1. Setting up the Project

Prerequisites

- Node.js installed
- Ganache to deploy the contract
- Truffle (optional)

2. Smart Contract

Here's a simple Solidity smart contract for a calculator:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0.;

contract Calculator{
    struct Calculation{
        uint256 int1;
        uint256 int2;
        string operation;
        uint256 result;
    }

    Calculation[] public calculations;

    event CalculationPerformed(uint256 int1, uint256 int2, string operation,
uint256 result);

    function add(uint256 a, uint256 b) public returns (uint256) {
        uint256 result = a+b;
        Calculation memory newCalculation = Calculation(a, b, "Addition",
result);

        calculations.push(newCalculation);
        emit CalculationPerformed(a, b, "Addition", result);
        return result;
    }

    function subtract(uint256 a, uint256 b) public returns (uint256) {
```

```
        uint256 result = a-b;
        Calculation memory newCalculation = Calculation(a, b, "Subtraction",
result);

        calculations.push(newCalculation);
        emit CalculationPerformed(a, b, "Subtraction", result);
        return result;
    }

    function multiply(uint256 a, uint256 b) public returns (uint256) {
        uint256 result = a*b;
        Calculation memory newCalculation = Calculation(a, b, "Multiplication",
result);

        calculations.push(newCalculation);
        emit CalculationPerformed(a, b, "Multiplication", result);
        return result;
    }

    function divide(uint256 a, uint256 b) public returns (uint256) {
        require(b != 0, "Division by zero");
        uint256 result = a/b;
        Calculation memory newCalculation = Calculation(a, b, "Division",
result);

        calculations.push(newCalculation);
        emit CalculationPerformed(a, b, "Division", result);
        return result;
    }

    function viewLatestCalculation() public view returns (uint256 int1, uint256
int2, string memory operation, uint256 result) {
        require(calculations.length > 0, "No calculations yet.");
        Calculation memory lastCalculation = calculations[calculations.length -
1];
        return (lastCalculation.int1, lastCalculation.int2,
lastCalculation.operation, lastCalculation.result);
    }
```

```
    function viewCalculation(uint256 index) public view returns (uint256 int1,
uint256 int2, string memory operation, uint256 result) {
        require(index < calculations.length, "Calculation does not exist.");
        Calculation memory selectedCalculation = calculations[index];
        return (selectedCalculation.int1, selectedCalculation.int2,
selectedCalculation.operation, selectedCalculation.result);
    }
 }
```

3. Deploy the Smart Contract

To deploy this contract locally (or on a test network), you can use Truffle or Remix.

Using Truffle:

1. Compile the contract:

```
truffle compile
```

2. Deploy the contract:

```
truffle migrate --network development
```

Once deployed, note down the contract address and ABI.

4. Frontend Setup

Now, the smart contract needs to be integrated with the frontend using Web3.js.

1. Install web3.js:

```
npm install web3
```

2. Create a basic HTML file for the calculator interface

5. JavaScript Integration (app.js)

Then, we need to write the logic to interact with the smart contract via Web3.js.

Open the frontend in your browser and interact with the contract.