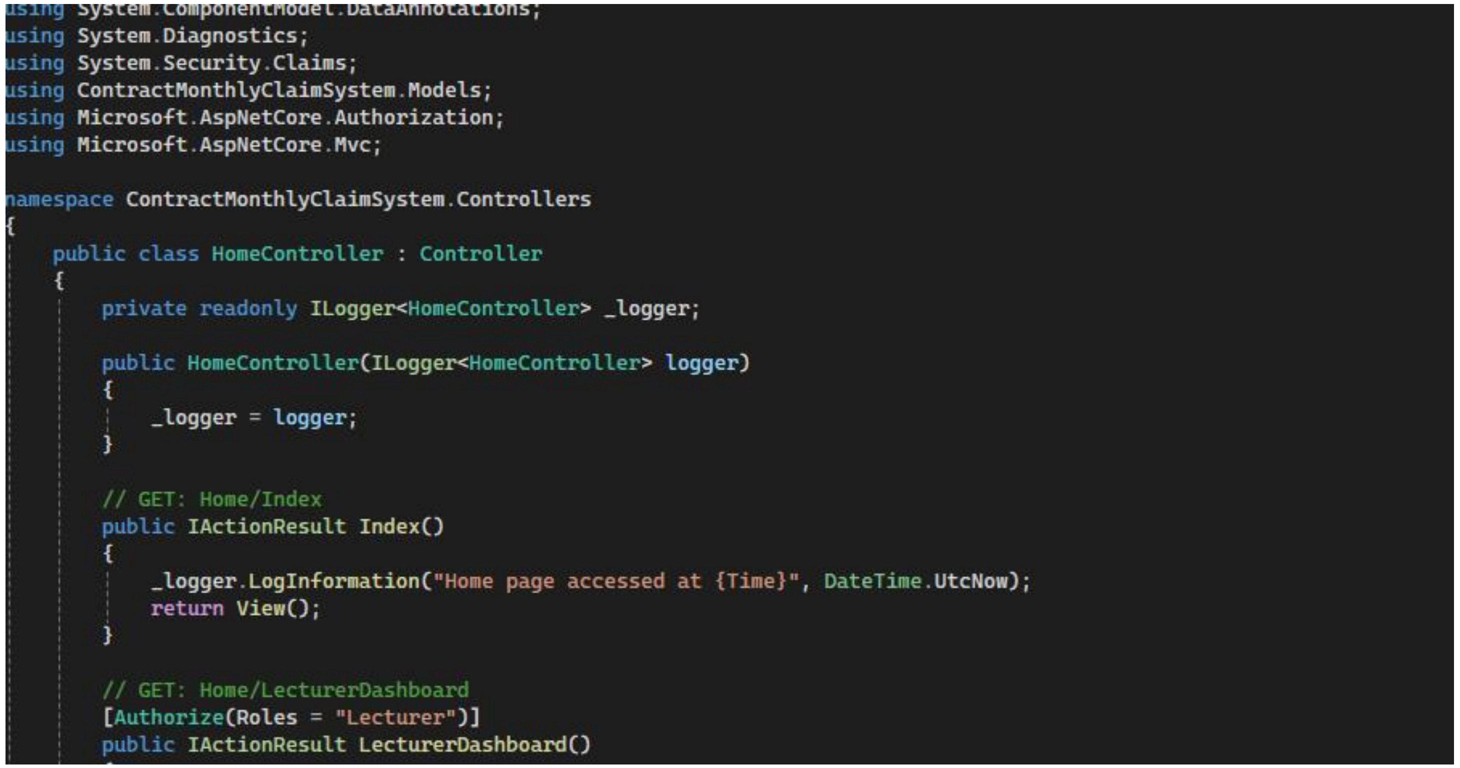
CONTRACT MONTHLY CLAIM SYSTEM HomeController CoreNavigation Hub

- 🎯 PURPOSE:
- • Role-based dashboard system for claim management
- • Secure access control for different user types
- • Central navigation for the entire application

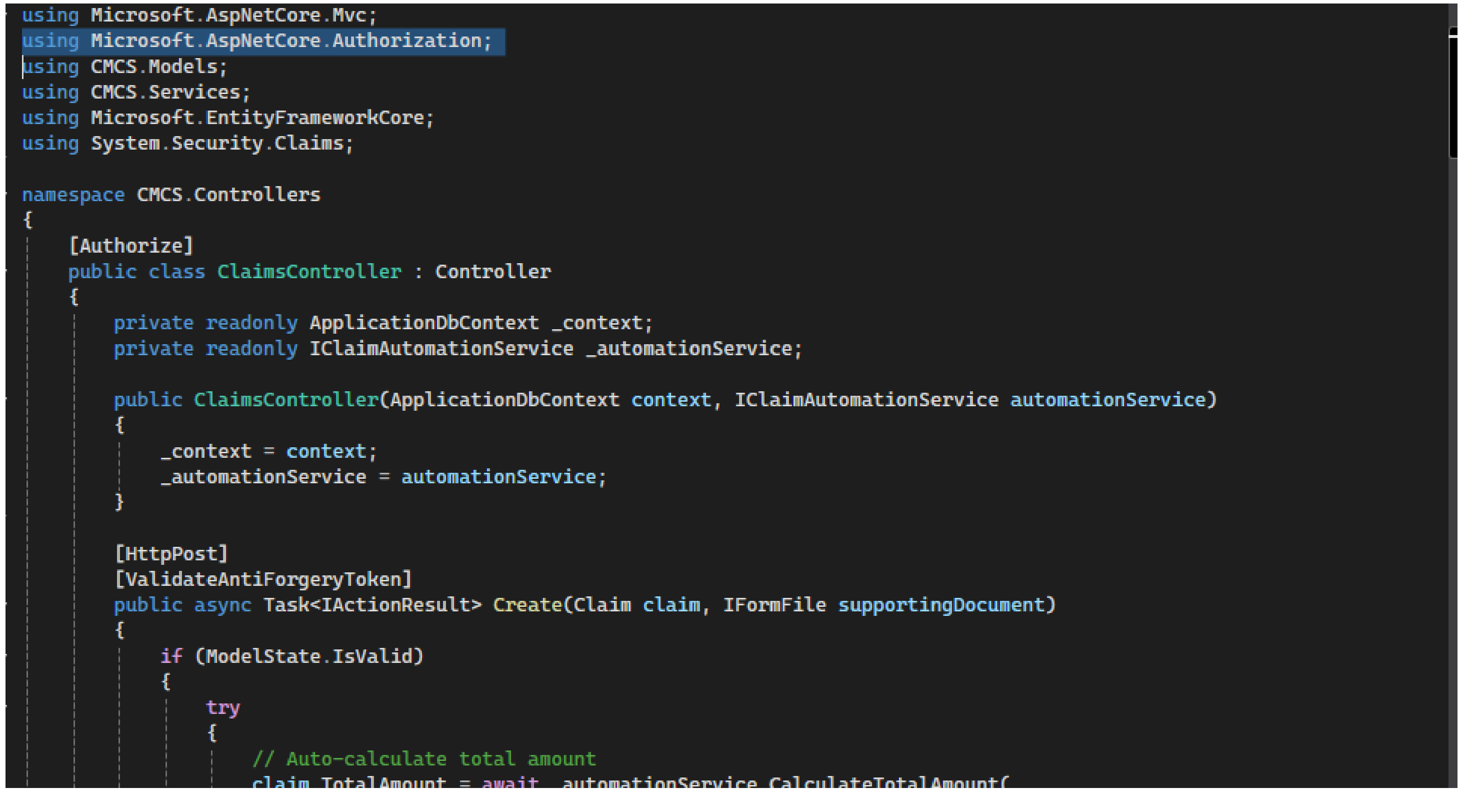
- 👥 TARGET USERS:
- • Lecturers (Claim Submission)
- • Programme Coordinators (Claim Review)
- • Academic Managers (Final Approval)
- • HR Staff (Analytics & Reporting)



```
using System.ComponentModel.DataAnnotations;
using System.Diagnostics;
using System.Security.Claims;
using ContractMonthlyClaimSystem.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace ContractMonthlyClaimSystem.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }

        // GET: Home/Index
        public IActionResult Index()
        {
            _logger.LogInformation("Home page accessed at {Time}", DateTime.UtcNow);
            return View();
        }

        // GET: Home/LecturerDashboard
        [Authorize(Roles = "Lecturer")]
        public IActionResult LecturerDashboard()
```

Controller Architecture

- 🏗️ TECHNICAL ARCHITECTURE ASP.NET Core MVC Controller ├── Dependency Injection │ └── ILogger for auditing ├── Role-Based Authorization │ └── [Authorize(Roles = "")] ├── Model-View-Controller Pattern │ └── Controller → View with ViewBag └── Exception Handling └── Try-Catch with logging 📁 Namespace: ContractMonthlyClaimSystem.Controllers 📁 Inherits from: Microsoft.AspNetCore.Mvc.Controller



```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Authorization;
using CMCS.Models;
using CMCS.Services;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;

namespace CMCS.Controllers
{
    [Authorize]
    public class ClaimsController : Controller
    {
        private readonly ApplicationDbContext _context;
        private readonly IClaimAutomationService _automationService;

        public ClaimsController(ApplicationDbContext context, IClaimAutomationService automationService)
        {
            _context = context;
            _automationService = automationService;
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create(Claim claim, IFormFile supportingDocument)
        {
            if (ModelState.IsValid)
            {
                try
                {
                    // Auto-calculate total amount
                    claim.TotalAmount = await _automationService.CalculateTotalAmount(
```

Authentication & Security

🔐 SECURITY FEATURES ROLE-BASED ACCESS: [Authorize(Roles = "Lecturer")] [Authorize(Roles = "ProgrammeCoordinator,AcademicManager")] [Authorize(Roles = "AcademicManager,HR")] SECURITY MEASURES: • Anti-Forgery Tokens ([ValidateAntiForgeryToken]) • User Identification (ClaimTypes.NameIdentifier) • Access Logging for audit trails • Custom Error Handling with secure messages 🚫 AccessDenied() - Handles unauthorized access attempts

Dashboard System

📊 ROLE-SPECIFIC DASHBOARDS LECTURER DASHBOARD: • Claim status overview (Pending/Approved/Rejected) • Earnings summary • Recent claims history COORDINATOR DASHBOARD: • Pending approvals queue • Approval/rejection statistics • Recent action timeline MANAGER DASHBOARD: • System-wide analytics • Department performance • Processing metrics & KPIs

Data Flow & Communication

🔄 DATA HANDLING CURRENT IMPLEMENTATION: • Mock data for demonstration • ViewBag for view communication • Anonymous objects for structured data EXAMPLE DATA FLOW: User Request → Controller Method → Mock Data → ViewBag → Razor View CONTACT FORM FLOW: GET Contact() → Display Form POST Contact() → Validate → Log → Redirect (PRG Pattern)

Error Handling Strategy

🚨 ROBUST ERROR MANAGEMENT COMPREHENSIVE ERROR HANDLING: • Global try-catch blocks • Structured logging with user context • User-friendly error messages • Detailed ErrorViewModel for debugging ERROR() METHOD FEATURES: • HTTP status code mapping • Request tracing for support • Secure error information exposure • No caching of error pages 📝 Logs include: UserId, Timestamp, Error Details, Stack Trace

Code Quality Features

⭐ CODE QUALITY & MAINTAINABILITY LOGGING STRATEGY: • Information logging for user actions • Error logging with exceptions • Warning logging for security events • Structured logging with parameters VALIDATION: • Model validation with data annotations • Client-side and server-side validation • Custom error messages MAINTAINABILITY: • Consistent error handling pattern • Separation of concerns • Clear method responsibilities

Real-World Enhancements

🚀 FUTURE ENHANCEMENTS PLANNED INTEGRATIONS: ✅ Entity Framework with SQL Database ✅ ASP.NET Identity for real authentication ✅ Email service integration (SMTP/SendGrid) ✅ Real-time data updates (SignalR) ✅ Business logic for claim calculations ✅ File upload handling for documents ✅ Reporting and analytics engine SCALABILITY: • Repository pattern for data access • Service layer for business logic • API endpoints for mobile access • Caching for performance

Key Technical Takeaways

💡 TECHNICAL HIGHLIGHTS BEST PRACTICES IMPLEMENTED: ✓ Role-based authorization ✓ Comprehensive logging strategy ✓ Consistent error handling ✓ Model validation with data annotations ✓ PRG pattern for form submissions ✓ Dependency injection ✓ Secure coding practices TECHNOLOGY STACK: • ASP.NET Core 6+ MVC • C# with modern features • Razor Views for UI • Built-in authentication system • Structured logging system

Demo & Next Steps

🎬 DEMONSTRATION READY READY TO SHOW: • Multi-role dashboard navigation • Secure access control in action • Contact form with validation • Error handling scenarios • System status monitoring NEXT DEVELOPMENT PHASE: 1. Database integration 2. Real authentication setup 3. Business logic implementation 4. File upload functionality 5. Email service integration ⏱️ Estimated completion: 2-3 weeks for full functionality