu.be/GS_Ra0-ZKQY

ASP.NET Web Applications (.NET Framework)

ASP.NET is a web application framework developed by Microsoft, which allows developers to build dynamic web sites, applications, and services. It is part of the .NET Framework and provides a robust, scalable, and secure environment for web development.

ASP.NET Web Applications using the .NET Framework provide a powerful and flexible platform for building modern web applications. Whether you're developing a simple website or a complex enterprise application, ASP.NET offers the tools and features you need to succeed

Key Features of ASP.NET Web Applications

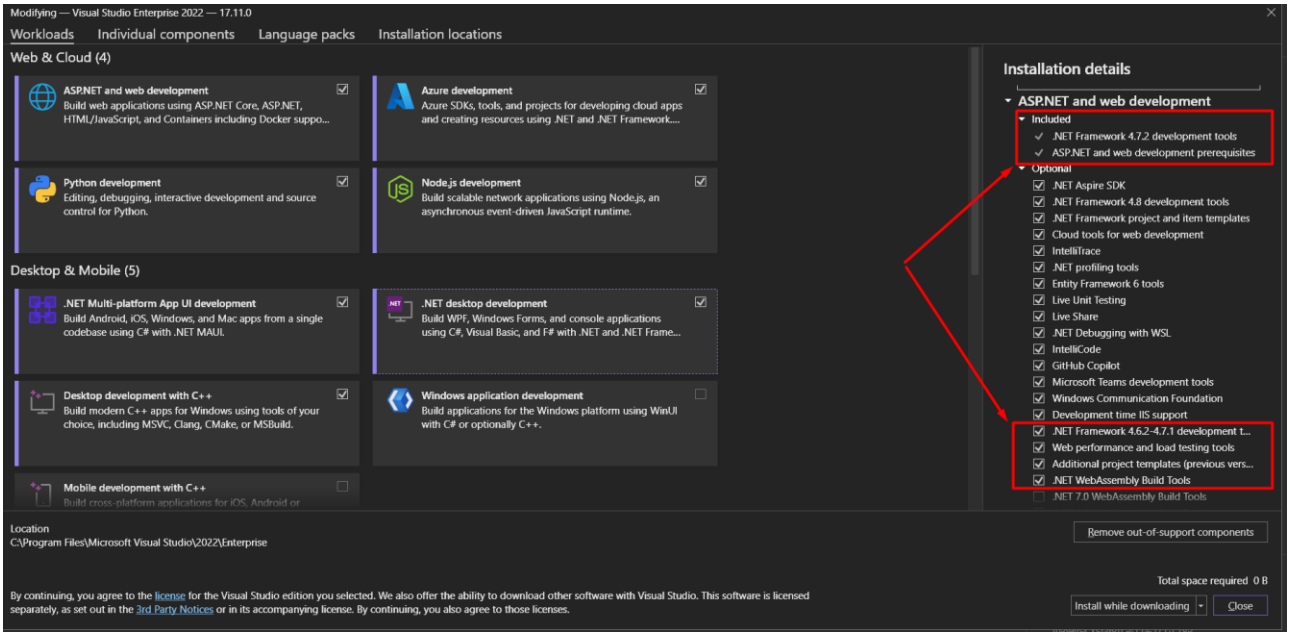
1. Server-Side Technology: ASP.NET runs on the server and generates dynamic web pages that can be viewed in any browser. This allows for more powerful and flexible web applications compared to client-side technologies.
2. Rich Toolbox and Designer: Visual Studio, the integrated development environment (IDE) for .NET, offers a rich set of tools and a designer for building web applications. This includes drag-and-drop controls, automatic deployment, and debugging tools.
3. State Management: ASP.NET provides various ways to manage state, such as session state, application state, and view state, which help maintain user data across different requests.
4. Security: Built-in security features like authentication and authorization, membership and roles, and secure communication (HTTPS) ensure that web applications are safe and secure.
5. Performance: ASP.NET applications are compiled, which means they run faster than interpreted scripts. Additionally, features like caching and asynchronous programming improve performance and scalability.
6. Extensibility and Customization: ASP.NET is highly extensible, allowing developers to create custom controls, modules, and handlers to meet specific needs.
7. Integration with .NET Framework: ASP.NET seamlessly integrates with the .NET Framework, providing access to a vast library of classes and functions for various tasks, such as file I/O, database interaction, and XML processing.

Getting Started with ASP.NET Web Applications

1. Install Visual Studio: Download and install Visual Studio, which is the primary IDE for developing ASP.NET applications.
2. Create a New Project: Open Visual Studio, select "Create a new project," and choose "ASP.NET Web Application (.NET Framework)" from the list of templates.
3. Choose a Template: Visual Studio offers several templates, such as Empty, Web Forms, MVC, and Web API. Choose the one that best fits your project requirements.
4. Develop Your Application: Use the rich set of tools and controls in Visual Studio to design and develop your web application. Write your code in C# or VB.NET, and use HTML, CSS, and JavaScript for the front-end.

5. Test and Debug: Use Visual Studio's built-in debugging tools to test your application and fix any issues.
6. Deploy Your Application: Once your application is ready, you can deploy it to a web server using various methods, such as FTP, Web Deploy, or cloud services like Azure.

Open Visual Studio Installer:



1. ASP.NET – Database First approach

Create an ASP.NET Web Application (.NET Framework) using the Database First approach with a Students database. Here are the steps to guide you through the process:

Step 1: Setting Up the Students Database

Objective: Create a Students database and a table to store student information.

Steps:

1. Create a Database:
    - Open SQL Server Management Studio (SSMS).
    - Create a new database named StudentsDB.
2. Create Students Table:
    - In the StudentsDB database, create a table named tblStudents with the following columns:

```
CREATE TABLE tblStudents (
    StudentID INT PRIMARY KEY IDENTITY(1,1),
```

```
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    Email NVARCHAR(100) NOT NULL,
    EnrollmentDate DATE NOT NULL
);
```

3. Insert Sample Data:
    - Insert sample student data into the tblStudents table:

```
INSERT INTO tblStudents (FirstName, LastName, Email, EnrollmentDate) VALUES ('Trung', 'Pham', 'trung.pham@seuni.com', '2024-01-15');
INSERT INTO tblStudents (FirstName, LastName, Email, EnrollmentDate) VALUES ('Michael', 'Pham', 'michael.pham@seuni.com', '2024-02-20');
```

Step 2: Creating the ASP.NET Web Application

Objective: Create a new ASP.NET Web Application (.NET Framework) project and set up the necessary packages.

Steps:

1. Create a New Project:
    - Open Visual Studio and create a new ASP.NET Web Application (.NET Framework) project.
    - Name your project StudentsApp and choose a location to save it.
    - Select the MVC template and click Create.
2. Install Entity Framework:
    - Right-click on your project in the Solution Explorer and select Manage NuGet Packages.
    - Search for EntityFramework and install it.

Step 3: Setting Up the Database Connection

Objective: Configure the connection to the StudentsDB database using Entity Framework.

Steps:

1. Define the Connection String in Web.config:
    - Open the Web.config file in your project.
    - Add the connection string within the <connectionStrings> section:

```
<connectionStrings>
```

```
 <add name="StudentsDBConnectionString" connectionString="data
source=YourServer;initial catalog=StudentsDB;integrated security=True;"
providerName="System.Data.SqlClient" />

</connectionStrings>
```

2. Create the Entity Framework Model:
    - Right-click on your project in the Solution Explorer, select Add > New Item.
    - Choose Data from the left menu and select ADO.NET Entity Data Model. Name it StudentsModel and click Add.
    - In the Entity Data Model Wizard, select EF Designer from database and click Next.
    - Choose your data connection or create a new connection to your StudentsDB database. This will automatically generate the connection string in your Web.config file.
    - Select the tblStudents table and click Finish.

Step 4: Creating the Students Controller and Views

Objective: Create a controller and views to display and manage student information.

1. Create the Students Controller:
    - Right-click on the Controllers folder, select Add > Controller, and choose MVC 5 Controller with views, using Entity Framework. Name it StudentsController.
    - Select StudentsModel as the model class and StudentsDBEntities as the data context class. Click Add.
2. Modify the Index View:
    - Open the Views/Students/Index.cshtml file.
    - Modify the view to display a list of students with their details.

Step 5: Testing the Application

Objective: Run the application and test the functionality.

Steps:

1. Run the Application:
    - Press F5 or click the Start button to run the application.
    - Navigate to /Students to access the list of students.
2. Verify the Output:
    - Check the list of students displayed on the page to verify that the data is being retrieved from the database correctly.

2. Exercises

2.1. Exercise 1: Setting Up GitHub Repository for ASP.NET MVC (.Net Framework)

Objective: Create a GitHub repository and manage the source code for the ASP.NET MVC (.Net Framework) application.

Steps:

1. Create a New Repository:
    - Go to your GitHub profile and click on the Repositories tab.
    - Click the New button to create a new repository.
    - Name the repository ASPNETMVCStudent.
    - Add a description (optional) and choose whether the repository should be public or private.
    - Check the box to initialize the repository with a README file.
    - Click Create repository.
2. Clone the Repository Locally:
    - Open Visual Studio.
    - Go to File > Clone Repository.
    - Enter the URL of your GitHub repository and choose a local path to clone the repository.
    - Click Clone.
3. Add Your ASP.NET MVC(.Net Framework) Project:
    - Open your ASP.NET MVC (.Net Framework) project in Visual Studio.
    - Right-click on the solution in the Solution Explorer and select Add > Existing Project.
    - Navigate to the cloned repository folder and add your project.
4. Commit and Push Changes:
    - Go to the Git Changes window in Visual Studio.
    - Stage all changes by clicking the + button.
    - Enter a commit message, e.g., "Initial commit of ASP.NET MVC (.Net Framework) Student App".
    - Click Commit All and then Push to push the changes to GitHub.

2.2. Exercise 2: School Management Web App with ASP.NET MVC (.Net Framework)

This exercise aims to learn how to create an ASP.NET Web MVC application, connect it to an MSSQL database using Entity Framework (Database First approach), and implement CRUD operations for managing Students, Courses, Instructors, Enrollments, and Departments.

Objective: Create an ASP.NET MVC (.Net Framework) application to manage student information, including CRUD operations for Students, Courses, Instructors, Enrollments, and Departments. Source Code management with Github.

2.1. Functional Requirements

1. Student Management:
    - Add, edit, delete, and view student information.
    - Fields: StudentID, FirstName, LastName, EnrollmentDate.
2. Course Management:
    - Add, edit, delete, and view course information.
    - Fields: CourseID, Title, Credits, DepartmentID.
3. Instructor Management:
    - Add, edit, delete, and view instructor information.
    - Fields: InstructorID, FirstName, LastName, HireDate.
4. Department Management:
    - Add, edit, delete, and view department information.
    - Fields: DepartmentID, Name, Budget, StartDate.
5. Enrollment Management:
    - Add, edit, delete, and view enrollment information.
    - Fields: EnrollmentID, CourseID, StudentID, Grade.
6. Database Integration:
    - Use Entity Framework to connect to an MSSQL database.
    - Implement CRUD operations using Entity Framework.

2.2. Non-Functional Requirements

1. Usability:
    - User-friendly interface with clear labels and instructions.
    - Consistent layout and design across all views.

2. Performance:
    - Fast response time for database operations.
    - Efficient handling of large datasets.
3. Security:
    - Secure database connection using connection strings.
    - Input validation to prevent SQL injection and other attacks.

2.3. Implementation Solution

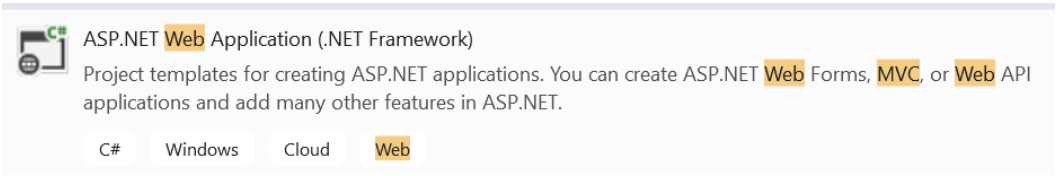
Create an ASP.NET Web App (.Net Framework) using the Database First approach with a more comprehensive database schema that includes Students, Courses, Instructors, Enrollments, and Departments.

Step 1: Setting ASP.NET MVC with GitHub

Create an ASP.NET MVC project in Visual Studio, initialize a local Git repository, and then map it to a GitHub repository.

Steps:

1. Create a New ASP.NET Core MVC Project:
    - Open Visual Studio.
    - Go to File > New > Project.
    - Select ASP.NET Web Application (.Net Framework) from the list of templates.



    - Name your project ASPNETWebAppMVCStudentApp , choose a location to save it, and Choose "MVC" as the project template
    - Click Create.
2. Install Entity Framework:
    - Right-click on your project in the Solution Explorer and select Manage NuGet Packages.
    - Search for EntityFramework and install it.
3. Initialize a Local Git Repository:
    - In Visual Studio, go to View > Git Changes to open the Git Changes window.
    - Click the Initialize Repository button to create a local Git repository for your project.

4. Create a GitHub Repository:
    - Go to your GitHub profile and click on the Repositories tab.
    - Click the New button to create a new repository.
    - Name the repository ASPNETWebAppMVCStudentApp .
    - Add a description (optional) and choose whether the repository should be public or private.
    - Do not initialize the repository with a README file, .gitignore, or license.
    - Click Create repository.
5. Map the Local Repository to GitHub:
    - In Visual Studio, go to the Git Changes window.
    - Click the Settings icon (gear icon) and select Repository Settings.
    - Under Remotes, click Add.
    - Enter origin as the remote name and paste the URL of your GitHub repository.
    - Click Save.
6. Commit and Push Changes:
    - Go to the Git Changes window in Visual Studio.
    - Stage all changes by clicking the + button.
    - Enter a commit message, e.g., "Initial commit of ASP.NET MVC Student App".
    - Click Commit All and then Push to push the changes to GitHub.

Step 2: Modify the Menu in _Layout.cshtml

1. Add a Navigation Menu:
    - Open the _Layout.cshtml file in the Views/Shared folder.
    - Add a navigation menu in the <nav> section to link to each controller's index view.

```
<nav>
    <ul>
        <li>@Html.ActionLink("Departments", "Index", "Departments")</li>
        <li>@Html.ActionLink("Instructors", "Index", "Instructors")</li>
        <li>@Html.ActionLink("Courses", "Index", "Courses")</li>
        <li>@Html.ActionLink("Students", "Index", "Students")</li>
        <li>@Html.ActionLink("Enrollments", "Index", "Enrollments")</li>
    </ul>
</nav>
```

Step 3: Setting Up the Database

Objective: Create a database with tables for Students, Courses, Instructors, Enrollments, and Departments.

Steps:

1. Create a Database:
    - Open SQL Server Management Studio (SSMS).
    - Create a new database named SchoolDB.
2. Create Tables:
    - Execute the following SQL scripts to create the necessary tables:

```
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY IDENTITY(1,1),
    Name NVARCHAR(50) NOT NULL,
    Budget DECIMAL(18, 2) NOT NULL,
    StartDate DATE NOT NULL
);
CREATE TABLE Instructors (
    InstructorID INT PRIMARY KEY IDENTITY(1,1),
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    HireDate DATE NOT NULL
);
CREATE TABLE Courses (
    CourseID INT PRIMARY KEY IDENTITY(1,1),
    Title NVARCHAR(100) NOT NULL,
    Credits INT NOT NULL,
    DepartmentID INT FOREIGN KEY REFERENCES Departments(DepartmentID)
);
CREATE TABLE Students (
    StudentID INT PRIMARY KEY IDENTITY(1,1),
```

```
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    EnrollmentDate DATE NOT NULL
);

CREATE TABLE Enrollments (
    EnrollmentID INT PRIMARY KEY IDENTITY(1,1),
    CourseID INT FOREIGN KEY REFERENCES Courses(CourseID),
    StudentID INT FOREIGN KEY REFERENCES Students(StudentID),
    Grade DECIMAL(3, 2)
);
```

3. Insert Sample Data:
    - Insert sample data into the tables:

```
INSERT INTO Departments (Name, Budget, StartDate) VALUES ('Computer Science', 120000.00, '2023-01-01');

INSERT INTO Instructors (FirstName, LastName, HireDate) VALUES ('John', 'Doe', '2020-08-15');

INSERT INTO Courses (Title, Credits, DepartmentID) VALUES ('Introduction to Programming', 3, 1);

INSERT INTO Students (FirstName, LastName, EnrollmentDate) VALUES ('Jane', 'Smith', '2023-09-01');

INSERT INTO Enrollments (CourseID, StudentID, Grade) VALUES (1, 1, 3.5);
```

Step 4: Setting Up the Database Connection

Objective: Configure the connection to the SchoolDB database using Entity Framework.

Steps:

1. Define the Connection String in Web.config:
    - Open the Web.config file in your project.
    - Add the connection string within the <connectionStrings> section:

```
<connectionStrings>
  <add name="SchoolDBConnectionString" connectionString="data source=YourServer;initial catalog=SchoolDB;integrated security=True;" providerName="System.Data.SqlClient" />
</connectionStrings>
```

2. Create the Entity Framework Model:
    - Right-click on your project in the Solution Explorer, select Add > New Item.
    - Choose Data from the left menu and select ADO.NET Entity Data Model. Name it SchoolModel and click Add.
    - In the Entity Data Model Wizard, select EF Designer from database and click Next.
    - Choose your data connection or create a new connection to your SchoolDB database. This will automatically generate the connection string in your Web.config file.
    - Select the tables (Departments, Instructors, Courses, Students, Enrollments) you want to include in your model and click "Finish".

Step 5: Scaffolding Controllers and Views

Objective: Create controllers and views to manage Students, Courses, Instructors, Enrollments, and Departments.

1. Create Controllers:
    - Right-click on the Controllers folder, select Add > Controller, and choose MVC 5 Controller with views, using Entity Framework. Name it StudentsController.
    - Select Student as the model class and SchoolDBEntities as the data context class. Click Add.
    - Repeat this step for each model (Instructor, Course, Student, Enrollment).
2. Modify the Views:
    - Open the views for each controller (e.g., Views/Students/Index.cshtml) and ensure they display the relevant data.

Step 6: Testing the Application

Objective: Run the application and test the functionality.

Steps:

1. Run the Application:
    - Press F5 or click the Start button to run the application.
    - Navigate to the respective URLs (e.g., /Students, /Courses) to access the list of entities.
    - Test CRUD operations for each entity.
2. Verify the Output:
    - Check the lists displayed on the pages to verify that the data is being retrieved from the database correctly.

3. Refine the UI:
    - Customize the views and layout to improve the user interface.
    - Add validation and error handling as needed.

2.3. Exercise 3: Challenge to get credit points

In this exercise, we aim to enhance the functionality of the Exercise 2 School App by implementing a user authentication system. This involves three key steps:

Step 1: Add a tblUser Table and Its Data

- Create a new table named tblUser in your database.
- Define the necessary columns for the table, such as UserID, Username, Password, Email, and any other relevant fields.
- Populate the tblUser table with sample data for testing purposes.

Step 2: Make a Login Form for Exercise 2 School App

- Design a login form that includes fields for the username and password.
- Implement functionality to validate the user credentials against the data in the tblUser table.
- Provide appropriate feedback for successful and unsuccessful login attempts.

Step 3: Set the Login Form as the First Screen

- Configure the application to display the login form as the initial screen when the app is launched.
- Ensure that upon successful login, the user is redirected to the main application interface.
- If the login fails, prompt the user to re-enter their credentials or provide an option to reset the password.

2.4. Exercise 4: Report in ASP.NET MVC

In this exercise, we will enhance the functionality of the School App by adding a feature to generate a filtered student report. This involves creating a new tab in the application that links to a filter form. The filter form will allow users to select a course and generate a report listing the students enrolled in that course. The report can be previewed or exported to a PDF file.

The steps include setting up the project, installing necessary reporting tools, designing the RDLC report, creating the filter form, adding a tab with an action link, and implementing the controller actions to handle report generation. By the end of this exercise, you will have a fully functional feature that enables users to generate and export student reports based on selected courses, enhancing the overall usability and functionality of the School App.

1. Set Up the Project

- Create a New ASP.NET MVC Project:
  - Open Visual Studio and create a new ASP.NET MVC project if you haven't already.
  - Ensure Entity Framework is installed and configured to connect to your MSSQL database.

2. Install Reporting Tools

- Install RDLC Reporting Tools:
  - Install the necessary NuGet packages for RDLC:

```
Install-Package Microsoft.ReportingServices.ReportViewerControl.WebForms
```

Steps for Report Design

1. Choose a Reporting Tool:
   - Select a reporting tool that suits your needs, such as Crystal Reports, RDLC (Report Definition Language Client-side), or any other preferred tool.
2. Create a New Report:
   - Open your reporting tool and create a new report file.
   - Define the data source for the report, which will be the list of students fetched based on the selected course.
3. Design the Report Layout:
   - Header Section:
     - Add a title to the report, such as "Student List for Selected Course".
     - Include the course title and other relevant information.
   - Detail Section:
     - Add a table or grid to display the student information.
     - Define columns for StudentID, FirstName, LastName, and Grade.
   - Footer Section:
     - Optionally, add a footer with page numbers or other summary information.
4. Bind Data to the Report:
   - Connect the report to the data source.
   - Bind the fields in the report to the corresponding data fields.
5. Add Formatting and Styling:
   - Apply formatting to the report elements to improve readability.
   - Use fonts, colors, and borders to make the report visually appealing.
6. Preview and Test the Report:
   - Preview the report to ensure it displays the data correctly.
   - Test with different courses to verify the filtering and data binding.
7. Implement Print Functionality:
   - Add a button to your form to print the report.

- Use the reporting tool's API to handle the print operation.
- Example for RDLC:

```
private void PrintReport()
{
    LocalReport report = new LocalReport();
    report.ReportPath = "PathToYourReport.rdlc";
    report.DataSources.Add(new ReportDataSource("StudentDataSet", students));
    PrintDocument printDoc = new PrintDocument();
    printDoc.PrintPage += (sender, e) => {
        e.Graphics.DrawImage(report.Render("Image"), new Point(0, 0));
    };
    printDoc.Print();
}
```

3.Create the RDLC Report

- Add a New Report:
  - Right-click on your project in Solution Explorer, select Add > New Item.
  - Choose Report and name it `StudentReport.rdlc`.
  - Location: `~/Reports/StudentReport.rdlc`
- Design the Report Layout:
  - Open the `StudentReport.rdlc` file.
  - Use the Report Data pane to add a new dataset.
  - Define the dataset query to fetch student data based on the selected course:

```
SELECT s.StudentID, s.FirstName, s.LastName, e.Grade
FROM Students s
JOIN Enrollments e ON s.StudentID = e.StudentID
WHERE e.CourseID = @CourseID
```

  - Add a table to the report and bind the fields (`StudentID`, `FirstName`, `LastName`, `Grade`) to the corresponding columns.
- Bind Data to the Report:
  - Connect the report to the data source.
  - Bind the fields in the report to the corresponding data fields.
  - Example for RDLC:

```
<Table Name="StudentTable">
  <TableColumns>
    <TableColumn Width="2in"/>
    <TableColumn Width="2in"/>
    <TableColumn Width="2in"/>
    <TableColumn Width="1in"/>
  </TableColumns>
  <Header>
```

```
    <TableRows>
      <TableRow>
        <TableCells>
          <TableCell>
            <ReportItems>
              <Textbox Name="StudentIDHeader">
                <Value>Student ID</Value>
              </Textbox>
            </ReportItems>
          </TableCell>
          <TableCell>
            <ReportItems>
              <Textbox Name="FirstNameHeader">
                <Value>First Name</Value>
              </Textbox>
            </ReportItems>
          </TableCell>
          <TableCell>
            <ReportItems>
              <Textbox Name="LastNameHeader">
                <Value>Last Name</Value>
              </Textbox>
            </ReportItems>
          </TableCell>
          <TableCell>
            <ReportItems>
              <Textbox Name="GradeHeader">
                <Value>Grade</Value>
              </Textbox>
            </ReportItems>
          </TableCell>
        </TableCells>
      </TableRow>
    </TableRows>
  </Header>
  <Details>
    <TableRows>
      <TableRow>
        <TableCells>
          <TableCell>
            <ReportItems>
              <Textbox Name="StudentID">
                <Value>=Fields!StudentID.Value</Value>
              </Textbox>
            </ReportItems>
          </TableCell>
          <TableCell>
            <ReportItems>
              <Textbox Name="FirstName">
                <Value>=Fields!FirstName.Value</Value>
              </Textbox>
            </ReportItems>
          </TableCell>
          <TableCell>
            <ReportItems>
              <Textbox Name="LastName">
                <Value>=Fields!LastName.Value</Value>
              </Textbox>
```

```
            </ReportItems>
          </TableCell>
          <TableCell>
            <ReportItems>
              <Textbox Name="Grade">
                <Value>=Fields!Grade.Value</Value>
              </Textbox>
            </ReportItems>
          </TableCell>
        </TableCells>
      </TableRow>
    </TableRows>
  </Details>
</Table>
```

4. Create the Filter Form

- Design the Filter Form:
  - Create a new view named `Index.cshtml` for the filter form.
  - Location: `~/Views/Report/Index.cshtml`
  - Add a dropdown list to select the course and two buttons for Print Preview and Export to PDF:

```
@model YourNamespace.Models.CourseViewModel
<form method="post" action="/Report/Generate">
    <select asp-for="SelectedCourseID" asp-items="Model.Courses"></select>
    <button type="submit" name="action" value="preview">Print Preview</button>
    <button type="submit" name="action" value="export">Export to PDF</button>
</form>
```

5. Add a Tab with Action Link

- Modify the Layout:
  - Add a new tab in your layout file named `_Layout.cshtml` with an action link to the filter form.
  - Location: `~/Views/Shared/_Layout.cshtml`

```
<ul class="nav navbar-nav">
    <li>@Html.ActionLink("Student Report", "Index", "Report")</li>
</ul>
```

6. Controller Actions

- Create Controller Actions:
  - Create a controller named ReportController.cs to handle the report generation.
  - Location: `~/Controllers/ReportController.cs`
  - Fetch the list of courses and pass it to the view:

```
public class ReportController : Controller
{
```

```
    private readonly SchoolDBEntities _context;

    public ReportController(SchoolDBEntities context)
    {
        _context = context;
    }

    public IActionResult Index()
    {
        var model = new CourseViewModel
        {
            Courses = new SelectList(_context.Courses, "CourseID", "Title")
        };
        return View(model);
    }

    [HttpPost]
    public IActionResult Generate(int selectedCourseID, string action)
    {
        var reportPath = Path.Combine(HostingEnvironment.WebRootPath, "Reports",
"StudentReport.rdlc");
        var localReport = new LocalReport { ReportPath = reportPath };

        var students = _context.Enrollments
                               .Where(e => e.CourseID == selectedCourseID)
                               .Select(e => new
                               {
                                   e.Student.StudentID,
                                   e.Student.FirstName,
                                   e.Student.LastName,
                                   e.Grade
                               }).ToList();

        localReport.DataSources.Add(new ReportDataSource("StudentDataSet",
students));

        var reportBytes = localReport.Render("PDF");

        return File(reportBytes, "application/pdf", "StudentReport.pdf");
    }
}
```

7. View for Report Generation

- Create the View:
  - Create a view for the report generation form named `Index.cshtml`.
  - Location: `~/Views/Report/Index.cshtml`

```
@model YourNamespace.Models.CourseViewModel
@{
    ViewBag.Title = "Student Report";
}

<h2>Student Report</h2>

<form method="post" action="/Report/Generate">
    <div class="form-group">
        <label for="SelectedCourseID">Select Course:</label>
        <select asp-for="SelectedCourseID" asp-items="Model.Courses"
class="form-control"></select>
```

```
    </div>
    <button type="submit" name="action" value="preview" class="btn btn-
primary">Print Preview</button>
    <button type="submit" name="action" value="export" class="btn btn-
secondary">Export to PDF</button>
</form>
```

8. Run and Test

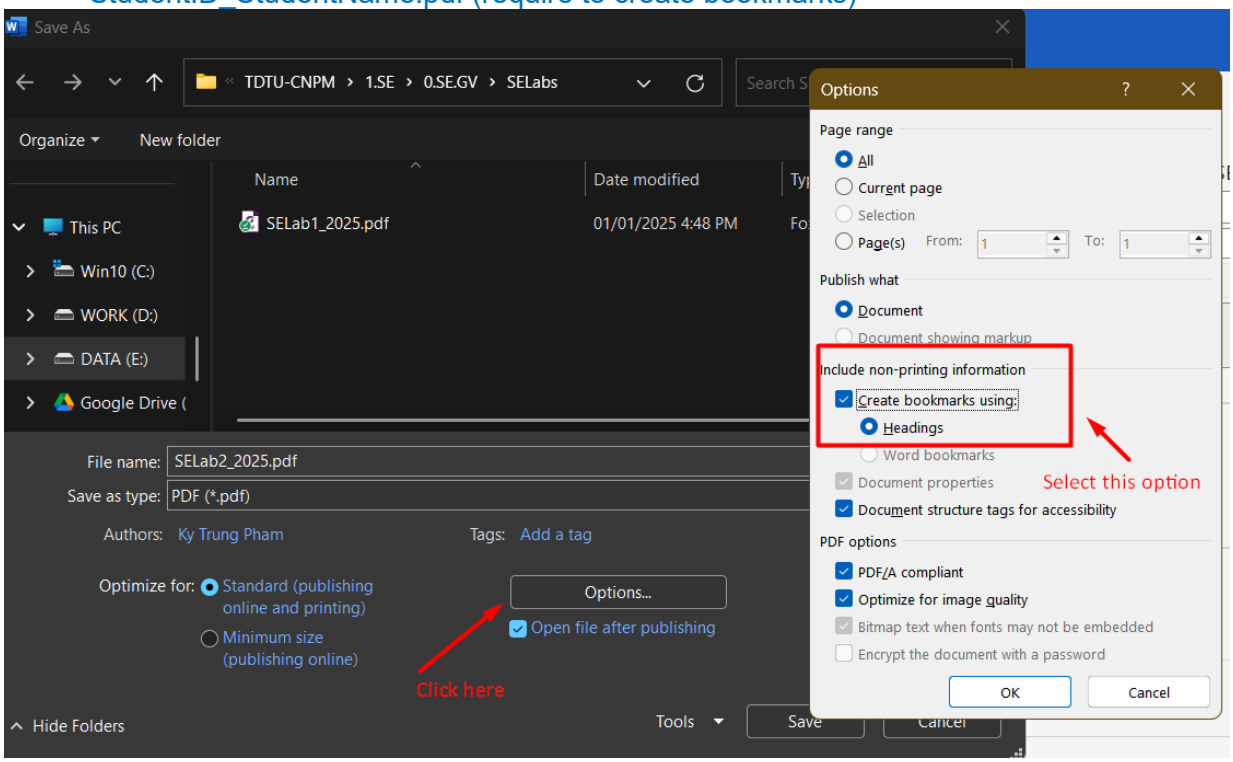
- Run Your Application:
  - Navigate to the report generation page, select a course, and generate the report.

3. Submit on eLearning: Summary Report

- Introduction: Briefly describe the purpose of the exercises.
- Exercises: List each exercise with a brief description of what you did and learned.
- Screenshots: Include the screenshots of every steps for each exercise and steps (Index each step with sequential number) and the output when press F5
- Conclusion: Summarize your overall learning experience and any challenges you faced.

Submit 2 files on eLearning:

- StudentID_StudentName.zip (contain Windows Web Project & .sql file) for both Class Activity and Homework.
- StudentID_StudentName.pdf (require to create bookmarks)



Notes: There may be a typo/mistake during creating this document. Please correct it by yourself.

Enjoy ☺ Email: tg_phamthaikytrung@tdtu.edu.vn