S17

Security and Maintainability Compliance At Labman

Arthur Clarkson

3rd October 2024

On the intranet we follow the N-Layer architecture, with our main layers being: View => Controller => Business => Data, to maximise maintainability we enforce the rule of layers only being able to send data from each other if they're next to each other, that means the View layer isn't able to get data from the Data layer, without going through the Controller and Business layer.

This helps improve security because each layer is isolated from one another, meaning if one layer is compromised, it will only be that layer affected, rather than the hacker gaining access to the whole system.

Layers communicate through well-defined interfaces, meaning that there is a reduced risk of unintended data exposure or unauthorized access between layers. One way we ensure this, is by handling the permission checks within the Controller layer, this is because: from this layer we are able to dynamically render the View layer's template via a model created in the controller, and we are able to stop users from accessing the business layer.

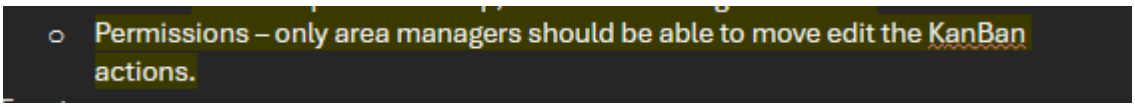
For example if a user is missing a certain permission, e.g. a permission to be able to delete a record, http GET endpoint will return the view without the delete button within the HTML, and if they attempt to send a request to the http DELETE endpoint, it will do a permissions check and return a 403 Forbidden response.

To make this maintainable, we have a decorator for our controller actions, this decorator runs before the action is executed, checking the user's claims to see if the HR permission is on of them.

```
[HttpGet]
[Authorize(Roles = Permissions.HR)]      You, 4 months ago • Merged PR 1591: Rename role to permission - Dep…
1 reference
public IActionResult GetUserEntries(int userId, DateTime? startDate = null, DateTime? endDate = null)
{
    return PartialView("_TimesheetEntriesTable", ApiControllers.Timesheet.GetTimesheetEntriesForUser(userId, startDate, endDate));
}

#endregion
```

On some of my recent work which I had completed, I was told to add permissions to the page, this was to stop users from accessing an edit menu, and stop users from creating/deleting certain records:



With this in mind, the permission for area manager on the intranet is called "Compliance Issue Manager", to make sure I complied with maintainability, I used the Authorize decorator to all endpoints we don't want normal users to access:

```
[HttpPost]
public IActionResult AssignUserToIssue(AssignUserToIssue form)
{
        ... Endpoint logic
}

[HttpGet]
public ActionResult EditIssue(int id, bool editingFromDashboard = false)
```

```
{
        ... Endpoint logic
}
```

Converted to

```
[HttpPost]
[Authorize(Roles = Permissions.ComplianceIssueManager)]
public IActionResult AssignUserToIssue(AssignUserToIssue form)
{
        ... Endpoint logic
}

[HttpGet]
[Authorize(Roles = Permissions.ComplianceIssueManager)]
public ActionResult EditIssue(int id, bool editingFromDashboard = false)
{
        ... Endpoint logic
}
```

I then needed to updated the UI to hide anything normal users shouldn't be seeing, to do this I added `bool HasEditPermissions` to the ViewBag so that it can be read in the view:

```
ViewBag.HasEditPermissions = HasCurrentUserGotRole(Permissions.ComplianceIssueManager);
```

This is how it is then read in the view:

```
@{
    bool hasEditPermissions = ViewBag.HasEditPermissions ?? false;
}
```

With this variable, I was then able to dynamically render the html:

```
<div class="kanban_card" style="grid-column: 1; cursor: @(hasEditPermissions ? "pointer" : "default");"
onclick="@(hasEditPermissions ? $"openEditIssuePopup({ticket.Id})" : "")">
                    <div class="card-content">
```