PROG6212 POE Part 2

St10438960

Jemaine Ethan Naidoo

YouTube video Link: <https://youtu.be/3BqH4SDArp0>

GitHub Repo Link: <https://github.com/Jemaine08/VCDN-PROG6212-POE-Part2>

Unit testing:

A screenshot of a computer

AI-generated content may be incorrect.

**Documentation for Feedback Part 1:**

**Feedback 1:**

**Feedback Summary:**

The lecturer showed that the form of submitting claims was poorly designed and practical. The user interface had to be more intuitive, particularly with file uploads and input of claim information.

**Implementation in Part 2:**

Improved Claim Submission Form:

-The layout and responsiveness were enhanced in the Create.cshtml view using Bootstrap 5. This guarantees the form will adapt properly to various sizes of the screens and deliver a more user-friendly experience.

-The file upload control was improved with good instructions and validation messages have been included to make sure that users upload valid files (the implementation is of IFormFile).

-UI validation includes added form validation with the support of data-val attribute and custom messages and the possibility to view feedback instantly.

-The form fields were not only labeled but also, I included placeholders to direct users on the process of submitting claims.

**Visual Example:**

<div class="mb-3">

<label for="LecturerName" class="form-label">Lecturer Name</label>

<input type="text" class="form-control" id="LecturerName" name="LecturerName" placeholder="Enter the name of the lecturer" required>

</div>

**Feedback 2:**

**Feedback 2: Provide Validation and Response to Claims.**

**Feedback Summary:**

The lecturer requested validation and error correction to be applied to the submission forms particularly to such fields as Lecturer Name, Hours Worked, and Hourly Rate.

**Implementation in Part 2:**

**Server-Side Validation:**

-I also added stipulation of the form on the Create.cshtml to make sure that a user cannot post a claim without typing the necessary information.

-Model Binding applied to Hourlyrate and HoursWorked using Used ASP.NET Core: Model Binding and Data Annotations ([Required], [Range]) to ensure that the numbers are valid.

**Client-Side Validation:**

-JavaScript was used to do client-side validation, which allows the user to get immediate feedback when he/she attempted to submit the form and one of the fields (e.g. non-numeric values in HourlyRate) contained invalid data.

-When an error occurs, it will be displayed on the top of the corresponding fields and will enhance the user experience.

**Visual Example:**

@if (!ViewData.ModelState.IsValid)

{

<div class="alert alert-danger">Please correct the highlighted errors.</div>

}

**Feedback 3: UI better Claim Verification and Rejection.**

**Feedback Summary:**

The lecturer asked to have more clear UI components to check claims and reject them. This would entail easy buttons, right messages, and status symbols.

**Implementation in Part 2:**

**Better Verification/Rejection Buttons:**

I have added colored buttons (Bootstrap classes) to the Verify and Reject action to make them conspicuous. The buttons have also been explained with the words verify and reject in bold fonts, with the correct CSS classes, such as btn-success (verify) or btn-danger (reject).

**Status Indicator:**

I added a status column to the claims listing view (MyClaims.cshtml), where iconography (FontAwesome icons) is used to include a visual representation of the claims in the Submitted, Verified or Rejected status.

A claim entry also has a status label which dynamically updates as the status of the claim changes to give the coordinator visual feedback.

**Visual Example**:

<button class="btn btn-success" @onclick="VerifyClaim">Verify</button>

<button class="btn btn-danger" @onclick="RejectClaim">Reject</button>

<td>

@if (claim.Status == ClaimStatus.Verified)

{

<span class="badge bg-success">Verified</span>

}

else if (claim.Status == ClaimStatus.Rejected)

{

<span class="badge bg-danger">Rejected</span>

}

else

{

<span class="badge bg-warning">Submitted</span>

}

</td>

**Feedback 4: Be More responsive in Views.**

**Feedback Summary:**

The lecturer emphasized that it is necessary to make the UI responsive to enable it to work effectively both on a desktop and a mobile device.

**Implementation in Part 2:**

**Responsive Design:**

-To see the views, particularly the claim forms and the claims listing, can fit the different screen sizes, I adopted the responsive grid system of Bootstrap.

-In the case of the Claims listing page (MyClaims.cshtml), I made sure that the table will wrap appropriately on smaller monitors and that the columns will be stacked on mobile phones to make it easier to read.

-Implemented responsive navigation based on the navbar component of Bootstrap that folds into a hamburger menu when there is limited space.

**Visual example**

<div class="container-fluid">

<div class="row">

<div class="col-md-6">

<!-- Form Inputs here -->

</div>

<div class="col-md-6">

<!-- Claim Data Table here -->

</div>

</div>

</div**>**

**Feedback 5: Consistency of the UI.**

**Feedback Summary:**

The lecturer proposed standardization in design, such that the buttons, forms, and other elements of the UI should have the same design.

**Implementation in Part 2:**

**Consistent UI Elements:**

-The predefined form controls, buttons, and alerts offered by Bootstrap allowed me to have a uniform style of all pages because they were styled in a similar way.

-The buttons in actions such as Verify, Reject and Download, I made sure that the same color schemes (green in verify, red in reject) and rounded corners have been used in the entire app.

**Visual example**

<button class="btn btn-primary" type="submit">Submit Claim</button>

<button class="btn btn-success" type="submit">Verify Claim</button>

<button class="btn btn-danger" type="submit">Reject Claim</button>

**Feedback 6: Enhance UI in File Upload.**

**Feedback Summary:**

According to the lecturer, improving the file upload interface, which is not completely clear and intuitive, was recommended.

**Implementation in Part 2:**

**More Intuitive File Upload Interface:**

-I added a file input into the claim submission form with easily understandable guidelines on which types of files will be accepted ( e.g., PDF, DOCX ).

-Added client-side validity to prevent the possibility of users uploading invalid file types or empty files.

-Once it is uploaded, the file name appears next to the file input field to allow users to have instant feedback of the file that they selected.

**Visual example**

<div class="mb-3">

<label for="Upload" class="form-label">Upload Claim Document</label>

<input type="file" class="form-control" id="Upload" name="Upload" required accept=".pdf,.docx,.jpg">

<small class="form-text text-muted">Only PDF, DOCX, or JPG files are allowed.</small>

</div>

**References List:**

Dam, R. and Siang T., (2020). 10 Insightful Design Thinking Frameworks: A Quick Overview. Available at: <<https://www.interaction-design.org/literature/article/design-thinking-a-quick-overview/>> [Accessed 8 September 2024].

Satzinger, J.W., Jackson, R.B., and Burd, S.D., (2016). Introduction to Systems Analysis and Design: An Agile, Iterative Approach. 7th edition. Toronto: Cengage.

Troelsen, A. and Japikse, P., (2021). Pro C# 9 with .NET 5: Foundational Principles and Practice in Programming. 10th edition. New York: Apress.

Microsoft, (2024). Introduction to Identity on ASP.NET Core. Available at: <<https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-8.0&tabs=visual-studio>> [Accessed 10 October 2024].

Microsoft, (2023). Tutorial: Get Started with EF Core in an ASP.NET MVC Web App. Available at: <<https://learn.microsoft.com/en-us/aspnet/core/data/ef-mvc/intro?view=aspnetcore-8.0>> [Accessed 1 October 2024].

Microsoft, (2023). Create a Complex Data Model - ASP.NET MVC with EF Core. Available at: <<https://learn.microsoft.com/en-us/aspnet/core/data/ef-mvc/complex-data-model?view=aspnetcore-8.0>> [Accessed 1 October 2024].

Microsoft, (2007). Crystal Reports SDK. Available at: <<https://learn.microsoft.com/en-us/previous-versions/ms225335(v%3dvs.90)>> [Accessed 17 November 2024].

Microsoft, (2024). Model Binding in ASP.NET Core. Available at: <<https://learn.microsoft.com/en-us/aspnet/core/mvc/models/model-binding?view=aspnetcore-9.0>> [Accessed 11 November 2024].

Ibrahim, (2024). Contract Monthly Claim System: Web Application Documentation and Workflow. Unpublished documentation for internal use.

Interaction Design Foundation, (2020). Design Thinking: A Quick Overview. Available at: <<https://www.interaction-design.org/literature/article/design-thinking-a-quick-overview>> [Accessed: 22 October 2025].

**Bibliography for Code and Concepts Used:**

**UnitTesting\_MVC.pdf**, (2024). *How to do Unit Testing in Visual Studio MVC with a DB*. Available at: /mnt/data/UnitTesting\_MVC.pdf [Accessed 22 October 2025].

**PROG6212\_Workshop\_Part2.pptx**, (2024). *PROG6212 Workshop Part 2: MVC UI Implementation and Unit Testing*. Available at: /mnt/data/PROG6212\_Workshop\_Part2.pptx [Accessed 22 October 2025].

**Prog6212.pdf**, (2025). *POE – Programming 2B, PROG6212*. Available at: /mnt/data/Prog6212.pdf [Accessed 22 October 2025].

**README.md**, (2025). *Project Documentation for Contract Monthly Claims System*. Available at: /mnt/data/README.md [Accessed 22 October 2025].