Encoding object source scope by generating Resource Id

Status	APPROVED
Owner	@Sławomir Pawluk
Contributors	@Piotr Litwinski @Conor Brady @Colm Nolan @Desmond Farrell @Mohamed Shalaby
Approved	@Piotr Litwinski
Due date	
Decision	We should explore: • how to handle 1 object linked to multiple entities (links table or similar?) - Tasks Manager • dig in a bit more into data migration strategies - moving data around e.g. an issue moved from one inspection to another We can start implementing this
On this page	

❓ Problem statement

iQMS System requires 2 levels of authorization:

- permission based authorization - to protect endpoints from unauthorized execution
- resource based authorization - filter out the data that user is not allowed to see (ex. Task or Action plan for a Risk Assessment, when user is not allowed to view all tasks nor risk assessments)

As long as our REST API/GQL refactor to have smaller endpoints with as little responsibility as possible allows us to cover most of the cases with simple permission based authorization, there are parts of the system that will require resource based authorization.

As an example a Task manager is a component, where user can view tasks that are created from multiple different sources. Depending on list of permissions that user is assigned with, the list of the tasks will need to be filtered.

Since Tasks Manager is considered a separate Product or Service, the proper design is required to avoid chatty interface across multiple services to check the access to the object.

🏅 Design proposal

A proposal to consider might be encoding the objects scope as a Resource Id, similarly how Azure identifies resources in the cloud.

The Azure Resource Id looks as follows:
`/subscriptions/e2889e3e-37a9-4d55-b0ba-892c5e90101d/resourceGroups/IQMS_COMPUTE_DEV/providers/microsoft.insights/components/iqms-ai-dev`

From this identifier we can read information:

- what is the resource subscription

- what is the resource group of the resource
- what is the type of the resource
- etc.

Taking this approach as an example, the iQMS objects could be identified by similar Resource Id, like:

`/tenant/b7fd2d08-e266-4059-8283-0aef30034678/entity/bc249325-c73f-46cf-97b2-c20de468d6c9/inspection/8076dc65-6240-4704-9055-1ebd78aac177/requests/ab134024-fd4e-40c2-bef9-932d2034c956/issues/faa88650-1217-4edf-b2b1-efe462035d50`

Using this identifier the object contains accurate information about the scope of it's belongingness.

Using Resource Id, the Resource Based authorization can be implemented using Authorization Policies to perform post-fetch filtering:

```
public class EntityAuthorizationHandler : AuthorizationHandler<TenantRequirement, Entity>
{
  protected override Task HandleRequirementAsync(AuthorizationHandlerContext context, TenantRequirement requirement, Entity resource)
  {
    var resourceId = resource.ResourceId;
    var userTenantId = context.User.FindFirstValue("tid") ?? "unknown";

    if (context.User.HasClaim(x => x.Type == "permission" && x.Value == "entities.view.all"))
    {
      context.Succeed(requirement);
      return Task.CompletedTask;
    }

    if (resourceId.Contains($"/tenant/{userTenantId}"))
    {
      context.Succeed(requirement);
    }

    return Task.CompletedTask;
  }
}

////

builder.Services.AddAuthorization()
  .AddAuthorizationBuilder()
  .AddPolicy("ViewEntityPolicy", c => c.AddRequirements(new ViewEntityRequirement()));
builder.Services.AddSingleton<IAuthorizationHandler, EntityAuthorizationHandler>();

app.MapGet(
  "/{id}",
  async (Guid id, Repository repo, IAuthorizationService authService, ClaimsPrincipal user) =>
  {
    var entity = repo.GetEntity(id);

    var accessResult = await authService.AuthorizeAsync(user, entity, "ViewEntityPolicy");

    return accessResult switch
    {
      { Succeeded: true } => Results.Ok(entity),
      { Succeeded: false } => Results.StatusCode(403),
      _ => Results.NotFound()
    };
```

```
});
```

This solution works very well with HotChocolate authorization engine. HotChocolate would batch fetch Tasks by list of Id's, then filter the result collection based on batched call to authorization policy.

🥈 Alternative

Alternative use of Resource Id would be filtering entities when fetching data for components such as Tasks Manager

```
public async Task<IActionResult> GetTasks(CancellationToken cancellationToken)
{
  var user = HttpContext.User;

  if(user.HasPermission("tasks.view"))
  {
    var tasks = await _mediator.Send(new GetAllTasksQuery(), cancellationToken);
    // ...
    return result;
  }

  var inspectionsTasks = user.HasPermission("inspections.view") ? await _mediator.Send(new GetInspectionsTasksQuery()) : Array.Empty<GetTaskQueryResult>();
  var requestsTasks = user.HasPermission("requests.view") ? await _mediator.Send(new GetRequestsTasksQuery()) : Array.Empty<GetTaskQueryResult>();
  var checklistsTasks = user.HasPermission("checklists.view") ? await _mediator.Send(new GetChecklistsTasksQuery()) : Array.Empty<GetTaskQueryResult>();
  var issuesTasks = user.HasPermission("issues.view") ? await _mediator.Send(new GetIssuesTasksQuery()) : Array.Empty<GetTaskQueryResult>();

  var result = Array.Empty<GetTaskQueryResult>()
    .Concat(inspectionsTasks)
    .Concat(requestsTasks)
    .Concat(checklistsTasks)
    .Concat(issuesTasks);

  // ...

  return result;
}
```

This solution works very well with REST API approach. The authorization is performed on API layer and does not leak to application layers.

👀 Considerations

Considerations that should be taken into account when implementing this type of the solution:

- Indexing
- Searching (Full-Text?)
- Parsing (simple string.Contains() or RegEx?)
- Consistent namespaces used in the URI across multiple services
- Max length of the identifier (ideally less than 4000 chars to avoid truncation, Full-Text search limitations)
- Task Manager Tasks - a single Task can be connected with multiple objects - The Resource Id should belong to Links table
- What happens when the parent resource is deleted

✅ Follow up

Decision	Status	Next steps
	DECIDED / IN REVIEW / OTHER	☐

💎 Source files

https://dev.azure.com/irlca/iQMS/_git/irlca.rnd.resource_based_authorization