

TMO Task Report

Vilda Duan

1. Introduction

This project implements a full-stack solution to display a report of the top-performing sellers by month, filtered by branch. The frontend is built using React with TypeScript, while the backend is a .NET 8 API following clean architecture principles. The backend reads order data from a CSV file and exposes RESTful endpoints consumed by the frontend.

The goal is to build a responsive frontend that allows users to select a branch and view seller performance by month. First, a backend API was developed which reads and processes CSV data, exposing clean APIs for branch listing and seller performance. And then a frontend interface was developed using React and TypeScript, of which the UX fetches and displays branches, and user can select a specific branch to view top seller name, total orders, and total price in each month.

Figure 1 shows a user interface generated for this project, after implementing backend and frontend. It clearly shows which seller has the most total sales in each month, filtered by Branch.

Monthly Top Sellers Report

Branch 1

Top Sellers for Branch 1

Month	Top Seller	Orders	Total Sales
January	Seller B	4	\$1084.50
February	Seller C	2	\$492.28
March	Seller B	3	\$1106.91
April	Seller C	4	\$914.41
May	Seller B	1	\$422.70
June	Seller D	2	\$688.58
July	Seller C	3	\$1023.37
August	Seller D	1	\$271.17
September	Seller B	2	\$786.10
October	Seller D	1	\$115.23
November	Seller C	2	\$600.61
December	Seller C	1	\$291.34

Figure 1. User interface displaying top-sellers by month, filtered by branch.

2. Backend Implementation (.NET Core)

The backend is built on .NET 8, utilizing CsvHelper for efficient CSV parsing, following clean architecture principles with clearly separated layers including Controllers for API endpoints, Services for business logic and Models for data structures, ensuring maintainability and testability throughout the application. The overall structure of Backend is shown in Figure 2.

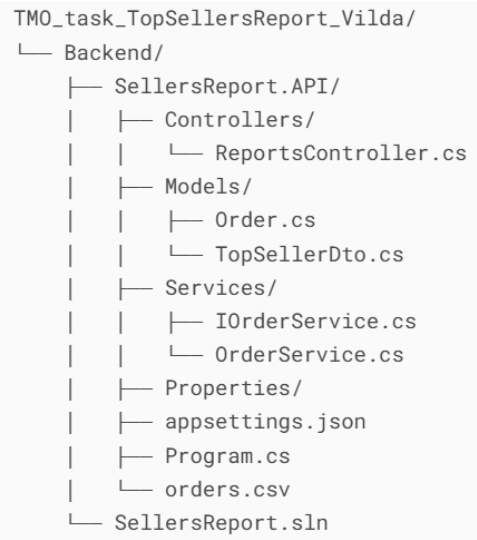


Figure 2. The structure layout for backend.

The orders.csv file was loaded using CsvHelper and mapped to an internal data model within OrderService.cs. A dedicated service layer was implemented to parse the CSV, validate input rows, group data by month and branch, and aggregate seller data by calculating total orders and total price. The processed data was structured to enable efficient monthly grouping and quick filtering by branch.

Two RESTful endpoints were implemented in ReportsController.cs: GET /api/branches retrieves the list of unique branches dynamically extracted from the CSV file, while GET /api/sellers/top?branch={branchName} returns the top-performing seller for each month based on the specified branch.

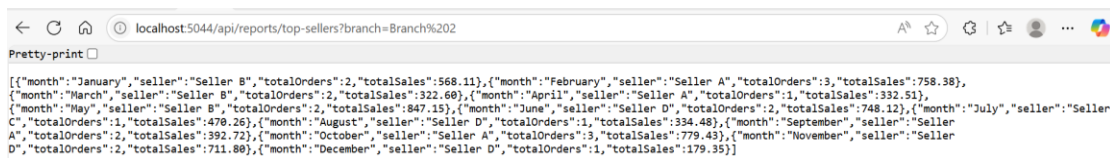
Unit tests were implemented using xUnit to validate core business logic, including CSV data processing and top-seller aggregation. The tests were executed using the “dotnet test” command, and the test suite completed successfully with all tests passing. The output confirmed that one test was discovered and passed with no failures or skips, ensuring reliable functionality of the implemented logic, as shown in Figure 3.

```
E:\TMO_task_TopSellersReport_Vilda\Backend\SellersReport.API>cd /d "E:\TMO_task_TopSellersReport_Vilda\Backend\SellersReport.tests"
E:\TMO_task_TopSellersReport_Vilda\Backend\SellersReport.Tests>dotnet test
Restore complete (0.7s)
SellersReport.API succeeded (0.5s) → E:\TMO_task_TopSellersReport_Vilda\Backend\SellersReport.API\bin\Debug\net9.0\SellersReport.API.dll
SellersReport.Tests succeeded (0.5s) → bin\Debug\net9.0\SellersReport.Tests.dll
[xUnit.net 00:00:00.00] xUnit.net VSTest Adapter v2.8.2+699d445a1a (64-bit .NET 9.0.5)
[xUnit.net 00:00:00.85]   Discovering: SellersReport.Tests
[xUnit.net 00:00:00.89]   Discovered: SellersReport.Tests
[xUnit.net 00:00:00.89]   Starting: SellersReport.Tests
[xUnit.net 00:00:00.95]   Finished: SellersReport.Tests
SellersReport.Tests test succeeded (4.4s)

Test summary: total: 1, failed: 0, succeeded: 1, skipped: 0, duration: 4.4s
Build succeeded in 7.2s
```

Figure 3. Dotnet test results of backend.

The backend run smoothly on a browser, showing branches from the CSV file, as well as a brief top-sellers layout, as shown in Figure 4.



```
localhost:5044/api/reports/top-sellers?branch=Branch%202
Pretty-print
[{"month":"January","seller":"Seller B","totalOrders":2,"totalSales":568.11}, {"month":"February","seller":"Seller A","totalOrders":3,"totalSales":758.38}, {"month":"March","seller":"Seller B","totalOrders":2,"totalSales":322.68}, {"month":"April","seller":"Seller A","totalOrders":1,"totalSales":332.51}, {"month":"May","seller":"Seller B","totalOrders":2,"totalSales":847.15}, {"month":"June","seller":"Seller D","totalOrders":2,"totalSales":748.12}, {"month":"July","seller":"Seller C","totalOrders":1,"totalSales":470.26}, {"month":"August","seller":"Seller D","totalOrders":1,"totalSales":334.48}, {"month":"September","seller":"Seller A","totalOrders":2,"totalSales":392.72}, {"month":"October","seller":"Seller A","totalOrders":3,"totalSales":779.43}, {"month":"November","seller":"Seller D","totalOrders":2,"totalSales":711.88}, {"month":"December","seller":"Seller D","totalOrders":1,"totalSales":179.35}]
```

Figure 4. Backend run results showing top-sellers on Branch 2.

3. Frontend Implementation

```
sellers-report-frontend/
├─ node modules/
├─ src/
│   ├─ api/                # API service calls to the backend
│   ├─ components/         # (dropdowns, seller tables)
│   ├─ models/             # TypeScript interfaces
│   ├─ pages/              # Page-level components
│   ├─ styles/
│   ├─ App.tsx
│   ├─ index.tsx
│   ├─ setupTests.ts
│   └─ react-app-env.d.ts
├─ jest.config.js
├─ jest.setup.js
├─ tsconfig.json
└─ package.json
```

Figure 5. The structure layout for frontend.

The sellers-report-frontend is a React-based frontend application built with TypeScript. It is designed to interact with backend APIs, display seller data, and provide an intuitive UI for users to manage and view reports on sellers. The project is structured to promote modularity, maintainability, and scalability. The structure layout of frontend is shown in Figure 5.

The project was started by setting up it with Create React App using the TypeScript template. This provided a solid base with strict typing and better tooling. The tsconfig.json file was configured for strict type safety, and necessary dependencies such as React Router, Axios, and styling/testing libraries were added.

API service modules were placed in the src/api/ directory to handle all HTTP requests to the backend. This approach keeps the logic centralized, reusable, and easier to maintain.

Reusable UI components including dropdowns (BranchDropdown.tsx) and seller tables (SellersTable.tsx) were built in the src/components/ folder. These components are stateless, styled for reusability, and help maintain a consistent UI.

Testing is set up using Jest and React Testing Library. Several test files including BranchDropdown.test.tsx, SellersTable.test.tsx, ReportPage.tsx were implemented to test critical UI functionality. Figure 6 shows that all test cases are passed.

```
PASS src/components/__tests__/SellersTable.test.tsx
PASS src/pages/ReportPage.test.tsx
PASS src/components/__tests__/BranchDropdown.test.tsx

Test Suites: 3 passed, 3 total
Tests:      8 passed, 8 total
Snapshots:  0 total
Time:       8.031 s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

Figure 6. Frontend test results showing a robust frontend design.

The final UI results after running both backend and frontend are showing in Figure 7 and Figure 8

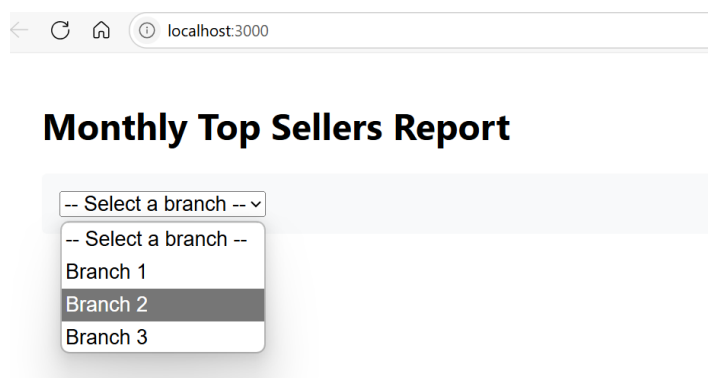


Figure 7. UI interface showing a dropdown menu.

Figure 7 shows a dropdown menu for a user to select, and there are 3 branches to select from. Figure 8 shows the final results when one of the branches is selected, with names of the top seller in each month within that branch. In addition, the number of orders is also displayed.

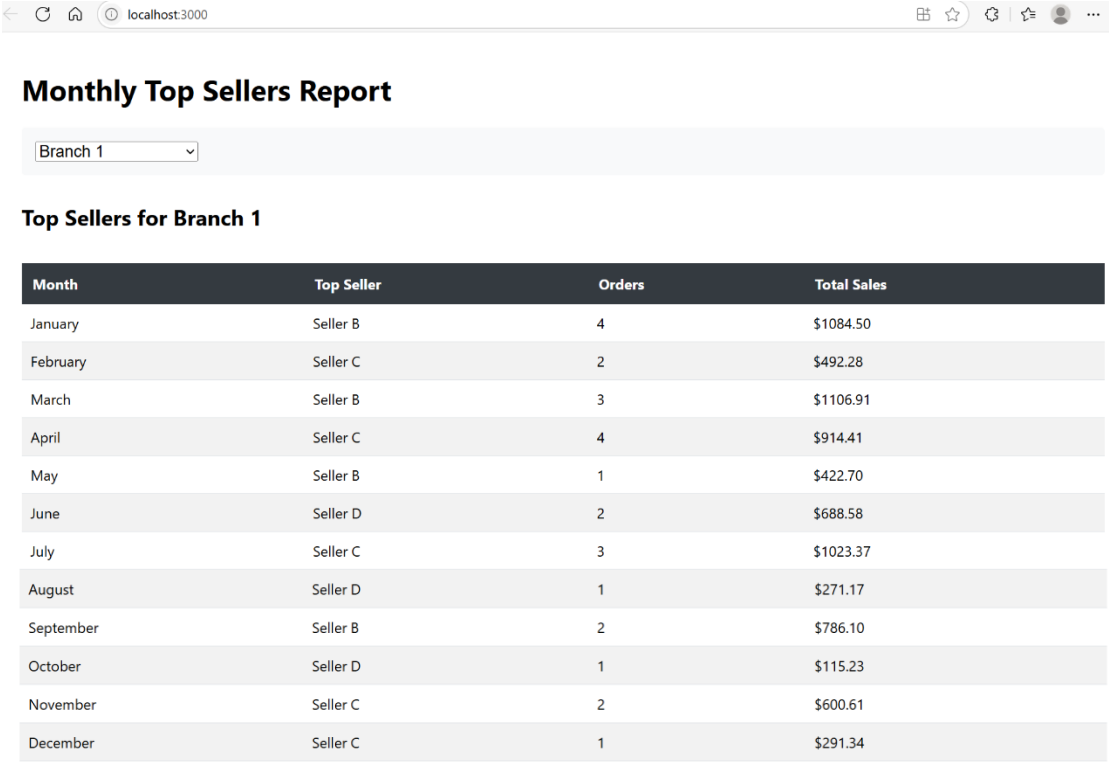


Figure 8. UI interface showing top sellers' information after selecting a branch.

4. Conclusion.

This project successfully delivers a complete full-stack solution for visualizing monthly top-performing sellers filtered by branch. By integrating a robust .NET 8 backend with a clean, modular React and TypeScript frontend, the system ensures both maintainability and a smooth user experience. The backend efficiently processes CSV data and provides structured, reliable endpoints, while the frontend offers a responsive and intuitive interface for users to interact with the data.