PROG6212 POE: CONTRACT MONTHLY CLAIM SYSTEM

Presented By

Keona Mackan (ST10445050)

THIS PRESENTATION COVERS:

- **Feedback** received from Part 2 and the **fixes** implemented in Part 3.
- **Detailed updates** and **improvements** in the system claim submission and approva processes.
- New features introduced in **Part 3**, including **HR functionality**, **unit testing**, **error h** and **automated claim approval**.
- Screenshots and **code explanations** for the main improvements and functionality.

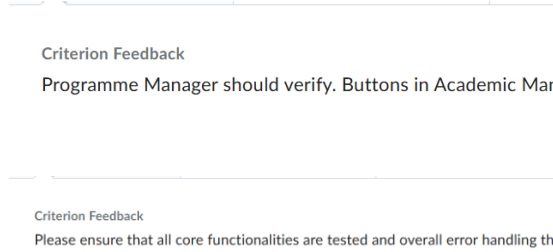
LECTURER FEEDBACK RECEIVED IN PART 2:

In Part 2, feedback highlighted several key issues that were addressed in Part 3:

Programme Manager Verification: The Programme Manager workflow was unclear, and the manager's UI was not fully structured. This led to confusion and inefficiency in the approval process.

Button Overlap in the Manager View: Buttons in the Academic Manager view overlapped, negatively impacting the user experience and causing a cluttered interface.

Error Handling and Unit Testing: Part 2 lacked sufficient error handling and unit testing, which left gaps in the system's robustness and reliability.



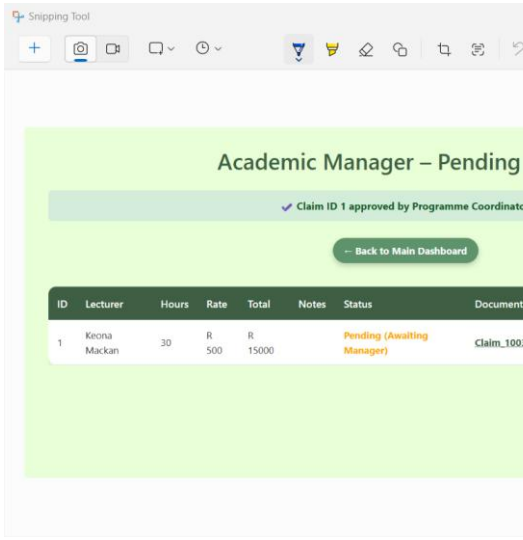
These issues have been fully resolved and significantly improved in Part 3, ensuring a more efficient, user-friendly system with comprehensive error handling and testing.

FIX 1: PROGRAMME MANAGER VERIFICATION & BUTTON OVERLAP

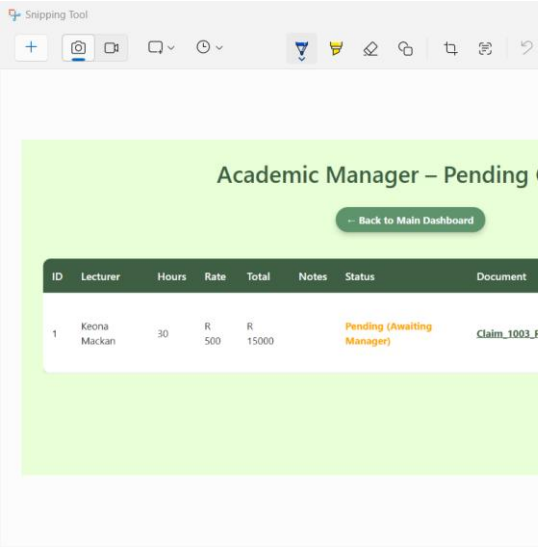
In Part 3, a **dedicated Manager view** was created to streamline the **approval workflow** (Lecturer → Coordinator → Manager), ensuring a clearer and more structured process.

The **UI layout** was redesigned using flexbox to eliminate overlapping buttons, providing a more responsive and intuitive interface. Buttons are now properly aligned and visually separated, improving readability and usability.

BEFORE:



AFTER:

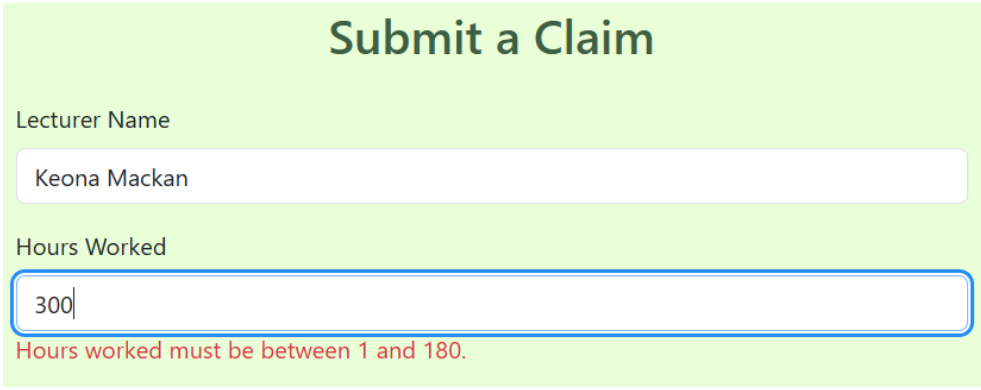


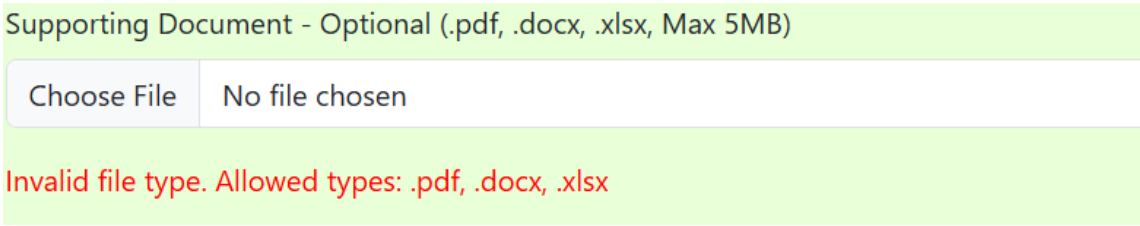
FIX 2: IMPLEMENTED IN PART 3, UNIT TESTING & ERROR HANDLING

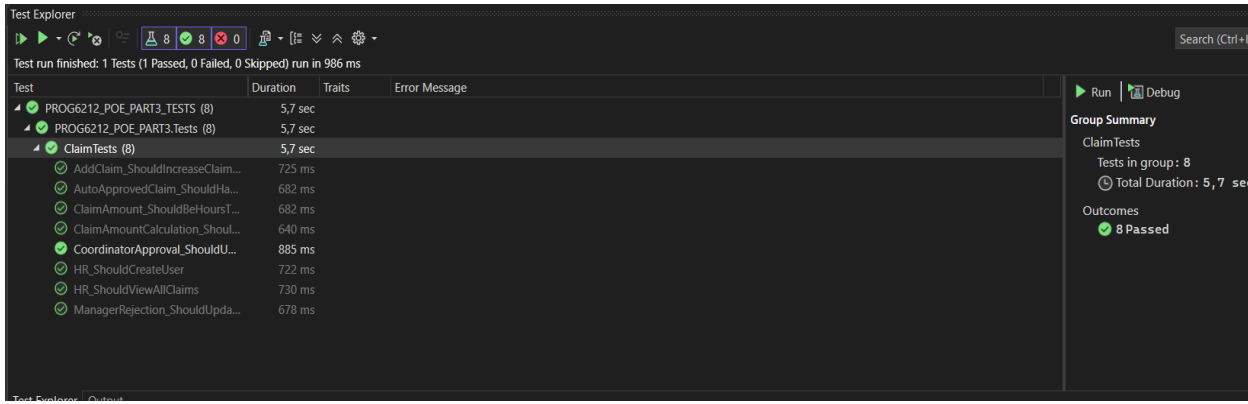
In Part 3, several key improvements were made to enhance the system's reliability and user experience:

Additional unit tests were added to cover more functionalities, including the claim total calculation and HR LINQ queries, ensuring that all critical processes are properly tested.. Additionally, error handling was integrated around critical processes, such as **claim creation and file upload**, to catch and display meaningful error messages. This improves system robustness and ensures users are informed of any issues.

The UI was also updated to **display validation messages** for invalid files or missing values, contributing to a more user-friendly experience. These updates fully meet the rubric requirement for effective unit testing and error handling.







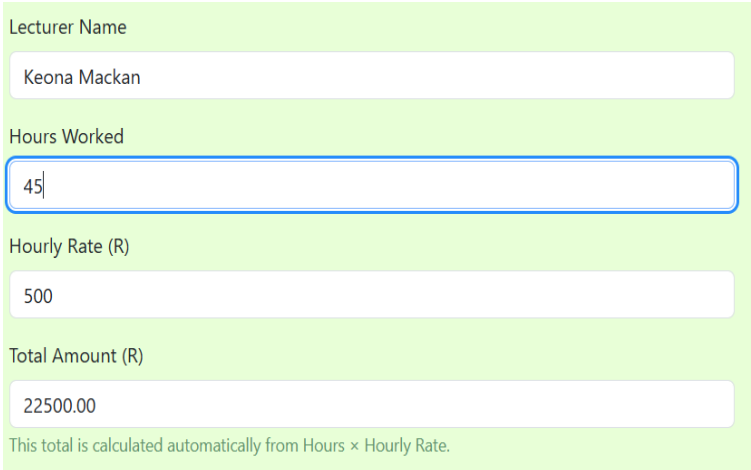
PART 3 FEATURES:

FEATURE 1: AUTOMATED CLAIM SUBMISSION

In Part 3, the Lecturer's **claim submission** process was **automated**. The Lecturer's name, surname, and hourly rate are auto-filled from HR data whe Lecturer logs in, streamlining the claim submission. The hours worked are manually entered by the Lecturer, and the claim total is automatically calcu based on the hourly rate. **Validation** ensures that only valid data is submitte preventing errors during claim creation.

```
// Ensure lecturer + rate are taken from DB, not from posted values
model.LecturerId = user.Id;
model.LecturerName = $"{user.FirstName} {user.LastName}";
model.HourlyRate = user.HourlyRate;
```

```
// Set submission date and calculate the total claim amount
model.DateSubmitted = DateTime.UtcNow;
model.StoredClaimAmount = model.HoursWorked * model.HourlyRate;
```



FEATURE 2 :

COORDINATOR APPROVAL WORKFLOW

In **Part 3**, the Coordinator approval process is triggered only if the claim's hours fall outside the defined range of 40 to 180 hours. If the Lecturer’s claim hours are within this range, **the claim is automatically approved**. **If the hours fall outside this range,** the claim is sent to the Coordinator for approval or rejection. Once the Coordinator approves or rejects the claim, the status is automatically updated, and a success message is displayed using TempData.

```
try
{
    // Automatic approval logic based on HoursWorked
    if (model.HoursWorked >= 40 && model.HoursWorked <= 180)
    {
        // Automatically approve if hours are within the ran
        model.Status = "Approved";
    }
    else
    {
        // Otherwise, it goes through the normal approval pr
        model.Status = "Pending";
    }
```

```
public IActionResult Approve(int id)
{
    if (HttpContext.Session.GetString("Role") != "Coordinator")
        return RedirectToAction("Login", "Home");

    var claim = _context.Claims.FirstOrDefault(c => c.Id == id);

    if (claim != null)
    {
        claim.Status = "CoordinatorApproved";
        _context.SaveChanges();

        TempData["Message"] = $"✔ Claim ID {id} approved by Programme Coor
    }

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

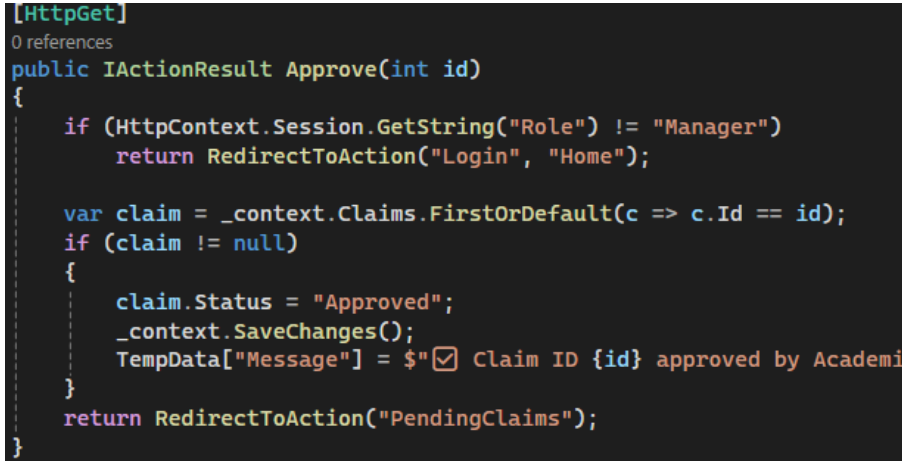
FEATURE 3:

ACADEMIC MANAGER FINAL APPROVAL

The **Academic Manager** is responsible for verifying claims that have already been approved by the Coordinator. However, this only occurs if the hours fall outside the allowed range. If the claim is within the accepted range, the Academic Manager's approval is skipped.

Improvement in Part 3:

Claims with hours outside the predefined range are forwarded to the **Coordinator**, and then to the **Academic Manager for final approval**. The Manager's view has been redesigned for clarity, ensuring that the approval process is straightforward.



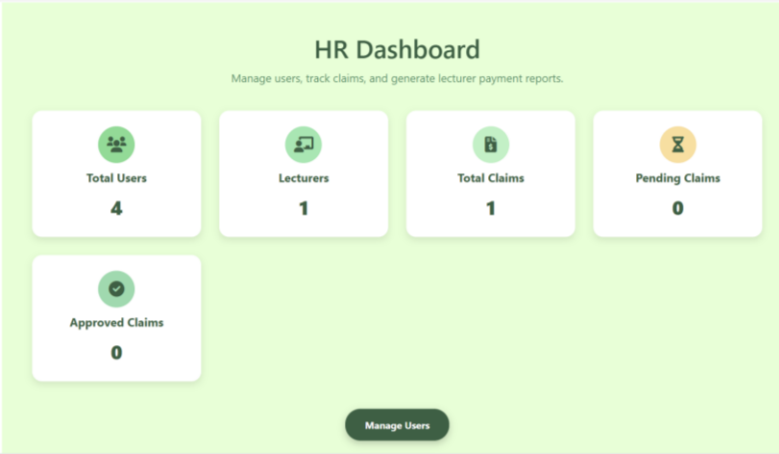
```
[HttpGet]
0 references
public IActionResult Approve(int id)
{
    if (HttpContext.Session.GetString("Role") != "Manager")
        return RedirectToAction("Login", "Home");

    var claim = _context.Claims.FirstOrDefault(c => c.Id == id);
    if (claim != null)
    {
        claim.Status = "Approved";
        _context.SaveChanges();
        TempData["Message"] = $"☑ Claim ID {id} approved by Academi
    }
    return RedirectToAction("PendingClaims");
}
```

FEATURE 4:

HR FUNCTIONALITY

Part 3 introduces a **new HR functionality** where HR is responsible f **creating users** in the system, including Lecturers, Coordinators, an Program Managers, and **providing** them **with login details**. HR can **update user information** as needed, ensuring all records are accu and up to date. The HR Dashboard has been designed with a clean a intuitive interface, **improving navigation** and ease of use. It include summary cards that display key metrics such as Total Users, Total Lecturers, Total Claims, Pending Claims, and Approved Claims. HR also responsible for **generating PDF reports** on approved claims




```
[HttpPost]
0 references
public IActionResult CreateUser(User model)
{
    if (!IsHR())
        return RedirectToAction("Login", "Home");

    if (!ModelState.IsValid)
        return View(model);

    _context.Users.Add(model);
    _context.SaveChanges();

    TempData["Message"] = "User created successfully
    return RedirectToAction("ManageUsers");
}
```

```
[HttpPost]
0 references
public IActionResult EditUser(User model)
{
    if (!IsHR())
        return RedirectToAction("Login", "Home");

    if (!ModelState.IsValid)
        return View(model);

    var user = _context.Users.FirstOrDefault(u => u.Id ==
    if (user == null)
        return NotFound();

    user.Username = model.Username;
    user.Password = model.Password;
    user.Role = model.Role;
    user.FirstName = model.FirstName;
    user.LastName = model.LastName;
    user.Email = model.Email;
    user.HourlyRate = model.HourlyRate;

    _context.SaveChanges();

    TempData["Message"] = "User updated successfully.";
    return RedirectToAction("ManageUsers");
}
```

```
public IActionResult DownloadClaimRe
{
    if (!IsHR())
        return RedirectToAction("Log

    var claim = _context.Claims
        .Include(c => c.Lecturer)
        .FirstOrDefault(c => c.Id ==

    if (claim == null)
        return NotFound();

    if (claim.Status != "Approved")
        return BadRequest("Reports c

    using (var ms = new MemoryStream
    {
        Document doc = new Document(
        PdfWriter.GetInstance(doc, m
```

FEATURE 5:

SESSION & ROLE SECURITY

Role-based security has been implemented in Part 3 to ensure that only **authorized users** can access **specific areas** of the system. Lecturers, for example, cannot access HR, Coordinator, or Manager pages, and vice versa. This ensures that each user only interacts with the pages relevant to their role. Unauthorized users attempting to access restricted pages are automatically redirected to the Login page, maintaining a secure environment for sensitive data.

```
// Store session info
HttpContext.Session.SetString("Username", user.Username);
HttpContext.Session.SetString("Role", user.Role);
HttpContext.Session.SetInt32("UserId", user.Id);
HttpContext.Session.SetString("FullName", $"{user.FirstName} {user.LastNam

if (user.Role == "Lecturer")
{
    HttpContext.Session.SetString("LecturerHourlyRate", user.HourlyRate.To
}

// Redirect based on role
return user.Role switch
{
    "Lecturer" => RedirectToAction("MainDashboard", "Lecturer"),
    "Coordinator" => RedirectToAction("MainDashboard", "Coordinator"),
    "Manager" => RedirectToAction("ManagerDashboard", "Manager"),
    "HR" => RedirectToAction("Index", "HR"), // HR controller coming next
    _ => View()
};
```

CONCLUSION:

Part 3 successfully addresses all **Part 2** feedback and introduces new features such as:

- Automated claim submission.
- Automated approval (if specified criteria are met).
- HR functionality.
- PDF export for approved claims.

The **UI** is streamlined with a clean green theme to improve usability. Role-based access ensures secure claim processing and efficient administrative tasks.

THE END