E-commerce Transaction Analyzer

Mini Project 1

Submitted by:
Group Name: Ctrl + Alt + F4 😀

Group Members:

- Cameron Carnegie
- Manbir Sidhu
- Balkar Singh
- Michael Parent
- Gurveer Chahal

Course: Functional & Logic Programming (COMP-481)
Instructor: Majid Babaei
Term: Fall 2025
Date: October 10th, 2025

Institution: University of Fraser Valley

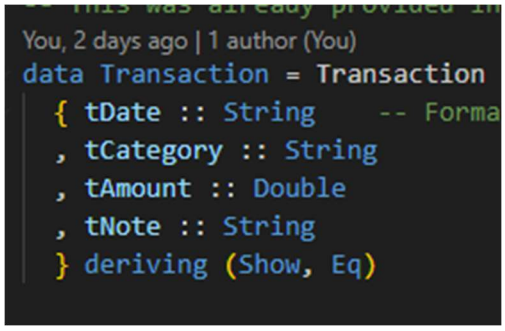
Overview

This project implements a transaction management system that allows users to filter transactions by category and amount, calculate spending totals by category, and format transaction data for display.

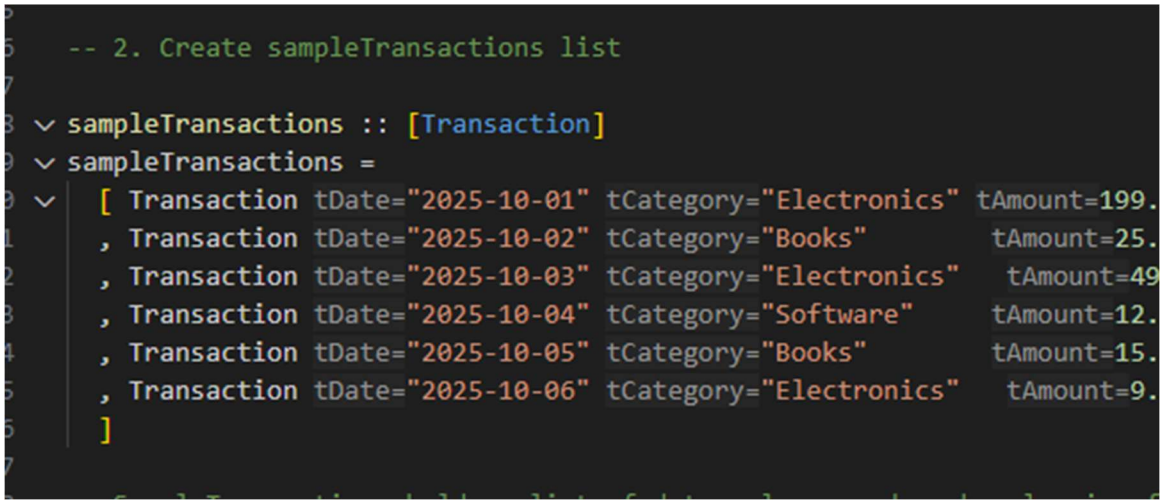
Features

Transaction Management

- **Custom Transaction Type**: Defines structured transaction records with date, category, amount, and notes



- **Sample Data**: Includes realistic sample transactions across multiple categories



Filtering Operations

- **Category Filter**: Filter transactions by specific category name
- **Amount Filter**: Filter transactions meeting minimum amount thresholds
- **Composable Filters**: Combine multiple filters for complex queries. [4]

Analysis Capabilities

- **Category Totals**: Calculate total spending per category using strict left-fold evaluation (`foldl'`) [2,4]
- **Sorted Output**: Automatically groups and sorts transactions by category for analysis

Formatted Output

- **Table Formatting**: Generates aligned, padded tables using `printf` for readable display
- **Consistent Layout**: Fixed-width columns for dates, categories, amounts, and notes [3]

Technical Highlights

1. Performance Optimization

Uses `foldl'` (strict left fold) instead of lazy `foldl` to prevent stack overflow on large transaction lists, ensuring efficient memory usage during aggregation operations[12].

2. Functional Composition

Demonstrates function composition with the `$` operator and chaining filters to build complex queries from simple, reusable functions. [4]

3. Type Safety

Leverages Haskell's strong type system with custom data types deriving `Show` and `Eq` for automatic string conversion and equality comparisons. [4]

Requirements

- GHC (Glasgow Haskell Compiler) 8.0 or higher
- Standard Haskell libraries (included with GHC)

Dependencies

```
Data.List        -- sortBy, groupBy, intercalate
Data.Ord         -- comparing
Data.Monoid      -- Sum
Data.Function    -- on
Data.Foldable    -- foldl'
Text.Printf      -- printf
```

Usage

Running the Application

Execute the compiled binary or run through GHCi to see demonstration output including all sample transactions, filtered views by category and amount, and spending totals[11].

Code Examples from the Code

The proper explanation of the code is given in the file “main.hs”. For simplicity sake we have attached the screenshot of the code with the headings.

Filter by Category:

```
filterByCategory "Electronics" sampleTransactions
-- Returns only Electronics transactions
```

```haskell
-- 3. Implement filterByCategory :: String -> [Transaction] -> [Transaction]
-- Filters transactions where tCategory matches the given category string.
filterByCategory :: String -> [Transaction] -> [Transaction]
filterByCategory categoryName =
  filter (\t -> tCategory t == categoryName)

-- Little Explainer:
-- Here our function filterByCategory takes two parameters
-- 1. categoryName of type String
-- 2. A list of Transaction objects [Transaction]
-- and then returns a filtered list of Transaction objects [Transaction]
```

Filter by Minimum Amount:

```
filterByMinAmount 50.00 sampleTransactions
-- Returns transactions >= $50.00
```

```haskell
-- 4. Implement filterByMinAmount :: Double -> [Transaction] -> [Transaction]
-- Filters transactions where tAmount is greater than or equal to the minimum amount.
filterByMinAmount :: Double -> [Transaction] -> [Transaction]
filterByMinAmount minAmount =
  filter (\t -> tAmount t >= minAmount)

  -- Little Explainer:
-- Here our function filterByMinAmount takes two parameters
-- 1. minAmount of type Double
-- 2. A list of Transaction objects [Transaction]
-- and then returns a filtered list of Transaction objects [Transaction]
```

Combined Filters:

```
filterByMinAmount 20.00 $ filterByCategory "Books" sampleTransactions
-- Returns Books transactions >= $20.00
```

```haskell
let expensiveBooks =
      filterByMinAmount 20.00 $
      filterByCategory "Books" sampleTransactions
putStrLn (formatTransactions expensiveBooks)
```

Calculate Category Totals:

```
totalByCategory sampleTransactions
-- Returns [(String, Double)] tuples of (category, total)
```

```haskell
-- 5. Implement totalByCategory :: [Transaction] -> [(String, Double)]
-- Calculates the total amount spent per category.
totalByCategory :: [Transaction] -> [(String, Double)]
totalByCategory transactions =
  -- a. Sort by category (required for groupBy to work correctly)
  -- sortBy is a built in function in Haskell that sorts a list based on a comparison function.
  -- comparing is a helper function that creates a comparison
  -- between two transactions based on the tCategory field.
  let sorted = sortBy (comparing tCategory) transactions
      -- b. Group consecutive transactions with the same category
      -- groupBy is a built in function that groups consecutive elements
      -- in a list based on a predicate (which is category).
      -- (==) means equality check
      -- on is a helper function that transforms a binary function
      -- (like (==)) into a function that operates on the results
      -- of applying another function (like tCategory).
      grouped = groupBy ((==) `on` tCategory) sorted
      -- c. Map groups to (category, sum amounts)
      -- calculateTotal is expecting a group of arguments (Transaction objects)
      -- grabs the category from the first element of the list
      -- foldl' is
      calculateTotal group =
        let category = tCategory (head group)
            total = foldl' (\acc t -> acc + tAmount t) 0.0 group
            -- fodl' is a strict version of foldl that helps avoid stack overflows
            -- on large lists by evaluating the accumulator at each step.
            -- It is a lambda function that takes an accumulator acc and a transaction t
            -- Basically acc (current total) + tAmount t (amount of current transaction)
            -- 0.0 is the initial value of the accumulator
            -- group is the list of transactions in the current category      You, 2 days ago •
        in (category, total)
  in map calculateTotal grouped
```

Project Structure

```
main.hs              -- Complete implementation
├── Transaction      -- Custom data type definition
├── Filters          -- filterByCategory, filterByMinAmount
```

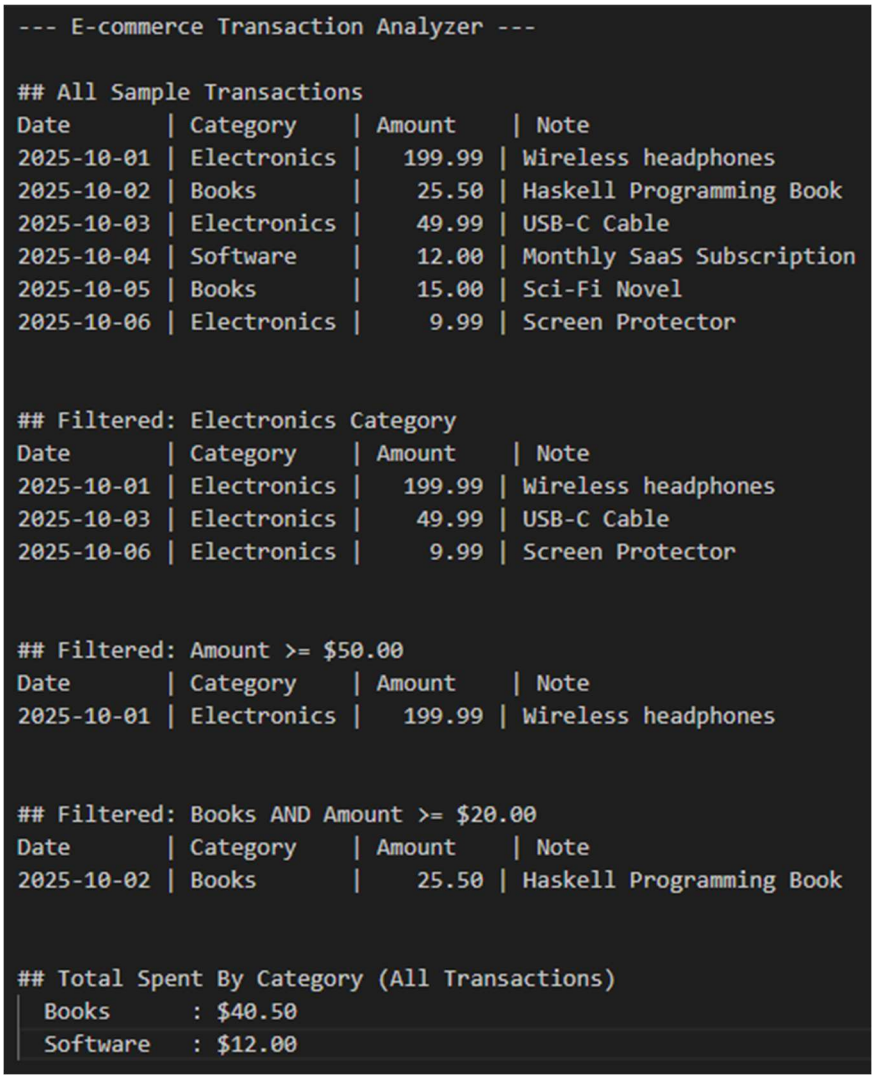
```
├── Analysis          -- totalByCategory
├── Formatting        -- formatTransactions
└── Main             -- Demonstration and output
```

Learning Objectives

This project demonstrates understanding of: - Custom algebraic data types with record syntax - Higher-order functions (`filter`, `map`, `foldl'`) - List processing with sorting and grouping - Strict evaluation for performance - Function composition and lambda expressions - Type signatures and type safety - String formatting with `printf`

Sample Output

The actual output of the file has been attached in the zip file whereas for the ease, I have attached the screen shot.



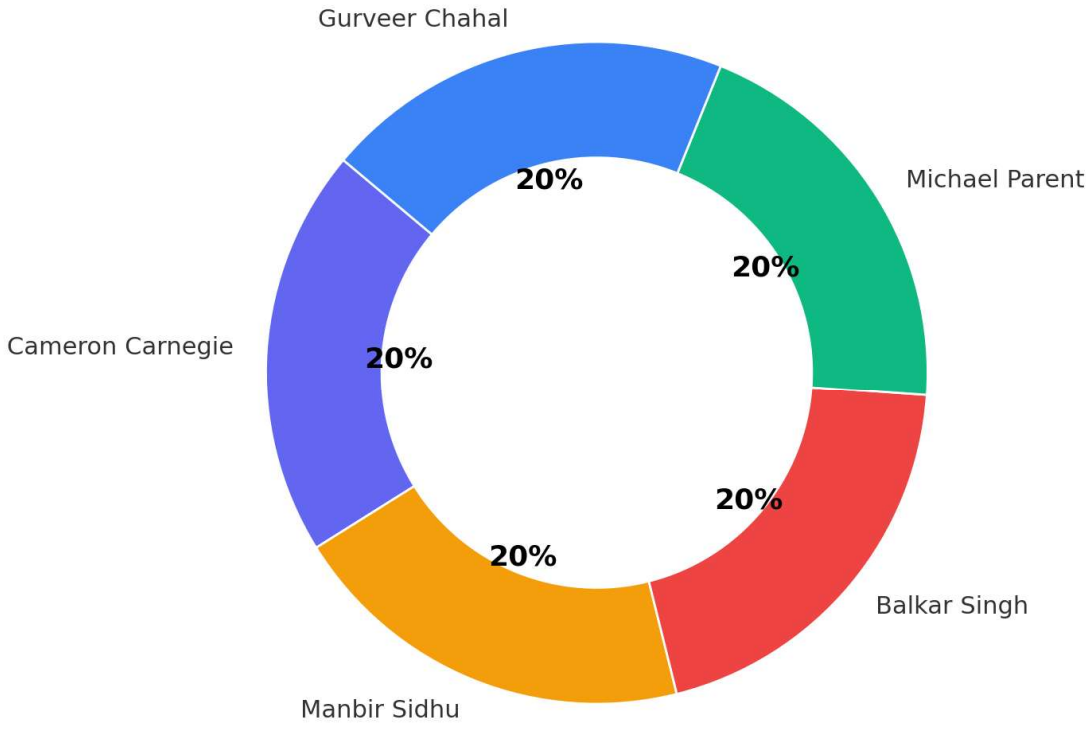
```
--- E-commerce Transaction Analyzer ---

## All Sample Transactions
Date       | Category    | Amount    | Note
2025-10-01 | Electronics |   199.99 | Wireless headphones
2025-10-02 | Books       |    25.50 | Haskell Programming Book
2025-10-03 | Electronics |    49.99 | USB-C Cable
2025-10-04 | Software    |    12.00 | Monthly SaaS Subscription
2025-10-05 | Books       |    15.00 | Sci-Fi Novel
2025-10-06 | Electronics |     9.99 | Screen Protector


## Filtered: Electronics Category
Date       | Category    | Amount    | Note
2025-10-01 | Electronics |   199.99 | Wireless headphones
2025-10-03 | Electronics |    49.99 | USB-C Cable
2025-10-06 | Electronics |     9.99 | Screen Protector


## Filtered: Amount >= $50.00
Date       | Category    | Amount    | Note
2025-10-01 | Electronics |   199.99 | Wireless headphones


## Filtered: Books AND Amount >= $20.00
Date       | Category    | Amount    | Note
2025-10-02 | Books       |    25.50 | Haskell Programming Book


## Total Spent By Category (All Transactions)
  Books      : $40.50
  Software   : $12.00
```

Extending the Project

We can improve this project further with these:

- Add date range filtering
- Implement transaction persistence (file I/O)
- Calculate statistics (average, median spending)
- Add transaction editing and deletion
- Export reports to CSV format
- Implement tag-based filtering

Contribution Chart



References

[1] https://blog.haskell.org/documentation-best-practices-in-2024/

[2] https://hackage.haskell.org/package/base-4.12.0.0/docs/src/Data.Foldable.html

[3] https://stackoverflow.com/questions/7947981/how-do-i-import-the-foldable-class-into-my-module

[4] https://hackage.haskell.org/package/base/docs/Data-Foldable.html

[5] https://xmonad.github.io/xmonad-docs/base-4.16.4.0/src/Data-Foldable.html

[6] https://stackoverflow.com/questions/28118063/how-can-i-make-this-fold-more-generic

[7] https://nikivazou.github.io/CMSC498V/lectures/MonoidsAndFoldables.html

[8] https://news.ycombinator.com/item?id=36773022

[9] https://haskell-haddock.readthedocs.io

[10] https://stackoverflow.com/questions/37874974/best-practices-for-distributing-a-haskell-application-and-updating-it

[11] https://www.youtube.com/watch?v=YYyMngA0HxM

[12] https://stackoverflow.com/questions/20356742/foldr-foldl-haskell-explanation

Note: I have not included in-text citations for all of the sources, as I was primarily focused on finding solutions and exploring better approaches during the research process.