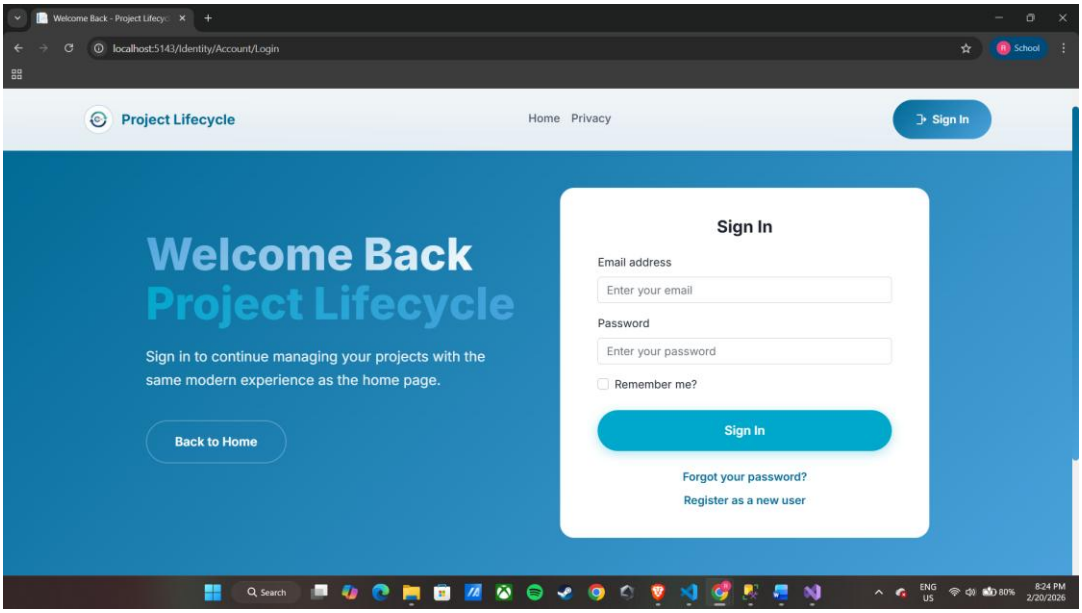
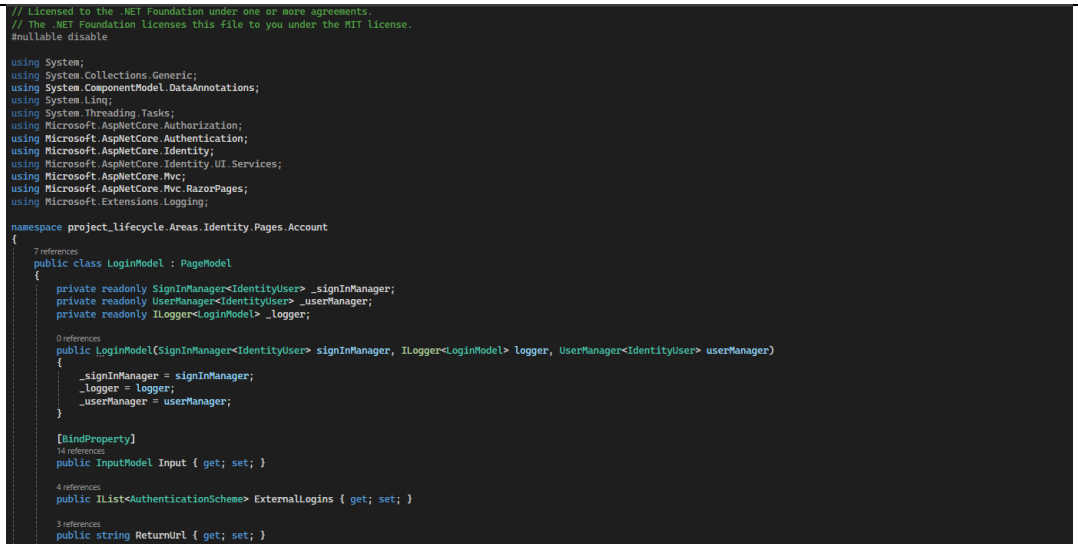
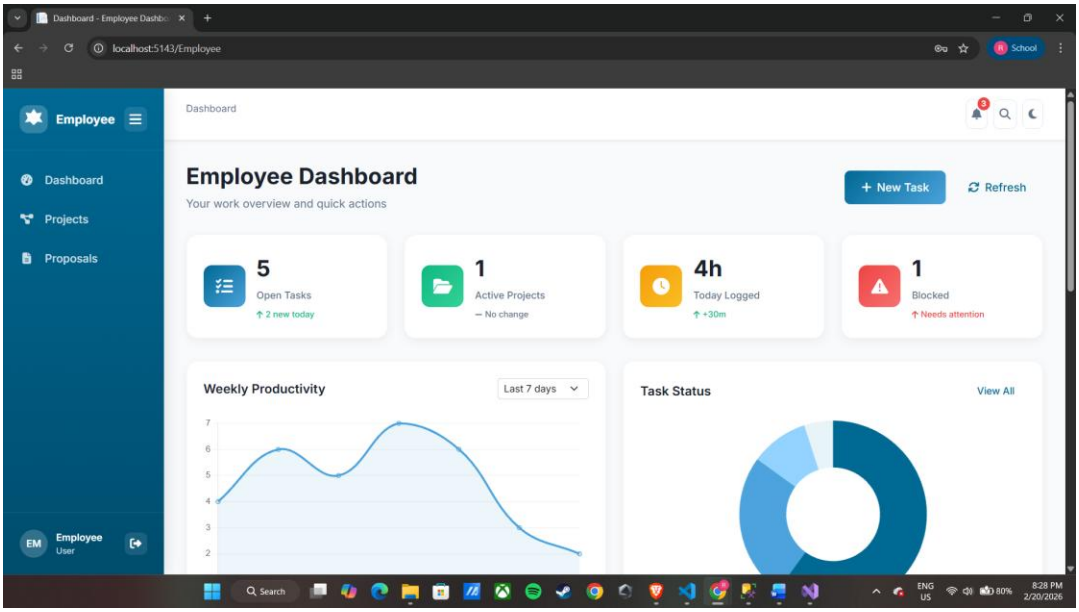
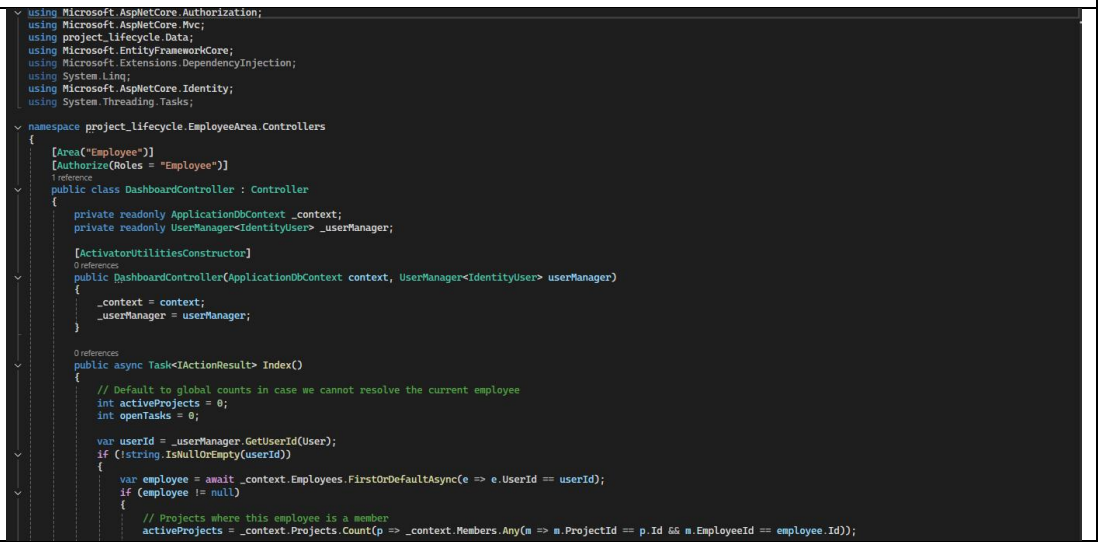
NAME	**ROCKY BANTILAN ADAYA**
PROJECT TITLE	**Adaya IT Solution: Corporate Project Lifecycle & Workflow Management System**
SUBJECT: **CODE:** **TIME:**	**IT15/L Integrative Programming and Technologies** **8441** **10:00 AM – 12:00 PM**
TOPIC (Type of Business Process)	**#37 Collaborative and Workflow Transactions System**
Products/Servics	**Corporate Enterprises, Project-Based Organizations, and HR/Project Management Services**
Deployed Link / Website link	**http://adaya-101.runasp.net**
Role Based-Access	**Super Admin Acct. (if done)** **Username: superadmin@gmail.com** **Password: @Admin123** **Admin Acct. (Department Head)** **Username: dh@gmail.com** **Password: @Depthead123** **Admin Acct. (Project Manager)** **Username: pm@gmail.com** **Password: @Projman123** **Employee Acct.** **Username: emp@gmail.com** **Password: @Emp123**
3rd Deliverables-Deployed Transactions (CRUD)	

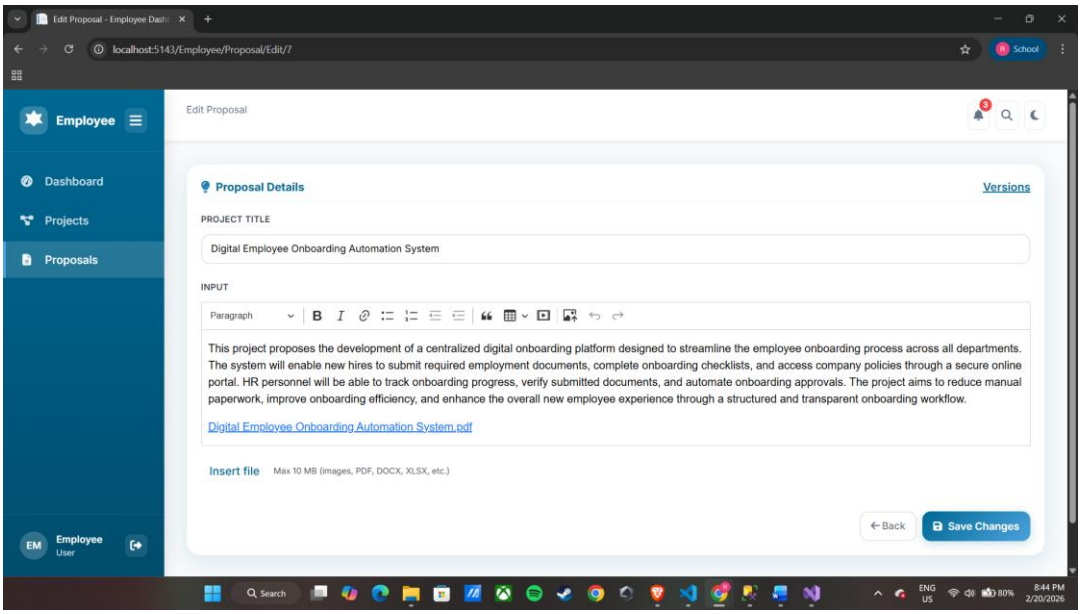
Log in-Page (Screenshot)	 The **Login Page** enables internal corporate personnel to securely access the system using their assigned account credentials. Employee accounts are pre-registered and managed by the Super Administrator or HR/Admin, ensuring that only authorized users can log in to the platform. Upon entering a valid username and password, the system authenticates the user's identity and grants access based on their designated role within the organization, such as Employee, Project Manager, Department Head, HR/Admin, Executive, or Super Admin. This role-based authentication ensures that users are directed to the appropriate dashboard and are only able to access features and data relevant to their responsibilities. In cases of invalid login credentials, the system displays an error message prompting the user to provide correct login information.
Log in-Page - Source Code (Screenshot)	

```
    [TempData]
    2 references
    public string ErrorMessage { get; set; }

    1 reference
    public class InputModel
    {
        [Required]
        [EmailAddress]
        5 references
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        4 references
        public string Password { get; set; }

        [Display(Name = "Remember me?")]
        5 references
        public bool RememberMe { get; set; }
    }

    0 references
    public async Task OnGetAsync(string returnUrl = null)
    {
        if (!string.IsNullOrEmpty(ErrorMessage))
        {
            ModelState.AddModelError(string.Empty, ErrorMessage);
        }

        returnUrl ??= Url.Content("~/");

        // Clear the existing external cookie to ensure a clean login process
        await HttpContext.SignOutAsync(IdentityConstants.ExternalScheme);

        ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();

        ReturnUrl = returnUrl;
    }

    0 references
    public async Task<IActionResult> OnPostAsync(string returnUrl = null)
    {
        returnUrl ??= Url.Content("~/");

        ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();

        if (ModelState.IsValid)
        {
            var result = await _signInManager.PasswordSignInAsync(Input.Email, Input.Password, Input.RememberMe, lockoutOnFailure: false);
            if (result.Succeeded)
            {
                var user = await _userManager.FindByEmailAsync(Input.Email);
                if (await _userManager.IsInRoleAsync(user, "SuperAdmin"))
                {
                    return RedirectToAction("Index", "SuperAdmin", new { area = "SuperAdmin" });
                }

                if (await _userManager.IsInRoleAsync(user, "Admin"))
                {
                    return RedirectToAction("Index", "Dashboard", new { area = "Admin" });
                }

                if (await _userManager.IsInRoleAsync(user, "Employee"))
                {
                    return RedirectToAction("Index", "Dashboard", new { area = "Employee" });
                }

                if (await _userManager.IsInRoleAsync(user, "HumanResource"))
                {
                    return RedirectToAction("Index", "Dashboard", new { area = "HumanResource" });
                }

                if (await _userManager.IsInRoleAsync(user, "DepartmentHead"))
                {
                    return RedirectToAction("Index", "Dashboard", new { area = "DepartmentHead" });
                }

                if (await _userManager.IsInRoleAsync(user, "Executive"))
                {
                    return RedirectToAction("Index", "Dashboard", new { area = "Executive" });
                }

                if (await _userManager.IsInRoleAsync(user, "ProjectManager"))
                {
                    return RedirectToAction("Index", "Dashboard", new { area = "ProjectManager" });
                }

                _logger.LogInformation("User logged in.");
                return LocalRedirect(returnUrl);
            }
            if (result.RequiresTwoFactor)
            {
                return RedirectToPage("./LoginWith2fa", new { ReturnUrl = returnUrl, RememberMe = Input.RememberMe });
            }
            if (result.IsLockedOut)
            {
                _logger.LogWarning("User account locked out.");
                return RedirectToPage("./Lockout");
            }
            else
            {
                ModelState.AddModelError(string.Empty, "Invalid login attempt.");
                return Page();
            }
        }

        return Page();
    }
}
```

Dashboard (Client/End-User) (Screenshot)	 The **Employee Dashboard** serves as the primary interface for employees to access and manage their assigned tasks and project-related responsibilities within the system. It provides an overview of ongoing projects, assigned tasks, deadlines, and project updates relevant to the employee's role. Through this dashboard, employees can track the progress of their tasks, view project details, and stay informed about notifications such as new assignments, task updates, or project announcements. Additionally, it allows employees to collaborate within project workflows by participating in assigned activities and monitoring their individual contributions to the overall project lifecycle. This feature ensures that employees remain updated and actively engaged in project execution and organizational workflow processes.
Dashboard - Source Code **(Client/End-User) (Screenshot)**	

```
                // Open tasks assigned to this employee via TaskMembers and not Checked
                openTasks = _context.TaskMembers
                    .Where(tm => tm.Member != null && tm.Member.EmployeeId == employee.Id && tm.ProjectTask != null && tm.ProjectTask.Status != "Checked")
                    .Select(tm => tm.ProjectTaskId)
                    .Distinct()
                    .Count();
            }
        }

        ViewData["ActiveProjects"] = activeProjects;
        ViewData["OpenTasks"] = openTasks;

        return View();
    }
  }
}
```

Update/Edit Page (Screenshot)



The **Edit Proposal** page provides a dedicated interface for employees to modify and refine their submitted project proposals. This page allows users to update the project title and use a rich text editor to expand upon the project's description, objectives, and workflows—such as the "Digital Employee Onboarding Automation System" shown in the interface.

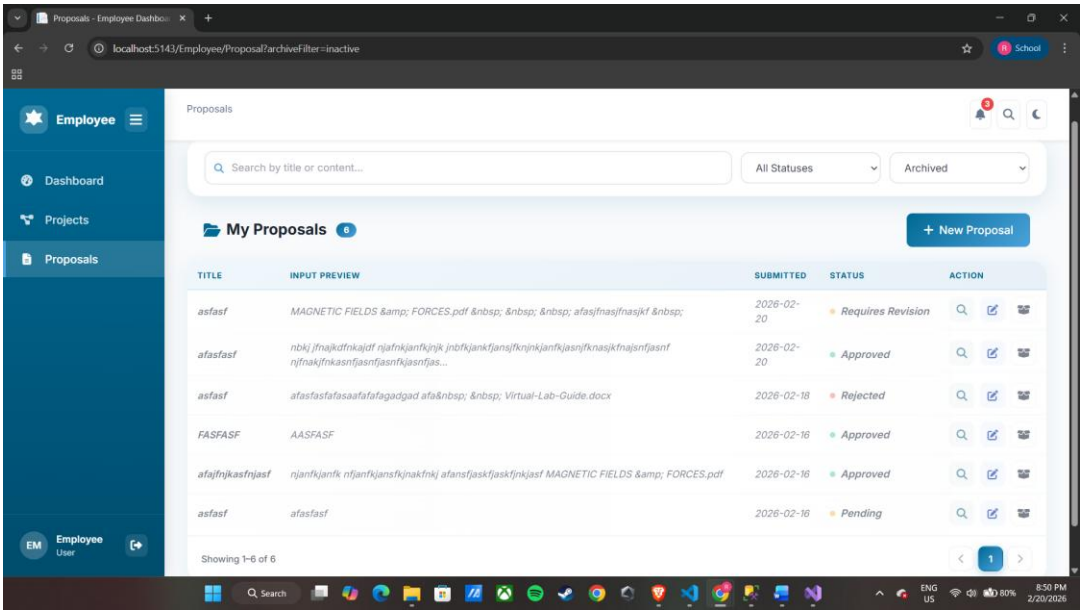
By offering tools for text formatting and file attachments (supporting images, PDFs, and documents up to 10 MB), the page ensures that proposals can be presented with professional clarity and supporting documentation. The inclusion of a "Versions" feature allows employees to track historical changes, while the "Save Changes" functionality ensures that updates are synchronized with the system’s central database. This interface is critical for maintaining the accuracy of project documentation and allowing for iterative improvements to proposals before they move through the organizational approval lifecycle.

Update/Edit Page - Source Code (Screenshot)

```
[HttpGet]
0 references
public async Task<IActionResult> Edit(int id)
{
    try
    {
        var userId = _userManager.GetUserId(User);
        if (string.IsNullOrEmpty(userId)) return Challenge();

        var employee = await _context.Employees.FirstOrDefaultAsync(e => e.UserId == userId);
        if (employee == null) return Forbid();

        var proposal = await _context.ProjectProposals.FirstOrDefaultAsync(p => p.Id == id && p.EmployeeId == employee.Id);
        if (proposal == null) return NotFound();

        var versions = await _context.ProjectProposalVersions
            .Where(v => v.ProjectProposalId == proposal.Id)
            .OrderByDescending(v => v.VersionNumber)
            .ToListAsync();

        var nextVersion = (versions.Any() ? versions.Max(v => v.VersionNumber) : 0) + 1;

        var vm = new ProjectProposalEditViewModel
        {
            Proposal = proposal,
            Versions = versions,
            NextVersionNumber = nextVersion
        };

        return View(vm);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error loading proposal edit for id {Id}", id);
        return StatusCode(500);
    }
}
```

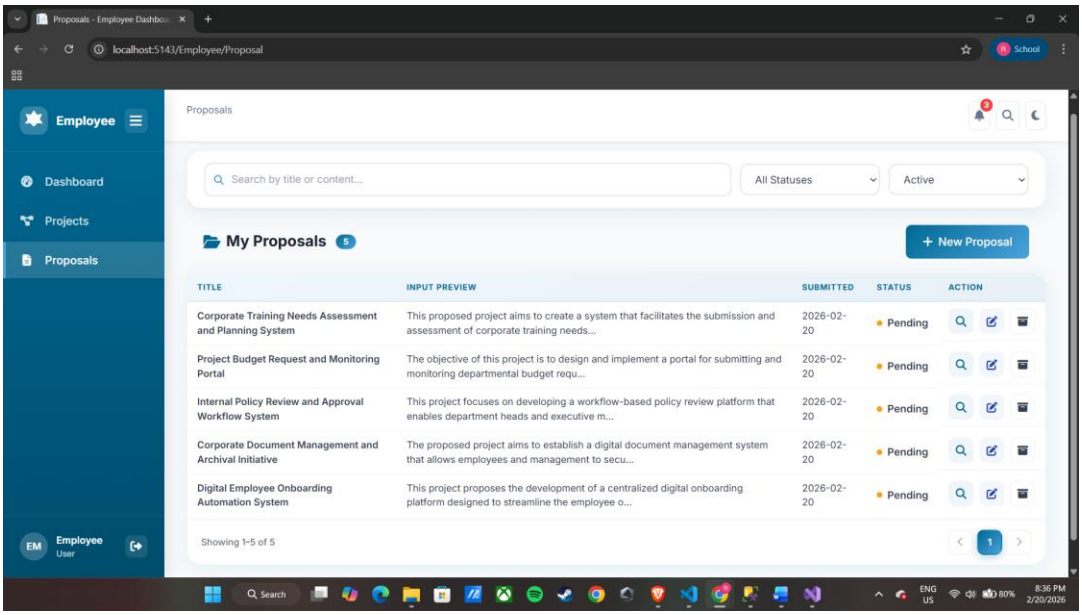
```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Edit(int id, ProjectProposalEditViewModel model)
{
    try
    {
        var userId = _userManager.GetUserId(User);
        if (string.IsNullOrEmpty(userId)) return Challenge();

        var employee = await _context.Employees.FirstOrDefaultAsync(e => e.UserId == userId);
        if (employee == null) return Forbid();

        var proposal = await _context.ProjectProposals.FirstOrDefaultAsync(p => p.Id == id && p.EmployeeId == employee.Id);
        if (proposal == null) return NotFound();

        if (model?.Proposal == null)
        {
            TempData["ErrorMessage"] = "Invalid submission.";
            return RedirectToAction("Index");
        }

        if (string.IsNullOrWhiteSpace(model.Proposal.Title) || string.IsNullOrWhiteSpace(model.Proposal.Input))
        {
            TempData["ErrorMessage"] = "Title and Input are required.";
            return RedirectToAction("Edit", new { id });
        }

        // Save current content as a previous version
        var existingVersions = await _context.ProjectProposalVersions.Where(v => v.ProjectProposalId == proposal.Id).ToListAsync();
        var nextVersion = (existingVersions.Any() ? existingVersions.Max(v => v.VersionNumber) : 0) + 1;

        var previousVersion = new ProjectProposalVersion
        {
            ProjectProposalId = proposal.Id,
            VersionNumber = nextVersion,
            Title = proposal.Title,
            Input = proposal.Input,
            EmployeeId = employee.Id,
            DateCreated = DateTime.Now
        };
```

```
        _context.ProjectProposalVersions.Add(previousVersion);

        // Update the main proposal with the new content
        proposal.Title = model.Proposal.Title;
        proposal.Input = model.Proposal.Input;
        proposal.DateCreated = DateTime.Now;

        await _context.SaveChangesAsync();

        TempData["SuccessMessage"] = "Proposal updated and previous version saved.";
        return RedirectToAction("Details", new { id = proposal.Id });
    }
    catch (DbUpdateException dbEx)
    {
        _logger.LogError(dbEx, "DB error updating proposal {Id}", id);
        TempData["ErrorMessage"] = dbEx.InnerException?.Message ?? dbEx.Message;
        return RedirectToAction("Edit", new { id });
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Unexpected error updating proposal {Id}", id);
        TempData["ErrorMessage"] = ex.Message;
        return RedirectToAction("Edit", new { id });
    }
}
```

Archive (Screenshot)	 The **Archive** functionality within the Proposals module serves as a critical organizational tool, allowing employees to manage the visibility and lifecycle of their submissions. By utilizing the archive filter and the dedicated archive action icon located in the proposal table, users can transition inactive, completed, or outdated proposals out of their primary workspace. This feature provides a structured way to declutter the "My Proposals" list without permanently deleting records, ensuring that historical data—such as submission dates, past statuses (e.g., Approved or Rejected), and associated documents—remains accessible for future reference or audits. Through the archive toggle and status filters, employees can seamlessly switch between viewing active projects and archived entries, maintaining an efficient and focused environment for current project execution.
Archive – Source Code (Screenshot)	

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Unarchive(int id)
{
    try
    {
        var userId = _userManager.GetUserId(User);
        if (string.IsNullOrEmpty(userId)) return Challenge();

        var employee = await _context.Employees.FirstOrDefaultAsync(e => e.UserId == userId);
        if (employee == null) return Forbid();

        var proposal = await _context.ProjectProposals.FirstOrDefaultAsync(p => p.Id == id && p.EmployeeId == employee.Id);
        if (proposal == null) return NotFound();

        proposal.IsArchived = false;
        await _context.SaveChangesAsync();

        TempData["SuccessMessage"] = "Proposal restored from archive.";
        return RedirectToAction("Index", new { archiveFilter = "inactive" });
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error unarchiving proposal {Id}", id);
        TempData["ErrorMessage"] = "Could not restore proposal.";
        return RedirectToAction("Index");
    }
}
```

Display Records (Screenshot)



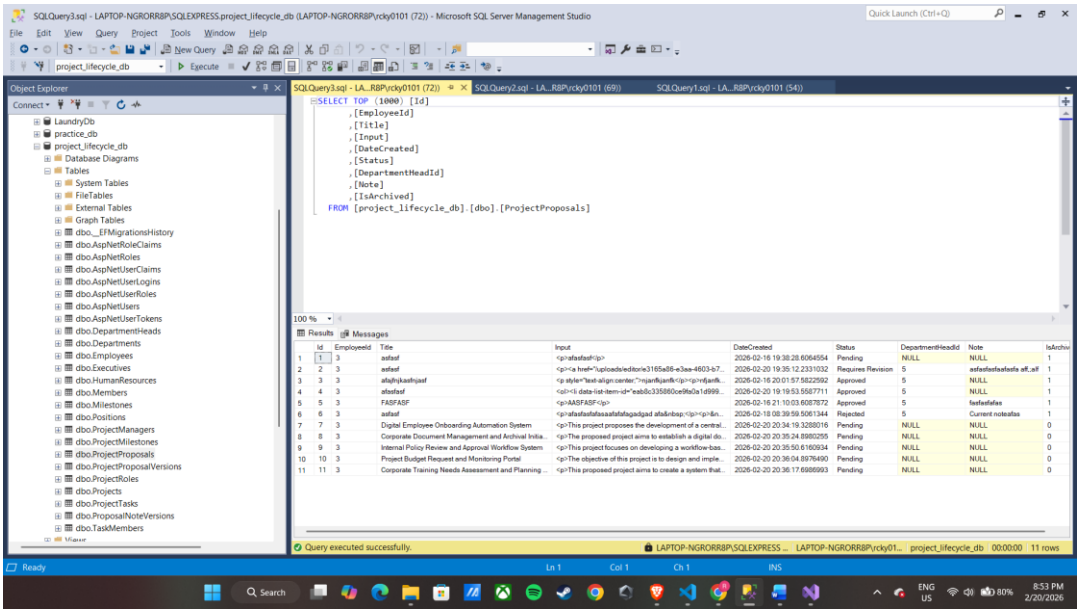
The **My Proposals** page features a structured **Display Records** interface that provides a comprehensive overview of all project proposals authored by the employee. This centralized table is designed for high scannability, allowing users to monitor their submission history and current progress at a glance. Each record in the display includes several critical fields, such as the full **Project Title**, an **Input Preview** that highlights snippets of the proposal's core content or attached filenames, the specific **Date Submitted**, and a color-coded **Status** badge indicating whether the item is Pending, Approved, Rejected, or Requires Revision.

To maintain an organized workspace, the interface is supported by dynamic management tools, including a global search bar and dropdown menus that allow users to filter records by their active or archived state. Furthermore, a dedicated **Action** column provides immediate access to view, edit, or archive individual entries, while pagination controls ensure that large sets of data remain easy to navigate. This systematic display of records ensures that employees can efficiently manage their professional contributions and stay informed about the lifecycle of their organizational proposals.

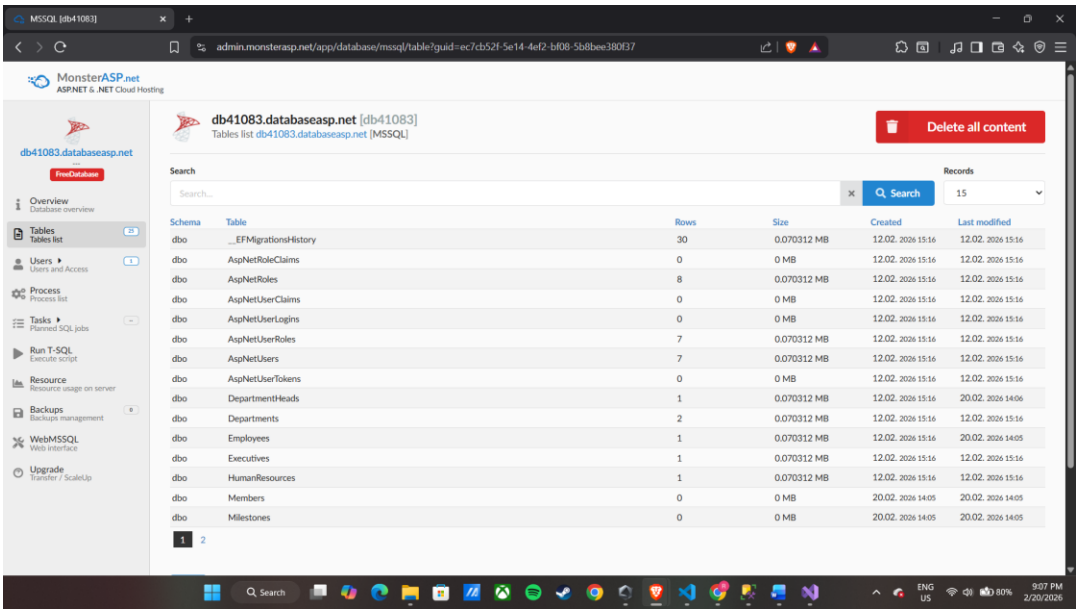
Display Records - Source Code (Screenshot)

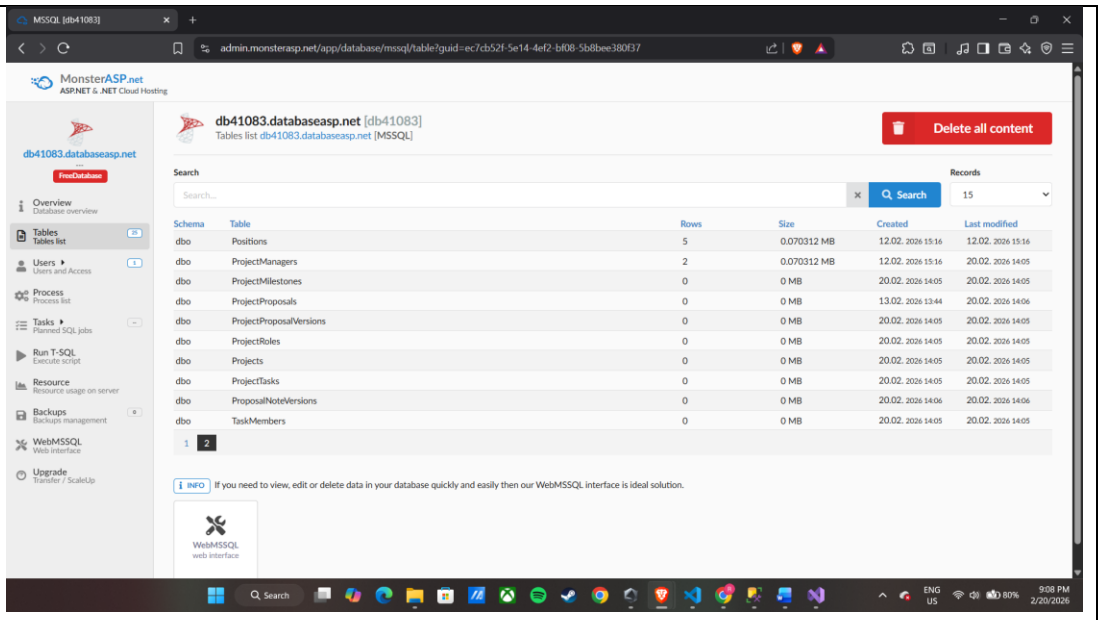
```
public IActionResult Index(string archiveFilter = "active")
{
    ViewData["Title"] = "Proposals";

    try
    {
        // Get current employee ID from user
        var userId = _userManager.GetUserId(User);
        _logger.LogInformation($"Getting proposals for userId: {userId}");

        if (string.IsNullOrEmpty(userId))
        {
            _logger.LogWarning("UserId is null or empty");
            return View(new List<ProjectProposal>());
        }

        var employee = _context.Employees.FirstOrDefault(e => e.UserId == userId);

        if (employee == null)
        {
            _logger.LogWarning($"No employee found for userId: {userId}");
            return View(new List<ProjectProposal>());
        }

        _logger.LogInformation($"Found employee: {employee.Id}");

        // archiveFilter: "all" | "active" | "inactive" (default: active)
        IQueryable<ProjectProposal> query = _context.ProjectProposals.Where(p => p.EmployeeId == employee.Id);
        if (!string.Equals(archiveFilter, "all", StringComparison.OrdinalIgnoreCase))
        {
            if (string.Equals(archiveFilter, "inactive", StringComparison.OrdinalIgnoreCase))
            {
                query = query.Where(p => p.IsArchived);
            }
            else // active
            {
                query = query.Where(p => !p.IsArchived);
            }
        }

        var proposals = query.OrderByDescending(p => p.DateCreated).ToList();
        ViewData["ArchiveFilter"] = archiveFilter;

        _logger.LogInformation($"Found {proposals.Count} proposals for employee {employee.Id}");

        return View(proposals);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error in ProposalController.Index");
        ViewData["ErrorMessage"] = $"Error loading proposals: {ex.Message}";
        return View(new List<ProjectProposal>());
    }
}
```

Database – SQL Server (Local) Screenshot



The **Database** module for the proposal system is powered by **SQL Server (Local)**, providing a robust and structured relational storage environment for all project-related data. Utilizing Microsoft SQL Server Management Studio (SSMS), administrators and developers can manage the project_lifecycle_db database, which serves as the central

	repository for critical system information. The core of this storage is the ProjectProposals table, which systematically records essential attributes for every submission, including unique identifiers like the **EmployeeId**, the project **Title**, and the detailed **Input** content. Furthermore, the database handles lifecycle and organizational metadata by storing the **DateCreated**, the current **Status** (such as Pending, Approved, or Rejected), and departmental links through the **DepartmentHeadId** field. It also features built-in logic for content management, such as a **Note** field for feedback and a bit-based **IsArchived** column, which directly controls whether a record appears in the active or archived view on the employee dashboard. By maintaining this detailed schema, the SQL Server backend ensures data integrity, supports complex filtering across different proposal versions, and facilitates seamless synchronization between the user interface and the persistent storage layer.
Database – Deployed/Online **Screenshot**	



The **Database** infrastructure is hosted in a **Deployed/Online** environment through **MonsterASP.net**, providing a high-availability cloud solution for the project lifecycle system. This online implementation utilizes a remote **MSSQL** instance, identified as db41083.databaseasp.net, which mirrors the local schema to ensure seamless data transitions between development and production. Through the MonsterASP.net management interface, administrators can monitor real-time database statistics, including row counts, storage sizes, and precise timestamps for the last modification of table data.

The cloud database maintains the integrity of the organizational hierarchy and project workflows by hosting several critical tables, including User and Role Management tables such as AspNetUsers and AspNetRoles for secure access, as well as Organizational Structure tables like Departments and Employees to facilitate proposal routing. The Project and Proposal Lifecycle is meticulously tracked through the ProjectProposals and ProjectProposalVersions tables, while specific execution data is managed via Projects, ProjectTasks, and ProjectMilestones. Additionally, Administrative Tracking is handled through tables like ProposalNoteVersions and ProjectManagers. This cloud-based deployment ensures that all project data remains synchronized and accessible to remote users, providing a scalable backbone for the organization's digital transformation efforts.

Date Submitted:	**February 20, 2026**
Teacher's Feedback	_____ _____ _____ _____

	_____ _____ _____ _____ Student's Signature (after feedbacking) Teacher's Signature