Varsity College - School of IT

Programming 2B (PROG6212) POE Part 2

Bachelor of Computer Science in Application Development

Submitted by:
Cameron Chetty - ST10251759

October 18, 2024

TABLE OF CONTENTS

I. SUBMISSION INFORMATION

II. WEB APP PAGES

III. BIBLIOGRAPHY

PART 2: IMPLEMENT A PROTOTYPE WEB APPLICATION

Links

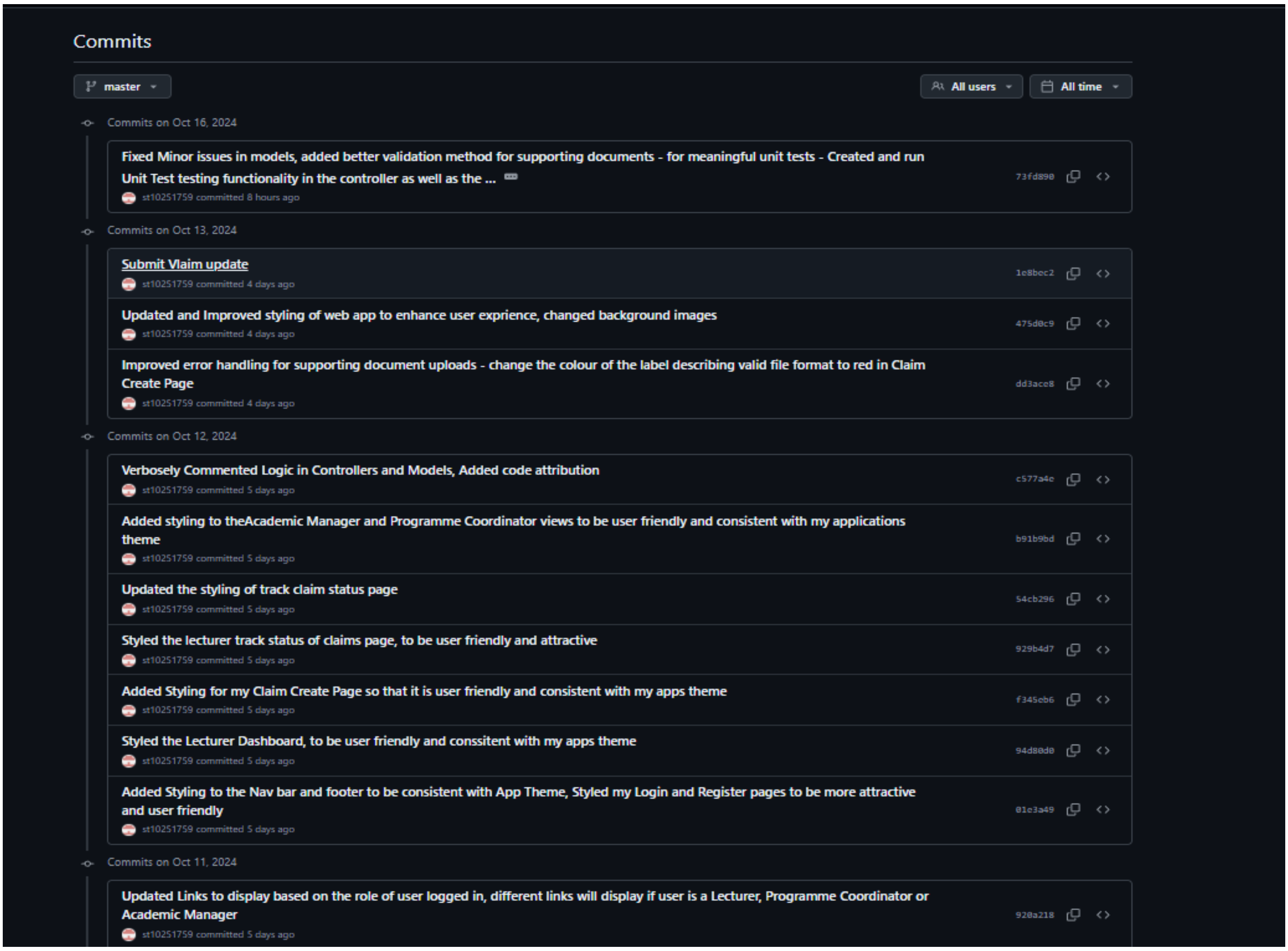
Please see below links for submission

- GitHub Link: https://github.com/st10251759/prog6212-poe-part-2
- YouTube Demo Link: https://youtu.be/6Fp5pLVE_Tw

Credentials for Testing With Different Role

1. Lecturer
   Username/Email: lecturer@gmail.com
   Password: N*BkM,T(L9:Jm=HF
2. Programme Coordinator
   Username/Email: coordinator@gmail.com
   Password: NeeuBgyFlE,HB7Uj
3. Academic Manager
   Username/Email: manager@gmail.com
   Password: R4,yNZcyoh77*zkD

Commit History



Commits

Commits on Oct 16, 2024

Fixed Minor issues in models, added better validation method for supporting documents - for meaningful unit tests - Created and run Unit Test testing functionality in the controller as well as the ...
st10251759 committed 8 hours ago — 73fd890

Commits on Oct 13, 2024

Submit Vlaim update
st10251759 committed 4 days ago — 1e8bec2

Updated and Improved styling of web app to enhance user exprience, changed background images
st10251759 committed 4 days ago — 475d0c9

Improved error handling for supporting document uploads - change the colour of the label describing valid file format to red in Claim Create Page
st10251759 committed 4 days ago — dd3ace8

Commits on Oct 12, 2024

Verbosely Commented Logic in Controllers and Models, Added code attribution
st10251759 committed 5 days ago — c577a4e

Added styling to theAcademic Manager and Programme Coordinator views to be user friendly and consistent with my applications theme
st10251759 committed 5 days ago — b91b9bd

Updated the styling of track claim status page
st10251759 committed 5 days ago — 54cb296

Styled the lecturer track status of claims page, to be user friendly and attractive
st10251759 committed 5 days ago — 929b4d7

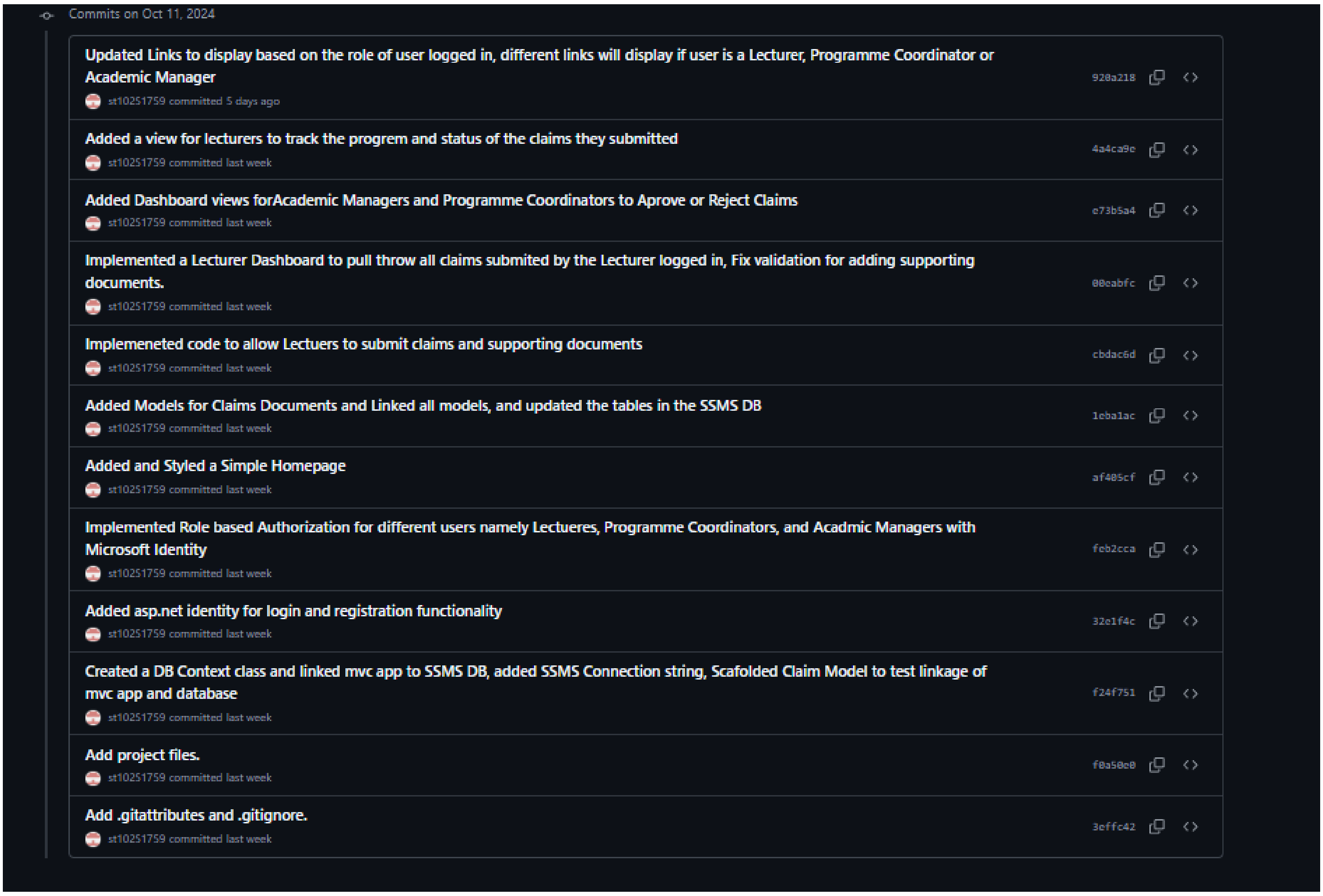
Added Styling for my Claim Create Page so that it is user friendly and consistent with my apps theme
st10251759 committed 5 days ago — f345eb6

Styled the Lecturer Dashboard, to be user friendly and conssitent with my apps theme
st10251759 committed 5 days ago — 94d80d0

Added Styling to the Nav bar and footer to be consistent with App Theme, Styled my Login and Register pages to be more attractive and user friendly
st10251759 committed 5 days ago — 01e3a49

Commits on Oct 11, 2024

Updated Links to display based on the role of user logged in, different links will display if user is a Lecturer, Programme Coordinator or Academic Manager
st10251759 committed 5 days ago — 920a218



Commits on Oct 11, 2024

Updated Links to display based on the role of user logged in, different links will display if user is a Lecturer, Programme Coordinator or Academic Manager
st10251759 committed 5 days ago — 920a218

Added a view for lecturers to track the progrem and status of the claims they submitted
st10251759 committed last week — 4a4ca9e

Added Dashboard views forAcademic Managers and Programme Coordinators to Aprove or Reject Claims
st10251759 committed last week — e73b5a4

Implemented a Lecturer Dashboard to pull throw all claims submited by the Lecturer logged in, Fix validation for adding supporting documents.
st10251759 committed last week — 80eabfc

Implemeneted code to allow Lectuers to submit claims and supporting documents
st10251759 committed last week — cbdac6d

Added Models for Claims Documents and Linked all models, and updated the tables in the SSMS DB
st10251759 committed last week — 1eba1ac

Added and Styled a Simple Homepage
st10251759 committed last week — af405cf

Implemented Role based Authorization for different users namely Lectueres, Programme Coordinators, and Acadmic Managers with Microsoft Identity
st10251759 committed last week — feb2cca

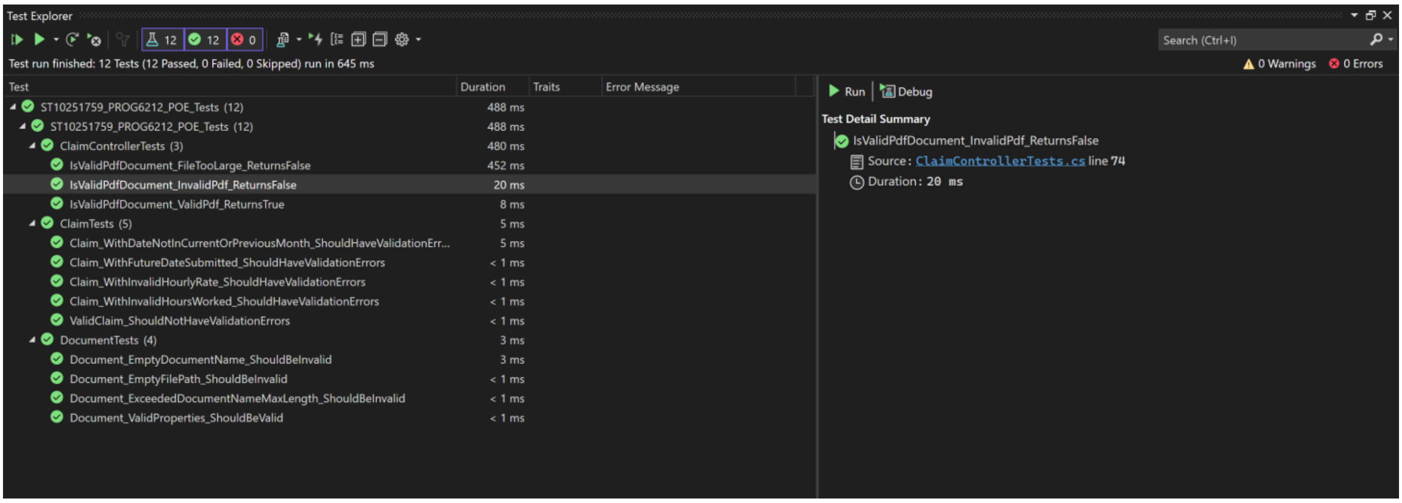
Added asp.net identity for login and registration functionality
st10251759 committed last week — 32e1f4c

Created a DB Context class and linked mvc app to SSMS DB, added SSMS Connection string, Scafolded Claim Model to test linkage of mvc app and database
st10251759 committed last week — f24f751

Add project files.
st10251759 committed last week — f0a50e0

Add .gitattributes and .gitignore.
st10251759 committed last week — 3effc42

Screenshots of Unit Tests Running



```
Test run finished: 12 Tests (12 Passed, 0 Failed, 0 Skipped) run in 645 ms

Test                                                                   Duration   Traits   Error Message
ST10251759_PROG6212_POE_Tests (12)                                       488 ms
  ST10251759_PROG6212_POE_Tests (12)                                     488 ms
    ClaimControllerTests (3)                                             480 ms
      IsValidPdfDocument_FileTooLarge_ReturnsFalse                       452 ms
      IsValidPdfDocument_InvalidPdf_ReturnsFalse                          20 ms
      IsValidPdfDocument_ValidPdf_ReturnsTrue                              8 ms
    ClaimTests (5)                                                         5 ms
      Claim_WithDateNotInCurrentOrPreviousMonth_ShouldHaveValidationErr...  5 ms
      Claim_WithFutureDateSubmitted_ShouldHaveValidationErrors            < 1 ms
      Claim_WithInvalidHourlyRate_ShouldHaveValidationErrors              < 1 ms
      Claim_WithInvalidHoursWorked_ShouldHaveValidationErrors             < 1 ms
      ValidClaim_ShouldNotHaveValidationErrors                            < 1 ms
    DocumentTests (4)                                                      3 ms
      Document_EmptyDocumentName_ShouldBeInvalid                           3 ms
      Document_EmptyFilePath_ShouldBeInvalid                              < 1 ms
      Document_ExceededDocumentNameMaxLength_ShouldBeInvalid              < 1 ms
      Document_ValidProperties_ShouldBeValid                              < 1 ms

Test Detail Summary
  IsValidPdfDocument_InvalidPdf_ReturnsFalse
    Source: ClaimControllerTests.cs line 74
    Duration: 20 ms
```

Code of Unit Test

```
============== ClaimControllerTests ==============

namespace ST10251759_PROG6212_POE_Tests

{//namepsace begin

    // TestFixture attribute indicates that this class contains test methods

    [TestFixture]
    public class ClaimControllerTests
    {
        // Declaring private fields for the controller, context, user manager, and test user
        private ClaimController _controller; // Instance of the ClaimController being tested
        private Prog6212DbContext _context; // In-memory database context for testing
        private UserManager<IdentityUser> _userManager; // User manager to manage identity users
        private IdentityUser _testUser; // Test user instance for performing tests

        // SetUp attribute indicates that this method runs before each test method
        [SetUp]
        public void Setup()
        {
            // Configuring the in-memory database for testing with a unique name for each run
            var options = new DbContextOptionsBuilder<Prog6212DbContext>()
                .UseInMemoryDatabase(databaseName: "TestDatabase" + Guid.NewGuid().ToString()) // Ensure a unique database name
                .Options; // Creating options for the context

            _context = new Prog6212DbContext(options); // Initializing the in-memory database context
            _userManager = CreateUserManager(); // Creating a mock UserManager instance

            // Initializing a test user with a unique ID and username
            _testUser = new IdentityUser
            {
                Id = Guid.NewGuid().ToString(), // Generating a unique ID for the test
user
                UserName = "testuser@example.com" // Setting a sample username for
the test user
            };

            // Adding the test user to the in-memory database context
            _context.Users.Add(_testUser);
            _context.SaveChanges(); // Saving changes to the in-memory database

            // Initializing the ClaimController with the context, user manager, and
null for the third parameter
            _controller = new ClaimController(_context, _userManager, null);
        }

        // TearDown attribute indicates that this method runs after each test method
        [TearDown]
        public void Dispose()
        {
            _context?.Dispose(); // Disposing of the context if it is not null to free
resources
            _controller?.Dispose(); // Disposing of the controller instance
            _userManager?.Dispose(); // Disposing of the user manager instance
        }

        // Test method for validating a correctly formatted PDF document
```

```
    [Test]
    public void IsValidPdfDocument_ValidPdf_ReturnsTrue()
    {
      // Arrange: Creating a valid PDF file as a FormFile object
      var validFile = new FormFile(new MemoryStream(new byte[100]), 0,
100, "Data", "valid.pdf")
      {
        Headers = new HeaderDictionary(), // Initializing headers for the
FormFile
        ContentType = "application/pdf" // Setting the content type to PDF
      };

      // Act: Calling the IsValidDocument method with the valid file
      var result = _controller.IsValidDocument(validFile);

      // Assert: Verifying that the result is true, indicating the document is valid
      Assert.IsTrue(result);
    }

    // Test method for validating an incorrectly formatted PDF document
    [Test]
    public void IsValidPdfDocument_InvalidPdf_ReturnsFalse()
    {
      // Arrange: Creating an invalid file (not a PDF)
      var invalidFile = new FormFile(new MemoryStream(new byte[100]), 0,
100, "Data", "invalid.txt")
      {
            Headers = new HeaderDictionary(), // Initializing headers for the
FormFile
            ContentType = "text/plain" // Setting the content type to plain text
        };

        // Act: Calling the IsValidDocument method with the invalid file
        var result = _controller.IsValidDocument(invalidFile);

        // Assert: Verifying that the result is false, indicating the document is
invalid
        Assert.IsFalse(result);
    }

    // Test method for validating a file that exceeds the maximum size limit
    [Test]
    public void IsValidPdfDocument_FileTooLarge_ReturnsFalse()
    {
        // Arrange: Creating a large PDF file as a FormFile object (20 MB)
        var largeFile = new FormFile(new MemoryStream(new byte[20 * 1024 *
1024]), 0, 20 * 1024 * 1024, "Data", "large.pdf") // 20 MB
        {
            Headers = new HeaderDictionary(), // Initializing headers for the
FormFile
            ContentType = "application/pdf" // Setting the content type to PDF
        };

        // Act: Calling the IsValidDocument method with the large file
        var result = _controller.IsValidDocument(largeFile);
```

```
            // Assert: Verifying that the result is false, indicating the file size exceeds
the limit
            Assert.IsFalse(result);
        }

        // Helper method to create a mock UserManager<IdentityUser>
        private UserManager<IdentityUser> CreateUserManager()
        {
            // Creating a mock object for IUserStore<IdentityUser>
            var store = new Mock<IUserStore<IdentityUser>>(); // Mocking the user
store interface
            // Initializing the UserManager with the mocked store and other
parameters set to null
            var userManager = new UserManager<IdentityUser>(
                store.Object, // Passing the mocked store object
                null, // Other parameters can be set as needed but are null for testing
                null,
                null,
                null,
                null,
                null,
                null,
                null
            );

            return userManager; // Returning the mocked UserManager instance
        }
```

```
    }
}//namespace end
```

============== ClaimControllerTests ==============

============== DocumentTests ==============

```
namespace ST10251759_PROG6212_POE_Tests
{
    [TestFixture]
    public class DocumentTests
    {
        [Test]
        public void Document_ValidProperties_ShouldBeValid()
        {
            // Arrange
            // Create a Document object with valid properties
            var document = new Document
            {
                DocumentName = "Test Document", // Valid name
                FilePath = "C:/Users/chett/source/repos/ST10251759_PROG6212_POE/wwwroot/uploads/161c81d7-50ab-4151-90b2-5e558a88d5f1_Test Document 2.pdf", // Valid file path
                UploadedOn = DateTime.Now, // Current date is valid
                ClaimId = 1 // Valid claim ID
            };

            // Act
            // Validate the document model
            var validationResults = ValidateModel(document);
            // Assert
            // Assert that there are no validation errors
            Assert.IsEmpty(validationResults);
        }

        [Test]
        public void Document_EmptyDocumentName_ShouldBeInvalid()
        {
            // Arrange
            // Create a Document object with an empty DocumentName
            var document = new Document
            {
                DocumentName = string.Empty, // Invalid: DocumentName is required
                FilePath = "C:/Users/chett/source/repos/ST10251759_PROG6212_POE/wwwroot/uploads/161c81d7-50ab-4151-90b2-5e558a88d5f1_Test Document 2.pdf", // Valid file path
                UploadedOn = DateTime.Now, // Current date is valid
                ClaimId = 1 // Valid claim ID
            };

            // Act
            // Validate the document model
            var validationResults = ValidateModel(document);

            // Assert
            // Assert that there are validation errors and check the error message
            Assert.IsNotEmpty(validationResults);
            Assert.AreEqual("Document Name is required.",
validationResults[0].ErrorMessage);
        }

        [Test]
        public void Document_EmptyFilePath_ShouldBeInvalid()
        {
            // Arrange
            // Create a Document object with an empty FilePath
            var document = new Document
            {
                DocumentName = "Test Document", // Valid name
                FilePath = string.Empty, // Invalid: FilePath is required
                UploadedOn = DateTime.Now, // Current date is valid
                ClaimId = 1 // Valid claim ID
            };

            // Act
            // Validate the document model
            var validationResults = ValidateModel(document);

            // Assert
            // Assert that there are validation errors and check the error message
            Assert.IsNotEmpty(validationResults);
            Assert.AreEqual("File Path is required.",
validationResults[0].ErrorMessage);
        }
        [Test]
        public void Document_ExceededDocumentNameMaxLength_ShouldBeInvalid()
        {
            // Arrange
            // Create a Document object with a DocumentName that exceeds the maximum length
            var document = new Document
            {
                DocumentName = new string('A', 256), // Invalid: Exceeds max length of 255
                FilePath = "C:/Documents/TestDocument.pdf", // Valid file path
                UploadedOn = DateTime.Now, // Current date is valid
                ClaimId = 1 // Valid claim ID
            };

            // Act
            // Validate the document model
            var validationResults = ValidateModel(document);

            // Assert
            // Assert that there are validation errors and check the error message
            Assert.IsNotEmpty(validationResults);
            Assert.AreEqual("The field DocumentName must be a string or array type with a maximum length of '255'.", validationResults[0].ErrorMessage);
        }

        private IList<ValidationResult> ValidateModel(Document document)
        {
            var validationResults = new List<ValidationResult>();
            var validationContext = new ValidationContext(document); // Create a validation context for the document
            Validator.TryValidateObject(document, validationContext, validationResults, true); // Perform validation
            return validationResults; // Return the list of validation results
        }
    }
}
============== DocumentTests ==============
============== ClaimsModelTest ==============
namespace ST10251759_PROG6212_POE_Tests
{
    [TestFixture]
    public class ClaimTests
    {
        [Test]
        public void ValidClaim_ShouldNotHaveValidationErrors()
        {
            // Arrange
            // Creating a valid Claim object with proper values
            var claim = new Claim
            {
                HoursWorked = 10,  // Valid hours worked
                HourlyRate = 200,  // Valid hourly rate
                TotalAmount = 1000, // Total amount calculated correctly (HoursWorked * HourlyRate)
                Notes = "This is a valid claim.", // Valid notes
                DateSubmitted = DateTime.Now // Current date for valid submission
            };

            // Act
            // Prepare to validate the claim object
            var validationResults = new List<ValidationResult>();
            var validationContext = new ValidationContext(claim);
            // Try to validate the object
            var isValid = Validator.TryValidateObject(claim, validationContext,
validationResults, true);

            // Assert
            // Ensure the claim is valid and has no validation errors
            Assert.IsTrue(isValid); // Expect isValid to be true
            Assert.IsEmpty(validationResults); // Expect validationResults to be
empty
        }

        [Test]
        public void
Claim_WithInvalidHoursWorked_ShouldHaveValidationErrors()
        {
            // Arrange
            // Creating a Claim object with invalid hours worked (0 hours)
            var claim = new Claim
            {
                HoursWorked = 0, // Invalid value (must be greater than 0)
                HourlyRate = 100, // Valid hourly rate
                TotalAmount = 1000, // This value may be incorrect but not validated in this case
                Notes = "Invalid claim due to hours worked.", // Valid notes
                DateSubmitted = DateTime.Now // Current date for submission
            };

            // Act
            // Prepare to validate the claim object
            var validationResults = new List<ValidationResult>();
            var validationContext = new ValidationContext(claim);
            // Try to validate the object
            var isValid = Validator.TryValidateObject(claim, validationContext, validationResults, true);

            // Assert
            // Ensure the claim is invalid and has validation errors
            Assert.IsFalse(isValid); // Expect isValid to be false
            Assert.IsNotEmpty(validationResults); // Expect validationResults to have errors
            Assert.IsTrue(validationResults.Exists(v => v.ErrorMessage == "Hours Worked must be between 1 and 100.")); // Expect specific error message
        }

        [Test]
        public void Claim_WithInvalidHourlyRate_ShouldHaveValidationErrors()
        {
            // Arrange
            // Creating a Claim object with an invalid hourly rate (less than the
minimum allowed)
            var claim = new Claim
            {
                HoursWorked = 10, // Valid hours worked
                HourlyRate = 40, // Invalid value (must be between 50 and 1000)
                TotalAmount = 400, // This value may be incorrect but not validated in
this case
                Notes = "Invalid claim due to hourly rate.", // Valid notes
                DateSubmitted = DateTime.Now // Current date for submission
            };

            // Act
            // Prepare to validate the claim object
            var validationResults = new List<ValidationResult>();
            var validationContext = new ValidationContext(claim);
            // Try to validate the object
            var isValid = Validator.TryValidateObject(claim, validationContext,
validationResults, true);

            // Assert
            // Ensure the claim is invalid and has validation errors
            Assert.IsFalse(isValid); // Expect isValid to be false
            Assert.IsNotEmpty(validationResults); // Expect validationResults to
have errors
            Assert.IsTrue(validationResults.Exists(v => v.ErrorMessage == "Hourly
Rate must be between 50 and 1000.")); // Expect specific error message
        }
```

```
    [Test]

    public void
Claim_WithFutureDateSubmitted_ShouldHaveValidationErrors()

    {

      // Arrange

      // Creating a Claim object with a future submission date

      var claim = new Claim

      {

        HoursWorked = 10, // Valid hours worked

        HourlyRate = 100, // Valid hourly rate

        TotalAmount = 1000, // Total amount calculated correctly
(HoursWorked * HourlyRate)

        Notes = "This claim has a future submission date.", // Valid notes

        DateSubmitted = DateTime.Now.AddDays(1) // Future date (invalid)

      };

      // Act

      // Prepare to validate the claim object

      var validationResults = new List<ValidationResult>();

      var validationContext = new ValidationContext(claim);

      // Try to validate the object

      var isValid = Validator.TryValidateObject(claim, validationContext,
validationResults, true);


      // Assert

      // Ensure the claim is invalid and has validation errors

      Assert.IsFalse(isValid); // Expect isValid to be false

      Assert.IsNotEmpty(validationResults); // Expect validationResults to
have errors
```

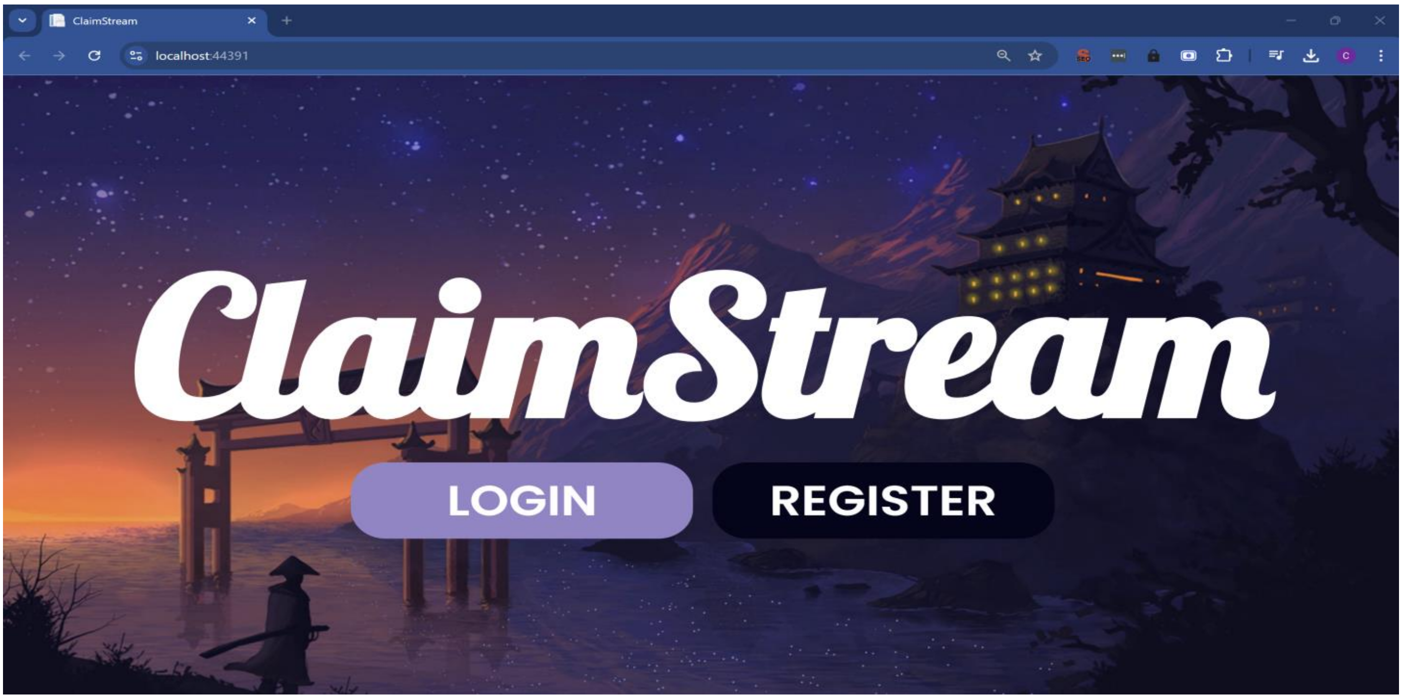
```
            Assert.IsTrue(validationResults.Exists(v => v.ErrorMessage == "Date
Submitted cannot be in the future.")); // Expect specific error message
        }


        [Test]
        public void
Claim_WithDateNotInCurrentOrPreviousMonth_ShouldHaveValidationErrors()
        {
            // Arrange
            // Creating a Claim object with a submission date not in the current or
previous month
            var claim = new Claim
            {
                HoursWorked = 10, // Valid hours worked
                HourlyRate = 100, // Valid hourly rate
                TotalAmount = 1000, // Total amount calculated correctly
(HoursWorked * HourlyRate)
                Notes = "Invalid submission date.", // Valid notes
                DateSubmitted = new DateTime(DateTime.Now.Year,
DateTime.Now.Month - 2, 1) // Submission date is two months ago (invalid)
            };
            // Act
            // Prepare to validate the claim object
            var validationResults = new List<ValidationResult>();
            var validationContext = new ValidationContext(claim);
            // Try to validate the object
            var isValid = Validator.TryValidateObject(claim, validationContext,
validationResults, true);
            // Assert
            // Ensure the claim is invalid and has validation errors
            Assert.IsFalse(isValid); // Expect isValid to be false
            Assert.IsNotEmpty(validationResults); // Expect validationResults to have errors
            Assert.IsTrue(validationResults.Exists(v => v.ErrorMessage == "Date Submitted can only be from the current month or previous month.")); // Expect specific error message
        }
    }
}
```

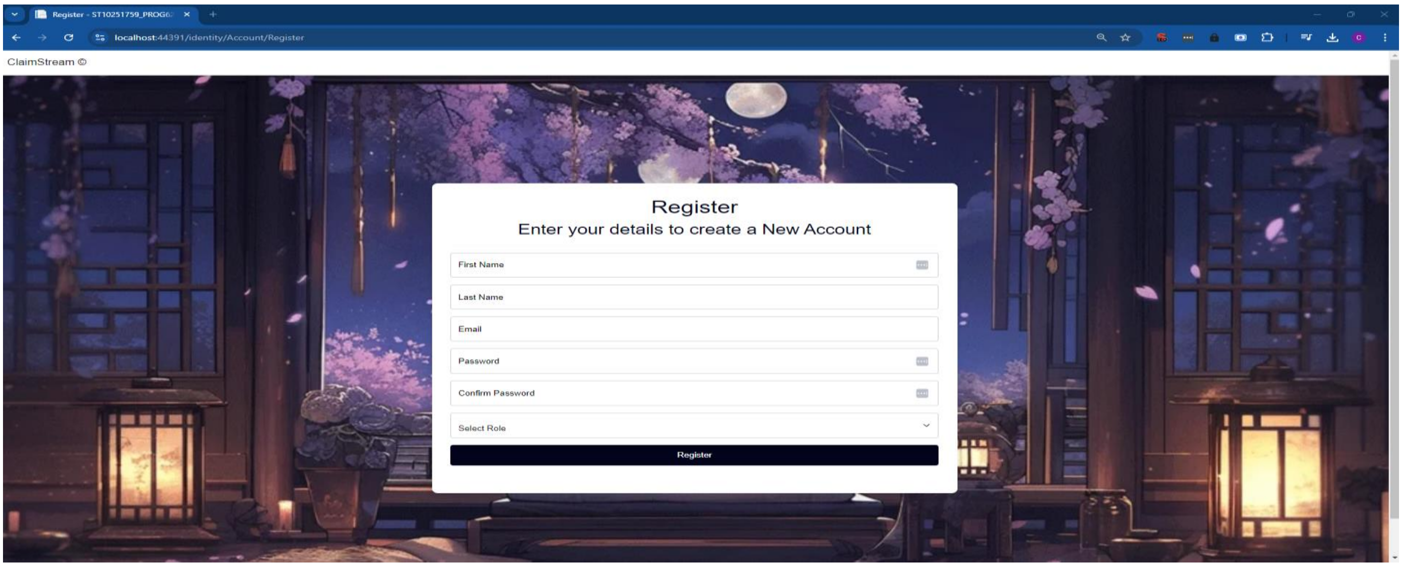
============= ClaimsModelTest=============

Screenshots of Web App Running

Home Screen



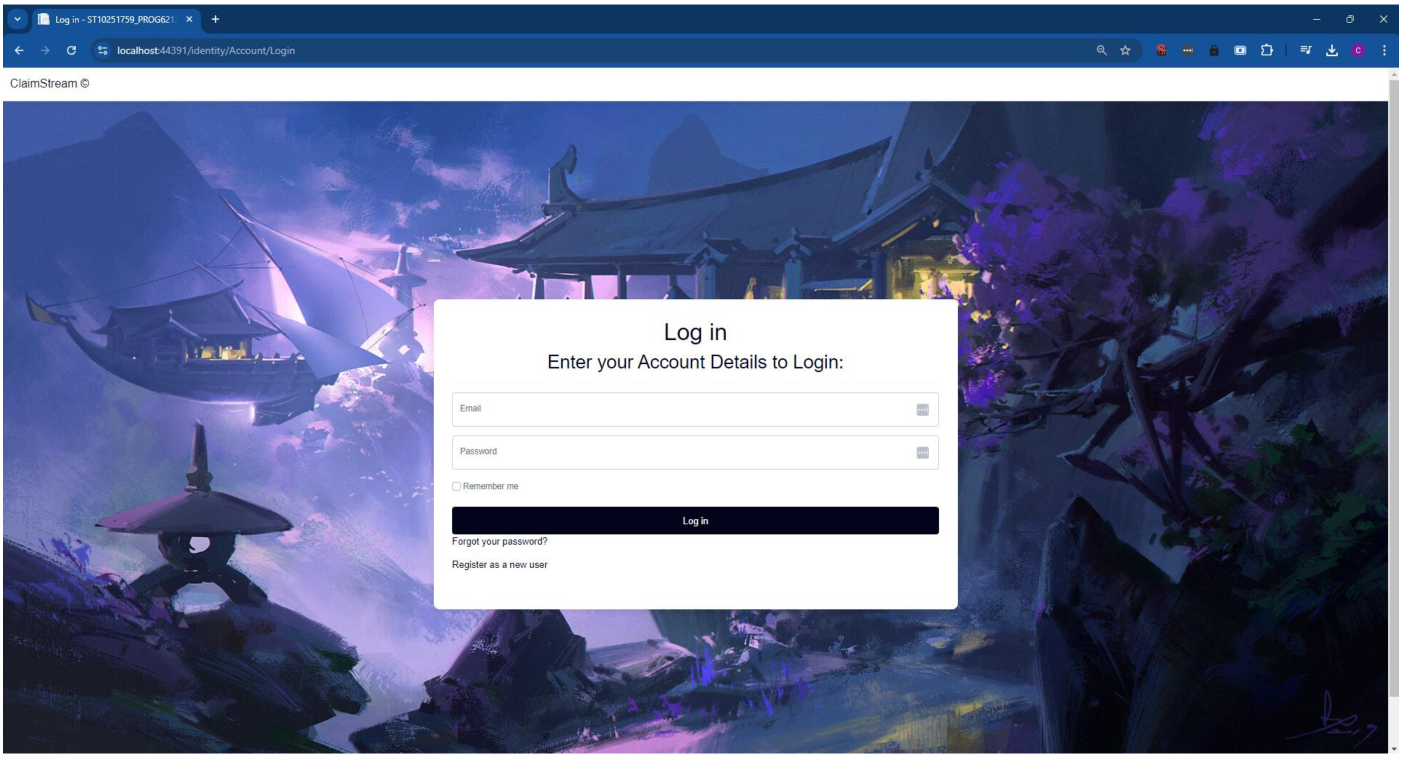
Register Page


Register - ST10251759_PROG6
localhost:44391/identity/Account/Register
ClaimStream ©

Register

Enter your details to create a New Account

Login Screen



localhost:44391/identity/Account/Login

ClaimStream ©

Log in

Enter your Account Details to Login:

Email

Password

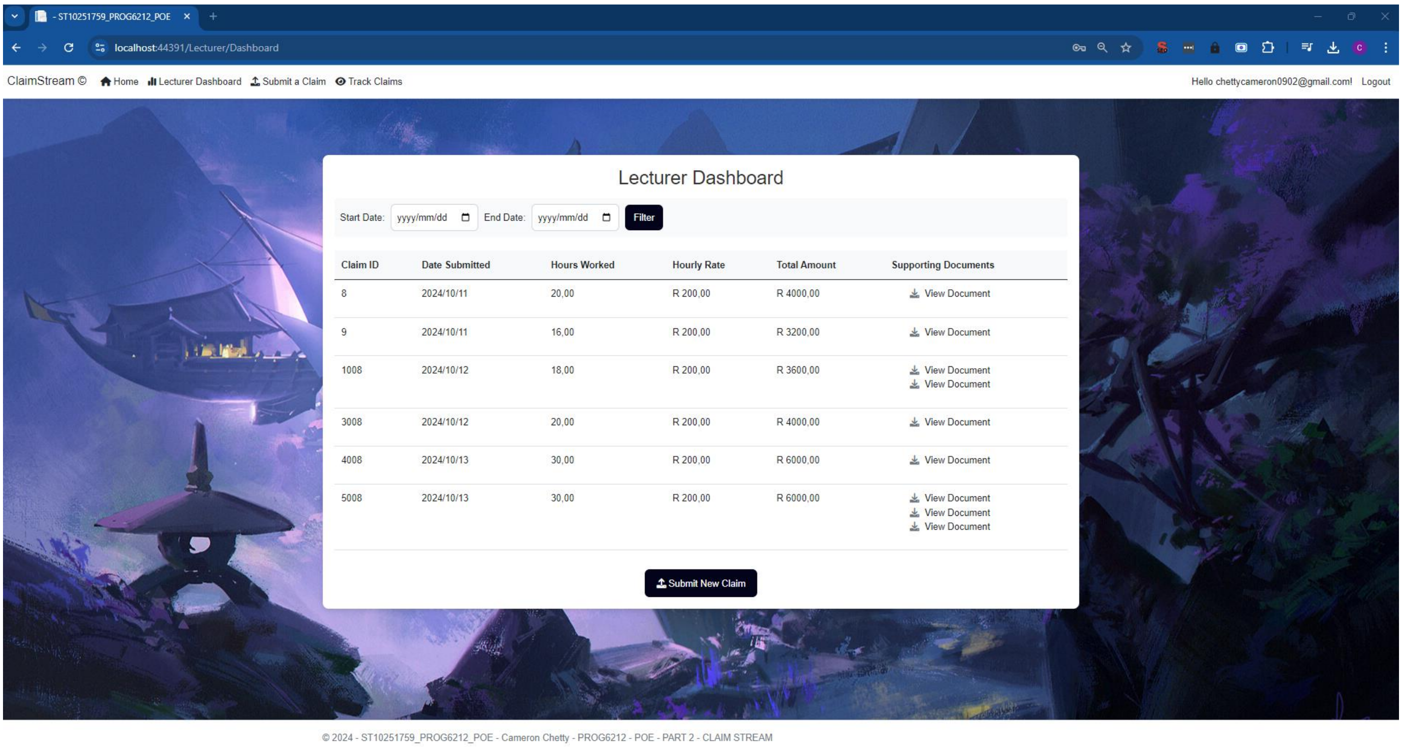
□ Remember me

Forgot your password?

Register as a new user

Lecturer Views:

Lecturer Dashboard Page



localhost:44391/Lecturer/Dashboard

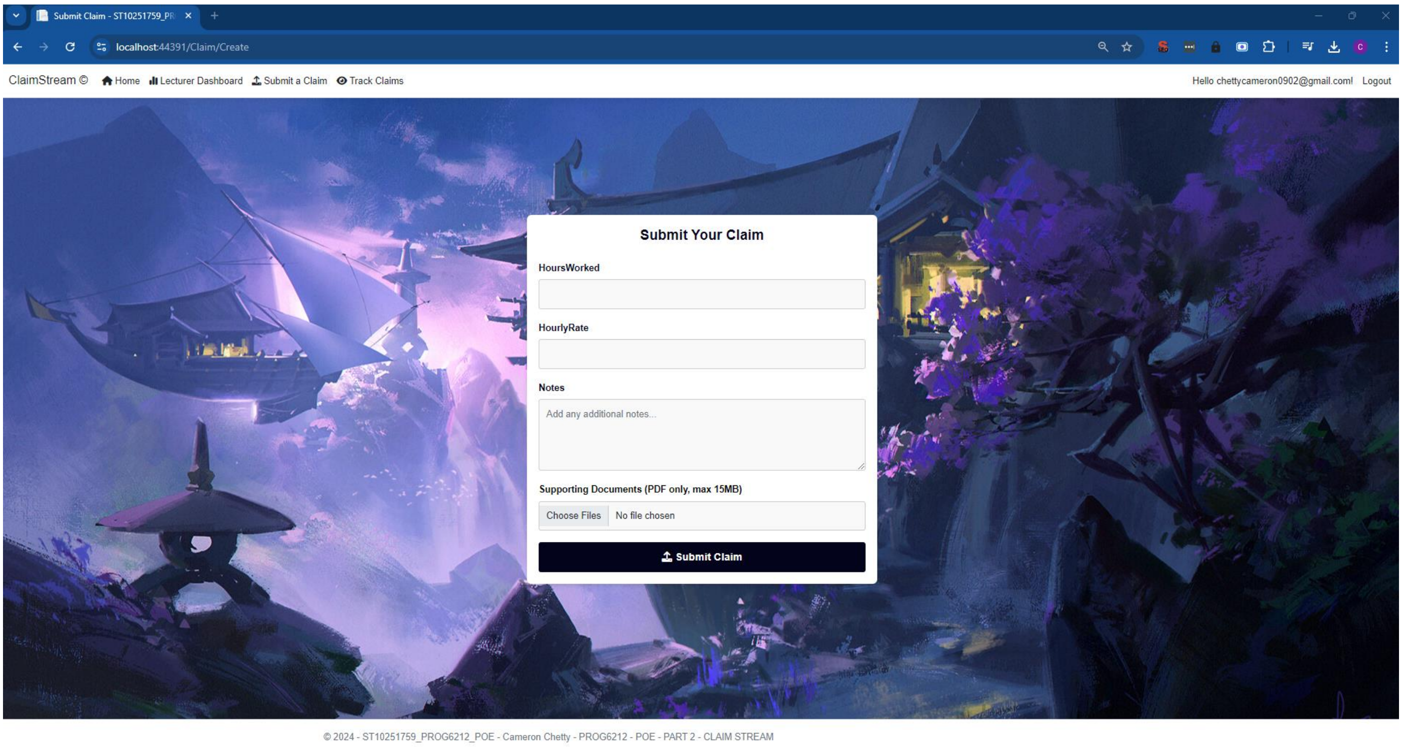
Lecturer Dashboard

Start Date: yyyy/mm/dd End Date: yyyy/mm/dd

Claim ID	Date Submitted	Hours Worked	Hourly Rate	Total Amount	Supporting Documents
8	2024/10/11	20,00	R 200,00	R 4000,00	View Document
9	2024/10/11	16,00	R 200,00	R 3200,00	View Document
1008	2024/10/12	18,00	R 200,00	R 3600,00	View Document View Document
3008	2024/10/12	20,00	R 200,00	R 4000,00	View Document
4008	2024/10/13	30,00	R 200,00	R 6000,00	View Document
5008	2024/10/13	30,00	R 200,00	R 6000,00	View Document View Document View Document

© 2024 - ST10251759_PROG6212_POE - Cameron Chetty - PROG6212 - POE - PART 2 - CLAIM STREAM

Submit Claim Page



Submit Your Claim

HoursWorked

HourlyRate

Notes

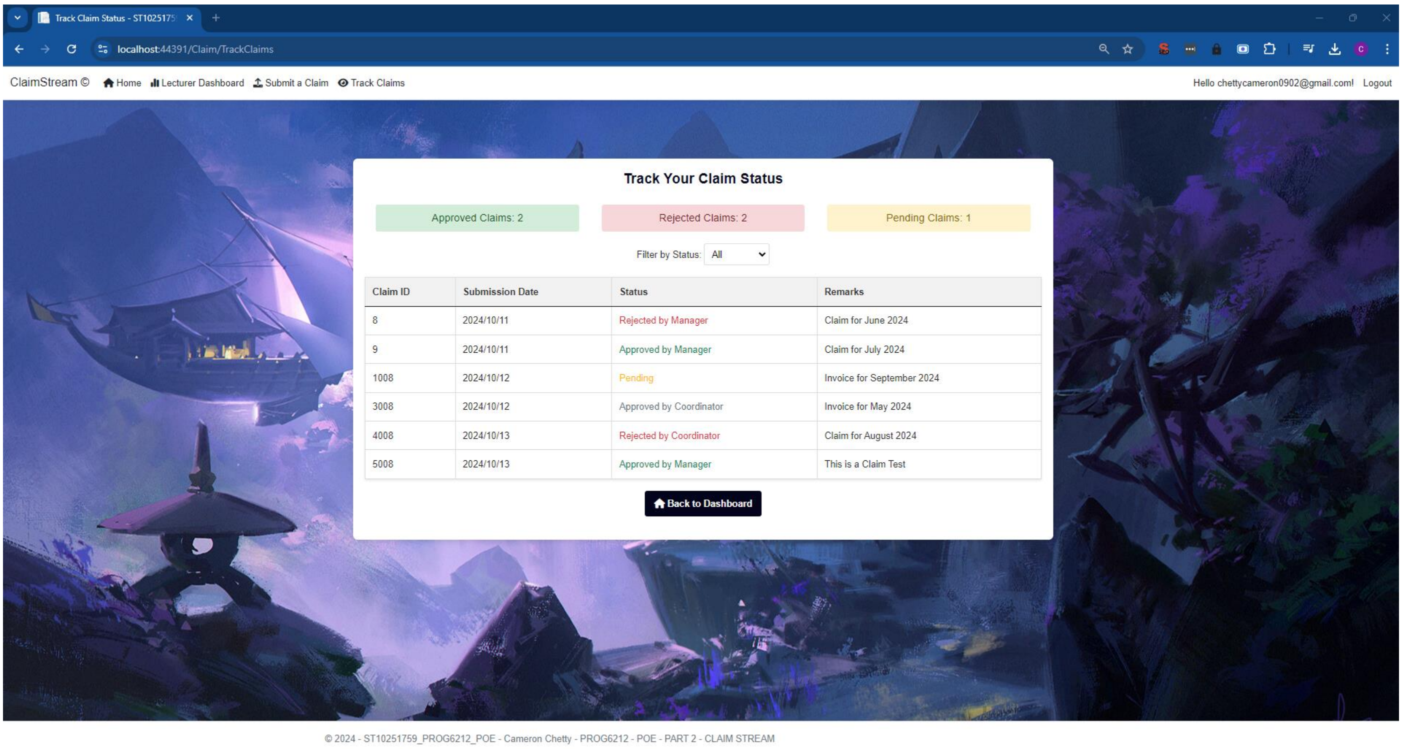
Add any additional notes...

Supporting Documents (PDF only, max 15MB)

No file chosen

© 2024 - ST10251759_PROG6212_POE - Cameron Chetty - PROG6212 - POE - PART 2 - CLAIM STREAM

Track Status Page


Track Claim Status - ST1025175
localhost:44391/Claim/TrackClaims
ClaimStream ©
Home
Lecturer Dashboard
Submit a Claim
Track Claims
Hello chettycameron0902@gmail.com!
Logout
Back to Dashboard

Track Your Claim Status

Approved Claims: 2

Rejected Claims: 2

Pending Claims: 1

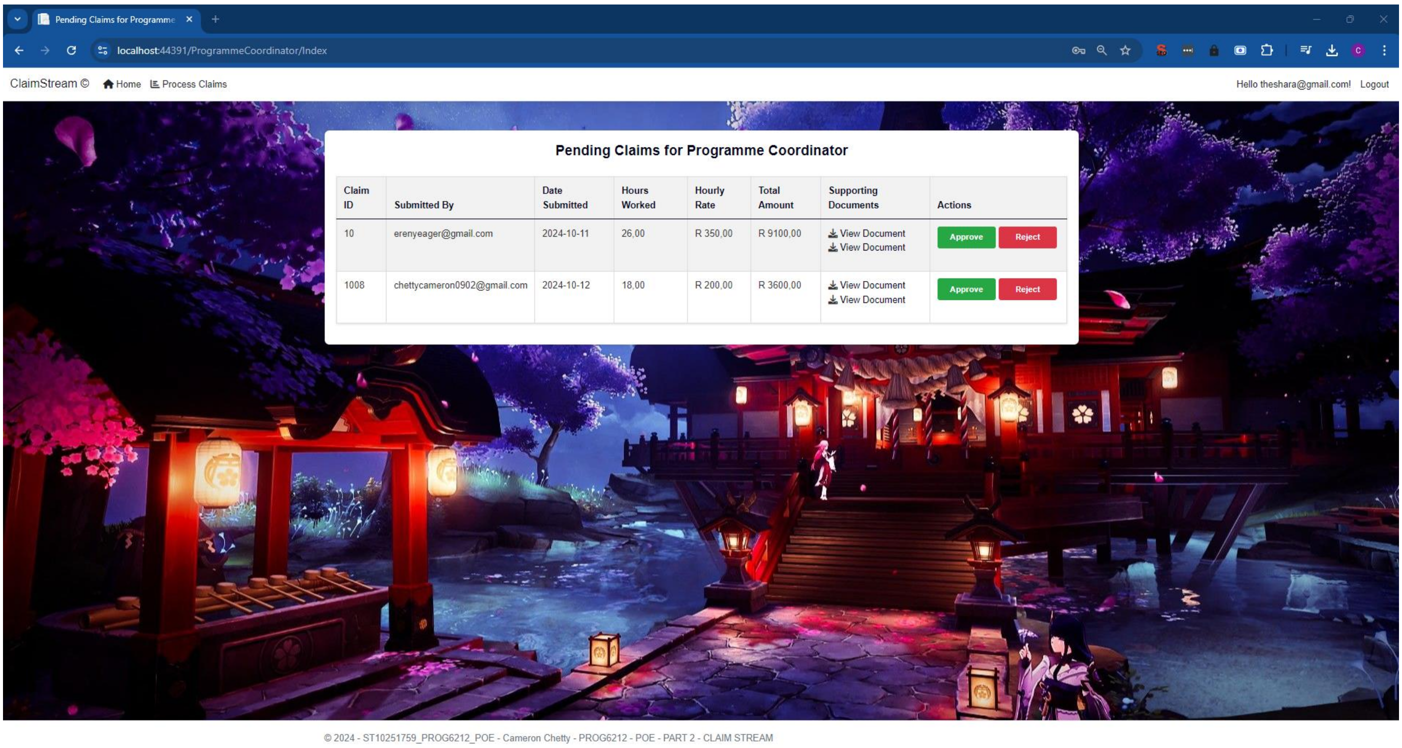
Filter by Status: All

Claim ID	Submission Date	Status	Remarks
8	2024/10/11	Rejected by Manager	Claim for June 2024
9	2024/10/11	Approved by Manager	Claim for July 2024
1008	2024/10/12	Pending	Invoice for September 2024
3008	2024/10/12	Approved by Coordinator	Invoice for May 2024
4008	2024/10/13	Rejected by Coordinator	Claim for August 2024
5008	2024/10/13	Approved by Manager	This is a Claim Test

© 2024 - ST10251759_PROG6212_POE - Cameron Chetty - PROG6212 - POE - PART 2 - CLAIM STREAM

Programme Coordinator View:

Programme Coordinator Dashboard Page


Pending Claims for Programme
localhost:44391/ProgrammeCoordinator/Index
ClaimStream ©
Home
Process Claims
Hello theshara@gmail.com!
Logout

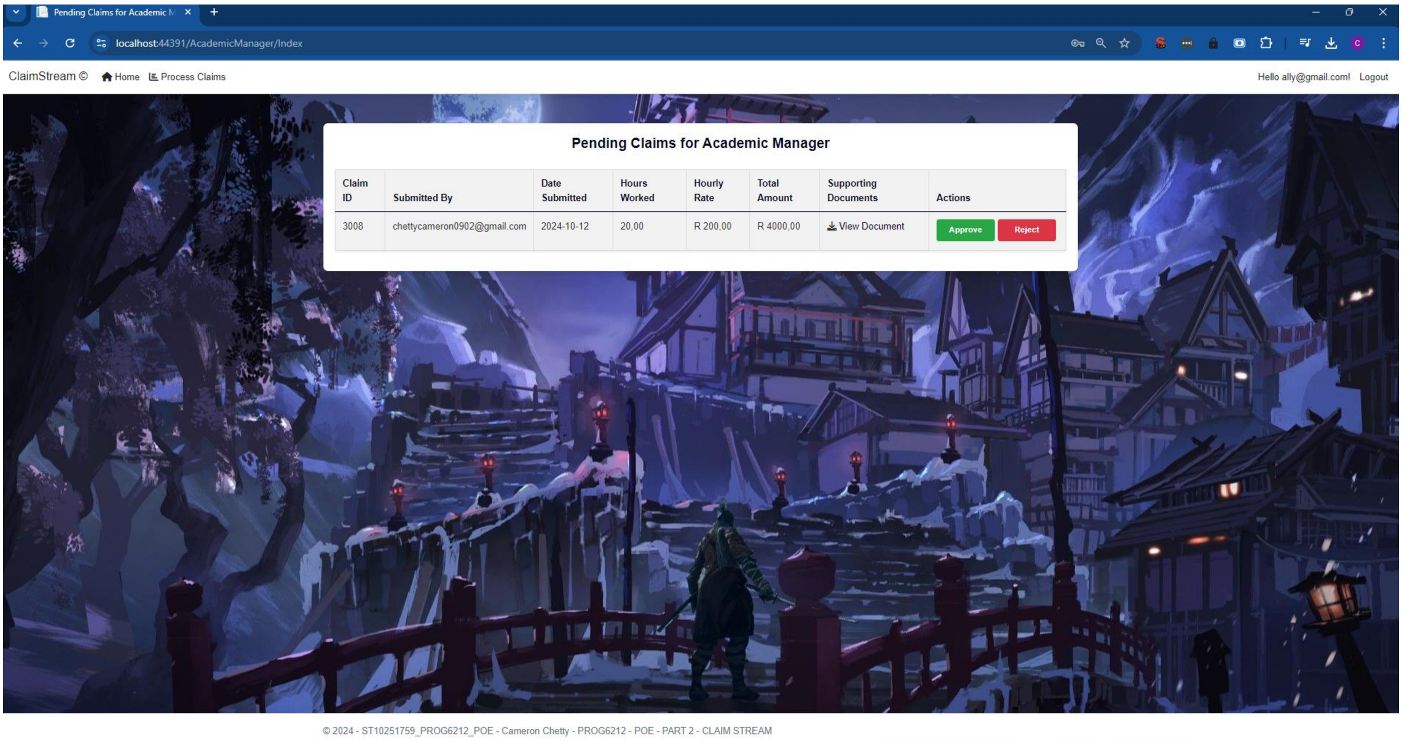
Pending Claims for Programme Coordinator

Claim ID	Submitted By	Date Submitted	Hours Worked	Hourly Rate	Total Amount	Supporting Documents	Actions
10	erenyeager@gmail.com	2024-10-11	26,00	R 350,00	R 9100,00	View Document View Document	Approve Reject
1008	chettycameron0902@gmail.com	2024-10-12	18,00	R 200,00	R 3600,00	View Document View Document	Approve Reject

© 2024 - ST10251759_PROG6212_POE - Cameron Chetty - PROG6212 - POE - PART 2 - CLAIM STREAM

Academic Manager View:

Academic Manager Dashboard Page


Pending Claims for Academic M
localhost:44391/AcademicManager/Index
ClaimStream ©
Home
Process Claims
Hello ally@gmail.com!
Logout

Pending Claims for Academic Manager

Claim ID	Submitted By	Date Submitted	Hours Worked	Hourly Rate	Total Amount	Supporting Documents	Actions
3008	chettycameron0902@gmail.com	2024-10-12	20,00	R 200,00	R 4000,00	View Document	Approve Reject

© 2024 - ST10251759_PROG6212_POE - Cameron Chetty - PROG6212 - POE - PART 2 - CLAIM STREAM

Code for Functionality

Application User Model

```
using Microsoft.AspNetCore.Identity; // Importing the ASP.NET Core Identity namespace for user authentication and management


namespace ST10251759_PROG6212_POE.Models // Defining a namespace for the project models
{
    // Defining a class ApplicationUser that extends the IdentityUser class provided by ASP.NET Core Identity
    public class ApplicationUser : IdentityUser
    {
        // Property to store the user's first name
        public string Firstname { get; set; } // Represents the user's first name

        // Property to store the user's last name
        public string Lastname { get; set; } // Represents the user's last name

        // Navigation property to establish a one-to-many relationship with the Claim class
        // This property will hold a collection of claims associated with this user
        public virtual ICollection<Claim> Claims { get; set; } // Represents the list of claims linked to the user
    }
}
```

Claim Model

```
using System.ComponentModel.DataAnnotations; // Importing data annotation attributes for model validation
using System.ComponentModel.DataAnnotations.Schema; // Importing attributes for defining database schema details
using System.Reflection.Metadata; // Importing metadata features, although this is not used in the current code
```

```
namespace ST10251759_PROG6212_POE.Models // Defining a namespace for the project models
{
    // Defining a Claim class to represent a claim in the application
    public class Claim
    {
        // ClaimId property: a unique identifier for each claim
        public int ClaimId { get; set; }


        // HoursWorked property: represents the number of hours worked
        [Required(ErrorMessage = "Hours Worked is required.")] // Validation to ensure this field is not empty
        [Range(1, 100, ErrorMessage = "Hours Worked must be between 1 and 100.")] // Validation to ensure the value is between 1 and 100
        public decimal HoursWorked { get; set; }


        // HourlyRate property: represents the rate per hour for the claim
        [Required(ErrorMessage = "Hourly Rate is required.")] // Validation to ensure this field is not empty
        [Range(50, 1000, ErrorMessage = "Hourly Rate must be between 50 and 1000.")] // Validation to ensure the value is between 50 and 1000
        public decimal HourlyRate { get; set; }


        // TotalAmount property: the total amount to be paid for the claim, calculated from HoursWorked and HourlyRate
        [Required] // Validation to ensure this field is not empty
        public decimal TotalAmount { get; set; }


        // Notes property: additional notes related to the claim, with a maximum length
        [MaxLength(500, ErrorMessage = "Notes can't exceed 500 characters.")] // Validation to restrict length to 500 characters
        public string Notes { get; set; }
        // DateSubmitted property: the date the claim was submitted
        [Required] // Validation to ensure this field is not empty
        [CustomValidation(typeof(Claim), nameof(ValidateSubmissionDate))] // Custom validation to check the submission date
        public DateTime DateSubmitted { get; set; }

        // Status property: tracks the current status of the claim, defaulting to "Pending"
        public string Status { get; set; } = "Pending";

        // Approval tracking properties to indicate if the claim has been approved by the coordinator and manager
        public bool IsApprovedByCoordinator { get; set; } = false; // Default is false (not approved)
        public bool IsApprovedByManager { get; set; } = false; // Default is false (not approved)

        // ApplicationUserId property: foreign key linking the claim to the user who submitted it
        [ForeignKey("ApplicationUser")] // Indicates that this property is a foreign key
        public string ApplicationUserId { get; set; } // User ID of the person making the claim

        // Navigation property: establishes a relationship with the ApplicationUser model
        public virtual ApplicationUser ApplicationUser { get; set; }

        // Documents property: a collection of documents associated with the claim
        public virtual ICollection<Document> Documents { get; set; } // Allows for multiple documents to be linked to a claim

        // Custom validation method for the DateSubmitted property
        public static ValidationResult ValidateSubmissionDate(DateTime dateSubmitted, ValidationContext context)
        {
            var currentDate = DateTime.Now; // Get the current date and time
            // Check if the submitted date is in the future
            if (dateSubmitted > currentDate)
            {
                return new ValidationResult("Date Submitted cannot be in the future."); // Return an error if it is
            }

            // Check if the submitted date is within the current or previous month
            if (dateSubmitted.Month != currentDate.Month && dateSubmitted.Month != currentDate.AddMonths(-1).Month)
            {
                return new ValidationResult("Date Submitted can only be from the current month or previous month."); // Return an error if it isn't
            }

            return ValidationResult.Success; // Return success if all validations pass
        }
    }
}
```

ClaimViewModel

```
using Microsoft.AspNetCore.Http; // Importing the namespace for HTTP-related functionalities, including file uploads
using System.Collections.Generic; // Importing the namespace for generic collections like List
using System.ComponentModel.DataAnnotations; // Importing the namespace for data annotations used for validation

namespace ST10251759_PROG6212_POE.Models // Defining the namespace for the models used in the project
{
    // Defining a view model class named ClaimViewModel
    // This class is used to transfer data between the view and the controller
    public class ClaimViewModel
    {
        // Property to hold the number of hours worked by the user
        // The Required attribute enforces that this field must be filled out
        [Required(ErrorMessage = "Hours Worked is required.")] // Custom error message if validation fails
        [Range(1, 100, ErrorMessage = "Hours Worked must be between 1 and 100.")] // Validates that the value must be between 1 and 100
        public decimal HoursWorked { get; set; } // Decimal property for storing hours worked

        // Property to hold the user's hourly rate
        // The Required attribute enforces that this field must be filled out
        [Required(ErrorMessage = "Hourly Rate is required.")] // Custom error message if validation fails
        [Range(50, 1000, ErrorMessage = "Hourly Rate must be between 50 and 1000.")] // Validates that the value must be between 50 and 1000
        public decimal HourlyRate { get; set; } // Decimal property for storing the hourly rate

        // Property to hold additional notes related to the claim
        // The MaxLength attribute restricts the length of the notes to a maximum of 500 characters
        [MaxLength(500, ErrorMessage = "Notes can't exceed 500 characters.")] // Custom error message if validation fails
        public string Notes { get; set; } // String property for storing notes

        // Property to hold a list of supporting documents for the claim
        // This property will accept a list of files uploaded by the user
        [Display(Name = "Supporting Documents")] // This attribute specifies the display name for the property in the view
        public List<IFormFile> SupportingDocuments { get; set; } // List of IFormFile to hold uploaded documents
    }
}
```

Document Model

```
using System.ComponentModel.DataAnnotations.Schema; // Importing the namespace for attributes related to database mapping, such as ForeignKey
using System.ComponentModel.DataAnnotations; // Importing the namespace for data annotations used for validation


namespace ST10251759_PROG6212_POE.Models // Defining the namespace for the models used in the project
{
    // Defining a class named Document
    // This class represents a document associated with a claim
    public class Document
    {
        // Property to uniquely identify each document in the database
        public int DocumentId { get; set; } // Auto-implemented property for storing the document's unique identifier


        // Property to store the name of the document
        // The Required attribute ensures that this field must be filled out
        [Required(ErrorMessage = "Document Name is required.")] // Custom error message to display if validation fails
        [MaxLength(255)] // Specifies that the maximum length for the document name is 255 characters
        public string DocumentName { get; set; } // String property for storing the name of the document


        // Property to store the file path of the uploaded document
        // The Required attribute ensures that this field must be filled out
        [Required(ErrorMessage = "File Path is required.")] // Custom error message to display if validation fails
        public string FilePath { get; set; } // String property for storing the file path of the document

        // Property to store the date and time when the document was uploaded
        [Required] // The Required attribute ensures that this field must be filled out
        public DateTime UploadedOn { get; set; } // DateTime property for storing the timestamp of when the document was uploaded

        // Property to establish a relationship between the Document and Claim classes
        [ForeignKey("Claim")] // This attribute indicates that the ClaimId property is a foreign key referencing the Claim table
        public int ClaimId { get; set; } // Integer property for storing the ID of the associated claim

        // Navigation property to represent the relationship between Document and Claim
        public virtual Claim Claim { get; set; } // Virtual property for accessing the related Claim object, allowing Entity Framework to load it automatically
    }
}
```

Lecturer Controller

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore; // Add this line
using ST10251759_PROG6212_POE.Data;
using ST10251759_PROG6212_POE.Models;
using System;
using System.Linq;
using System.Threading.Tasks;
```

```
namespace ST10251759_PROG6212_POE.Controllers
{//namespace begin
    //Using Micrisoft Idenitity with Roles - This line means that only users with the "Lecturer" role can access the actions in this controller.
    [Authorize(Roles = "Lecturer")]
    public class LecturerController : Controller
    {//Lecturer Controller begin

        //Private Field Declaration
        private readonly Prog6212DbContext _context; //This field holds an instance of Prog6212DbContext, which is used to interact with the database
        private readonly UserManager<IdentityUser> _userManager; //This field holds an instance of UserManager<IdentityUser>, which is part of ASP.NET Identity and is used for managing user information, including retrieving user details.


        //Constructor Method - initializes the _context and _userManager fields with instances provided via dependency injection. This allows the controller to use the database context and user management functionalities throughout its methods.
        public LecturerController(Prog6212DbContext context, UserManager<IdentityUser> userManager)
        {
            _context = context;
            _userManager = userManager;
        }

        //This action method retrieves claims related to the currently logged-in lecturer and can optionally filter them by submission date.
        public async Task<IActionResult> Dashboard(DateTime? startDate, DateTime? endDate)
        {
            // Get current logged-in user's ID
            var user = await _userManager.GetUserAsync(User);
            var userId = await _userManager.GetUserIdAsync(user);


            // Fetch claims for the logged-in lecturer
            var claimsQuery = _context.Claims
                .Include(c => c.Documents) // Include documents assosciated witht the claim
                .Where(c => c.ApplicationUserId == userId); //where the user logged in


            //Apply date filtering if dates are provided - This block checks if both startDate and endDate parameters are provided (i.e., not null).
            if (startDate.HasValue && endDate.HasValue)
            {
                claimsQuery = claimsQuery.Where(c => c.DateSubmitted >= startDate.Value && c.DateSubmitted <= endDate.Value);
            }


            // this line executes the query asynchronously and retrieves the results as a list of claims.
            var claims = await claimsQuery.ToListAsync();


            //eturns the claims list to the view, which will display the claims data to the user.
            return View(claims);
        }
    }//Lecturer Controller end
}//namespace end
```

Lecturer View

```
@model IEnumerable<ST10251759_PROG6212_POE.Models.Claim>


<div class="container-dashboard">
    <h2>Lecturer Dashboard</h2>


    <div class="filter-section">
```

```
        <form method="get" class="filter-form">
            <label for="start-date">Start Date:</label>
            <input type="date" id="start-date" name="startDate" value="@Context.Request.Query["startDate"]" class="form-control" />
            <label for="end-date">End Date:</label>
            <input type="date" id="end-date" name="endDate" value="@Context.Request.Query["endDate"]" class="form-control" />
            <button type="submit" class="btn-filter">Filter</button>
        </form>
    </div>

    <table class="table table-hover">
        <thead class="table-header">
            <tr>
                <th>Claim ID</th>
                <th>Date Submitted</th>
                <th>Hours Worked</th>
                <th>Hourly Rate</th>
                <th>Total Amount</th>
                <th>Supporting Documents</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var claim in Model)
            {
                <tr>
                    <td>@claim.ClaimId</td>
                    <td>@claim.DateSubmitted.ToShortDateString()</td>
                    <td>@claim.HoursWorked</td>
                    <td>R @claim.HourlyRate</td>
                    <td>R @claim.TotalAmount</td>
                    <td>
                        @if (claim.Documents != null && claim.Documents.Any())
                        {
                            <ul style="list-style-type: none;">
                                @foreach (var doc in claim.Documents)
                                {
                                    <li>
                                        <i class="fa-solid fa-download"></i>
                                        <a href="@Url.Content("~/uploads/" + doc.DocumentName)" target="_blank" style="text-decoration: none; color: #212529;">View Document</a>
                                    </li>
                                }
                            </ul>
                        }
                        else
                        {
                            <span>No Documents</span>
                        }
                    </td>
                </tr>
            }
        </tbody>
    </table>

    <div class="action-section">
        <a href="/Claim/Create" class="btn-submit"><i class="fa-solid fa-upload"></i> Submit New Claim</a>
    </div>
</div>
```

```
@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}

<style>
    body {
        font-family: Arial, sans-serif;
        /* background-color: #03041c; */
        background: url('/images/temple3.jpg') no-repeat center center fixed;
        background-size: cover;
        margin: 0;
        padding: 0;
    }

    /* Container Styling */
    .container-dashboard {
        margin: 6rem auto;
        width: 100%;
        max-width: 1300px;
        padding: 20px;
        background-color: #fff;
        border-radius: 10px;
        box-shadow: 0px 4px 10px rgba(0, 0, 0, 0.1);
    }

    h2 {
        text-align: center;
        color: #333;
        margin-bottom: 1rem;
        font-size: 2rem;
    }

    /* Filter Section */
    .filter-section {
        margin-bottom: 1.5rem;
        background-color: #f9fafb;
        padding: 10px;
        border-radius: 8px;
    }

    .filter-form {
        display: flex;
        align-items: center; /* Align items vertically */
        justify-content: flex-start; /* Align items to the left */
        gap: 10px; /* Space between items */
    }

    .filter-section label {
        font-size: 1rem;
    }

    .form-control {
        padding: 10px;
        border-radius: 8px;
        border: 1px solid #ced4da;
        width: 150px; /* Adjust width to fit better */
    }

    .btn-filter {
        background-color: #03041c;
        border: none;
        padding: 10px 15px;
        color: white;
        border-radius: 8px;
        cursor: pointer;
        transition: background-color 0.3s ease;
    }

        .btn-filter:hover {
            background-color: #555;
        }

    /* Table Styling */
    .table {
        width: 100%;
        border-collapse: collapse;
        margin-bottom: 2rem;
    }

        .table th, .table td {
            padding: 12px;
            text-align: left;
            border-bottom: 1px solid #dee2e6;
        }

    .table-header {
        background-color: #f8f9fa;
    }
```

```
.table-hover tbody tr:hover {
    background-color: #f1f3f5;
}

/* Action Section */
.action-section {
    text-align: center;
}

.btn-submit {
    background-color: #03041c;
    border: none;
    padding: 12px 20px;
    color: white;
    font-size: 1rem;
    border-radius: 8px;
    text-decoration: none;
    display: inline-block;
    transition: background-color 0.3s ease;
}

    .btn-submit:hover {
        background-color: #fff;
        color: #03041c;
        border: 1px solid #03041c;
    }

/* Icons Styling */
.fa-download {
    margin-right: 5px;
```

```
        color: #6c757d;
    }
</style>
```

Claim Controller

```
//Import List
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using ST10251759_PROG6212_POE.Data;
using ST10251759_PROG6212_POE.Models;
using System;

namespace ST10251759_PROG6212_POE.Controllers
{//namespace begin
    //Using Micrisoft Idenitity with Roles - This line means that only users with the "Lecturer" role can access the actions in this controller.
    [Authorize(Roles = "Lecturer")]
    public class ClaimController : Controller
    {//ClaimController class begin


        //private fields declaration - These lines declare three private fields that will be used throughout the controller.
        private readonly Prog6212DbContext _context; //interact with the database.
        private readonly UserManager<IdentityUser> _userManager; //helps manage user accounts, like retrieving the currently logged-in user.
        private readonly IWebHostEnvironment _environment; //provides information about the web hosting environment
        //Constructor method - initializes the private fields with the values passed in, allowing the controller to use them for database access, user management, and environment information.
        public ClaimController(Prog6212DbContext context, UserManager<IdentityUser> userManager, IWebHostEnvironment environment)
        {//Construct begin
            _context = context;
            _userManager = userManager;
            _environment = environment;
        }//constructor end


        // GET: Claim/Create - This method responds to GET requests to the "Claim/Create" URL. It simply returns a view (web page) for creating a new claim.
        public IActionResult Create()
        {
            return View();
        }


        // POST: Claim/Create - This method handles POST requests to submit a new claim
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create(ClaimViewModel model)
        {
            // Validate model state - checks if the incoming data (from ClaimViewModel) is valid. If not, it returns the same view with the current model to show any validation errors.
            if (!ModelState.IsValid)
            {
                return View(model);
            }


            // Validate supporting documents - This block checks if the user has uploaded any supporting documents. If none are found, it adds an error message
```

```
if (model.SupportingDocuments == null || model.SupportingDocuments.Count == 0)
{
    ModelState.AddModelError("", "At least one supporting document must be attached.");
    return View(model);
}


// Validate file types and sizes - checks if the document uploaded is a valid document - of the type PDF and the size of the document is no greater than 15 MB - if this is not true returns current model with errors
bool isInvalidFile = false; //flag variable for invalid file (not a PDF)
foreach (var file in model.SupportingDocuments)
{
    if (file.ContentType != "application/pdf" || file.Length > 15 * 1024 * 1024)
    {
        ViewBag.InvalidFile = true; //assign a viewbag variable to true - indicated user is tryinhg to upload an invalid file - this variable in the view willbe used to change the label describing the correct file format to red
        isInvalidFile = true;
        ModelState.AddModelError("", "Only PDF files under 15 MB are allowed.");
        return View(model);
    }
}


// If the model is valid and documents are valid, proceed to create the claim
if (!isInvalidFile)
{
    var user = await _userManager.GetUserAsync(User);


    //Creates a new claim object - retrives the HoursWorked, HourlyRate and Notes from the view the user interacts with, and also stores the user id of the user currently logged in
```

```
var claim = new Claim
{
    HoursWorked = model.HoursWorked,
    HourlyRate = model.HourlyRate,
    Notes = model.Notes,
    DateSubmitted = DateTime.Now,
    ApplicationUserId = user.Id,
    TotalAmount = model.HourlyRate * model.HoursWorked

};

//adds the claim to the database table and saves changes
_context.Claims.Add(claim);
await _context.SaveChangesAsync();

// Handle file upload
var uploadsFolder = Path.Combine(_environment.WebRootPath, "uploads");

// for each loop that goes through each file in the SupportingDocuments list of the model. Each file represents a document that the user uploaded to support their claim.
foreach (var file in model.SupportingDocuments)
{
    //generates a new unique identifier (GUID). This ensures that every file has a unique name, even if multiple files with the same original name are uploaded.
    var uniqueFileName = Guid.NewGuid().ToString() + "_" + file.FileName;
    var filePath = Path.Combine(uploadsFolder, uniqueFileName);

    // Ensure directory exists
    Directory.CreateDirectory(uploadsFolder);
```

```
                    // Save file
                    using (var fileStream = new FileStream(filePath, FileMode.Create))
                    {
                        await file.CopyToAsync(fileStream);
                    }

                    // Create document entry and link it to the claim - A new Document object is created to represent the uploaded file in the database.
                    var document = new Document
                    {
                        ClaimId = claim.ClaimId,
                        DocumentName = uniqueFileName,
                        FilePath = filePath
                    };

                    //This line adds the newly created document object to the _context.Documents collection. This prepares the document to be saved in the database when changes are committed.
                    _context.Documents.Add(document);
                }

                await _context.SaveChangesAsync();

                //This line stores a success message in TempData. This is a temporary data storage mechanism that allows the message to be displayed to the user on the next page they visit
                TempData["SuccessMessage"] = "Claim submitted successfully!";
                //this line redirects the user to the "Dashboard" action of the "Lecturer" controller.
                return RedirectToAction("Dashboard", "Lecturer");
            }
            return View(model);
        }
        // GET: Claims/Track
        public async Task<IActionResult> TrackClaims()
        {
            // Get the logged-in user
            var currentUser = await _userManager.GetUserAsync(User);

            // Fetch claims for the current user
            var claims = _context.Claims
                .Where(c => c.ApplicationUserId == currentUser.Id)
                .ToList();
            //pass the list of claims to the view
            return View(claims);
        }
    }//ClaimController class end

}//namespace end
```

Claim Create View

```
@model ST10251759_PROG6212_POE.Models.ClaimViewModel

@{
    ViewData["Title"] = "Submit Claim";
}

<div class="container-claims">
    <h2>Submit Your Claim</h2>

    <form asp-action="Create" enctype="multipart/form-data" method="post">
        <div class="form-group">
```

```
            <label asp-for="HoursWorked"></label>
            <input asp-for="HoursWorked" class="form-control" required />
            <span asp-validation-for="HoursWorked" class="text-danger"></span>
        </div>

        <div class="form-group">
            <label asp-for="HourlyRate"></label>
            <input asp-for="HourlyRate" class="form-control" required />
            <span asp-validation-for="HourlyRate" class="text-danger"></span>
        </div>

        <div class="form-group">
            <label asp-for="Notes"></label>
            <textarea asp-for="Notes" class="form-control" rows="4" placeholder="Add any additional notes..."></textarea>
            <span asp-validation-for="Notes" class="text-danger"></span>
        </div>

        <div class="form-group">
            <label for="SupportingDocuments" style="@(ViewBag.InvalidFile != null && (bool)ViewBag.InvalidFile ? "color: red;" : "color: black;")">Supporting Documents (PDF only, max 15MB)</label>
            <input type="file" name="SupportingDocuments" class="form-control" multiple required />
            <span asp-validation-for="SupportingDocuments" class="text-danger"></span>
        </div>

        <button type="submit" class="btn-claims btn-primary"><i class="fa-solid fa-upload"></i> Submit Claim</button>
    </form>
</div>
```

```
@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}

<style>
    body {
        font-family: Arial, sans-serif;
        /* background-color: #03041c; */
        background: url('/images/temple3.jpg') no-repeat center center fixed;
        background-size: cover;
        color: #03041c;
        margin: 0;
        padding: 0;
    }

    .container-claims {
        max-width: 600px;
        margin: 200px auto;
        padding: 20px;
        background-color: #ffffff;
        box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
        border-radius: 8px;
    }

    h2 {
        text-align: center;
        color: #03041c;
        margin-bottom: 30px;
        font-size: 24px;
        font-weight: bold;
    }

    .form-group {
        margin-bottom: 20px;
    }

    label {
        display: block;
        margin-bottom: 8px;
        font-weight: bold;
        color: #03041c;
    }

    .form-control {
        width: 100%;
        padding: 12px;
        font-size: 16px;
        border: 1px solid #ccc;
        border-radius: 4px;
        background-color: #f9f9f9;
        transition: border-color 0.3s ease-in-out;
    }

        .form-control:focus {
            border-color: #03041c;
            outline: none;
        }

    .btn-claims {
        width: 100%;
        padding: 12px;
        font-size: 18px;
        font-weight: bold;
        color: #ffffff;
        background-color: #03041c;
        border: none;
        border-radius: 4px;
        cursor: pointer;
        transition: background-color 0.3s ease-in-out;
    }

        .btn-claims:hover {
            background-color: #fff;
            color: #03041c;
            border: 1px solid #03041c;
        }

    .text-danger {
        font-size: 14px;
        color: #d9534f; /* Bootstrap danger color */
    }
</style>
```

Claim TrackClaims View

```
@model IEnumerable<ST10251759_PROG6212_POE.Models.Claim>
@{
    ViewBag.Title = "Track Claim Status";

    // Helper methods to determine CSS classes based on status
    string GetStatusClass(string status)
    {
        return status switch
        {
            "Approved by Manager" => "approved",
            "Rejected by Coordinator" => "rejected",
            "Rejected by Manager" => "rejected",
            "Pending" => "pending",
            _ => "unknown" // Default class for any unexpected status
        };
    }

    string GetRowClass(string status)
    {
        return status switch
        {
            "Approved by Manager" => "text-success",
            "Rejected by Coordinator" => "text-danger",
            "Rejected by Manager" => "text-danger",
            "Pending" => "text-warning",
            _ => "text-muted" // Default text class
        };
    }
}

<div class="container-claims">
    <h2 style="color: #03041c;">Track Your Claim Status</h2>
    <!-- Status Summary Section -->
    <div class="status-summary">
        <div class="status-item approved">
```

```
            Approved Claims: <span id="approved-count">@Model.Count(c => c.Status == "Approved by Manager")</span>
        </div>
        <div class="status-item rejected">
            Rejected Claims: <span id="rejected-count">@Model.Count(c => c.Status == "Rejected by Coordinator" || c.Status == "Rejected by Manager")</span>
        </div>
        <div class="status-item pending">
            Pending Claims: <span id="pending-count">@Model.Count(c => c.Status == "Pending")</span>
        </div>
    </div>

    <!-- Filter Section -->
    <div class="filter-section">
        <label for="status-filter">Filter by Status:</label>
        <select id="status-filter" class="status-filter" onchange="filterClaims()">
            <option value="All">All</option>
            <option value="Approved">Approved</option>
            <option value="Rejected">Rejected</option>
            <option value="Pending">Pending</option>
        </select>
    </div>

    <!-- Claims Table -->
    <table class="table">
        <thead>
            <tr>
                <th>Claim ID</th>
                <th>Submission Date</th>
                <th>Status</th>
            <th>Remarks</th>
          </tr>
        </thead>
        <tbody id="claims-table-body">
          @foreach (var claim in Model)
          {
            <tr class="@GetStatusClass(claim.Status)" data-status="@claim.Status">
              <td>@claim.ClaimId</td>
              <td>@claim.DateSubmitted.ToString("yyyy/MM/dd")</td>
              <td class="@GetRowClass(claim.Status)">@claim.Status</td>
              <td>@claim.Notes</td>
            </tr>
          }
        </tbody>
    </table>

    <a href="/Lecturer/Dashboard" class="btn-submit" style="text-decoration:none;"><i
class="fa-solid fa-house"></i> Back to Dashboard</a>
</div>

<script>
    function filterClaims() {
        const filterValue = document.getElementById("status-filter").value.toLowerCase();
        const claimsTable = document.getElementById("claims-table-body");
        const rows = claimsTable.getElementsByTagName("tr");

        for (let i = 0; i < rows.length; i++) {
            const row = rows[i];
            const status = row.getAttribute("data-status").toLowerCase();
            if (filterValue === "all" || status === filterValue) {
                row.style.display = ""; // Show the row
            } else {
                row.style.display = "none"; // Hide the row
            }
        }
    }
</script>

<style>
    body {
        font-family: Arial, sans-serif;
        /* background-color: #03041c; */
        background: url('/images/temple3.jpg') no-repeat center center fixed;
        background-size: cover;
        color: #333;
        margin: 0;
        padding: 0;
    }

    .container-claims {
        max-width: 1200px;
        margin: 100px auto;
        padding: 20px;
        background-color: #ffffff;
        box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
        border-radius: 8px;
    }

    h2 {
```

```
    text-align: center;
    color: #03041c;
    margin-bottom: 30px;
    font-size: 24px;
    font-weight: bold;
}

.status-summary {
    display: flex;
    justify-content: space-around;
    margin-bottom: 20px;
}

.status-item {
    font-size: 18px;
    padding: 10px;
    border-radius: 4px;
    text-align: center;
    width: 30%;
}

.approved {
    background-color: #d4edda;
    color: #155724;
}

.rejected {
    background-color: #f8d7da;
    color: #721c24;
}
```

```
.pending {
    background-color: #fff3cd;
    color: #856404;
}

.filter-section {
    margin-bottom: 20px;
    text-align: center;
}

.status-filter {
    padding: 8px;
    font-size: 16px;
    border-radius: 4px;
    border: 1px solid #ddd;
}

.table {
    width: 100%;
    margin-bottom: 20px;
    border-collapse: collapse;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

    .table th, .table td {
        padding: 12px;
        border: 1px solid #ddd;
        text-align: left;
        color: #333;
    }

    .table th {
      background-color: #f2f2f2;
      color: #333;
      font-weight: bold;
    }

    .table td {
      background-color: #ffffff;
    }

    .table .approved {
      color: green; /* Text for Approved status */
    }

    .table .rejected {
      color: red; /* Text for Rejected status */
    }

    .table .pending {
      color: orangered; /* Text for Pending status */
    }

  .btn-submit {
    display: block;
    width: 200px;
    margin: 20px auto;
    padding: 10px;
    font-size: 16px;
```

```
        font-weight: bold;
        border: none;
        border-radius: 4px;
        cursor: pointer;
        background-color: #03041c;
        color: #ffffff;
        text-align: center;
        transition: background-color 0.3s ease-in-out;
    }

        .btn-submit:hover {
            background-color: #fff;
            color: #03041c;
            border: 1px solid #03041c;
        }
</style>

<script>
    function filterClaims() {
        const filterValue = document.getElementById("status-filter").value.toLowerCase();
        const claimsTable = document.getElementById("claims-table-body");
        const rows = claimsTable.getElementsByTagName("tr");

        for (let i = 0; i < rows.length; i++) {
            const row = rows[i];
            const status = row.getAttribute("data-status").toLowerCase();

            if (filterValue === "all" || status.includes(filterValue)) {
                row.style.display = ""; // Show the row
            } else {
                row.style.display = "none"; // Hide the row
            }
        }
    }

    // Optional: Add a method to determine the class based on the status
    function GetStatusClass(status) {
        if (status.includes("Approved by Manager")) return "approved";
        if (status.includes("Rejected")) return "rejected";
        if (status.includes("Pending")) return "pending";
        return "";
    }
</script>
```

Programme Coordinator Controller

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using ST10251759_PROG6212_POE.Data;
using System.Linq;
using System.Threading.Tasks;

namespace ST10251759_PROG6212_POE.Controllers
{//namespace begin
    //Using Micrisoft Idenitity with Roles - This line means that only users with the "Programme Coordinator" role can access the actions in this controller.
    [Authorize(Roles = "Programme Coordinator")]
    public class ProgrammeCoordinatorController : Controller
    {//ProgrammeCoordinator controller begin

        //private fields declaration - used throughout controller
        private readonly Prog6212DbContext _context;


        //constructor - - initializes the private fields with the values passed in, allowing the
controller to use them for database access
        public ProgrammeCoordinatorController(Prog6212DbContext context)
        {
            _context = context;
        }


        //This method is responsible for displaying a list of claims pending approval from the
programme coordinator.
        public IActionResult Index()
        {
            // Only show claims that are pending and not yet approved by the coordinator
            var pendingClaims = _context.Claims
                .Include(c => c.ApplicationUser) // Include the ApplicationUser
                .Include(c => c.Documents) // Include the Documents
                .Where(c => !c.IsApprovedByCoordinator && c.Status == "Pending") //These are
the conditions for claims that can be vuewd by the programme Coordinator
                .ToList();


            //returns the pendingClaims list to the view, which will display it to the user.
            return View(pendingClaims);
        }


        //This method processes the approval of a claim when a POST request is made - when
Approve button is clicked - takes an integer parameter representing the ID of the claim to be
approved.
        [HttpPost]
        public async Task<IActionResult> Approve(int claimId)
        {
            //This line retrieves the claim from the database using the provided claimId
            var claim = await _context.Claims.FindAsync(claimId);

            // checks if the claim exists
            if (claim != null)
            {
                //if the claim exisists it set the boolean variable IsApprovedByCoordinator to true, indicating it has been approved by the coordinator.
                claim.IsApprovedByCoordinator = true;
                //The status of the claim is updated to reflect this approval.
                claim.Status = "Approved by Coordinator";
                //This saves the changes made to the claim in the database asynchronously.
                await _context.SaveChangesAsync();
            }
            //After processing the approval, the user is redirected back to the Index action to see the updated list of claims.
            return RedirectToAction("Index");
        }

        //This method processes the rejection of a claim. Similar to the Approve method, it takes the ID of the claim to be rejected
        [HttpPost]
        public async Task<IActionResult> Reject(int claimId)
        {
            // It attempts to find the claim in the database using the claimId.
            var claim = await _context.Claims.FindAsync(claimId);

            //checks if the claim was found
            if (claim != null)
            {
                //If the claim exists, the status is updated to indicate that it has been rejected.
                claim.Status = "Rejected by Coordinator";
                //The changes are saved to the database
                await _context.SaveChangesAsync();
            }
            //the user is redirected to the Index action to see the updated list of claims.
            return RedirectToAction("Index");
        }
    }//ProgrammeCoordinator controller end
}//namespace end
```

Programme Coordinator View

```
@model IEnumerable<ST10251759_PROG6212_POE.Models.Claim>

@{
    ViewBag.Title = "Pending Claims for Programme Coordinator";
}

<body>
    <div class="container-claims">
        <h2>Pending Claims for Programme Coordinator</h2>

        <table class="table table-striped">
            <thead>
                <tr>
                    <th>Claim ID</th>
                    <th>Submitted By</th>
                    <th>Date Submitted</th>
                    <th>Hours Worked</th>
                    <th>Hourly Rate</th>
                    <th>Total Amount</th>
                    <th>Supporting Documents</th>
                    <th>Actions</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var claim in Model)
                {
                    <tr>
                        <td>@claim.ClaimId</td>
                        <td>@claim.ApplicationUser?.Email</td>
                        <td>@claim.DateSubmitted.ToString("yyyy-MM-dd")</td>
                        <td>@claim.HoursWorked</td>
                        <td>R @claim.HourlyRate</td>
                        <td>R @claim.TotalAmount</td>
                        <td>
                            @if (claim.Documents != null && claim.Documents.Any())
                            {
                                <ul style="list-style-type: none; padding: 0;">
                                    @foreach (var doc in claim.Documents)
                                    {
                                        <li>
                                            <i class="fa-solid fa-download"></i>
                                            <a href="@Url.Content("~/uploads/" + doc.DocumentName)"
target="_blank" style="text-decoration: none;" class="document-link">View Document</a>
                                        </li>
                                    }
                                </ul>
                            }
                            else
                            {
                                <span>No Documents</span>
                            }
                        </td>
                        <td>
                            <div class="action-buttons">
                                <!-- Approve form -->
                                <form asp-action="Approve" method="post">
                                    <input type="hidden" name="claimId" value="@claim.ClaimId" />
                                    <input type="submit" value="Approve" class="btn btn-approve" />
                                </form>

                                <!-- Reject form -->
                                <form asp-action="Reject" method="post" style="display:inline;">
                                    <input type="hidden" name="claimId" value="@claim.ClaimId" />
                                    <input type="submit" value="Reject" class="btn btn-reject" />
                                </form>
                            </div>
                        </td>
                    </tr>
                }
            </tbody>
        </table>

        @if (!Model.Any())
        {
            <p>No pending claims to review at this time.</p>
        }
    </div>

    <style>
```

```
body {
    font-family: Arial, sans-serif;
    /* background-color: #03041c; */
    background: url('/images/temple6.jpg') no-repeat center center fixed;
    background-size: cover;
    color: #333;
    margin: 0;
    padding: 0;
}

.container-claims {
    max-width: 1600px;
    margin: 50px auto;
    padding: 20px;
    background-color: #ffffff;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    border-radius: 8px;
}

h2 {
    text-align: center;
    color: #03041c;
    margin-bottom: 30px;
    font-size: 24px;
    font-weight: bold;
}

.table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.table th, .table td {
    padding: 12px;
    border: 1px solid #ddd;
    text-align: left;
    color: #333;
}

.table th {
    background-color: #f2f2f2;
    color: #03041c;
    font-weight: bold;
}

.table td {
    background-color: #ffffff;
}

.table a {
    color: #03041c;
    text-decoration: underline;
}

.table a:hover {
    color: #03041c;
}
```

```
.btn-approve, .btn-reject {
    padding: 8px 12px;
    font-size: 14px;
    font-weight: bold;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    transition: background-color 0.3s ease-in-out;
    color: #ffffff;
    width: 100px;
    margin-right: 5px;
}

.btn-approve {
    background-color: #28a745;
}

.btn-reject {
    background-color: #dc3545;
}

.btn-approve:hover {
    background-color: #218838;
}

.btn-reject:hover {
    background-color: #c82333;
}

.action-buttons {
```

```
            display: flex;
            justify-content: space-between;
            align-items: center;
        }

        .document-link {
            text-decoration: none;
            color: #212529;
        }

        .document-link:hover {
            color: #03041c;
        }

    </style>
</body>
```

Academic Manager Controller

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using ST10251759_PROG6212_POE.Data;
using System.Linq;
using System.Threading.Tasks;

namespace ST10251759_PROG6212_POE.Controllers
{//namespace begin

    //Using Micrisoft Idenitity with Roles - This line means that only users with the
"Academic Manager" role can access the actions in this controller.
```

```
    [Authorize(Roles = "Academic Manager")]
    public class AcademicManagerController : Controller
    {//AcademicManager Controller begin

        //private fields declaration - used throughout controller
        private readonly Prog6212DbContext _context;

        //constructor - - initializes the private fields with the values passed in, allowing the
controller to use them for database access
        public AcademicManagerController(Prog6212DbContext context)
        {
            _context = context;
        }

        //This method is responsible for displaying a list of claims pending approval from the
manager.
        public IActionResult Index()
        {
            // Only show claims that are approved by the coordinator but pending manager
approval
            var pendingClaims = _context.Claims
                .Include(c => c.ApplicationUser) // Include the ApplicationUser
                .Include(c => c.Documents) // Include the Documents
                .Where(c => c.IsApprovedByCoordinator && !c.IsApprovedByManager &&
c.Status == "Approved by Coordinator") //These are the conditions for claims that can be
vuewd by the manager
                .ToList();
            //returns the pendingClaims list to the view, which will display it to the user.
            return View(pendingClaims);
        }
        //This method processes the approval of a claim when a POST request is made - when
Approve button is clicked - takes an integer parameter representing the ID of the claim to be
approved.
        [HttpPost]
        public async Task<IActionResult> Approve(int claimId)
        {
            //This line retrieves the claim from the database using the provided claimId
            var claim = await _context.Claims.FindAsync(claimId);


            // checks if the claim exists
            if (claim != null)
            {
                //if the claim exisists it set the boolean variable IsApprovedByManager to true,
indicating it has been approved by the manager.
                claim.IsApprovedByManager = true;
                //The status of the claim is updated to reflect this approval.
                claim.Status = "Approved by Manager";
                //This saves the changes made to the claim in the database asynchronously.
                await _context.SaveChangesAsync();
            }
            //After processing the approval, the user is redirected back to the Index action to see
the updated list of claims.
            return RedirectToAction("Index");
        }


        //This method processes the rejection of a claim. Similar to the Approve method, it
takes the ID of the claim to be rejected
        [HttpPost]
        public async Task<IActionResult> Reject(int claimId)
        {
            // It attempts to find the claim in the database using the claimId.
```

```
            var claim = await _context.Claims.FindAsync(claimId);

            //checks if the claim was found
            if (claim != null)
            {
                //If the claim exists, the status is updated to indicate that it has been rejected.
                claim.Status = "Rejected by Manager";
                //The changes are saved to the database
                await _context.SaveChangesAsync();
            }

            //the user is redirected to the Index action to see the updated list of claims.
            return RedirectToAction("Index");
        }
    }////AcademicManager Controller end
}//namespace end
```

Academic Manager View

```
@model IEnumerable<ST10251759_PROG6212_POE.Models.Claim>

@{
    ViewBag.Title = "Pending Claims for Academic Manager";
}

<body>
    <div class="container-claims">
        <h2>Pending Claims for Academic Manager</h2>

        <table class="table table-striped">
            <thead>
    <tr>
        <th>Claim ID</th>
        <th>Submitted By</th>
        <th>Date Submitted</th>
        <th>Hours Worked</th>
        <th>Hourly Rate</th>
        <th>Total Amount</th>
        <th>Supporting Documents</th>
        <th>Actions</th>
    </tr>
</thead>
<tbody>
    @foreach (var claim in Model)
    {
        <tr>
            <td>@claim.ClaimId</td>
            <td>@claim.ApplicationUser?.Email</td>
            <td>@claim.DateSubmitted.ToString("yyyy-MM-dd")</td>
            <td>@claim.HoursWorked</td>
            <td>R @claim.HourlyRate</td>
            <td>R @claim.TotalAmount</td>
            <td>
                @if (claim.Documents != null && claim.Documents.Any())
                {
                    <ul style="list-style-type: none; padding: 0;">
                        @foreach (var doc in claim.Documents)
                        {
                            <li>
                                <i class="fa-solid fa-download"></i>
```

```
                                            <a href="@Url.Content("~/uploads/" + doc.DocumentName)"
target="_blank" style="text-decoration: none;" class="document-link">View Document</a>
                                        </li>
                                    }
                                </ul>
                            }
                            else
                            {
                                <span>No Documents</span>
                            }
                        </td>
                        <td>
                            <div class="action-buttons">
                                <!-- Approve form -->
                                <form asp-action="Approve" method="post">
                                    <input type="hidden" name="claimId" value="@claim.ClaimId" />
                                    <input type="submit" value="Approve" class="btn btn-approve" />
                                </form>

                                <!-- Reject form -->
                                <form asp-action="Reject" method="post" style="display:inline;">
                                    <input type="hidden" name="claimId" value="@claim.ClaimId" />
                                    <input type="submit" value="Reject" class="btn btn-reject" />
                                </form>
                            </div>
                        </td>
                    </tr>
                }
            </tbody>
        </table>
```

```
    @if (!Model.Any())
    {
        <p>No pending claims to review at this time.</p>
    }
</div>

<style>
    body {
        font-family: Arial, sans-serif;
        /* background-color: #03041c; */
        background: url('/images/temple4.jpg') no-repeat center center fixed;
        background-size: cover;
        color: #333;
        margin: 0;
        padding: 0;
    }

    .container-claims {
        max-width: 1400px;
        margin: 50px auto;
        padding: 20px;
        background-color: #ffffff;
        box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
        border-radius: 8px;
    }

    h2 {
        text-align: center;
        color: #03041c;
    margin-bottom: 30px;
    font-size: 24px;
    font-weight: bold;
}

.table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

    .table th, .table td {
        padding: 12px;
        border: 1px solid #ddd;
        text-align: left;
        color: #333;
    }

    .table th {
        background-color: #f2f2f2;
        color: #03041c;
        font-weight: bold;
    }

    .table td {
        background-color: #ffffff;
    }

    .table a {
        color: #03041c;
        text-decoration: underline;
    }

        .table a:hover {
            color: #03041c;
        }

.btn-approve, .btn-reject {
    padding: 8px 12px;
    font-size: 14px;
    font-weight: bold;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    transition: background-color 0.3s ease-in-out;
    color: #ffffff;
    width: 100px;
    margin-right: 5px;
}

.btn-approve {
    background-color: #28a745;
}

.btn-reject {
    background-color: #dc3545;
}

.btn-approve:hover {
```

```
            background-color: #218838;
        }

        .btn-reject:hover {
            background-color: #c82333;
        }

        .action-buttons {
            display: flex;
            justify-content: space-between;
            align-items: center;
        }

        .document-link {
            text-decoration: none;
            color: #212529;
        }

            .document-link:hover {
                color: #03041c;
            }
    </style>
</body>
```

CODE ATTRIBUTUION

Author: w3schools

Link: https://www.w3schools.com/html/

Date Accessed: 12 October 2024

Author: w3schools

Link: https://www.w3schools.com/css/

Date Accessed: 12 October 2024

MVC APP

Author: Fatima Shaik

Link: https://github.com/fb-shaik/PROG6212-Group1-2024/blob/main/EmployeeLeaveManagement_G1.zip

Date Accessed: 11 October 2024

Database Work

Author: Microsoft

Link: https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/working-with-sql?view=aspnetcore-8.0&tabs=visual-studio

Date Accessed: 11 October 2024

LINQ Resource

Author: Fatima Shaik

Link: https://github.com/fb-shaik/PROG6212-Group1-2024/tree/main/Employee_Management_LINQ_FileHandling_G1

Date Accessed: 11 October 2024

Microsfot Identity

Author: Andy Malone MVP

Link: https://www.youtube.com/watch?v=zS79FDhAhBs

Date Accessed: 11 October 2024

PDF Doc - File Handling Resource

Author: Fatima Shaik

Link: https://github.com/fb-shaik/PROG6212-Group1-2024/tree/main/FileHandlingApp

Date Accessed: 11 October 2024