PROGRAMMING 2B: PORTFOLIO OF EVIDENCE

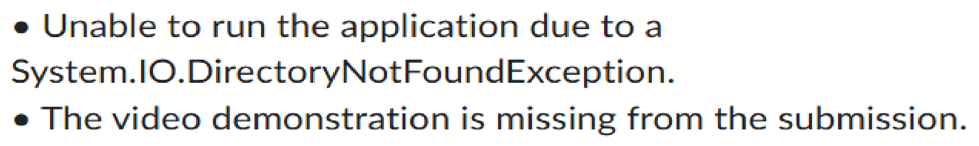
ST10273397

Nicholas Wolfaardt

GitHub Link: https://github.com/VCCT-PROG6212-2025-G1/ST10273397_PROG6212POE.git

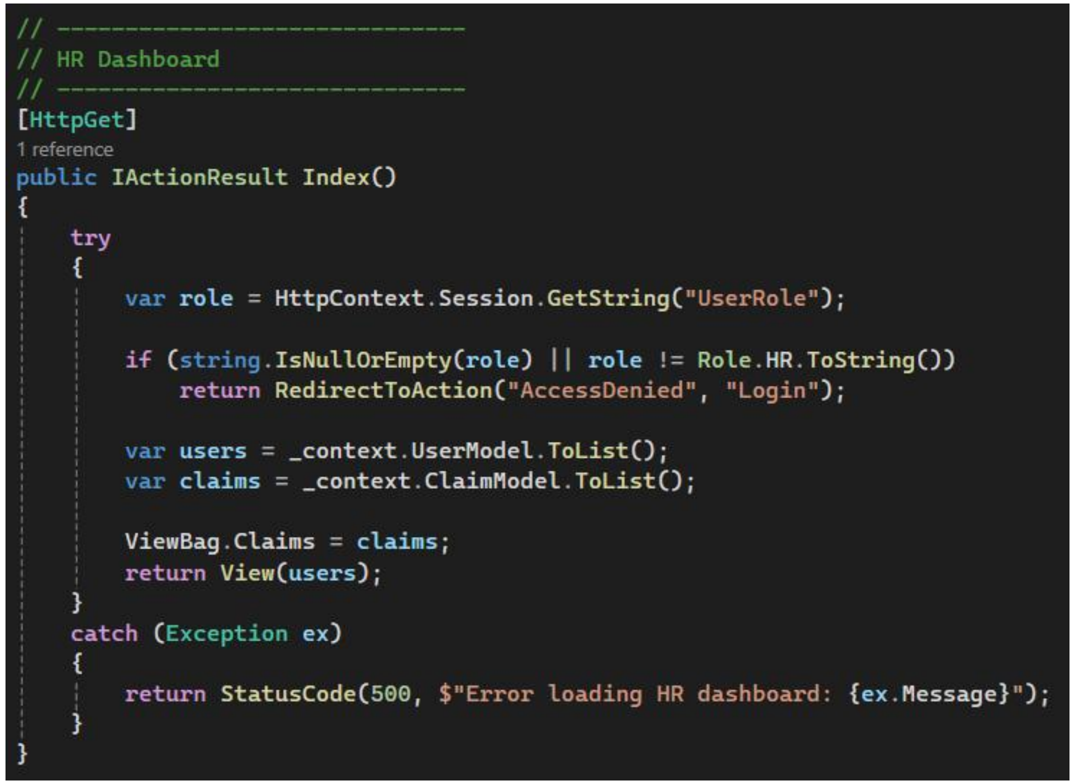
YouTube Link: https://youtu.be/8ewq0JXQozg

Lecturer Feedback



• Unable to run the application due to a System.IO.DirectoryNotFoundException.
• The video demonstration is missing from the submission.

- Application Now Runs.
- And If the Application Fails, a video is now provided as well.

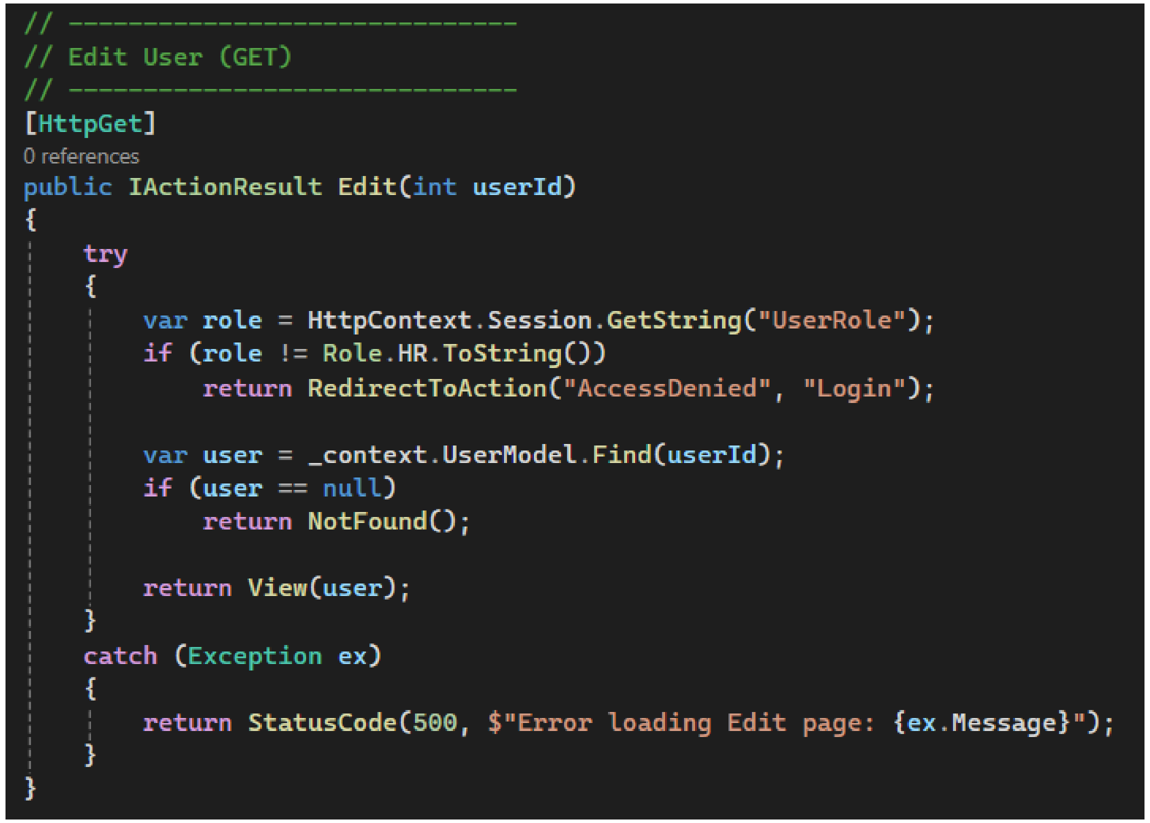


```
// -------------------------------
// HR Dashboard
// -------------------------------
[HttpGet]
1 reference
public IActionResult Index()
{
    try
    {
        var role = HttpContext.Session.GetString("UserRole");

        if (string.IsNullOrEmpty(role) || role != Role.HR.ToString())
            return RedirectToAction("AccessDenied", "Login");

        var users = _context.UserModel.ToList();
        var claims = _context.ClaimModel.ToList();

        ViewBag.Claims = claims;
        return View(users);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Error loading HR dashboard: {ex.Message}");
    }
}
```

HR Dashboard/Index

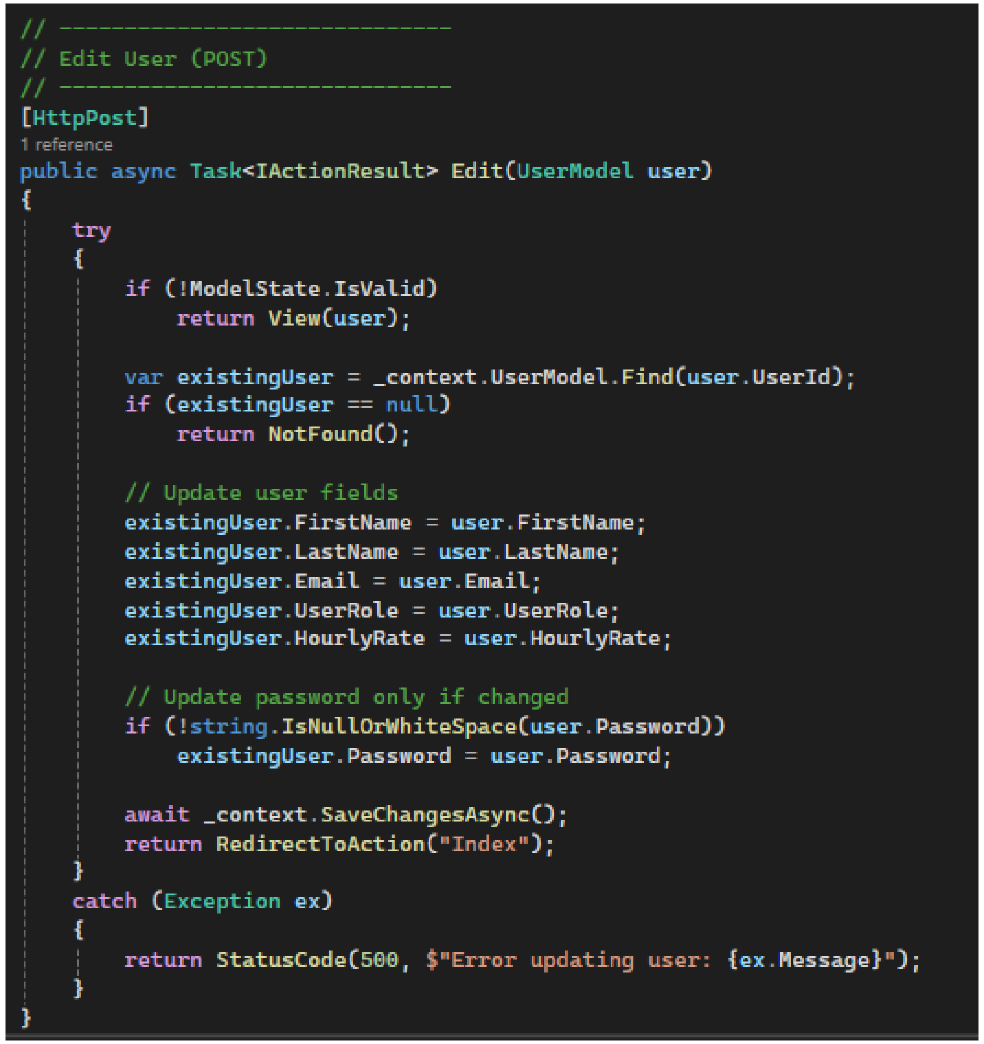
This method is the system's main entry point for all HR-related content. The Index() method is in charge of confirming that an HR employee has authorisation to view the dashboard when they log into the platform. To make sure that only someone with the HR role can view employee data or claims, it verifies the user's session role. After verification, it pulls all claims and registered users from the database and makes them accessible to the view. This establishes a central hub where HR can quickly monitor all of the company's workers and claims. Essentially, this approach provides the entire dataset required for the dashboard to operate while also establishing a secure gateway that guarantees sensitive information is only accessed by authorised staff.

Edit (Get) – Load the Edit page



```
// -----------------------------
// Edit User (GET)
// -----------------------------
[HttpGet]
0 references
public IActionResult Edit(int userId)
{
    try
    {
        var role = HttpContext.Session.GetString("UserRole");
        if (role != Role.HR.ToString())
            return RedirectToAction("AccessDenied", "Login");

        var user = _context.UserModel.Find(userId);
        if (user == null)
            return NotFound();

        return View(user);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Error loading Edit page: {ex.Message}");
    }
}
```

The HR staff can access a completely pre-loaded editing interface for any user in the system by using this GET method. Once the HR role has been verified, it uses the selected user's unique ID to search the database for them. To update the data without having to start from scratch, the procedure prepares the user's current details if they already exist and displays them in the view. Because it expedites the updating process and minimises data entry errors, this is a crucial usability aspect. By providing a smooth, safe, and effective means of accessing employee data for editing, the approach basically supports the HR workflow. Additionally, it creates the conditions for precise and controlled data alteration.

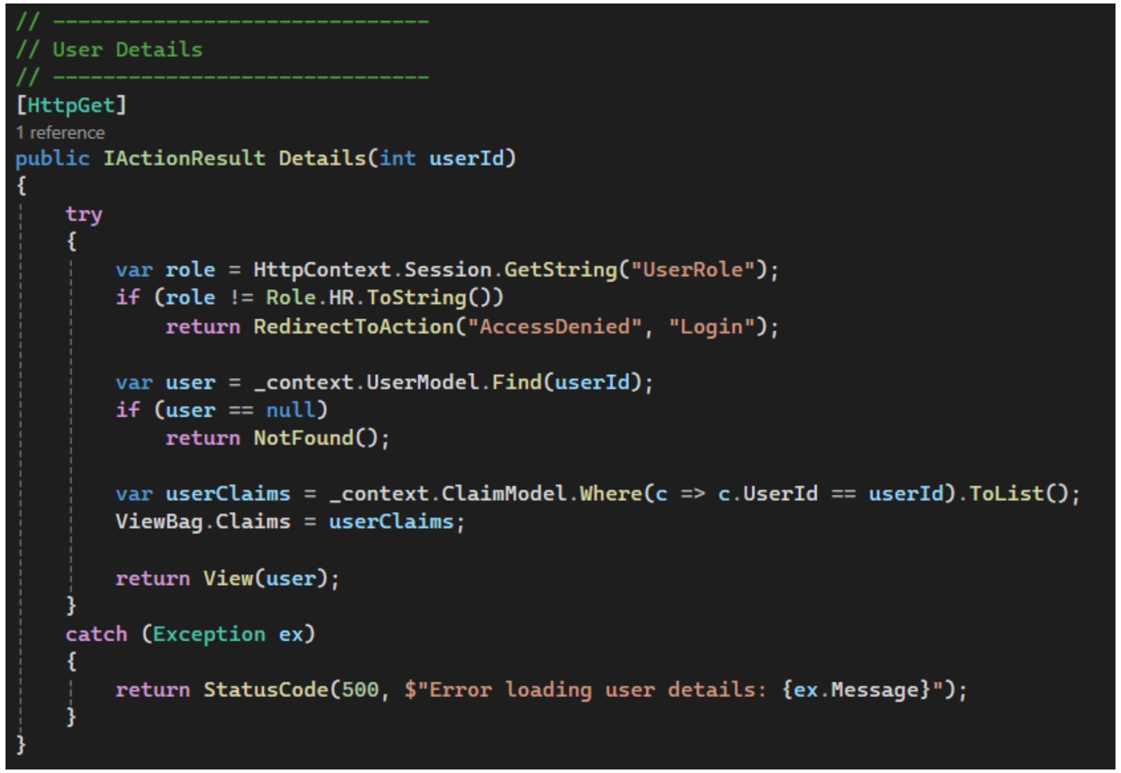


```
// ------------------------------
// Edit User (POST)
// ------------------------------
[HttpPost]
1 reference
public async Task<IActionResult> Edit(UserModel user)
{
    try
    {
        if (!ModelState.IsValid)
            return View(user);

        var existingUser = _context.UserModel.Find(user.UserId);
        if (existingUser == null)
            return NotFound();

        // Update user fields
        existingUser.FirstName = user.FirstName;
        existingUser.LastName = user.LastName;
        existingUser.Email = user.Email;
        existingUser.UserRole = user.UserRole;
        existingUser.HourlyRate = user.HourlyRate;

        // Update password only if changed
        if (!string.IsNullOrWhiteSpace(user.Password))
            existingUser.Password = user.Password;

        await _context.SaveChangesAsync();
        return RedirectToAction("Index");
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Error updating user: {ex.Message}");
    }
}
```

Edit (Post) – Save Updated User Data

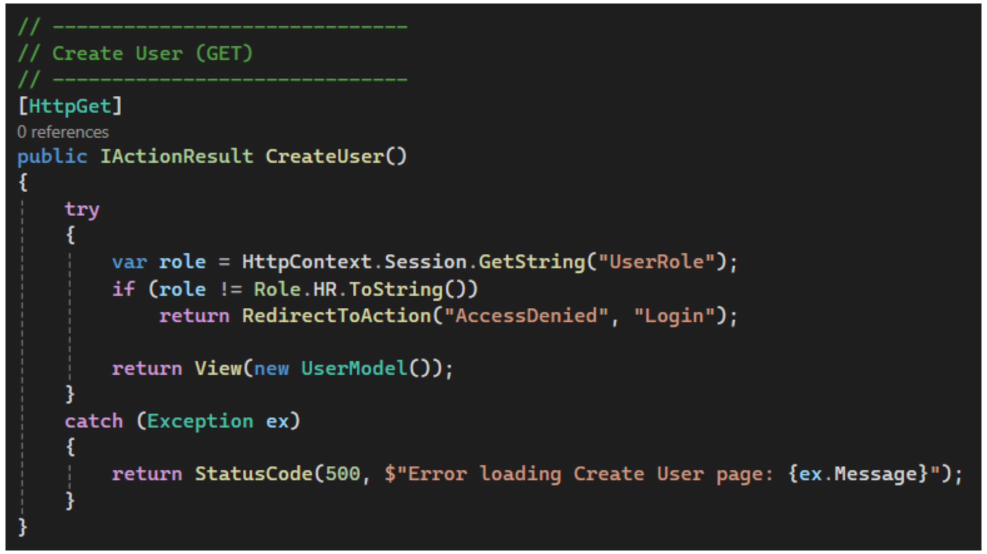
After HR submits the edit form, this function is in charge of actually applying changes to the database. The revised form finds the current database record linked to that user and re-validates the input once it is posted. In order to guarantee that the system consistently displays the most current version of that employee's profile, it then updates the relevant fields, including first name, last name, email, hourly rate, and role. In order to minimise unintentional overwrites, a crucial aspect of this approach is that the password is only changed when the HR staff purposefully enters a new one. The procedure saves the edits asynchronously and reroutes HR to the dashboard once all modifications have been made. This approach is essential because it manages data integrity, guards against unintentional data loss, and guarantees that all employee data is updated in a secure and regulated manner.

Details – User Profile & Claims Overview

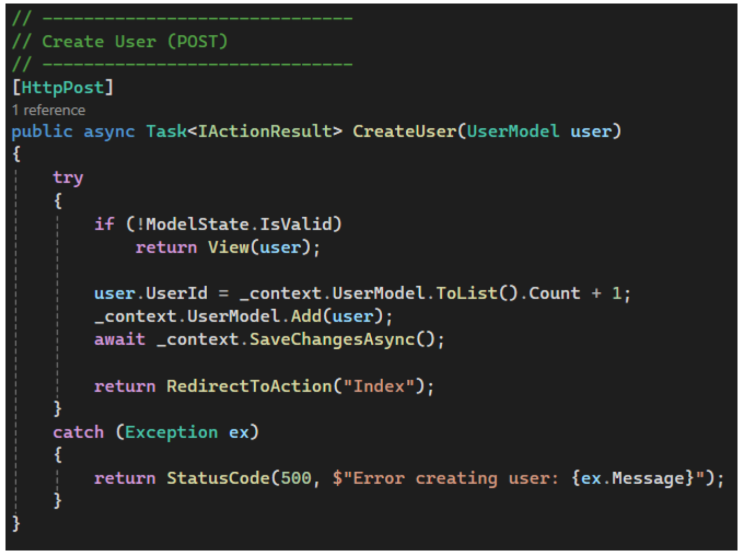


```
// --------------------------------
// User Details
// --------------------------------
[HttpGet]
1 reference
public IActionResult Details(int userId)
{
    try
    {
        var role = HttpContext.Session.GetString("UserRole");
        if (role != Role.HR.ToString())
            return RedirectToAction("AccessDenied", "Login");

        var user = _context.UserModel.Find(userId);
        if (user == null)
            return NotFound();

        var userClaims = _context.ClaimModel.Where(c => c.UserId == userId).ToList();
        ViewBag.Claims = userClaims;

        return View(user);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Error loading user details: {ex.Message}");
    }
}
```

For every employee, the Details() method serves as a personal profile viewer. This function receives a user's complete information and all claims related to that user once HR has chosen them. It gives HR a comprehensive, centralised view of the employee's activities, including their personal information, claim history, and any supporting data HR may require for decision-making or auditing. This approach promotes accountability and transparency from the standpoint of system design since HR can rapidly identify trends, validate claims, and comprehend how an employee interacts with the system. As a one-stop shop for examining an employee's whole file, it significantly contributes to the HR interface's usefulness and educational value.



```
// -----------------------------
// Create User (GET)
// -----------------------------
[HttpGet]
0 references
public IActionResult CreateUser()
{
    try
    {
        var role = HttpContext.Session.GetString("UserRole");
        if (role != Role.HR.ToString())
            return RedirectToAction("AccessDenied", "Login");

        return View(new UserModel());
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Error loading Create User page: {ex.Message}");
    }
}
```

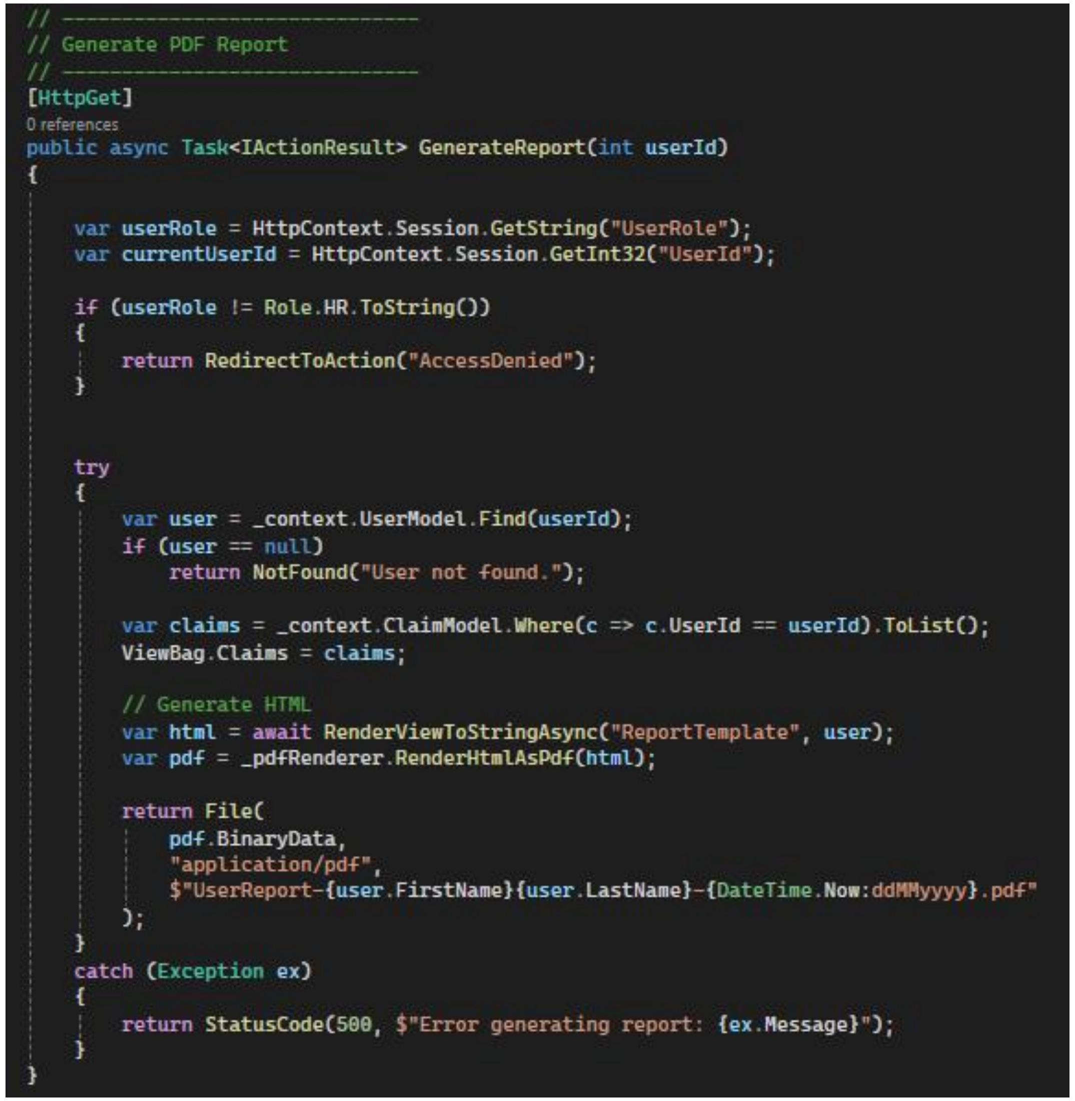


```
// -----------------------------
// Create User (POST)
// -----------------------------
[HttpPost]
1 reference
public async Task<IActionResult> CreateUser(UserModel user)
{
    try
    {
        if (!ModelState.IsValid)
            return View(user);

        user.UserId = _context.UserModel.ToList().Count + 1;
        _context.UserModel.Add(user);
        await _context.SaveChangesAsync();

        return RedirectToAction("Index");
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Error creating user: {ex.Message}");
    }
}
```

CreateUser (Get & Post) – Adding new Employees

This method's GET and POST versions work together to create the workflow for adding new employees to the system. The GET method enforces role-based access control and merely shows a blank registration form. The real creation logic is carried out by the POST method, which stores the employee's data in the database, generates a new unique ID, and validates the input. This approach guarantees a smooth and uniform hiring process. In order to stop incomplete or incorrect entries from entering the system, the workflow complies with validation requirements. These two techniques create a clear, safe, and effective onboarding procedure within the program, which is crucial for a professional HR system's capacity to add new users with ease and dependability.

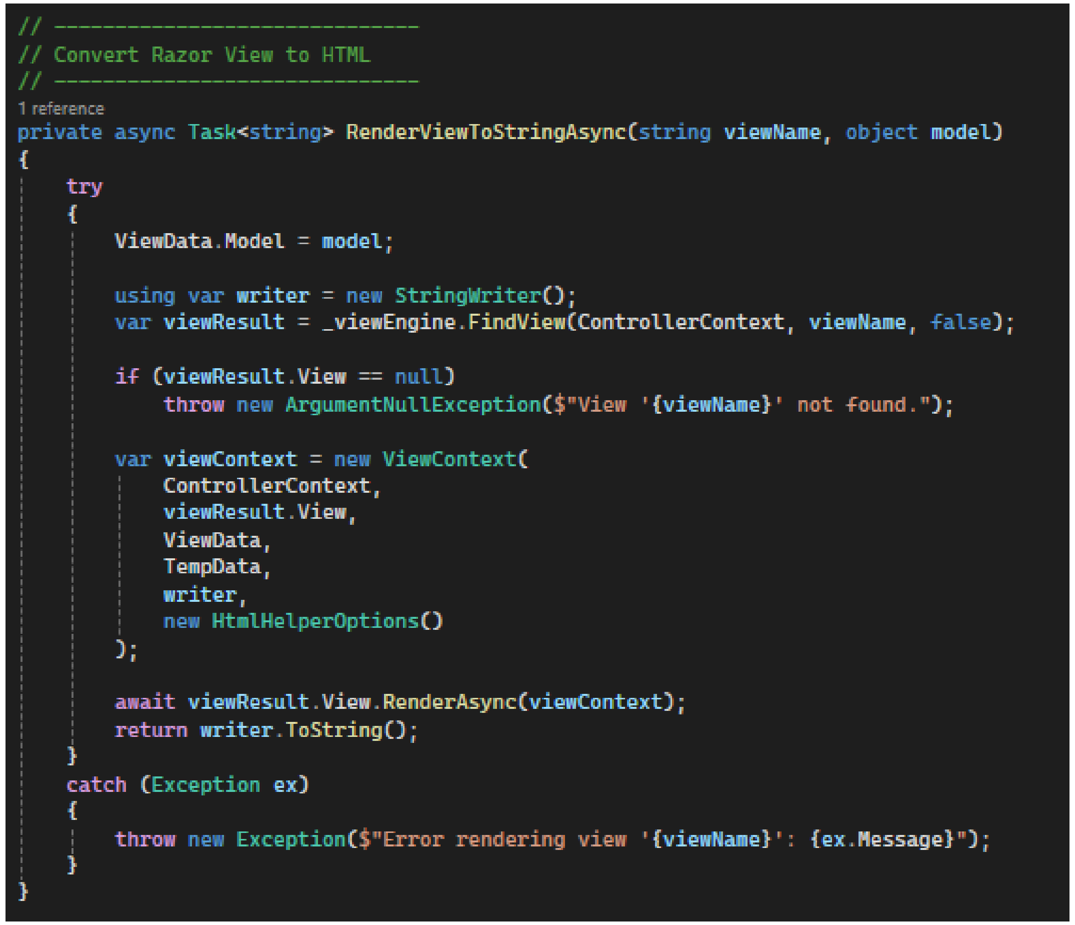
GenerateReport – PDF Creation



```
// ------------------------------
// Generate PDF Report
// ------------------------------
[HttpGet]
0 references
public async Task<IActionResult> GenerateReport(int userId)
{

    var userRole = HttpContext.Session.GetString("UserRole");
    var currentUserId = HttpContext.Session.GetInt32("UserId");

    if (userRole != Role.HR.ToString())
    {
        return RedirectToAction("AccessDenied");
    }


    try
    {
        var user = _context.UserModel.Find(userId);
        if (user == null)
            return NotFound("User not found.");

        var claims = _context.ClaimModel.Where(c => c.UserId == userId).ToList();
        ViewBag.Claims = claims;

        // Generate HTML
        var html = await RenderViewToStringAsync("ReportTemplate", user);
        var pdf = _pdfRenderer.RenderHtmlAsPdf(html);

        return File(
            pdf.BinaryData,
            "application/pdf",
            $"UserReport-{user.FirstName}{user.LastName}-{DateTime.Now:ddMMyyyy}.pdf"
        );
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Error generating report: {ex.Message}");
    }
}
```

This is one of the controller's best features since it turns ordinary online data into a polished, expert documents. Since reports frequently contain sensitive information, the process starts by determining whether the requester is actually HR, ensuring rigorous security. The chosen user and all of their claims are then retrieved, and they are ready to be incorporated in a PDF. The method converts a Razor view into HTML using another helper (RenderViewToStringAsync), which is then sent to IronPDF's ChromePdfRenderer to create a PDF that can be downloaded. This enables the system to generate documents that can be printed, shared, and archived, which is crucial for activities related to HR, auditing, payroll review, and compliance. With automated documentation that businesses can depend on, the functionality transforms your system from "just a web app" to something more enterprise-ready.

RenderViewToStringAsync – Converting Razor to HTML



```
// ------------------------------
// Convert Razor View to HTML
// ------------------------------
1 reference
private async Task<string> RenderViewToStringAsync(string viewName, object model)
{
    try
    {
        ViewData.Model = model;

        using var writer = new StringWriter();
        var viewResult = _viewEngine.FindView(ControllerContext, viewName, false);

        if (viewResult.View == null)
            throw new ArgumentNullException($"View '{viewName}' not found.");

        var viewContext = new ViewContext(
            ControllerContext,
            viewResult.View,
            ViewData,
            TempData,
            writer,
            new HtmlHelperOptions()
        );

        await viewResult.View.RenderAsync(viewContext);
        return writer.ToString();
    }
    catch (Exception ex)
    {
        throw new Exception($"Error rendering view '{viewName}': {ex.Message}");
    }
}
```

This technique is a backend engine that enables the creation of PDFs. This technique fills the gap left by Razor views' lack of direct compatibility with PDF generators. It selects the appropriate Razor template, ties the model (the user and their claims) to the view, and then renders it as plain HTML. IronPDF is able to comprehend that HTML. Because it uses the same Razor templates that the web pages currently use, there is no need to create unique representations for the PDF version, which makes this process quite effective. This is the approach you use in a presentation to emphasise reusability, high-quality architecture, and astute technical choices. It maintains the efficiency, cleanliness, and dryness of your system.

Thank you for your time.

ST10273397

Nicholas Wolfaardt

GitHub Link: https://github.com/VCCT-PROG6212-2025-G1/ST10273397_PROG6212POE.git

YouTube Link: https://youtu.be/8ewq0JXQozg