# SARS Specialist Developer Technical Assessment

Focus Areas: ReactJS, C#, Code Quality, Testing, Problem Solving

Time Limit: 5 working days

**Submission**: GitHub link including code, unit tests, and other necessary files.

## Part 1: ReactJS Application (Frontend)

Scenario

SARS maintains internal tools to process risk data, including transaction inputs. Your task is to build a small frontend tool that lets a user enter and sort numerical transaction values.

Requirements

* Form Input: Accept a comma-separated string of values.
* Validation: Accepts only numbers (decimals and integers). Returns an error if input contains non-numeric values.
* Display: Render the sorted numbers in both ascending and descending order. Show validation errors clearly.
* Toggle: Allow user to switch between ascending and descending views.

Technical Expectations

* Use functional components with React Hooks.
* Structure your project to reflect scalable, maintainable code.

Deliverables

* Source code in GitHub.
* A README.md explaining how to run the app and key design decisions.
* A text-based breakdown of how you would unit test your application.

## Part 2: C# Application (Backend / Logic)

Scenario

The SARS Customs Risk Engine interprets Reverse Polish Notation (RPN) rules for risk scoring. Your task is to build a parser that converts RPN to infix notation, ensuring clarity and accuracy.

Requirements

* Accepts an RPN expression as input.
* Converts the expression into a properly grouped infix string.
* Supports +, -, \*, / operators and numbers (integers and decimals).
* Handles invalid inputs gracefully.

Bonus Extensions (Optional)

* Provide an evaluation mode that calculates the result.

Technical Expectations

* Use C#.
* Emphasize object-oriented design.
* Include documentation and comments.

Deliverables

* Source code in GitHub.
* A README.md explaining architecture and how to run the app.
* A text-based breakdown of how you would unit test your application.

## Part 3: Code Review Scenario

Task

You are given the following pieces of code from a junior developer. They have issues in structure, naming, and logic. In a separate file, refactor them and explain what you changed and why.

JavaScript Code Sample:

function doThings(data){  
 let x = data.split(',');  
 x.sort();  
 return x.join(',');  
}

C# Code Sample:

public class RiskEngine {  
 public string Parse(string input){  
 string[] parts = input.Split(' ');  
 return string.Join(",", parts.Reverse());  
 }  
}

## Evaluation Criteria

We are assessing:

* Code clarity and organization
* Proper use of React and C# idioms
* Validation and error handling
* Ability to reason and explain code