Assignment 2: Calculator Goes to Production

1. The Mission

Congratulations, your calculator was a hit!

Sarah calls you into her office, beaming: "Great news! The client loved the calculator demo so much they want to integrate it into their enterprise system. But here’s the thing - their QA team requires all code to have unit tests before deployment. Also, their architects say the code needs to be modularized so other teams can reuse the calculation logic."

She slides a coffee across the desk: "Time to make this production-ready. Break out the logic into testable methods and prove it works with unit tests!"

Submission: GitHub repository URL

2. Learning Objectives

By completing this assignment, you will:

- Extract logic into reusable methods
- Understand method parameters and return values
- Write unit tests using xUnit
- Practice the Arrange-Act-Assert pattern

3. Stage 1: Extract the Calculator Logic

3.1. Requirements

Create a new class `Calculator` with the following methods:

```
public class Calculator
{
    public double Add(double a, double b)
    public double Subtract(double a, double b)
    public double Multiply(double a, double b)
    public double Divide(double a, double b)
    public double SquareRoot(double a)
    public double Power(double baseNum, double exponent)
}
```

The `Divide` method should throw an `ArgumentException` when dividing by zero. The `SquareRoot` method should throw an `ArgumentException` for negative numbers.

4. Stage 2: Write Unit Tests

4.1. Requirements

Create a test project and write unit tests for all calculator methods:

1. Create a new xUnit test project in your solution
2. Write at least **one test per method** (6 tests minimum)
3. Write **two additional tests** for error cases:
    - Division by zero throws `ArgumentException`
    - Square root of negative number throws `ArgumentException`

4.2. Example Test Structure

```
public class CalculatorTests
{
    [Fact]
    public void Add_TwoPositiveNumbers_ReturnsSum()
    {
        // Arrange
        double a = 5;
        double b = 3;

        // Act
        double result = Calculator.Add(a, b);

        // Assert
        Assert.Equal(8, result);
    }

    [Fact]
    public void Divide_ByZero_ThrowsArgumentException()
    {
        // Arrange
        double a = 10;
        double b = 0;

        // Act & Assert
        Assert.Throws<ArgumentException>(() => Calculator.Divide(a, b));
    }
}
```

4.3. Hints & APIs to Explore

- `dotnet new xunit -n Calculator.Tests` - Create test project
- `dotnet test` - Run all tests
- `Assert.Equal(expected, actual)` - Check values match
- `Assert.Throws<T>(() => …)` - Check exception is thrown

5. Tips for Success

- Keep methods simple - one calculation per method
- Test edge cases (zero, negative numbers, decimals)
- Run tests frequently as you develop
- Use descriptive test method names

Sarah’s parting words: "Once QA signs off on those tests, we’re shipping this to production. No pressure!"

Good luck!