T.C

CUKUROVA UNIVERSITY
ENGINEERING AND ARCHITECTURE FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

GRADUATION THESIS

STUDENT MANAGEMENT/INFORMATION SYSTEM

By
Nezaket KAYA

Advisor
Associate Prof. Fatih ABUT

May, 2024
ADANA

CONTENTS

1. INTRODUCTION

In this report, the design and development processes of a Student Management and Information System developed using ASP.NET Core MVC are discussed in technical details.

a. **Purpose and Technical Scope of the Project**

The primary objective of this project is to architect and implement a web application that facilitates bidirectional data flow between educators and learners.

On the server side, the ASP.NET Core's Model-View-Controller (MVC) pattern was leveraged to provide educators with a suite of CRUD (Create, Read, Update, Delete) operations for comprehensive student data management.

In this project, Bootstrap was used to achieve a more aesthetic appearance and implement responsive design on the frontend. Furthermore, JavaScript was employed to perform client-side validation of user input, ensuring data integrity and enhancing user experience by validating form fields in real-time before submission to the server. And session management was utilized to enable users to access only their own information.

There are also 3 different layouts in the project: _Layout, AdminLayout and StudentLayout. _Layout is used for the login screen, AdminLayout for the Admin Panel and StudentLayout for the student pages.

We will delve into these aspects in detail in the subsequent sections.

b. **Technologies**

1. Front-End: Html, CSS, Javascript, Boostrap, jQuery

2. Back-End: C#, ASP.NET Core, MVC (Model-View-Controller)

3. Data Management: MS SQL Server, Entity Framework

4. Web Server: MS ISS

2. CREATING TABLES, DATABASE CONNECTION AND CONFIGURATIONS

Initially Model classes representing the data structures of the Student Management System were first added to the ASP.NET Core MVC project created in Visual Studio. In this way, the Code-First approach was used when adding a database to the project. And I created a student-advisor relationship by adding the AdvisorId property to my Student class. I have established this relationship so that each student can have an advisor and each advisor can have more than one relationship.

Below, you can see the entity classes and their respective properties.

```
public class Admin
{
    0 references
    public int Id { get; set; }
    4 references
    public string NameSurname { get; set; }
    4 references
    public DateTime DateOfBirth { get; set; }
    5 references
    public string Phone { get; set; }
    4 references
    public string Email { get; set; }
    4 references
    public string Address { get; set; }
    0 references
    public string Image { get; set; }
    4 references
    public string Password { get; set; }
}
```

```
public class Course
{
    0 references
    public int Id { get; set; }
    3 references
    public string Title { get; set; }
    3 references
    public string Day { get; set; }
    3 references
    public string Time { get; set; }
    0 references
    public ICollection<Student> Students { get; set; }
}
```

```
public class Student
{

    public int Id { get; set; }

    public string Number { get; set; }

    public string NameSurname { get; set; }

    public string Gender { get; set; }

    public DateTime DateOfBirth { get; set; }
    6 references
    public string Phone { get; set; }
    6 references
    public string Email { get; set; }
    6 references
    public string Address { get; set; }
    5 references
    public int AdvisorId { get; set; }
    3 references
    public Advisor Advisor { get; set; }
    5 references
    public string Password { get; set; }
}
```

```
public class Announcement
{
    1 reference
    public int Id { get; set; }
    3 references
    public DateTime Date { get; set; }
    5 references
    public string Title { get; set; }
    5 references
    public string Description { get; set; }
    14 references
    public string File { get; set; }
}
```

```
public class Advisor
{
    0 references
    public int Id { get; set; }
    4 references
    public string Name { get; set; }
    4 references
    public string Department { get; set; }
    1 reference
    public ICollection<Student> Students { get; set; }
}
```

The AppDbContext class comes with a tool called Entity Framework Core and allows us to operate with database tables. Each DbSet represents a database table.

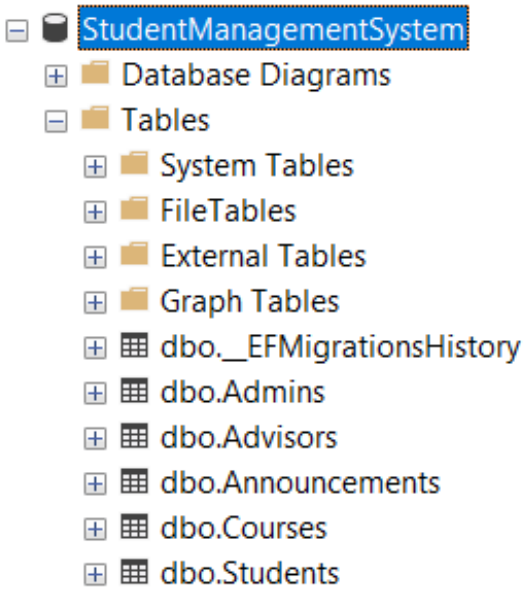
```
public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)
{
}
2 references
public DbSet<Admin> Admins { get; set; }
3 references
public DbSet<Announcement> Announcements { get; set; }
2 references
public DbSet<Course> Courses { get; set; }
4 references
public DbSet<Student> Students { get; set; }
3 references
public DbSet<Advisor> Advisors { get; set; }
0 references
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    modelBuilder.Entity<Advisor>()
        .HasMany(a => a.Students)
        .WithOne(s => s.Advisor)
        .HasForeignKey(s => s.AdvisorId);
}
```

After adding this class, a Connection String was added to the appsetting.json file to establish a database connection and the configurations were completed in the Program.cs file.

```
{
  "ConnectionStrings": {
    "SqlCon": "Data Source=DESKTOP-7R9PFD3\\SQLEXPRESSINS; Database = StudentManagementSystem;Integrated Security=True;Connect
    Timeout=30;Encrypt=False;Trust Server Certificate=False;Application Intent=ReadWrite;Multi Subnet Failover=False;"
  },
```

```
builder.Services.AddDbContext<AppDbContext>(options =>
{

    options.UseSqlServer(builder.Configuration.GetConnectionString("SqlCon"));

});
```

Afterwards, the migration was created and the tables were added to the database.



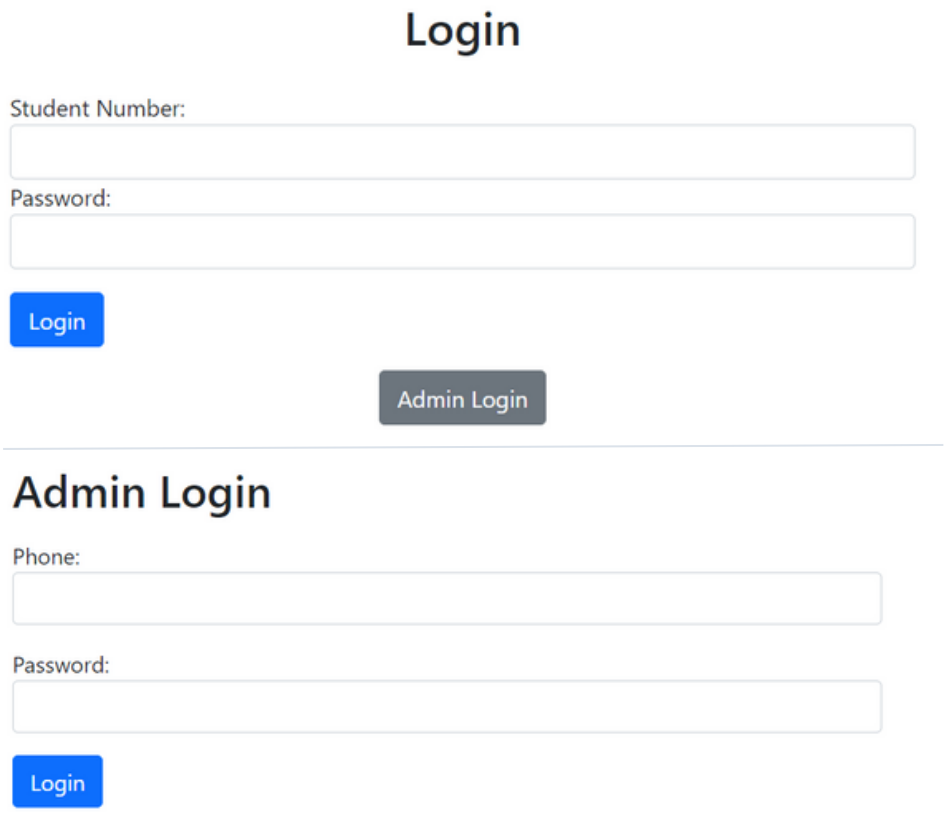
From this section onwards, the discussion will be based on application. The admin panel includes Admin Information, Announcement, Student Information, Course Program and Grades Management pages. In the student panel, the data created by the admin is displayed in a personalized way.

3. LOGIN PAGES

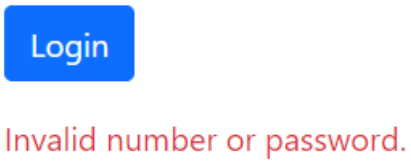
On the login pages, user information is checked and if the information is correct, the user is directed to the system. Students log in with their student number and password, and teachers log in with their phone number and password. In the code below, we set the default start screen for the application as the login page.

```
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Login}/{action=Login}/{id?}");
```

When the application is started, a login screen opens for students, and when the Admin Login button is clicked, they are directed to the Admin Login screen.



If the user information is entered incorrectly, a warning message will be displayed as follows.



This action in the Login Controller allows an student to log in using valid model data. If the information is correct, it stores the student session ID with Http.session and redirects to the student information page. Redirection to the Admin Login screen is also done on this page.

```
[HttpPost]
0 references
public IActionResult Login(string number, string password)
{
    var student = _context.Students.FirstOrDefault(s => s.Number == number && s.Password == password);
    if (student != null)
    {
        HttpContext.Session.SetInt32("StudentId", student.Id);
        return RedirectToAction("StudentInfo", "S_StudentInfo");
    }
    else
    {
        ViewBag.Error = "Invalid number or password.";
        return View();
    };
}
```

```
<form method="post" action="/Login/Login">
    <div class="form-group">
        <label for="number">Student Number:</label>
        <input type="text" class="form-control" id="number" name="number" required />
    </div>
    <div class="form-group">
        <label for="password">Password:</label>
        <input type="password" class="form-control" id="password" name="password" required />
    </div>
    <p></p>
    <button type="submit" class="btn btn-primary btn-block">Login</button>
</form>

@if (ViewBag.Error != null)
{
    <div class="text-danger mt-3">@ViewBag.Error</div>
}

<div class="text-center mt-3">
    <a href="/Login/LoginAdmin" class="btn btn-secondary">Admin Login</a>
</div>
```

Additionally, in order to use the Http.Session feature, the following code block must be added to the Program.cs file.

```
builder.Services.AddDistributedMemoryCache();

builder.Services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromMinutes(20);
    options.Cookie.HttpOnly = true;
    options.Cookie.IsEssential = true;
});
```

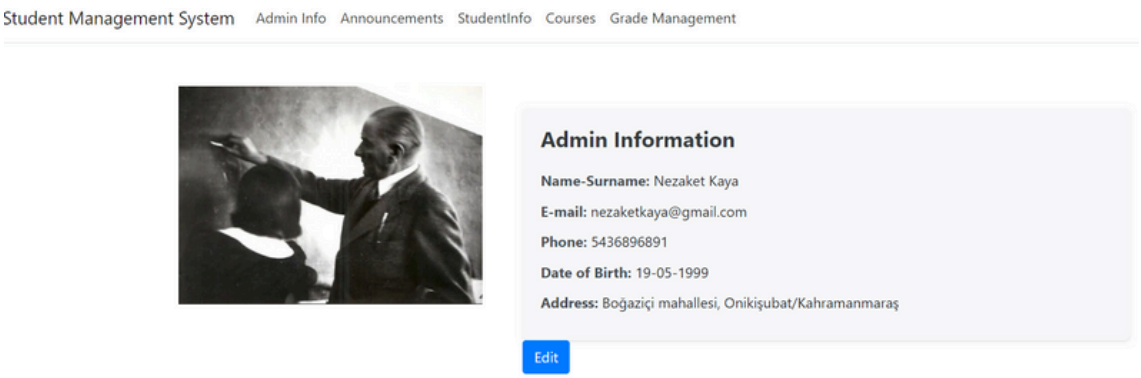
And finally the following action provides admin login.

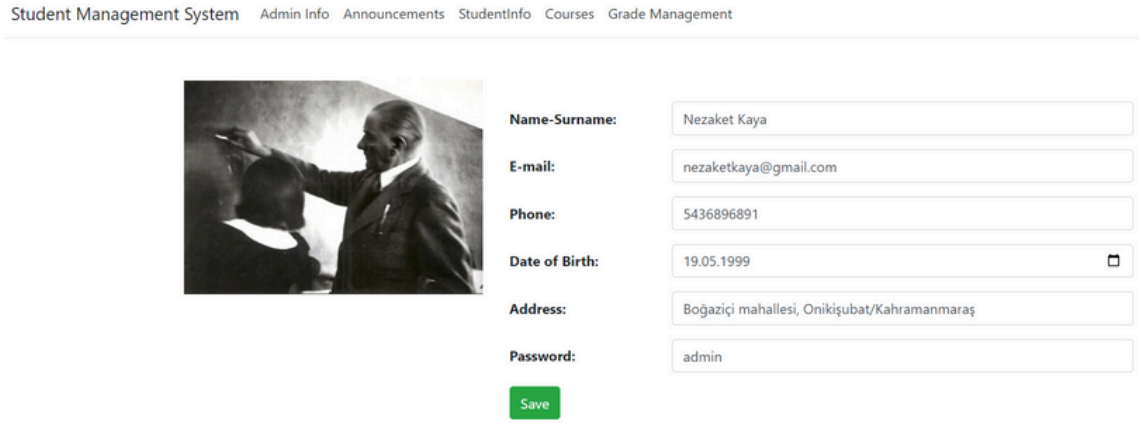
```
[HttpPost]
0 references
public IActionResult LoginAdmin(AdminLoginDto adminLoginDto)
{
    if (ModelState.IsValid)
    {
        var admin = _context.Admins.SingleOrDefault(u => u.Phone == adminLoginDto.Phone && u.Password == adminLoginDto.Password);

        if (admin != null)
        {
            return RedirectToAction("AdminInfo", "A_AdminInfo");
        }
        else
        {
            TempData["ErrorMessage"] = "Phone or password is wrong.";
        }
    }
    return View(adminLoginDto);
}
```

ADMIN PAGES

Admin Information:

When logged in as Admin, the Admin Information page opens first. On this page, when the **Edit** button is clicked, the information update screen appears and the Information area will be hidden. The updated information is saved to the database with the **Save** button.





This action retrieves the admin record from the database, passes this information to the view as ViewBag.AdminInfo, and returns the admin information page.

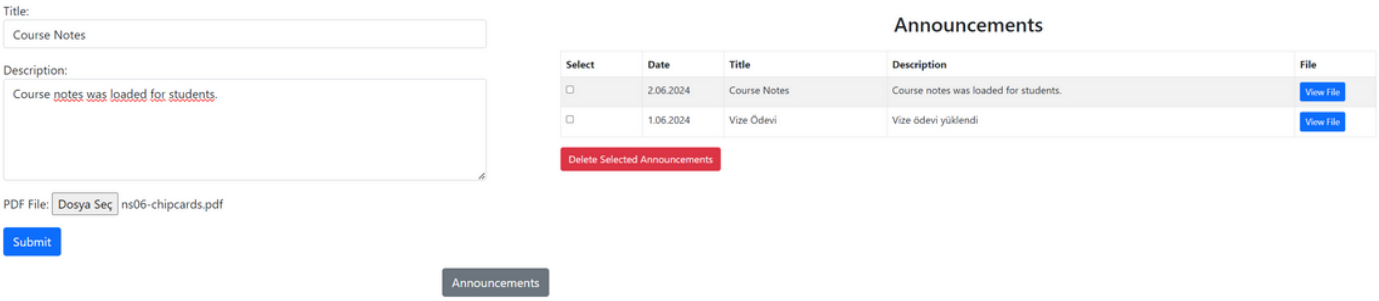
```
[HttpGet]
0 references
public IActionResult AdminInfo()
{
    var admin = _context.Admins.FromSqlRaw("SELECT TOP 1 * FROM Admins").FirstOrDefault();
    ViewBag.AdminInfo = admin;
    return View();
}
```

This code creates an HTML template to display admin information and adds an edit button.

```
<div class="container mt-4">
    <div class="info-card view-mode">
        <h4>Admin Information</h4>
        <p><span class="label">Name-Surname:</span> <span class="value">@ViewBag.AdminInfo.NameSurname</span></p>
        <p><span class="label">E-mail:</span> <span class="value">@ViewBag.AdminInfo.Email</span></p>
        <p><span class="label">Phone:</span> <span class="value">@ViewBag.AdminInfo.Phone</span></p>
        <p><span class="label">Date of Birth:</span> <span class="value">@ViewBag.AdminInfo.DateOfBirth.ToString("dd-MM-yyyy")</span></p>
        <p><span class="label">Address:</span> <span class="value">@ViewBag.AdminInfo.Address</span></p>
    </div>

    <button class="btn btn-primary edit-button">Edit</button>
</div>
```

Announcements:

On the announcements page, the admin can create announcements and perform actions such as adding assignments, including a pdf file. When the **Announcements** button is clicked, a list of added announcements appears, and on this screen, the announcement deletion process can be performed.



With the CreateAnnouncement action, the announcement and file data received from the form are processed. If the model is valid, the current time is set as the announcement date, the file is uploaded, and the announcement is saved to the database. In this action, **FileHelper** is used to add a pdf file.

```
[HttpPost]
0 references
public IActionResult CreateAnnouncement(Announcement announcement, IFormFile File)
{
    if (ModelState.IsValid)
    {
        announcement.Date = DateTime.Now;
        announcement.File = FileHelper.FileLoader(File);
        _context.Announcements.Add(announcement);
        _context.SaveChanges();
        return RedirectToAction("AnnouncementPage");
    }
    return View();
}
```

```
<div class="row">
    <div class="col-md-6">
        <form method="post" asp-action="CreateAnnouncement" enctype="multipart/form-data">
            <div class="form-group">
                <label asp-for="Title">Title:</label>
                <input type="text" asp-for="Title" class="form-control" required>
                <span asp-validation-for="Title" class="text-danger"></span>
            </div>
            <p></p>
            <div class="form-group">
                <label asp-for="Description">Description:</label>
                <textarea class="form-control" asp-for="Description" rows="5" required></textarea>
                <span asp-validation-for="Description" class="text-danger"></span>
            </div>
            <p></p>
            <div class="form-group">
                <label asp-for="File">PDF File:</label>
                <input type="file" asp-for="File" class="form-control-file">
                <span asp-validation-for="File" class="text-danger"></span>
            </div>
            <p></p>
            <button type="submit" class="btn btn-primary">Submit</button>
        </form>
    </div>

    <div class="text-center mt-3">
        <a href="/A_Announcement/GetAnnouncement" class="btn btn-secondary">Announcements</a>
    </div>
```

The GetAnnouncement action displays the announcements, and the DeleteAnnouncement action deletes the selected announcements.

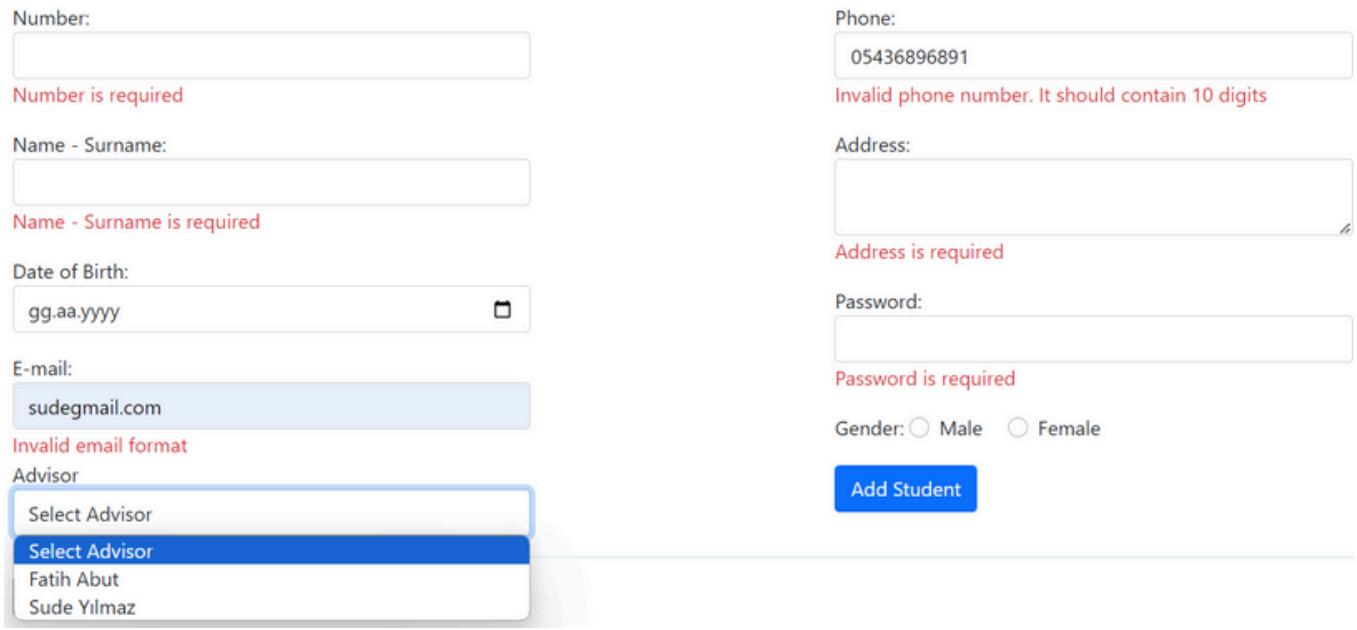
```
[HttpGet]
0 references
public IActionResult GetAnnouncement()
{
    var announcements = _context.Announcements
        .FromSqlRaw("SELECT * FROM Announcements ORDER BY Date DESC").ToList();

    foreach (var announcement in announcements)
    {
        if (!string.IsNullOrEmpty(announcement.File))
        {
            announcement.File = $"/pdf/{announcement.File}";
        }
    }

    return View(announcements);
}
```

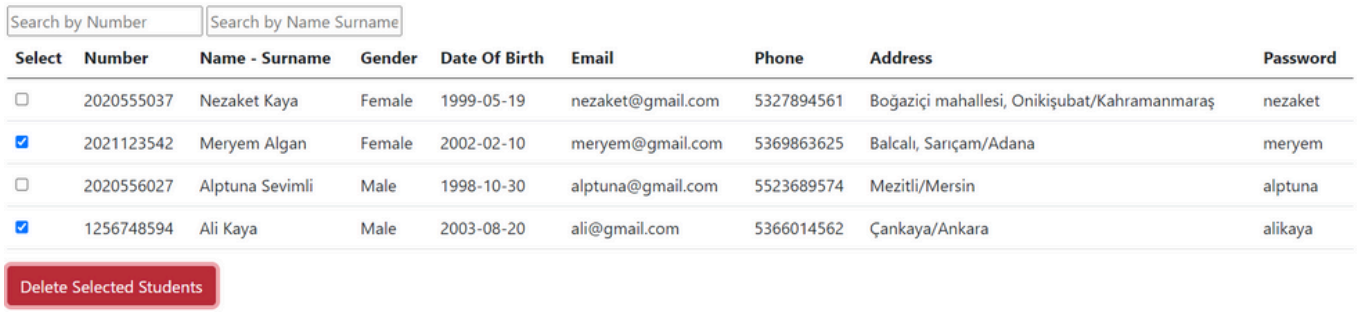
```
[HttpPost]
0 references
public IActionResult DeleteAnnouncement(List<int> selectedAnnouncement)
{
    if (selectedAnnouncement != null && selectedAnnouncement.Any())
    {
        foreach (var announcementId in selectedAnnouncement)
        {
            string sql = "DELETE FROM Announcements WHERE Id = @p0";

            _context.Database.ExecuteSqlRaw(sql, announcementId);
        }
    }
    return RedirectToAction("GetAnnouncement");
}
```

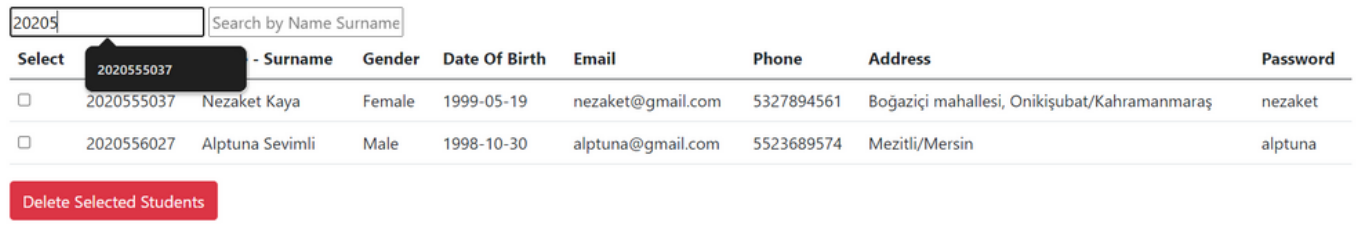
```
@if (Model != null && Model.Any())
{
    <form method="post" asp-action="DeleteAnnouncement">
        <table class="table table-striped table-bordered">
            <col style="width: 10%">
            <col style="width: 10%">
            <col style="width: 20%">
            <col style="width: 50%">
            <col style="width: 20%">
            <thead class="thead-dark">
                <tr>
                    <th scope="col">Select</th>
                    <th scope="col">Date</th>
                    <th scope="col">Title</th>
                    <th scope="col">Description</th>
                    <th scope="col">File</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var announcement in Model)
                {
                    <tr>
                        <td>
                            <input type="checkbox" name="selectedAnnouncement" value="@announcement.Id" />
                        </td>
                        <td>@announcement.Date.ToShortDateString()</td>
                        <td>@announcement.Title</td>
                        <td>@announcement.Description</td>
                        <td>
                            @if (!string.IsNullOrEmpty(announcement.File))
                            {
                                <a href="@announcement.File" class="btn btn-primary btn-sm" target="_blank">View File</a>
                            }
                            else
                            {
                                <span>No file</span>
                            }
                        </td>
                    </tr>
                }
            </tbody>
        </table>
        <button type="submit" class="btn btn-danger">Delete Selected Announcements</button>
    </form>
}
else
{
    <p class="text-center">No announcements available.</p>
}
```

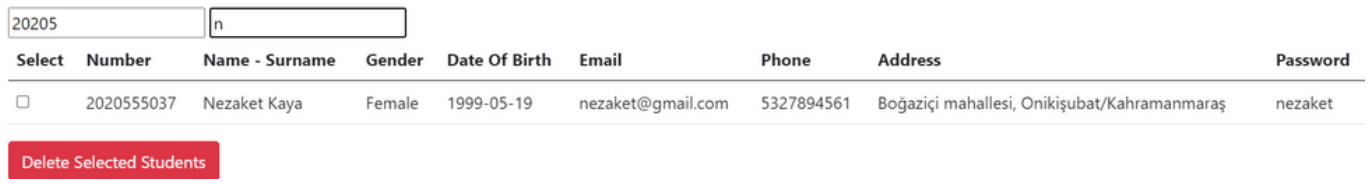
Student Information:

On this page, after entering student information in the relevant fields, the student is saved to the database with the Add Student button. JavaScript is used to display an instant error message if the inputs are not entered in the correct format. The students are listed in the table below and the selected students can be deleted using the checkBox. In addition, students can be filtered by number or name with search boxes.









In the **StudentInfo** action, all student records are retrieved from the database and converted into a list. This list of students is passed to the view via ViewBag. The **AddStudent** action adds the new student information to the database via SQL query if the information received from the form is valid. If successful, a success message is displayed and the student is directed to the information page. In case of an invalid model, an error message is displayed and the student is directed to the same page. And the **DeleteStds** action deletes the data selected with checkBoxes from the database.

```
[HttpGet]
0 references
public IActionResult StudentInfo()
{
    var advisors = _context.Advisors.FromSqlRaw("SELECT * FROM Advisors").ToList();
    ViewBag.Advisors = advisors;

    var students = _context.Students.FromSqlRaw("SELECT * FROM Students").ToList();
    ViewBag.StudentsInfo = students;

    return View();
}
```

```
[HttpPost]
0 references
public IActionResult AddStudent(Student newStudent)
{
    try
    {
        if (ModelState.IsValid)
        {
            string sql = @"
        INSERT INTO Students (Number, NameSurname, Gender, DateOfBirth, Phone, Email, Address, Password, AdvisorId)
        VALUES ({0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8})";

            _context.Database.ExecuteSqlRaw(
                sql,
                newStudent.Number,
                newStudent.NameSurname,
                newStudent.Gender,
                newStudent.DateOfBirth,
                newStudent.Phone,
                newStudent.Email,
                newStudent.Address,
                newStudent.Password,
                newStudent.AdvisorId
            );

            TempData["SuccessMessage"] = "Student has been successfully saved.";

            return RedirectToAction("StudentInfo");
        }
        else
        {
            TempData["ErrorMessage"] = "Student could not be saved. Please check your inputs.";

            return RedirectToAction("StudentInfo");
        }
    }
```

```
[HttpPost]
0 references
public IActionResult DeleteStds(List<int> selectedStudents)
{
    if (selectedStudents != null && selectedStudents.Any())
    {
        foreach (var studentId in selectedStudents)
        {
            string sql = "DELETE FROM Students WHERE Id = @p0";

            _context.Database.ExecuteSqlRaw(sql, studentId);
        }
    }
    return RedirectToAction("StudentInfo");
}
```

Student and Advisor:

On this page, students and advisors linked to each other appear in a single table.

Student and Advisor Information

Student Number	Student Name	Student Gender	Student Date of Birth	Student Phone	Student Email	Student Address	Advisor Name	Advisor Department
2020555037	Nezaket Kaya	Female	19.05.1999 00:00:00	5066322564	nezaket@gmail.com	Adana	Fatih Abut	Maths
2021123542	meryem algan	Female	10.02.2000 00:00:00	5436896892	meryem@gmail.com	Gaziantep	Sude Yılmaz	Biology

In this method of action, a list of students is taken along with their advisors. The Students table is joined to the Advisors table using a left join operation based on the AdvisorId foreign key relationship. This ensures that each student is associated with their own advisor.
After the SQL query is executed, the result is converted into a list of entities using the ToList() method. Finally, this list is transferred to the view for further processing and presentation. And the Student-Advisor table is created in the view with this information.

```
0 references
public IActionResult GetStudentAdvisor()
{
    var studentAdvisorList = _context.Students
        .Include(s => s.Advisor)
        .ToList();

    return View(studentAdvisorList);
}
```

```
@model List<Student>
@{
    Layout = "~/Views/Shared/AdminLayout.cshtml";
}
<h2>Student and Advisor Information</h2>

<table class="table">
    <thead>
        <tr>
            <th>Student Number</th>
            <th>Student Name</th>
            <th>Student Gender</th>
            <th>Student Date of Birth</th>
            <th>Student Phone</th>
            <th>Student Email</th>
            <th>Student Address</th>
            <th>Advisor Name</th>
            <th>Advisor Department</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var student in Model)
        {
            <tr>
                <td>@student.Number</td>
                <td>@student.NameSurname</td>
                <td>@student.Gender</td>
                <td>@student.DateOfBirth</td>
                <td>@student.Phone</td>
                <td>@student.Email</td>
                <td>@student.Address</td>
                <td>@student.Advisor.Name</td>
                <td>@student.Advisor.Department</td>
            </tr>
        }
    </tbody>
</table>
```

Client-side input validation and student filtering operations were performed with the following JavaScript codes.

```
document.addEventListener('DOMContentLoaded', function () {
    var form = document.getElementById('studentForm');

    form.querySelector('[name="Number"]').addEventListener('input', function (event) {
        var input = event.target;
        var validationMessage = input.nextElementSibling;
        if (input.value.trim() === '') {
            validationMessage.textContent = 'Number is required';
        } else {
            validationMessage.textContent = '';
        }
    });

    form.querySelector('[name="NameSurname"]').addEventListener('input', function (event) {
        var input = event.target;
        var validationMessage = input.nextElementSibling;
        if (input.value.trim() === '') {
            validationMessage.textContent = 'Name - Surname is required';
        } else {
            validationMessage.textContent = '';
        }
    });

    form.querySelectorAll('[name="Gender"]').forEach(function (radio) {
        radio.addEventListener('change', function () {
            var validationMessage = form.querySelector('[name="Gender"]').parentElement.querySelector('.text-danger');
            if (!form.querySelector('[name="Gender"]:checked')) {
                validationMessage.textContent = 'Gender is required';
            } else {
                validationMessage.textContent = '';
            }
        });
    });

    form.querySelector('[name="DateOfBirth"]').addEventListener('input', function (event) {
        var input = event.target;
        var validationMessage = input.nextElementSibling;
        if (input.value.trim() === '') {
            validationMessage.textContent = 'Date of Birth is required';
        } else {
            validationMessage.textContent = '';
        }
    });
```

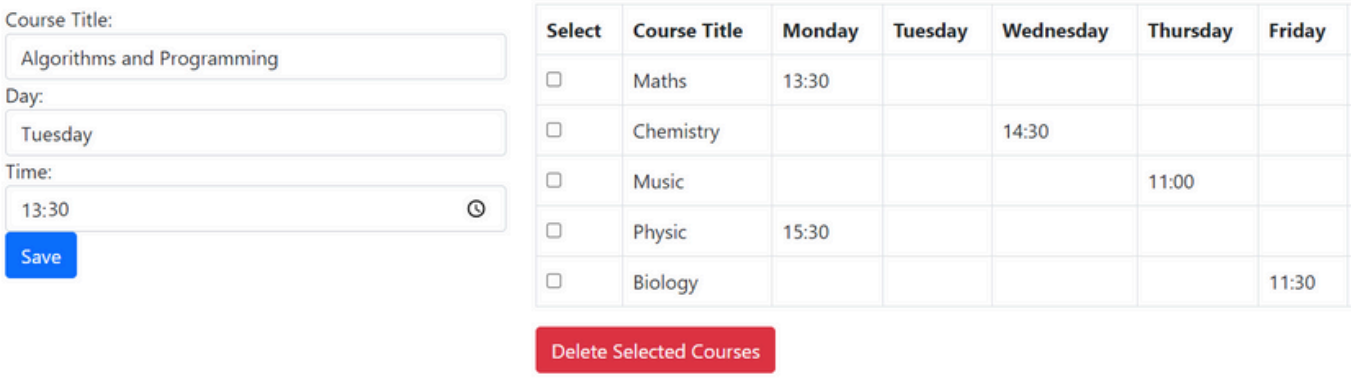
```
    form.querySelector('[name="Phone"]').addEventListener('input', function (event) {
        var input = event.target;
        var validationMessage = input.nextElementSibling;
        var phonePattern = /^\d{10}$/;
        if (!phonePattern.test(input.value.trim())) {
            validationMessage.textContent = 'Invalid phone number. It should contain 10 digits';
        } else {
            validationMessage.textContent = '';
        }
    });

    form.querySelector('[name="Address"]').addEventListener('input', function (event) {
        var input = event.target;
        var validationMessage = input.nextElementSibling;
        if (input.value.trim() === '') {
            validationMessage.textContent = 'Address is required';
        } else {
            validationMessage.textContent = '';
        }
    });

form.querySelector('[name="Password"]').addEventListener('input', function (event) {
    var input = event.target;
    var validationMessage = input.nextElementSibling;
    if (input.value.trim() === '') {
        validationMessage.textContent = 'Password is required';
    } else {
        validationMessage.textContent = '';
    }
});
```

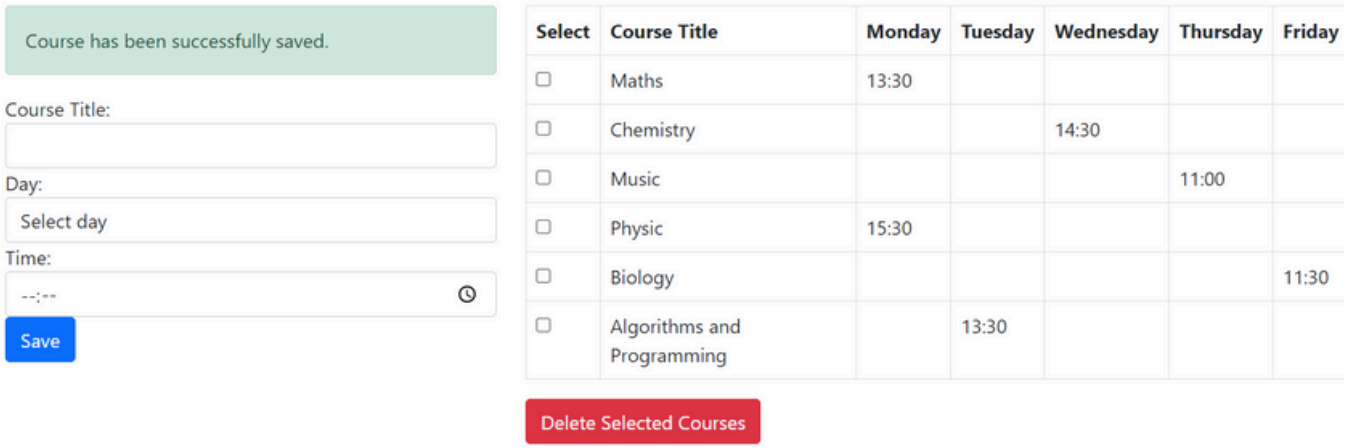
```
function searchByNumber() {
    var input, filter, table, tr, td, i, txtValue;
    input = document.getElementById("searchNumber");
    filter = input.value.toUpperCase();
    table = document.querySelector("table");
    tr = table.getElementsByTagName("tr");

    for (i = 0; i < tr.length; i++) {
        td = tr[i].getElementsByTagName("td")[1];
        if (td) {
            txtValue = td.textContent || td.innerText;
            if (txtValue.toUpperCase().indexOf(filter) > -1) {
                tr[i].style.display = "";
            } else {
                tr[i].style.display = "none";
            }
        }
```

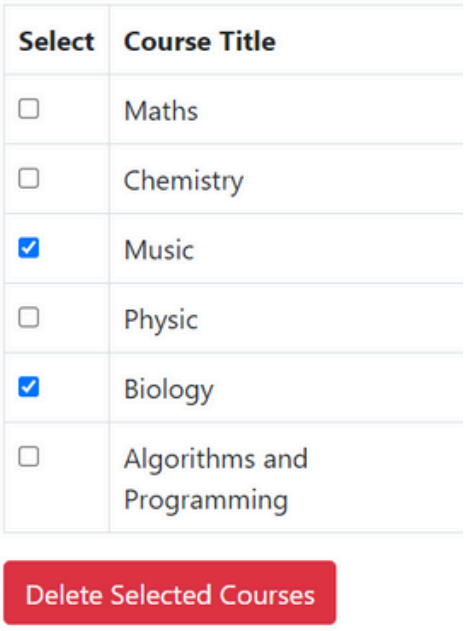
```
function searchByNameSurname() {
    var input, filter, table, tr, td, i, txtValue;
    input = document.getElementById("searchNameSurname");
    filter = input.value.toUpperCase();
    table = document.querySelector("table");
    tr = table.getElementsByTagName("tr");

    for (i = 0; i < tr.length; i++) {
        td = tr[i].getElementsByTagName("td")[2];
        if (td) {
            txtValue = td.textContent || td.innerText;
            if (txtValue.toUpperCase().startsWith(filter)) {
                tr[i].style.display = "";
            } else {
                tr[i].style.display = "none";
            }
```

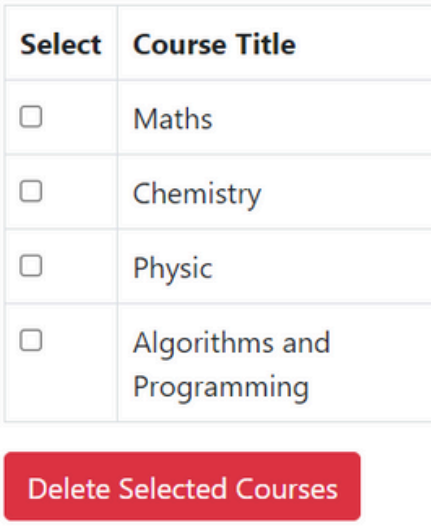
Course Management:

On this page, a new course is added to the database using the SQL command. After a successful addition, a redirect is made and a success message is displayed to the user. In case of an invalid model, an error message is displayed to the user and the same page is displayed again.









The CourseManagement method retrieves all courses from the database and returns them to a view to display as a list. This list is passed to the view via ViewBag.

The AddCourse method processes HTTP POST requests to add a course. The validity of the model is checked and if it is correct, a new course is added to the database using the SQL command. After a successful addition, a redirect is made and a success message is displayed to the user.

The DeleteCourse method is used to delete the selected courses from the database. The IDs of the selected courses are retrieved and a SQL DELETE command is run for each of them. As a result, after the courses are successfully deleted, a redirect is made and the user is returned to the course management page.

```
public IActionResult CourseManagement()
{
    var courses = _context.Courses.FromSqlRaw("SELECT * FROM Courses").ToList();
    ViewBag.Courses = courses;

    return View();
}
```

```
[HttpPost]
0 references
public IActionResult AddCourse(Course course)
{
    try
    {
        if (ModelState.IsValid)
        {
            string sql = @"
            INSERT INTO Courses (Title, Day, Time)
            VALUES ({0}, {1}, {2})";

            _context.Database.ExecuteSqlRaw(
               sql,
               course.Title,
               course.Day,
               course.Time
            );

            TempData["SuccessMessage"] = "Course has been successfully saved.";

            return RedirectToAction("CourseManagement");
        }
        else
        {
            TempData["ErrorMessage"] = "Course could not be saved. Please check your inputs.";

            return View(course);
        }
    }
    catch (Exception ex)
    {
        return View("Error");
    }
```

```
[HttpPost]
0 references
public IActionResult DeleteCourse(List<int> selectedCourses)
{
    if (selectedCourses != null && selectedCourses.Any())
    {
        foreach (var courseId in selectedCourses)
        {
            string sql = "DELETE FROM Courses WHERE Id = {0}";

            _context.Database.ExecuteSqlRaw(sql, courseId);
        }
        _context.SaveChanges();
    }
    return RedirectToAction("CourseManagement");
}
```

The form works with an HTTP POST request and sends data to the action named AddCourse. This action checks the model validity and if it is correct, adds a new course. In case of successful addition, a success message is displayed to the user. In the form, the select list element is used to select the day, and the time input type is used to select the hour.

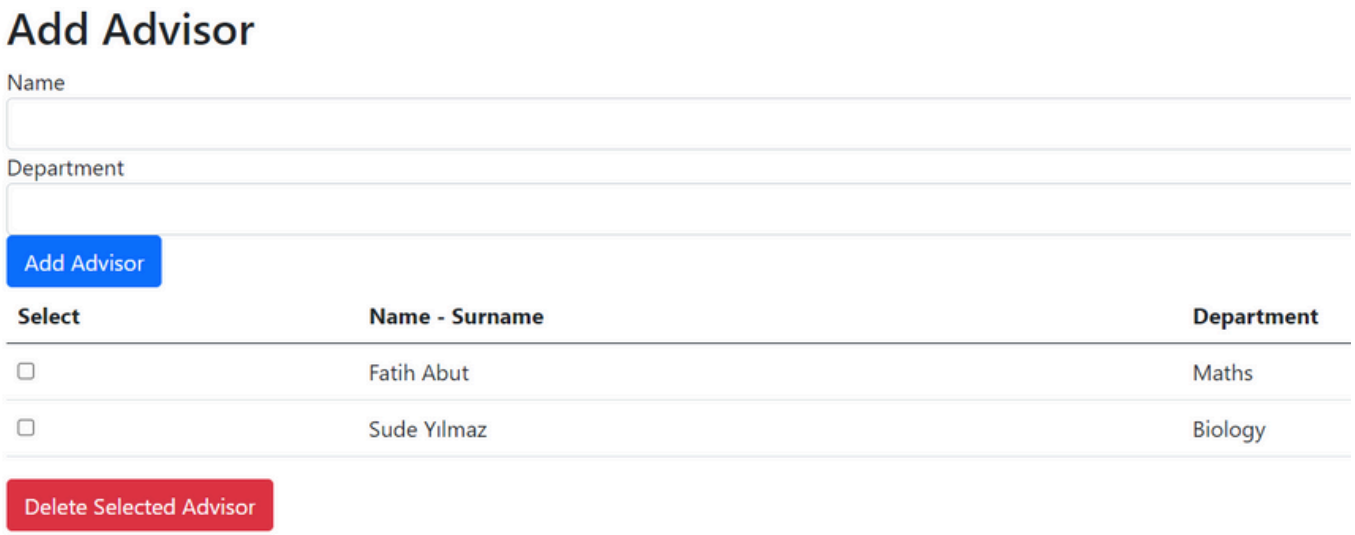
A table is used to list courses. The table shows all courses and their times on the relevant days.Also, there is a checkbox on the left side of each row next to the courses. The user can select the courses they want to delete and when they click on the "Delete Selected Courses" button, the selected courses will be deleted.

```
<form method="post" asp-action="AddCourse">
    <div class="form-group">
        <label asp-for="Title">Course Title:</label>
        <input asp-for="Title" type="text" class="form-control" required>
    </div>

    <div class="form-group">
        <label asp-for="Day">Day:</label>
        <select asp-for="Day" class="form-control" required>
            <option value="" selected disabled>Select day</option>
            <option value="Monday">Monday</option>
            <option value="Tuesday">Tuesday</option>
            <option value="Wednesday">Wednesday</option>
            <option value="Thursday">Thursday</option>
            <option value="Friday">Friday</option>
            <option value="Saturday">Saturday</option>
            <option value="Sunday">Sunday</option>
        </select>
    </div>

    <div class="form-group">
        <label asp-for="Time">Time:</label>
        <input asp-for="Time" type="time" class="form-control" required>
    </div>

    <button type="submit" class="btn btn-primary">Save</button>
</form>
</div>
```

```
<div class="col-md-8">
    @{

        var days = new List<string> { "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday" };
    }

    <form method="post" asp-action="DeleteCourse">
        <table class="table table-bordered">
            <thead>
                <tr>
                    <th>Select</th>
                    <th>Course Title</th>
                    @foreach (var day in days)
                    {
                        <th>@day</th>
                    }
                </tr>
            </thead>
            <tbody>
                @foreach (var course in ViewBag.Courses)
                {
                    <tr>

                        <td>
                            <input type="checkbox" name="selectedCourses" value="@course.Id" />
                        </td>
                        <td>@course.Title</td>
                        @foreach (var day in days)
                        {
                            var courseTime = "";

                            foreach (var courseP in ViewBag.Courses)
                            {
                                if (courseP.Day == day && courseP.Title == course.Title)
                                {
                                    courseTime = courseP.Time;
                                    break;
                                }
                            }
                            <td>@courseTime</td>
                        }
                    </tr>
                }
            </tbody>
        </table>
        <p></p>
        <button type="submit" class="btn btn-danger">Delete Selected Courses</button>
    </form>
```

Advisor:

On this page, the Advisor is added and the current advisor information along with the added advisor is displayed in a table. After selection is made with CheckBox, selected consultants can be deleted.



In Advisor action, all advisors are fetched from the database using a raw SQL query. The result is then converted to a list of Advisor entities using the ToList() method. Once the advisor list is received, it is transferred to the relevant view with ViewBag.Advisors.

```
[HttpGet]
0 references
public IActionResult Advisor()
{
    var advisors = _context.Advisors.FromSqlRaw("SELECT * FROM Advisors").ToList();
    ViewBag.Advisors = advisors;

    return View();
}
```

This HTTP POST action method "AddAdvisor" is used to add incoming advisor data to the database. The method checks the validity of the incoming model with ModelState.IsValid. If the model is valid, the consultant data is added to the database and the changes are saved.

```
[HttpPost]
0 references
public IActionResult AddAdvisor(Advisor advisor)
{
    if (ModelState.IsValid)
    {
        _context.Advisors.Add(advisor);
        _context.SaveChanges();

        return RedirectToAction("Advisor");
    }
    return View(advisor);
```

```
<form asp-action="AddAdvisor" method="post">
    <div class="form-group">
        <label asp-for="Name" class="control-label">Name</label>
        <input asp-for="Name" class="form-control" />
        <span asp-validation-for="Name" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Department" class="control-label">Department</label>
        <input asp-for="Department" class="form-control" />
        <span asp-validation-for="Department" class="text-danger"></span>
    </div>
    <div class="form-group">
        <button type="submit" class="btn btn-primary">Add Advisor</button>
    </div>
</form>

<form method="post" asp-action="DeleteAdvisor">
```

This HTTP POST action method "DeleteAdvisor" performs the function of deleting advisors from the database based on the incoming list of advisor ID numbers.

```
[HttpPost]
0 references
public IActionResult DeleteAdvisor(List<int> selectedAdvisors)
{
    if (selectedAdvisors != null && selectedAdvisors.Any())
    {
        foreach (var advisorId in selectedAdvisors)
        {
            string sql = "DELETE FROM Advisors WHERE Id = @p0";

            _context.Database.ExecuteSqlRaw(sql, advisorId);
        }
    }
    return RedirectToAction("Advisor");
}
```

```
<form method="post" asp-action="DeleteAdvisor">

<table class="table">
    <thead>
        <tr>
            <th>Select</th>
            <th>Name - Surname</th>
            <th>Department</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var advisors in ViewBag.Advisors)
        {
            <tr>
                <td>
                    <input type="checkbox" name="selectedAdvisors" value="@advisors.Id" />
                </td>

                    <td>@advisors.Name</td>
                    <td>@advisors.Department</td>
            </tr>
        }
    </tbody>
</table>
<button type="submit" class="btn btn-danger">Delete Selected Advisor</button>
</form>
```

STUDENT PAGES

Student Information:

When logged in as Student, the Student Information page opens first. On this page, the student displays his/her personal information.

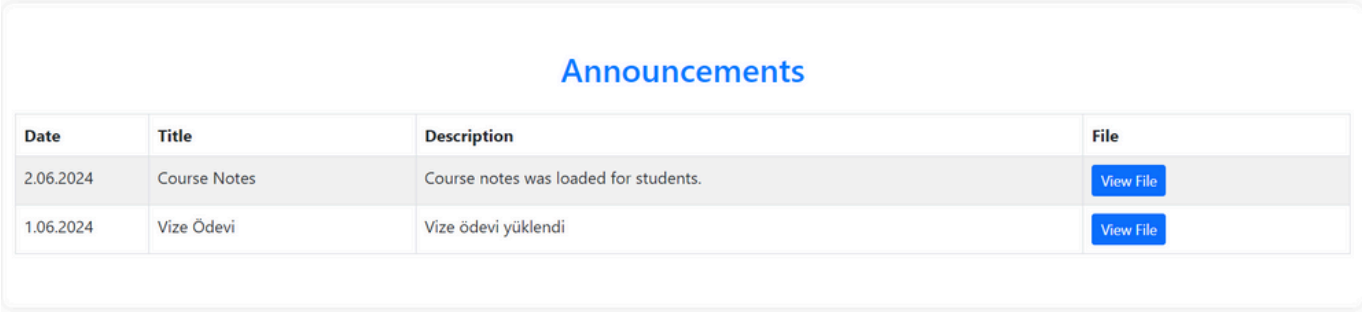
StudentInfo action is used to display student information. In the Login step, the session ID recorded with HttpContext.Session.Set is retrieved. If there is no student ID in the session, the user is directed to the login page. If the student ID is present, the student with this ID is found from the database and sent to the relevant view. If the student is not found or there is no student ID in the session, the user is directed to the login page.

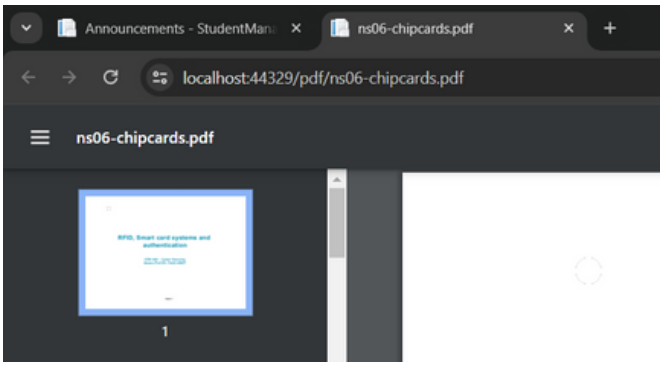
```
public class S_StudentInfoController : Controller
{
    private readonly AppDbContext _context;

    0 references
    public S_StudentInfoController(AppDbContext context)
    {
        _context = context;
    }

    [HttpGet]
    0 references
    public IActionResult StudentInfo()
    {
        var studentId = HttpContext.Session.GetInt32("StudentId");
        if (studentId == null)
        {
            return RedirectToAction("Login", "Login");
        }

        var student = _context.Students.FirstOrDefault(s => s.Id == studentId);
        if (student != null)
        {
            return View(student);
        }
       return RedirectToAction("Login", "Login");
    }
}
```

```
@{
    Layout = "~/Views/Shared/StudentLayout.cshtml";
}

@{
    ViewBag.Title = "StudentInfo";
}

<h2>Student Information</h2>

<div class="card">
    <div class="card-body">
        <div class="table-responsive">
            <table class="table table-bordered table-striped">
                <tbody>
                    <tr>
                        <th scope="row" style="width: 30%">Email</th>
                        <td>@Model.Number</td>
                    </tr>
                    <tr>
                        <th scope="row">Name</th>
                        <td>@Model.NameSurname</td>
                    </tr>
                    <tr>
                        <th scope="row">Gender</th>
                        <td>@Model.Gender</td>
                    </tr>
                    <tr>
                        <th scope="row">Date of Birth</th>
                        <td>@Model.DateOfBirth.ToShortDateString()</td>
                    </tr>
                    <tr>
                        <th scope="row">Phone</th>
                        <td>@Model.Phone</td>
                    </tr>
                    <tr>
                        <th scope="row">Email</th>
                        <td>@Model.Email</td>
                    </tr>
                    <tr>
                        <th scope="row">Address</th>
                        <td>@Model.Address</td>
                    </tr>
                </tbody>
            </table>
        </div>
    </div>
</div>
```

Announcements:

On this page, the student can view the announcements and assignments added by the teacher in chronological order, if a pdf file is added, the student can open it.





In the Announcements action, all announcements are retrieved from the database and sorted by date in descending order. Then, if the file field for each announcement is not empty and the file path is specified, the file path is prefixed with "/pdf/". Finally, these processed announcements are forwarded to the view and presented to the user. This allows users to view the announcements sorted by date and access the files if available.

```
[HttpGet]
0 references
public IActionResult Announcements()
{
    var announcements = _context.Announcements.FromSqlRaw("SELECT * FROM Announcements ORDER BY Date DESC").ToList();

    foreach (var announcement in announcements)
    {
        if (!string.IsNullOrEmpty(announcement.File))
        {
            announcement.File = $"/pdf/{announcement.File}";
        }
    }

    return View(announcements);
}
```

In the Announcements view file, a table is created where the announcements are displayed by adjusting the width ratio of each column.

When the View File button added to the last column is clicked, the pdf file attached to the announcement is displayed.

```
<table class="table table-striped table-bordered">
    <col style="width: 10%">
    <col style="width: 20%">
    <col style="width: 50%">
    <col style="width: 20%">
    <thead class="thead-dark">
        <tr>
            <th scope="col">Date</th>
            <th scope="col">Title</th>
            <th scope="col">Description</th>
            <th scope="col">File</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var announcement in Model)
        {
            <tr>
                <td>@announcement.Date.ToShortDateString()</td>
                <td>@announcement.Title</td>
                <td>@announcement.Description</td>
                <td>
                    @if (!string.IsNullOrEmpty(announcement.File))
                    {
                        <a href="@announcement.File" class="btn btn-primary btn-sm" target="_blank">View File</a>
                    }
                    else
                    {
                        <span>No file</span>
                    }
                </td>
            </tr>
        }
    </tbody>
</table>
```

Course Program:

On this page, students view the course schedule created by the teacher.

Course Title	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Maths	13:30						
Chemistry			14:30				
Physic	15:30						
Algorithms and Programming		13:30					

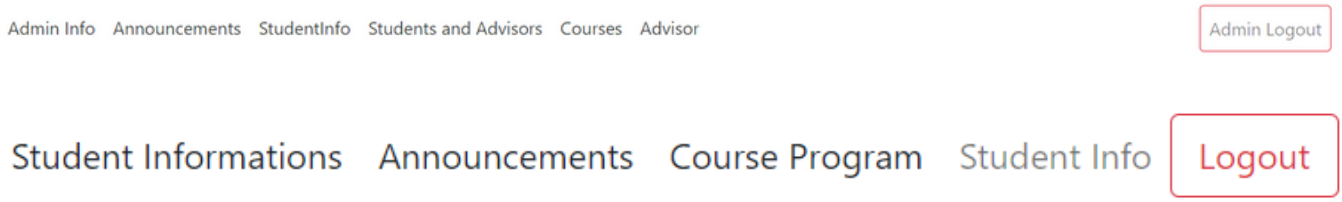
In the CourseProgram action, the courses are retrieved from the database as a list with the SQL query and sent to the view file with the ViewBag. Then, a course schedule table for students is created in the view file.

```
0 references
public IActionResult CourseProgram()
{
    var courses = _context.Courses.FromSqlRaw("SELECT * FROM Courses").ToList();
    ViewBag.Courses = courses;

    return View();
}
```

```
@{
    var days = new List<string> { "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday" };
}

    <div class="row justify-content-center">
        <div class="col-md-8">
            <table class="table table-striped">
                <thead class="thead-dark">
                    <tr>
                        <th>Course Title</th>
                        @foreach (var day in days)
                        {
                            <th>@day</th>
                        }
                    </tr>
                </thead>
                <tbody>
                    @foreach (var course in ViewBag.Courses)
                    {
                        <tr>
                            <td>@course.Title</td>
                            @foreach (var day in days)
                            {
                                var courseTime = "";

                                foreach (var courseP in ViewBag.Courses)
                                {
                                    if (courseP.Day == day && courseP.Title == course.Title)
                                    {
                                        courseTime = courseP.Time;
                                        break;
                                    }
                                }
                                <td>@courseTime</td>
                            }
                        </tr>
                    }
                </tbody>
            </table>
        </div>
    </div>
```

LOG OUT

Log out can be done with the buttons added to the navbar menu.



Two separate actions have been added to the LoginController for student and admin. And for this, a button that directs you to the Login page has been added to the navbar menu.

```
0 references
public IActionResult Logout()
{
    HttpContext.Session.Remove("StudentId");
    return RedirectToAction("Login", "Login");
}


0 references
public IActionResult LogoutAdmin()
{
    return RedirectToAction("LoginAdmin", "Login");
}
```

```
<ul class="navbar-nav mr-auto">
    @if (Context.Session.GetInt32("StudentId") != null)
    {
        <li class="nav-item">
            <a class="nav-link" href="@Url.Action("StudentInfo", "S_StudentInfo")"></a>
        </li>
    }
</ul>
<ul class="navbar-nav ml-auto">
    @if (Context.Session.GetInt32("StudentId") != null)
    {
        <li class="nav-item">
            <form method="post" asp-action="Logout" asp-controller="Login">
                <button type="submit" class="btn btn-outline-danger ml-2">Logout</button>
            </form>
        </li>
    }
</ul>
```

```
<ul class="navbar-nav ml-auto">
    <li class="nav-item">
        <a class="nav-link btn btn-outline-danger ml-2" href="@Url.Action("LogoutAdmin", "Login")">Admin Logout</a>
    </li>
</ul>
```

7. CONCLUSION

As a result, a Student Management System project implemented with general CRUD operations was created with ASP.NET Core MVC. With this project, teachers can add students, course schedules and announcements, and students can also view these data. In this way, a web application that can be used both ways was created.