WebAPI Presentation

A Case Study

What was given

- BriefingMetadata
- AssetMetadata
- ContentDistributionMetadata

Common member: AssetId

Complexities

- Aggregate data from different domains
- Handle concurrent requests efficiently
- Designed with fault tolerance in mind

REST API

REST API BENEFITS

- Stateless - allows for horizontal scaling
- Caching - clients can store requests
- Allows for different consumers
- Easily containerized

Honorable Mentions:

GraphQL

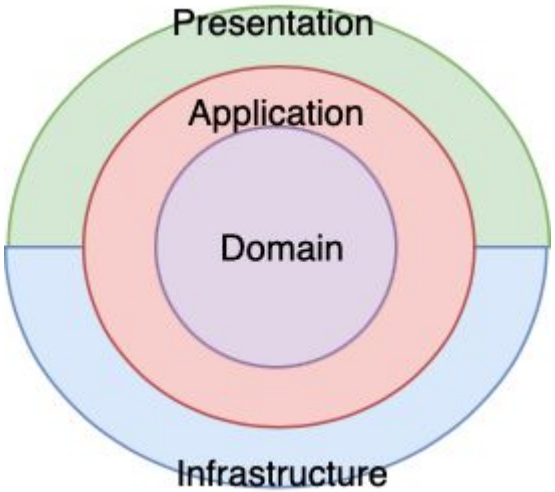
Challenges

- Understanding the data
  - Limited information
  - No stakeholder access
- Determining the size of the project
- Limited time
  - Coding issues

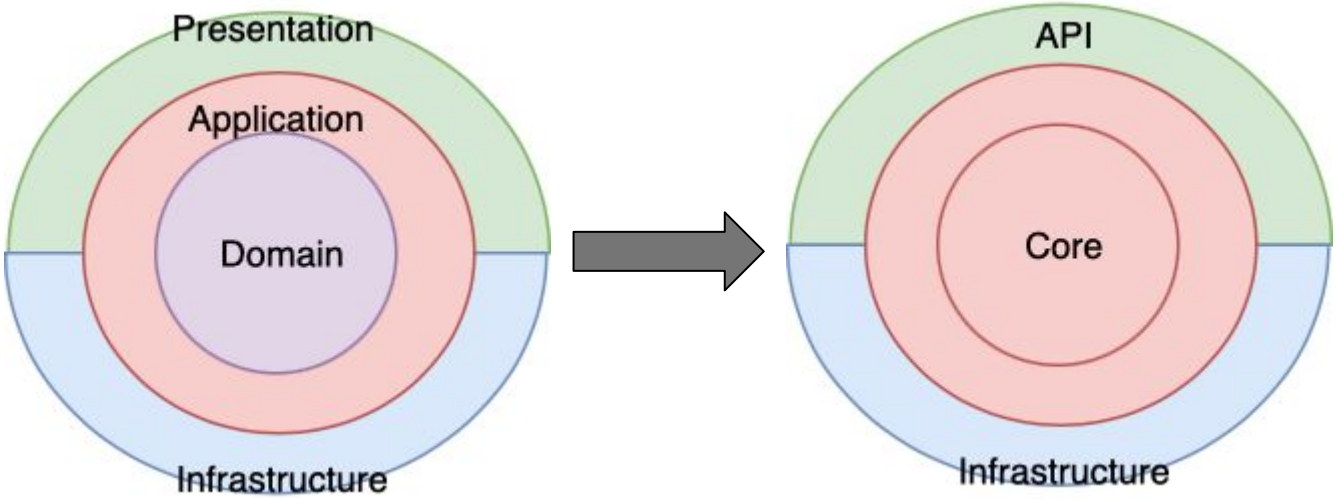
Approach

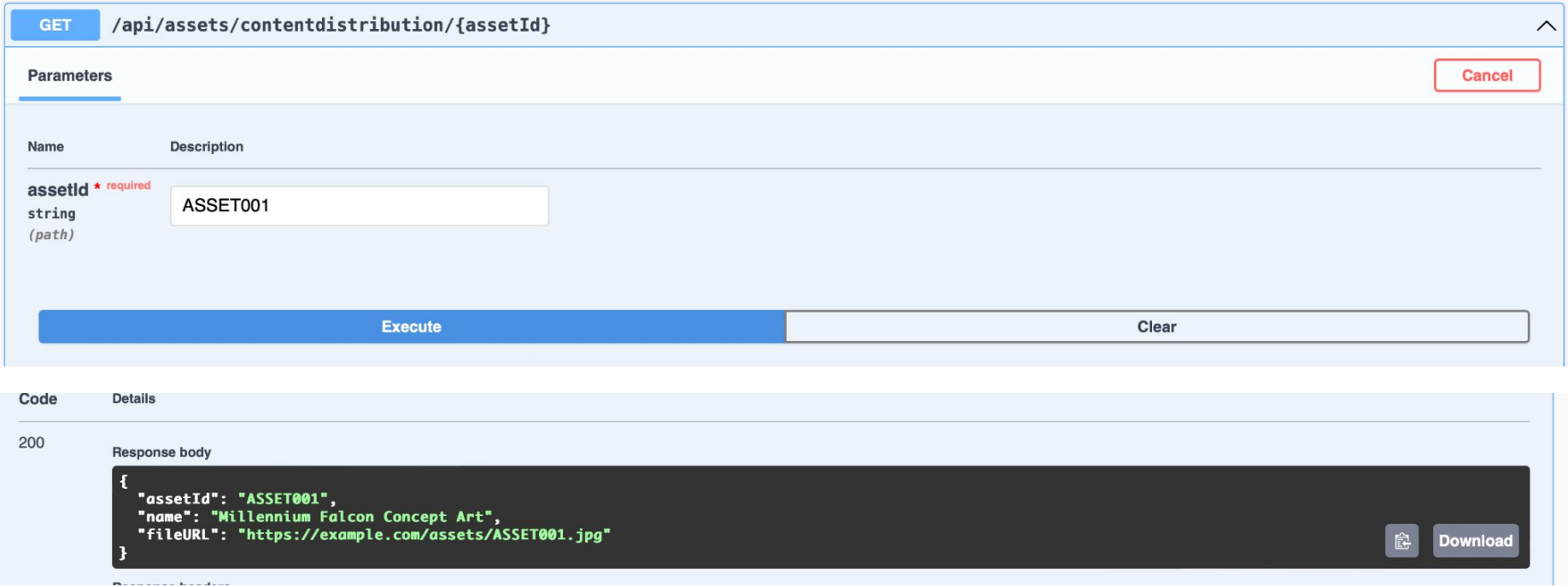
- Opt with an agile approach
  - Design, implement, reiterate
- Solution had to be adaptable.
- Used .NET 8 WebAPI
  - Security: Able to implement authentication
  - Platform agnostic
  - Mutli-threaded
- Used AI to brainstorm ideas

Clean Architecture

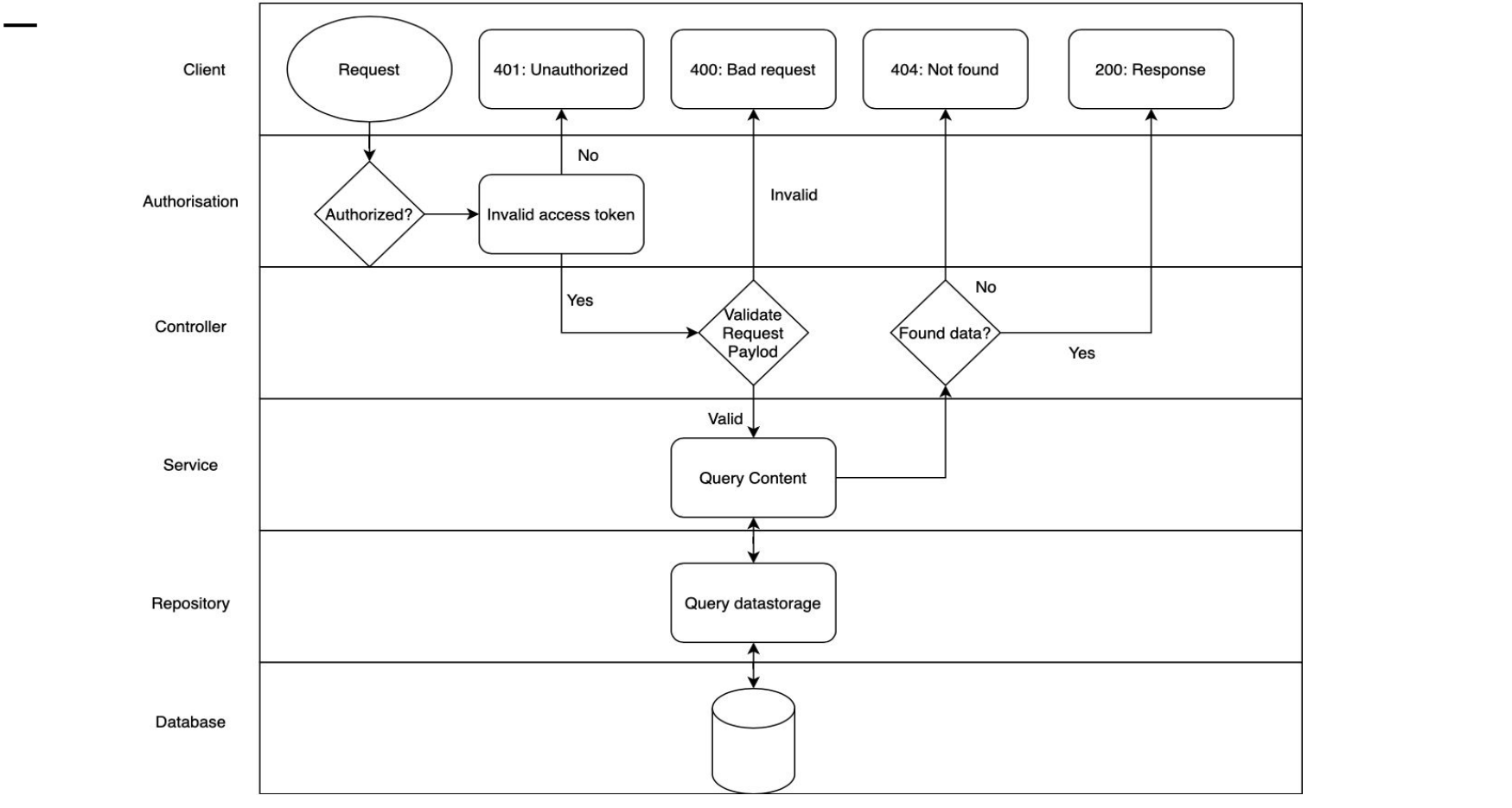


Clean Architecture




GET
/api/assets/contentdistribution/{assetId}
Parameters
Cancel
Name
Description
assetId * required
string
(path)
ASSET001
Execute
Clear
Code
Details
200
Response body
{
"assetId": "ASSET001",
"name": "Millennium Falcon Concept Art",
"fileURL": "https://example.com/assets/ASSET001.jpg"
}
Download

GET /assets/contentdistribution/{id}



```
[Fact]
0 references
public async Task GetAsset_ReturnsNotFound_WhenAssetDoesNotExist()
{
    // Arrange
    var assetId = "nonexistent";
    _mockAssetService
        .Setup(service => service.GetAssetByIdAsync(assetId.ToUpperInvariant()))
        .ReturnsAsync((Asset?)null);

    // Act
    var result = await _controller.GetAsset(assetId);

    // Assert
    var notFoundResult = Assert.IsType<NotFoundObjectResult>(result);
    Assert.Equal($"No asset found for ID {assetId.ToUpperInvariant()}", notFoundResult.Value);
}
```

Teamwork makes the dreamwork

- Breakdown the logic with a team mate or team mates
  - Refinement
- Whiteboard
- Pair-programming
  - Visual Studio Live Share

Limitations

- Clean architecture may make unit testing more complex
  - Integration tests are performance heavy
- Dependencies on third-parties
  - Swagger is no longer maintained.
- Potential overengineering for simple services.

Future Improvements

- Azure Application Insights
  - Added logging throughout the code
- Add caching
  - Redis
- Better structured API calls
- Better testing
  - Automated test
  - Integration test
- Packages like
  - FluentValidation
  - Polly