API Architecture and Framework Technical Design

Overview

This document outlines the API architecture and framework, detailing the rules, error handling, validation, and providing samples for each category or function.

# API Architecture

## Framework Choice

We have chosen the XYZ framework for the following reasons:

* Scalability
* Ease of integration
* Comprehensive documentation

# Design Rules

## IActionResult Return Type

Why use IActionResult Return Type:

* Standardization: Ensures a consistent return type across the API.
* Flexibility: Allows for a wide range of HTTP responses, making it easier to handle different scenarios such as success, client errors, and server errors.

Reasons to avoid custom user class returns:

* Complexity: Custom classes add unnecessary complexity and can lead to inconsistent return types.
* Maintenance: Standard types like IActionResult are well-supported and easier to maintain.

## Error Handling

Error Return Structure:

* Errors will be returned using a standardized format.
* All error responses will include a status code, error message, and an optional error code for further categorization.

{

"statusCode": 400,

"errorMessage": "Invalid request",

"errorCode": "ERR001"

}

## Validation

Validation will be performed at the following points:

* Request Level: Initial validation of request parameters and body.
* Service Level: Deeper validation within service logic to ensure data integrity.

# Sample Implementations

## Sample Controller

public class SampleController : ControllerBase

{

[HttpGet]

public IActionResult GetSample()

{

if (!ModelState.IsValid)

{

return BadRequest(new { statusCode = 400, errorMessage = "Invalid request", errorCode = "ERR001" });

}

return Ok(new { message = "Success" });

}

}

## Centralized Error Handling

// Centralized Error Handling Middleware

public class ErrorHandlingMiddleware

{

private readonly RequestDelegate \_next;

public ErrorHandlingMiddleware(RequestDelegate next)

{

\_next = next;

}

public async Task InvokeAsync(HttpContext httpContext)

{

try

{

await \_next(httpContext);

}

catch (Exception ex)

{

await HandleExceptionAsync(httpContext, ex);

}

}

private Task HandleExceptionAsync(HttpContext context, Exception exception)

{

context.Response.ContentType = "application/json";

context.Response.StatusCode = (int)HttpStatusCode.InternalServerError;

var response = new { statusCode = 500, errorMessage = "Internal Server Error", errorCode = "ERR002" };

return context.Response.WriteAsync(JsonConvert.SerializeObject(response));

}

}

## Validation Example

public class SampleRequest

{

[Required]

public string Name { get; set; }

}

public class SampleController : ControllerBase

{

[HttpPost]

public IActionResult CreateSample([FromBody] SampleRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(new { statusCode = 400, errorMessage = "Invalid request", errorCode = "ERR001" });

}

// Proceed with creating the sample

return Ok(new { message = "Sample created" });

}

}