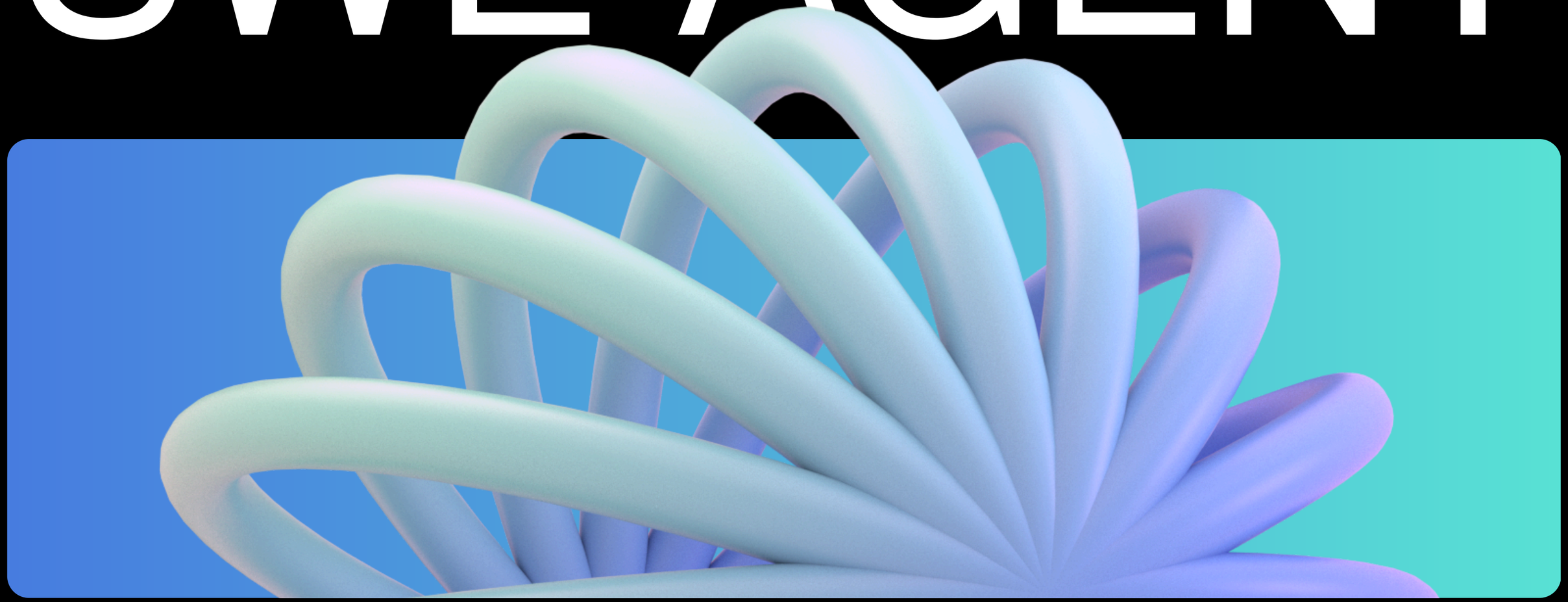
SWE AGENT



Sign me up

AI_57

- 1.Ngô Trần Xuân Hoà
- 2.Trương Anh Tuấn
- 3.Nguyễn Viết Thắng
- 4.Trần Nguyễn Hoàng Phúc
- 5.Đinh Quốc Huy

Phase 1:
Analyze & Select Feature

Why chosing this core feature?

Core

Its primary function is to generate the main deliverable

Complex

API Endpoint receives an HTTP request.
GeminiService generates a prompt.
An external call is made to the Google Gemini API.
The AI's text response is parsed and validated.
ExcelService transforms the parsed data into an Excel file (byte[]).
The API returns this file to the user. A failure at any link in this chain will cause the entire feature to fail.

High risk

The feature depends on the external Google Gemini API. This introduces risks that must be tested, such as network failures, API timeouts, and receiving malformed JSON or unexpected data from the AI.

Prompt

Analyze ExportUseCaseReportToExcel() function and class required and identify all functions that need unit testing:

[IMG]

For each function, identify:

1. Main functionality
2. Input parameters and types
3. Expected return values
4. Potential edge cases
5. Dependencies that need mocking

```
        // Secondary Actors
        ws.Cell(row, 1).Value = "Secondary Actors:";
        ws.Cell(row, 1).Style.Font.Bold = true;
        ws.Cell(row, 1).Style.Fill.BackgroundColor = XLColor.LightGray;
        ws.Cell(row, 2).Value = !string.IsNullOrEmpty(report.SecondaryActors) ? report.SecondaryActors : "None";
        row++;

        // ============== CORE USE CASE DETAILS ==============
        // Trigger
        ws.Cell(row, 1).Value = "Trigger:";
        ws.Cell(row, 1).Style.Font.Bold = true;
        ws.Cell(row, 1).Style.Fill.BackgroundColor = XLColor.LightGray;
        ws.Cell(row, 2).Value = report.Trigger;
        row++;

        // Description
        ws.Cell(row, 1).Value = "Description:";
        ws.Cell(row, 1).Style.Font.Bold = true;
        ws.Cell(row, 1).Style.Fill.BackgroundColor = XLColor.LightGray;
        ws.Cell(row, 2).Value = report.Description;
        row++;

        // Preconditions
        ws.Cell(row, 1).Value = "Preconditions:";
        ws.Cell(row, 1).Style.Font.Bold = true;
        ws.Cell(row, 1).Style.Fill.BackgroundColor = XLColor.LightGray;
        row++;

        for (int i = 0; i < (report.Preconditions?.Count ?? 0); i++)
        {
            ws.Cell(row, 2).Value = $"PRE-{i + 1}: {report.Preconditions?[i] ?? ""}";
            row++;
        }

        // Postconditions
        ws.Cell(row, 1).Value = "Postconditions:";
        ws.Cell(row, 1).Style.Font.Bold = true;
        ws.Cell(row, 1).Style.Fill.BackgroundColor = XLColor.LightGray;
        row++;

        for (int i = 0; i < (report.Postconditions?.Count ?? 0); i++)
        {
            ws.Cell(row, 2).Value = $"POST-{i + 1}: {report.Postconditions?[i] ?? ""}";
            row++;
        }

        // ============== FLOWS SECTION ==============
        // Normal Flow
        ws.Cell(row, 1).Value = "Normal Flow:";
        ws.Cell(row, 1).Style.Font.Bold = true;
        ws.Cell(row, 1).Style.Fill.BackgroundColor = XLColor.LightGray;
        row++;

        foreach (var step in report.NormalFlow ?? new List<FlowStep>())
        {
            ws.Cell(row, 2).Value = $"{step.Step}. {step.Description ?? ""}";
            row++;
        }

        // Alternative Flows
        if ((report.AlternativeFlows?.Count ?? 0) > 0)
        {
            ws.Cell(row, 1).Value = "Alternative Flows:";
            ws.Cell(row, 1).Style.Font.Bold = true;
            ws.Cell(row, 1).Style.Fill.BackgroundColor = XLColor.LightGray;
            row++;

            foreach (var altFlow in report.AlternativeFlows ?? new List<AlternativeFlow>())
            {
                ws.Cell(row, 2).Value = $"{altFlow.FlowId}. {altFlow.FlowName}";
                ws.Cell(row, 2).Style.Font.Bold = true;
                row++;

                foreach (var step in altFlow.Steps)
                {
                    ws.Cell(row, 2).Value = $"{step.Step}. {step.Description ?? ""}";
                    row++;
                }
            }
        }

        // ============== EXCEPTIONS SECTION ==============
        if ((report.Exceptions?.Count ?? 0) > 0)
        {
            ws.Cell(row, 1).Value = "Exceptions:";
            ws.Cell(row, 1).Style.Font.Bold = true;
            ws.Cell(row, 1).Style.Fill.BackgroundColor = XLColor.LightGray;
            row++;

            foreach (var exception in report.Exceptions ?? new List<ExceptionFlow>())
            {
                ws.Cell(row, 2).Value = $"{exception.ExceptionId}: {exception.ExceptionName}";
                ws.Cell(row, 2).Style.Font.Bold = true;
                row++;

                foreach (var description in exception.Descriptions)
                {
                    ws.Cell(row, 2).Value = $"- {description}";
                    row++;
                }
            }
        }

        // ============== ADDITIONAL INFORMATION SECTION ==============
        ws.Cell(row, 1).Value = "Priority:";
        ws.Cell(row, 1).Style.Font.Bold = true;
        ws.Cell(row, 1).Style.Fill.BackgroundColor = XLColor.LightGray;
        ws.Cell(row, 2).Value = report.Priority;
        row++;

        ws.Cell(row, 1).Value = "Frequency of Use:";
        ws.Cell(row, 1).Style.Font.Bold = true;
        ws.Cell(row, 1).Style.Fill.BackgroundColor = XLColor.LightGray;
        ws.Cell(row, 2).Value = report.FrequencyOfUse;
        row++;

        ws.Cell(row, 1).Value = "Business Rules:";
        ws.Cell(row, 1).Style.Font.Bold = true;
```

OutPut

ExportUseCaseReportToExcel(useCaseReport)
Main: Export use case report to Excel format
Inputs:
useCaseReport (UseCaseReport) - Complete use case data
Returns: byte[] - Excel file bytes
Edge cases:
Null use case report
Empty collections (preconditions, postconditions, flows)
Very long text fields
Missing required fields
Dependencies:
ClosedXML library
Excel formatting
Memory stream handling

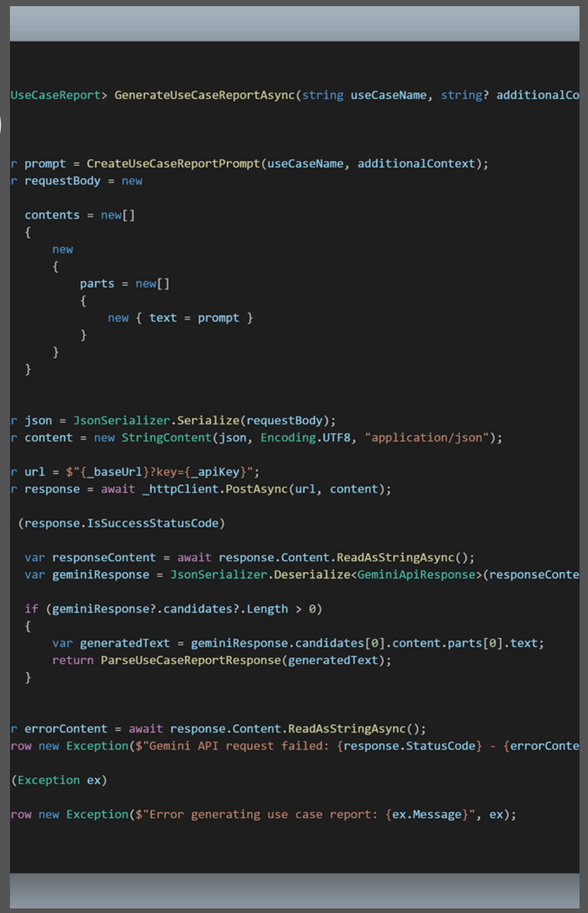
Prompt

Analyze GenerateUseCaseReportAsyncl() function and class required and identify all functions that need unit testing:

[IMG]

For each function, identify:

1. Main functionality
2. Input parameters and types
3. Expected return values
4. Potential edge cases
5. Dependencies that need mocking



```
UseCaseReport> GenerateUseCaseReportAsync(string useCaseName, string? additionalCo


r prompt = CreateUseCaseReportPrompt(useCaseName, additionalContext);
r requestBody = new

  contents = new[]
  {
      new
      {
          parts = new[]
          {
              new { text = prompt }
          }
      }
  }


r json = JsonSerializer.Serialize(requestBody);
r content = new StringContent(json, Encoding.UTF8, "application/json");

r url = $"{_baseUrl}?key={_apiKey}";
r response = await _httpClient.PostAsync(url, content);

 (response.IsSuccessStatusCode)

  var responseContent = await response.Content.ReadAsStringAsync();
  var geminiResponse = JsonSerializer.Deserialize<GeminiApiResponse>(responseConte

  if (geminiResponse?.candidates?.Length > 0)
  {
      var generatedText = geminiResponse.candidates[0].content.parts[0].text;
      return ParseUseCaseReportResponse(generatedText);
  }


r errorContent = await response.Content.ReadAsStringAsync();
row new Exception($"Gemini API request failed: {response.StatusCode} - {errorConte

(Exception ex)

row new Exception($"Error generating use case report: {ex.Message}", ex);
```

OutPut

GenerateUseCaseReportAsync(useCaseName, additionalContext)
Main: Generate comprehensive use case report using AI
Inputs:
useCaseName (string) - Name of the use case
additionalContext (string, optional) - Additional context for the use case
Returns: UseCaseReport object with complete use case specification
Edge cases:
Empty use case name
Null additional context
Very long use case names
Special characters in use case name
Dependencies:
Google Gemini API
HTTP client configuration
JSON parsing and validation

Phase 2: Test Cases Design

Prompt

Generate comprehensive unit test cases for the funtion ExportUseCaseReportToExcel () i give you before
Include:
- Happy path scenarios
- Edge cases (boundary values)
- Error scenarios
- Integration with cart state"

	Test Case	Input	Expected	Dependencies
Happy Path Tests	ExportUseCaseReportToExcel_ValidUs	Valid UseCaseReport	Excel bytes array	None
	ExportUseCaseReportToExcel_ValidUs	Valid report	Contains header	None
Edge Case Tests	ExportUseCaseReportToExcel_UseCas	Empty collections	Handles correctly	None
	ExportUseCaseReportToExcel_UseCas	10000+ character text	Handles correctly	None
	ExportUseCaseReportToExcel_UseCas	Special characters	Handles correctly	None
Data Validation Tests	ExportUseCaseReportToExcel_UseCas	Complex report	Contains all items	None
	ExportUseCaseReportToExcel_UseCas	Multiple flows	Contains all flows	None
Excel Formatting Tests	ExportUseCaseReportToExcel_ValidUs	Valid report	Proper formatting	None

Prompt

Generate comprehensive unit test cases for the funtion GenerateUseCaseReportAsync() i give you before
Include:
- Happy path scenarios
- Edge cases (boundary values)
- Error scenarios
- Integration with cart state"

	Test Case	Input	Expected	Dependencies
Happy Path Tests	GenerateUseCaseReportAsync_ValidR	Valid use case name	Attempts processing	None
	GenerateUseCaseReportAsync_ValidR	Request without context	Attempts processing	None
Input Validation Tests	GenerateUseCaseReportAsync_WithN	Null use case name	ArgumentNullException	None
	GenerateUseCaseReportAsync_WithE	Empty use case name	ArgumentException	None
Edge Case Tests	GenerateUseCaseReportAsync_WithVe	1000+ character name	Handles gracefully	None
	GenerateUseCaseReportAsync_WithSp	Special characters	Handles gracefully	None
Configuration Tests	GeminiService_WithValidConfiguratio	Valid configuration	Service initialized	None
	GeminiService_WithNullApiKey_Throw	Null API key	InvalidOperationException	None
	GeminiService_WithEmptyApiKey_Thr	Empty API key	InvalidOperationException	None
Service Initialization Tests	GeminiService_WithNullHttpClient_Th	Null HttpClient	ArgumentNullException	None
	GeminiService_WithNullConfiguration	Null configuration	ArgumentNullException	None
Error Handling Tests	GenerateUseCaseReportAsync_WithIn	Invalid API key	Exception thrown	None
	GenerateUseCaseReportAsync_WithIn	Invalid base URL	Exception thrown	None

Phase 3: Generate Test Code

GenerateUseCaseReportAsync – Happy Path Tests

GenerateUseCaseReportAsync – Input Validation Tests

GenerateUseCaseReportAsync – Edge Case Tests

Configuration Tests

Service Initialization Tests

Error Handling Tests

```
act]
    public async Task GenerateUseCaseReportAsync_ValidRequest_WillAttemptProcessing()
    {
        // Arrange
        var httpClient = new HttpClient();
        var service = new GeminiService(httpClient, _mockConfiguration.Object);
        var useCaseName = "User Login";
        var additionalContext = "Authentication system";

        // Act & Assert
        // This will fail due to no real AI service, but we can test the method exist
        await Assert.ThrowsAsync<Exception>(() =>
            service.GenerateUseCaseReportAsync(useCaseName, additionalContext));
    }
```

Prompt

You are a C# Unit Testing Expert using xUnit, Moq, and FluentAssertions.
I have a GeminiService.cs class that needs better unit testing. I also have an existing test file, GeminiServiceUseCaseReportTests.cs, but it has two major problems:
It only tests the GenerateUseCaseReportAsync method and the constructor. It completely misses GenerateTestCasesAsync and GenerateUseCaseTableAsync.
The existing tests for GenerateUseCaseReportAsync are flawed. They use new HttpClient() and just Assert.ThrowsAsync<Exception>. This doesn't actually test the method's logic (like success, API failure, or parsing).
Your task is to generate a new, complete test file named GeminiServiceTests.cs that fixes these problems and provides comprehensive coverage.
Requirements:
Mock HttpClient: You MUST NOT use new HttpClient(). You must mock the HttpClient's behavior by using a Mock<HttpMessageHandler>. This is critical for simulating API responses.
Cover All Public Methods: Write tests for GenerateTestCasesAsync, GenerateUseCaseTableAsync, and fix the tests for GenerateUseCaseReportAsync.
Use IConfiguration Mock: Continue using Mock<IConfiguration> as shown in the existing tests.
Test Scenarios: For each of the three Generate...Async methods, you must generate tests for the following scenarios:

- Success (200 OK):
  - Simulate a HttpStatusCode.OK response with a valid, well-formed JSON payload (based on the GeminiApiResponse structure).
  - Assert that the method returns the correctly parsed object (UseCaseReport or GeminiTestCaseResponse).
  - Assert that Success is true for GeminiTestCaseResponse.
- API Failure (e.g., 400 Bad Request):
  - Simulate a HttpStatusCode.BadRequest or HttpStatusCode.InternalServerError response.
  - Assert that the method throws an Exception containing the error message from the API.
- Malformed JSON Response:
  - Simulate a HttpStatusCode.OK response but with invalid JSON (e.g., "{"bad": "json"").
  - Assert that the method throws an Exception (from the Parse... methods) indicating a parsing error.
- Empty Candidates Response:
  - For GenerateTestCasesAsync and GenerateUseCaseTableAsync, simulate a HttpStatusCode.OK response with valid JSON but an empty candidates array.
  - Assert that the method returns a GeminiTestCaseResponse where Success is false and a proper Message is set.

Input Validation: The current GeminiService code does not have explicit null/empty checks for the string inputs in its public methods. Create tests that check for ArgumentNullException or ArgumentException for userRequirement and useCaseName. (This will require adding the validation to the GeminiService implementation, but you should write the tests as if it exists).

```
ExportUseCaseReportToExcel – Happy Path Tests

ExportUseCaseReportToExcel – Edge Case Tests

ExportUseCaseReportToExcel – Data Validation Tests

Excel Formatting Tests

Helper Methods
```

```
[Fact]
    public void ExportUseCaseReportToExcel_ValidUseCaseReport_ContainsCorrectHeader()
    {
        // Arrange
        var useCaseReport = CreateValidUseCaseReport();

        // Act
        var result = _excelService.ExportUseCaseReportToExcel(useCaseReport);

        // Assert
        using var workbook = new XLWorkbook(new MemoryStream(result));
        var worksheet = workbook.Worksheet("Use Case Report");

        worksheet.Cell("A1").Value.Should().Be("II. Requirement Specifications");
    }
```

Prompt

You are a C# Unit Testing expert using xUnit, FluentAssertions, and the ClosedXML.Excel library.
Your task is to create a unit test class named ExcelServiceUseCaseReportTests for the provided ExcelService.cs class.
Base Code (ExcelService.cs):
Instructions for the Test Class:

1. Test Focus: The test class must only contain tests for the ExportUseCaseReportToExcel(UseCaseReport report) method. Do not generate tests for ExportToExcel(TestCaseRequest request) or any of the private helper methods.
2. Instantiation: The ExcelService must be instantiated directly in the test class's constructor (_excelService = new ExcelService();). Do not use any mocks.
3. Test Scenarios (Must Include):
   - Happy Path:
     - ..._ReturnsExcelBytes: Checks that the byte[] result is not null or empty.
     - ..._ContainsCorrectHeader: Checks that the generated Excel file's cell A1 contains the "II. Requirement Specifications" title.
     - ..._ContainsBasicInformation: Checks for basic fields like "Created By:" and its value.
     - ..._ContainsPreconditions: Checks that precondition text is present.
     - ..._ContainsPostconditions: Checks that postcondition text is present.
     - ..._ContainsNormalFlow: Checks that normal flow steps are present.
     - ..._ContainsAlternativeFlows: Checks that alternative flow steps are present.
     - ..._ContainsExceptions: Checks that exception flow text is present.
   - Edge Cases:
     - ..._NullUseCaseReport_ThrowsException: Asserts that passing a null report throws an ArgumentNullException.
     - ..._UseCaseReportWithEmptyCollections_HandlesCorrectly: Confirms the method runs successfully when all List<> properties on the report are empty.
     - ..._UseCaseReportWithVeryLongText_HandlesCorrectly: Confirms the method runs successfully when report properties contain very long strings (e.g., 10,000+ characters).
     - ..._UseCaseReportWithSpecialCharacters_HandlesCorrectly: Confirms the method runs successfully when report properties contain special characters.
   - Data Validation:
     - ..._UseCaseReportWithMultipleItems_ContainsAllItems: Confirms that when a report has multiple preconditions, all of them are present in the file.
     - ..._UseCaseReportWithMultipleFlows_ContainsAllFlows: Confirms that all normal flow steps are present.
   - Formatting:
     - ..._HasProperFormatting: A simple test to check that the worksheet's RangeUsed() is not null.
4. Test Helper Methods (Crucial): The test class must include the following private helper methods to support the tests:
   - CreateValidUseCaseReport(): A helper that returns a standard, fully-populated UseCaseReport object with sample data for all fields.
   - CreateComplexUseCaseReport(): A helper that returns a report with multiple items in its collections (e.g., multiple preconditions, postconditions, flows) to test data validation.
   - FindRowContaining(IXLWorksheet worksheet, string text): A helper method that uses ClosedXML to iterate through the worksheet and find the row number containing a specific string.
   - GetWorksheetContent(IXLWorksheet worksheet): A helper method that reads all cell values from the worksheet into a single string.

Generate the complete ExcelServiceUseCaseReportTests.cs file, including all necessary using statements, the test class, the constructor, and all the tests and helper methods described.

Result

Phase 4: Run and Debug Test

Test overall:

cmd: dotnet test

```
ailed!  - Failed:     1, Passed:    31, Skipped:     0, Total:    32, Duration: 447 ms - TestcaseGenerator.Tests.dll (net9.0)
xUnit.net 00:00:01.00]
estcaseGenerator.Service.Tests.Services.ExcelServiceUseCaseReportTests.ExportUseCaseReportToExcel_UseCaseReportWithMultipleItems_ContainsAllItems
FAIL]
xUnit.net 00:00:01.03]
estcaseGenerator.Service.Tests.Services.ExcelServiceUseCaseReportTests.ExportUseCaseReportToExcel_ValidUseCaseReport_ContainsExceptions [FAIL]
xUnit.net 00:00:01.16]
estcaseGenerator.Service.Tests.Services.ExcelServiceUseCaseReportTests.ExportUseCaseReportToExcel_ValidUseCaseReport_ContainsPreconditions [FAIL]
Passed!  - Failed:     0, Passed:    12, Skipped:     0, Total:    12, Duration: 97 ms - TestcaseGenerator.Integration.Tests.dll (net9.0)
dotnet : [xUnit.net 00:00:00.65]
TestcaseGenerator.Service.Tests.Services.GeminiServiceUseCaseReportTests.GenerateUseCaseReportAsync_WithNullUseCaseName_ThrowsException [FAIL]
At line:1 char:1
+ dotnet test
+ ~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: ([xUnit.net 00:0...xception [FAIL]:String) [], RemoteException
    + FullyQualifiedErrorId : NativeCommandError

[xUnit.net 00:00:00.65]
TestcaseGenerator.Service.Tests.Services.GeminiServiceUseCaseReportTests.GenerateUseCaseReportAsync_WithEmptyUseCaseName_ThrowsException [FAIL]
[xUnit.net 00:00:00.58]
TestcaseGenerator.Tests.Controllers.TestcaseControllerUseCaseReportTests.TestcaseController_WithNullConfiguration_ThrowsException [FAIL]
  [xUnit.net 00:00:01.18]
  TestcaseGenerator.Service.Tests.Services.ExcelServiceUseCaseReportTests.ExportUseCaseReportToExcel_ValidUseCaseReport_ContainsNormalFlow [FAIL]
  [xUnit.net 00:00:01.20]
  TestcaseGenerator.Service.Tests.Services.ExcelServiceUseCaseReportTests.ExportUseCaseReportToExcel_ValidUseCaseReport_ContainsBasicInformation
  [FAIL]
  [xUnit.net 00:00:01.21]
  TestcaseGenerator.Service.Tests.Services.ExcelServiceUseCaseReportTests.ExportUseCaseReportToExcel_ValidUseCaseReport_ContainsCorrectHeader [FAIL]
  [xUnit.net 00:00:01.23]
  TestcaseGenerator.Service.Tests.Services.ExcelServiceUseCaseReportTests.ExportUseCaseReportToExcel_ValidUseCaseReport_ContainsPostconditions [FAIL]
  [xUnit.net 00:00:01.23]
  TestcaseGenerator.Service.Tests.Services.ExcelServiceUseCaseReportTests.ExportUseCaseReportToExcel_NullUseCaseReport_ThrowsException [FAIL]
  [xUnit.net 00:00:01.27]
  TestcaseGenerator.Service.Tests.Services.ExcelServiceUseCaseReportTests.ExportUseCaseReportToExcel_UseCaseReportWithMultipleFlows_ContainsAllFlows
  [FAIL]
  [xUnit.net 00:00:01.29]
  TestcaseGenerator.Service.Tests.Services.ExcelServiceUseCaseReportTests.ExportUseCaseReportToExcel_ValidUseCaseReport_ContainsAlternativeFlows
  [FAIL]
   at System.Net.Sockets.Socket.<ConnectAsync>g__WaitForConnectWithCancellation|285_0(AwaitableSocketAsyncEventArgs saea, ValueTask connect
CancellationToken cancellationToken)
   at System.Net.Http.HttpConnectionPool.ConnectToTcpHostAsync(String host, Int32 port, HttpRequestMessage initialRequest, Boolean async,
CancellationToken cancellationToken)

Failed!  - Failed:    13, Passed:    19, Skipped:     0, Total:    32, Duration: 1 s - TestcaseGenerator.Service.Tests.dll (net9.0)
```

Missing Input Validation (Guard Clauses)

Phase 5: Modify and Mocking

Prompt

Help me define error and code for each specific failed testcase and give me a modify way to update the test case to achieve better result

TestcaseGenerator\Service\GeminiService.cs

```
1  public GeminiService(HttpClient httpClient, IConfiguration configuration)
2  {
3      _httpClient = httpClient;
4      _configuration = configuration;
5      _apiKey = _configuration["GeminiApi:ApiKey"] ?? throw new InvalidOperationException("Gemini API key not configured");
6      _baseUrl = _configuration["GeminiApi:BaseUrl"] ?? "...";
7  }
8
```

Missing Input Validation (Guard Clauses)

Solution

You need to add "guard clauses" (input checks) at the beginning of your methods and constructors.

```
1  public GeminiService(HttpClient httpClient, IConfiguration configuration)
2  {
3      _httpClient = httpClient ?? throw new ArgumentNullException(nameof(httpClient));
4      _configuration = configuration ?? throw new ArgumentNullException(nameof(configuration));
5
6      var apiKey = _configuration["GeminiApi:ApiKey"];
7      if (string.IsNullOrEmpty(apiKey))
8      {
9          throw new InvalidOperationException("Gemini API key not configured");
10     }
11     _apiKey = apiKey;
12
13     _baseUrl = _configuration["GeminiApi:BaseUrl"] ?? "...";
14 }
15
```

Incorrect Excel Test Assertions (Type Mismatch)

Prompt

Help me define error and code for each specific failed testcase and give me a modify way to update the test case to achieve better result

TestcaseGenerator.Service.Tests\Services\ExcelServiceUseCaseReportTests.cs

```
worksheet.Cell("A1").Value.Should().Be("II. Requirement Specifications");
worksheet.Cell("A3").Value.Should().Be("Created By:");
worksheet.Cell($"B{precondition1Row}").Value.Should().Be("PRE-1: User has valid account");

```

Incorrect Excel Test Assertions (Type Mismatch)

Solution

You need to explicitly convert the cell's value to a string in your assertions.

```
// AFTER ✅
worksheet.Cell("A1").Value.ToString().Should().Be("II. Requirement Specifications");
worksheet.Cell("A3").Value.ToString().Should().Be("Created By:");
worksheet.Cell($"B{precondition1Row}").Value.ToString().Should().Be("PRE-1: User has valid account");

// Or even safer with ClosedXML:
worksheet.Cell("A1").GetText().Should().Be("II. Requirement Specifications");
worksheet.Cell("A3").GetText().Should().Be("Created By:");
worksheet.Cell($"B{precondition1Row}").GetText().Should().Be("PRE-1: User has valid account");

```

Incorrect Test Data (Logical Error)

Prompt

Help me define error and code for each specific failed testcase and give me a modify way to update the test case to achieve better result

TestcaseGenerator.Service.Tests\Services\ExcelServiceUseCaseReportTests.cs

```
// In ExportUseCaseReportToExcel_UseCaseReportWithMultipleItems_ContainsAllItems
// BEFORE ❌
content.Should().Contain("PRE-1: User has valid account");

```

```
// In ExportUseCaseReportToExcel_UseCaseReportWithMultipleFlows_ContainsAllFlows
// BEFORE ❌
content.Should().Contain("1. User navigates to login page");

```

Incorrect Test Data (Logical Error)

Solution

Change the strings in the Should().Contain() assertions to match the data from CreateComplexUseCaseReport.

```
// AFTER ✅
content.Should().Contain("PRE-1: Customer has valid account");
content.Should().Contain("PRE-3: Customer has sufficient funds");

```

```
// AFTER ✅
content.Should().Contain("1. Customer selects payment method");
content.Should().Contain("5. System confirms payment success");

```

Prompt

Generate comprehensive unit test cases for this funtion callAIAutoTaskGenerator()
Include:
- Happy path scenarios
- Edge cases (boundary values)
- Error scenarios
- Integration with cart state"

```
/**
 * Main function: Auto generate tasks từ project
 */
export async function autoGenerateTasksFromProject(
  request: AutoGenerateTasksRequest
): Promise<GenerateTasksResponse> {
  try {
    // 1. Lấy thông tin project
    const projectInfo = await getProjectInfo(request.project_id);

    // 2. Lấy team members
    const teamMembers = await getTeamMembersByProject(request.project_id);

    // 3. Default sprint params nếu không có
    const sprintParams = request.sprint_params || {
      duration_weeks: 2,
      start_date: new Date().toISOString().split('T')[0]
    };

    // 4. AI generate tasks
    const aiResponse = await callAIAutoTaskGenerator(
      projectInfo,
      teamMembers,
      sprintParams
    );

    // 5. Lưu vào database
    const savedTasks = await saveTasksToDatabase(aiResponse.tasks);

```

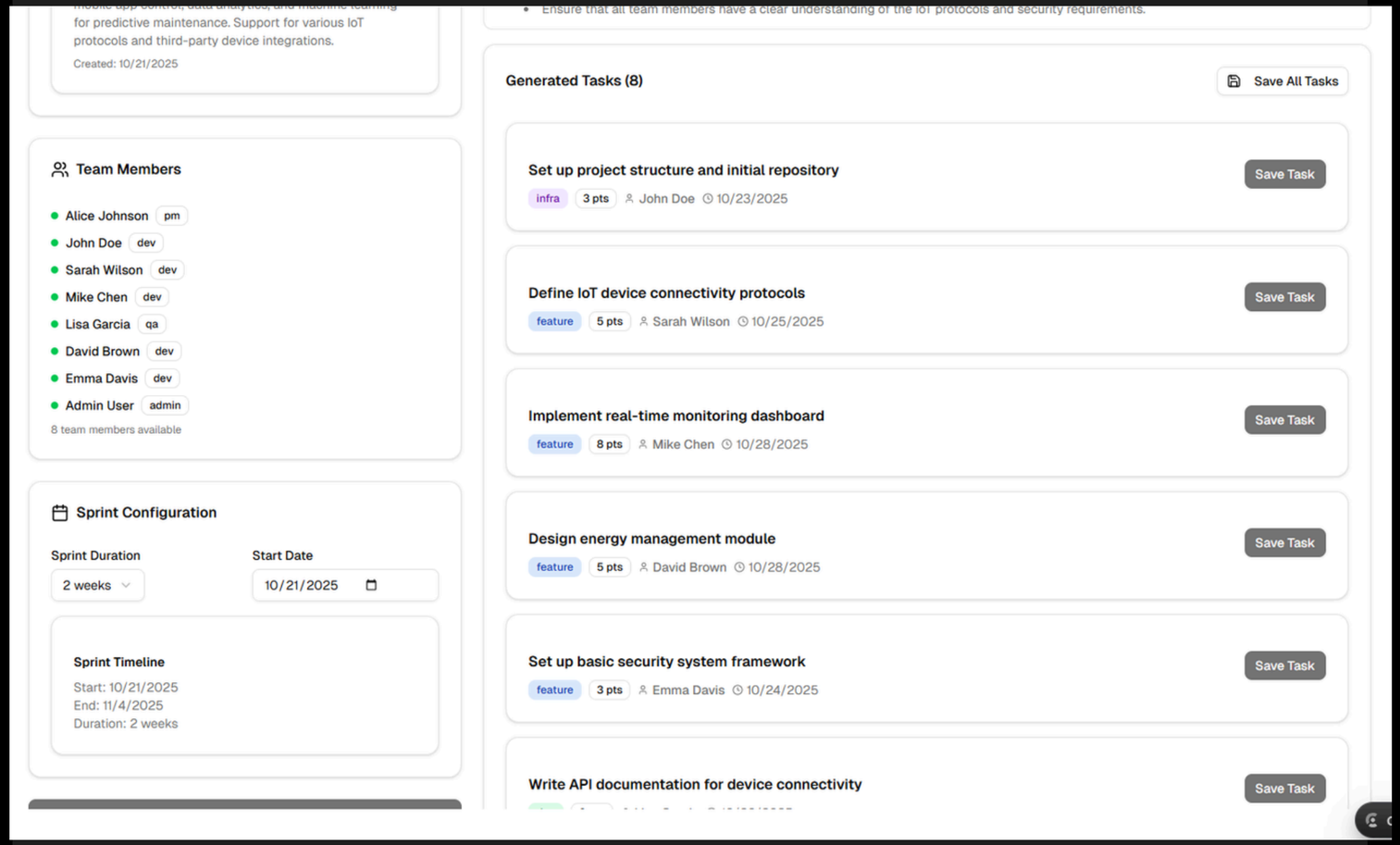
Prompt

Generate comprehensive unit test cases for this funtion callAIAutoTaskGenerator()
Include:
- Happy path scenarios
- Edge cases (boundary values)
- Error scenarios
- Integration with cart state"

```
    // 5. Lưu vào database
    const savedTasks = await saveTasksToDatabase(aiResponse.tasks);

    return {
      success: true,
      tasks: savedTasks,
      warnings: aiResponse.warnings,
      sprint_plan: aiResponse.sprint_plan,
      task_details: aiResponse.task_details
    };
  } catch (error) {
    console.error('Error auto generating tasks:', error);
    return {
      success: false,
      tasks: [],
      warnings: [],
      sprint_plan: {
        sprint_number: 1,
        duration_weeks: request.sprint_params?.duration_weeks || 2,
        start_date:
          request.sprint_params?.start_date ||
          new Date().toISOString().split('T')[0],
        end_date: '',
        total_capacity: 0,
        total_committed: 0,
        utilization_percentage: 0,
        tasks: []
      },
      task_details: {},
      error: error instanceof Error ? error.message : 'Unknown error'
    };
  }
}
```

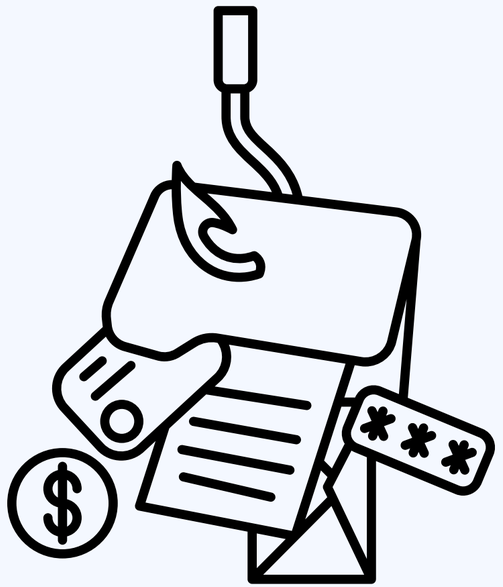
Prompt

Generate comprehensive unit test cases for this funtion callAIAutoTaskGenerator()
Include:
- Happy path scenarios
- Edge cases (boundary values)
- Error scenarios
- Integration with cart state"



```typescript
// Test suite cho AI Auto Task Generator - Function gọi AI để tự động tạo tasks
describe('AI Auto Task Generator', () => {
  const mockProjectInfo: ProjectInfo = {
    id: '1',
    name: 'Test Project',
    description: 'A test project',
    owner_id: 'user1',
    created_at: '2024-01-01'
  };

  const mockTeamMembers: TeamMember[] = [
    {
      id: 'user1',
      email: 'dev@test.com',
      display_name: 'Developer',
      role: 'dev',
      created_at: '2024-01-01'
    }
  ];

  const mockSprintParams: SprintParams = {
    duration_weeks: 2,
    start_date: '2024-01-01'
  };

  beforeEach(() => {
    vi.clearAllMocks();
  });

  // Test: Tạo tasks thành công khi AI API hoạt động bình thường
  it('should generate tasks using AI when API is available', async () => {
    const mockResponse = {
      candidates: [
        {
          content: {
            parts: [
              {
                text: JSON.stringify({
                  tasks: [
                    {
                      id: 'task_1',
                      title: 'Setup project',
                      type: 'infra',
                      estimate_pt: 5,
                      assignee: 'user1',
                      due_date: '2024-01-08',
                      project_id: '1',
                      status: 'todo',
                      created_at: '2024-01-01'
                    }
                  ],
                  task_details: {},
                  warnings: [],
                  sprint_plan: {}
                })
              }
            ]
          }
        }
      ]
    };

    vi.mocked(fetch).mockResolvedValueOnce({
      ok: true,
      json: () => Promise.resolve(mockResponse)
    } as Response);

    const result = await callAIAutoTaskGenerator(
      mockProjectInfo,
      mockTeamMembers,
      mockSprintParams
    );

    expect(result.tasks).toHaveLength(1);
    expect(result.tasks[0].title).toBe('Setup project');
  });

  // Test: Sử dụng fallback tasks khi AI API thất bại
  it('should fallback to basic tasks when AI fails', async () => {
    vi.mocked(fetch).mockRejectedValueOnce(new Error('API Error'));

    const result = await callAIAutoTaskGenerator(
      mockProjectInfo,
      mockTeamMembers,
      mockSprintParams
    );

    expect(result.tasks).toHaveLength(4); // Basic fallback tasks
    expect(result.warnings).toContain(
      'Using fallback auto task generation - AI service unavailable'
    );
  });
```

KEY PRINCIPLES OF CYBER SECURITY



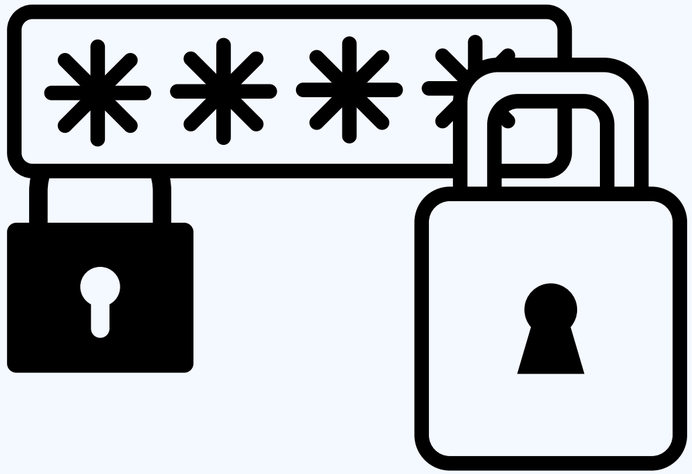
Core Security Principles

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi ac purus lacinia, egestas ipsum tincidunt, vehicula est. Donec quis laoreet turpis. Duis bibendum sem eget elit laoreet scelerisque. Proin eget mattis eros, eget condimentum nisl. Aenean semper sem a sem maximus hendrerit vitae id ex.

CYBER SECURITY TOOLS AND TECHNOLOGIES

Core Security Principles

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi ac purus lacinia, egestas ipsum tincidunt, vehicula est. Donec quis laoreet turpis. Duis bibendum sem eget elit laoreet scelerisque. Proin eget mattis eros, eget condimentum nisl. Aenean semper sem a sem maximus hendrerit vitae id ex.



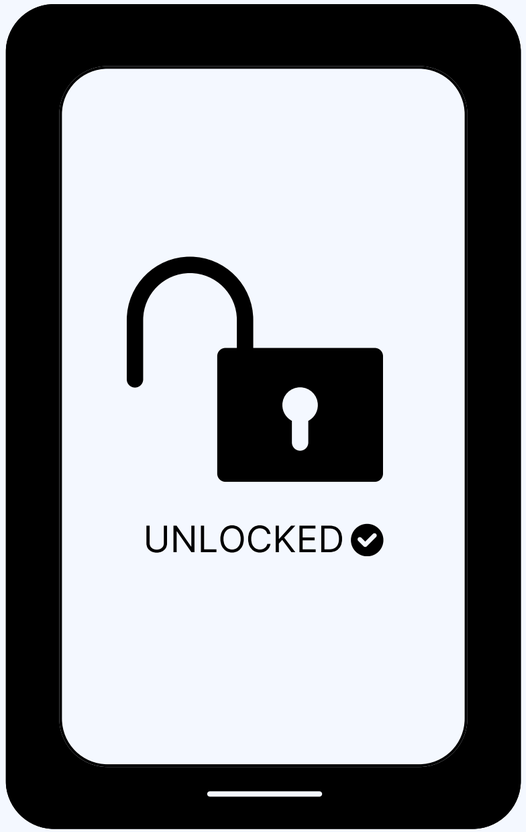
CONCLUSION

The Future of Cyber Security

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi ac purus lacinia, egestas ipsum tincidunt, vehicula est. Donec quis laoreet turpis. Duis bibendum sem eget elit laoreet scelerisque. Proin eget mattis eros, eget condimentum nisl. Aenean semper sem a sem maximus hendrerit vitae id ex.

Cyber | Protect | Data | Threat | Security | Attack | Firewall | Malware

THANKYOU



Stay Safe, Stay Secure

Thynk Unlimited

@reallygreatsite