User story

1. As a user I should be able to query for the sales of a specific year and display the value of the sales of the selected year in the screen.

   Total sales and a breakdown (list) of the total sales by State. (Orders.State)

   Sales DB field = Products.Sales

   Calculation

   Total Sales = Year Sales – Year Returns

Answer:

Step1: Create Database and Import Data

```
CREATE TABLE Products (
   ProductID INT PRIMARY KEY,
   ProductName NVARCHAR(100),
   Sales DECIMAL(18, 2)
);

CREATE TABLE Orders (
   OrderID INT PRIMARY KEY,
   ProductID INT,
   OrderDate DATE,
   State NVARCHAR(50),
   Returns DECIMAL(18, 2),
   FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);
```

Step2: Import Data from Excel to SQL Server

Stpe3: Create ASP.NET Core Web API Project

```
dotnet new webapi -n SalesAPI
```

Step4: Add Models and DbContext

```
public class Product
{
   public int ProductID { get; set; }
```

```
    public string ProductName { get; set; }
    public decimal Sales { get; set; }
}

public class Order
{
    public int OrderID { get; set; }
    public int ProductID { get; set; }
    public DateTime OrderDate { get; set; }
    public string State { get; set; }
    public decimal Returns { get; set; }
    public Product Product { get; set; }
}
```

Create a SalesDbContext:

```
public class SalesDbContext : DbContext
{
    public SalesDbContext(DbContextOptions<SalesDbContext> options) : base(options) { }

    public DbSet<Product> Products { get; set; }
    public DbSet<Order> Orders { get; set; }
}
```

Configure the DbContext in Startup.cs:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<SalesDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
    services.AddControllers();
}
```

```
"ConnectionStrings": {
```

```
  "DefaultConnection": "Server=your_server;Database=SalesDB;Trusted_Connection=True;"
}
```

Step 5: Create Sales Controller

```
[ApiController]
[Route("[controller]")]
public class SalesController : ControllerBase
{
    private readonly SalesDbContext _context;

    public SalesController(SalesDbContext context)
    {
        _context = context;
    }

    [HttpGet("{year}")]
    public async Task<IActionResult> GetSales(int year)
    {
        var totalSales = await _context.Orders
            .Where(o => o.OrderDate.Year == year)
            .SumAsync(o => o.Product.Sales - o.Returns);

        var salesByState = await _context.Orders
            .Where(o => o.OrderDate.Year == year)
            .GroupBy(o => o.State)
            .Select(g => new
            {
                State = g.Key,
                StateSales = g.Sum(o => o.Product.Sales - o.Returns)
            })
            .ToListAsync();
```

```
        return Ok(new
        {
            TotalSales = totalSales,
            SalesByState = salesByState
        });
    }
}
```

User stories

2. As a user I should be able to apply a percentage of increment to the total sales of the selected year, to simulate the increase of sales in the next year.

   Display the increment number of Sales after applying the percentage.

   Display the increment by state in a column after the actual value.

   Bonus: Apply individual percentages of increase per state.

   If you need to populate a “State” dropdown, use the Orders.State.

Step1: Create a SalesIncrementRequest Model**:**

```
public class SalesIncrementRequest
{
    public int Year { get; set; }
    public decimal OverallIncrement { get; set; }
    public Dictionary<string, decimal> StateIncrements { get; set; }
}
```

Step2: **Update Sales Controller**:

```
[ApiController]
[Route("[controller]")]
public class SalesController : ControllerBase
{
    private readonly SalesDbContext _context;

    public SalesController(SalesDbContext context)
```

```
    {
        _context = context;
    }

    [HttpGet("{year}")]
    public async Task<IActionResult> GetSales(int year)
    {
        var totalSales = await _context.Orders
            .Where(o => o.OrderDate.Year == year)
            .SumAsync(o => o.Product.Sales - o.Returns);

        var salesByState = await _context.Orders
            .Where(o => o.OrderDate.Year == year)
            .GroupBy(o => o.State)
            .Select(g => new
            {
                State = g.Key,
                StateSales = g.Sum(o => o.Product.Sales - o.Returns)
            })
            .ToListAsync();

        return Ok(new
        {
            TotalSales = totalSales,
            SalesByState = salesByState
        });
    }

    [HttpPost("increment")]
    public async Task<IActionResult> CalculateSalesIncrement([FromBody] SalesIncrementRequest
request)
    {
        var totalSales = await _context.Orders
```

```
            .Where(o => o.OrderDate.Year == request.Year)
            .SumAsync(o => o.Product.Sales - o.Returns);

        var salesByState = await _context.Orders
            .Where(o => o.OrderDate.Year == request.Year)
            .GroupBy(o => o.State)
            .Select(g => new
            {
                State = g.Key,
                StateSales = g.Sum(o => o.Product.Sales - o.Returns)
            })
            .ToListAsync();

        var totalIncrement = totalSales * (request.OverallIncrement / 100);

        var stateIncrements = salesByState.ToDictionary(
            s => s.State,
            s => s.StateSales * ((request.StateIncrements.ContainsKey(s.State) ?
request.StateIncrements[s.State] : request.OverallIncrement) / 100)
        );

        return Ok(new
        {
            TotalIncrement = totalIncrement,
            StateIncrements = stateIncrements
        });
    }

    [HttpGet("states")]
    public async Task<IActionResult> GetStates()
    {
        var states = await _context.Orders
            .Select(o => o.State)
```

```
            .Distinct()
            .ToListAsync();

        return Ok(states);
    }
}
```

User Story:

3. As a user I should be able to download the forecasted data to a csv, with the next columns State, Percentage increase, Sales value.

Step1: **Add a CSV Export Method**:

```
using System.IO;
using System.Text;

[HttpPost("export")]
public async Task<IActionResult> ExportToCsv([FromBody] SalesIncrementRequest request)
{
    var totalSales = await _context.Orders
        .Where(o => o.OrderDate.Year == request.Year)
        .SumAsync(o => o.Product.Sales - o.Returns);

    var salesByState = await _context.Orders
        .Where(o => o.OrderDate.Year == request.Year)
        .GroupBy(o => o.State)
        .Select(g => new
        {
            State = g.Key,
            StateSales = g.Sum(o => o.Product.Sales - o.Returns)
        })
        .ToListAsync();
```

```
    var stateIncrements = salesByState.ToDictionary(
        s => s.State,
        s => s.StateSales * ((request.StateIncrements.ContainsKey(s.State) ? request.StateIncrements[s.State] : request.OverallIncrement) / 100)
    );

    var csvBuilder = new StringBuilder();
    csvBuilder.AppendLine("State,Percentage Increase,Sales Value");

    foreach (var state in stateIncrements)
    {
        csvBuilder.AppendLine($"{state.Key},{request.StateIncrements.ContainsKey(state.Key) ? request.StateIncrements[state.Key] : request.OverallIncrement},{state.Value}");
    }

    var csvContent = csvBuilder.ToString();
    var bytes = Encoding.UTF8.GetBytes(csvContent);
    var result = new FileContentResult(bytes, "text/csv")
    {
        FileDownloadName = "forecasted_sales.csv"
    };

    return result;
}
```

Step2: **Add Download Button in HTML**:

```
<!DOCTYPE html>
<html>
<head>
    <title>Sales Query</title>
</head>
<body>
    <h1>Sales Query</h1>
```

```
    <input type="number" id="year" placeholder="Enter year" />
    <button onclick="fetchSales()">Get Sales</button>
    <h2>Total Sales: <span
    id="totalSales"></span></h2>
    <h2>Sales by State:</h2>
    <ul id="salesByState"></ul>

    <h1>Sales Increment Simulation</h1>
    <input type="number" id="overallIncrement" placeholder="Enter overall increment %" />
    <button onclick="applyIncrement()">Apply Increment</button>
    <h2>Total Increment: <span id="totalIncrement"></span></h2>
    <h2>State Increments:</h2>
    <ul id="stateIncrements"></ul>

    <h3>Individual State Increments</h3>
    <select id="stateDropdown"></select>
    <input type="number" id="stateIncrement" placeholder="Enter state increment %" />
    <button onclick="addStateIncrement()">Add State Increment</button>
    <ul id="individualStateIncrements"></ul>

    <button onclick="downloadCsv()">Download Forecasted Data</button>

    <script>
        const stateIncrements = {};

        async function fetchSales() {
            const year = document.getElementById('year').value;
            const response = await fetch(`/Sales/${year}`);
            const data = await response.json();

            document.getElementById('totalSales').innerText = data.totalSales;
            const stateSales = data.salesByState.map(sale => `<li>${sale.state}:
${sale.stateSales}</li>`).join('');
```

```
        document.getElementById('salesByState').innerHTML = stateSales;

        await fetchStates();
    }

    async function fetchStates() {
        const response = await fetch(`/Sales/states`);
        const states = await response.json();
        const stateDropdown = document.getElementById('stateDropdown');
        stateDropdown.innerHTML = states.map(state => `<option value="${state}">${state}</option>`).join('');
    }

    async function applyIncrement() {
        const year = document.getElementById('year').value;
        const overallIncrement = document.getElementById('overallIncrement').value;

        const response = await fetch(`/Sales/increment`, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                year: parseInt(year),
                overallIncrement: parseFloat(overallIncrement),
                stateIncrements: stateIncrements
            })
        });

        const data = await response.json();
        document.getElementById('totalIncrement').innerText = data.totalIncrement;
        const stateIncrementsList = Object.entries(data.stateIncrements).map(([state, increment]) => `<li>${state}: ${increment}</li>`).join('');
```

```
        document.getElementById('stateIncrements').innerHTML = stateIncrementsList;
    }

    function addStateIncrement() {
        const state = document.getElementById('stateDropdown').value;
        const increment = document.getElementById('stateIncrement').value;
        stateIncrements[state] = parseFloat(increment);
        const individualStateIncrements = Object.entries(stateIncrements).map(([state, increment]) =>
`<li>${state}: ${increment}%</li>`).join('');
        document.getElementById('individualStateIncrements').innerHTML =
individualStateIncrements;
    }

    async function downloadCsv() {
        const year = document.getElementById('year').value;
        const overallIncrement = document.getElementById('overallIncrement').value;

        const response = await fetch(`/Sales/export`, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                year: parseInt(year),
                overallIncrement: parseFloat(overallIncrement),
                stateIncrements: stateIncrements
            })
        });

        const blob = await response.blob();
        const url = window.URL.createObjectURL(blob);
        const a = document
            .createElement('a');
```

```
            a.href = url;
            a.download = 'forecasted_sales.csv';
            document.body.appendChild(a);
            a.click();
            a.remove();
            window.URL.revokeObjectURL(url);
        }
    </script>
</body>
</html>
```