Expense Tracker 1.0 (MVP)

Overview
This project is a full-stack mobile application for users to track and visualize their income and expenses. This project focuses on building an end-to-end product developing skills in front-end UI development, back-end API development, database management and cloud deployment.

Objectives

- Build an end-to-end mobile application.
- Deploy an app and database to the cloud.
- Build a clean and simplistic front-end with charts, graphs and tables.
- Gain experience with cloud deployment.
- Gain experience and skills using Angular and ASP.NET Core.

System Architecture
The application follows a 3-tier architecture ensuring security, scalability and maintainability. Both the back-end and the database are securely hosted on Microsoft Azure's cloud platform.

- Presentation Tier: The User Interface is built with Angular framework and handles data presentation and all user interactions.
- Application Tier: ASP.NET Core REST API running on .NET 10 is used for business logic and communicates between both the front-end and the database. The back-end is hosted on Microsoft Azure's cloud platform
- Data Tier: A PostgreSQL database reliably stores all user data.

Key features

- Can import transaction CSV files from banks.
- Fast loading charts and tables visualizing transactions for simple analysis.
- Expense aggregation including monthly totals, yearly totals and totals by transaction type.
- Securely cloud hosted REST API.

Challenges I faced and solutions.

One challenge I faced was making sure all charts and tables update dynamically when new transactions are added or removed. I solved this by using RxJS Observables as well as Angular Signals so components update whenever the data they rely on is changed. This increased scalability as I can create additional components without extra effort for syncing each one.

Another challenge was handling data validation from transaction data that users insert and import. Importing incorrect file attempts or malformed data could cause the system to fail. My solution was implementing validation within the back-end API and using well-defined models. Also when the user uploads an incorrect file type a clear message is returned informing the user exactly what went wrong.

Next Iteration Improvements

- Role-based access so users can save their data.
- Compatible with more NZ banks.
- Connect to open banking APIs.
- Extract keywords or use Natural Language Processing to classify transaction descriptions so I can categorize transactions more specifically.
- Add predictions for budgeting.
- Add more informative analytics and trends.
- Interactive payments map (future iteration)

Conclusion

During this project I gained experience developing using the latest versions of tools such as ASP.NET Core, Angular, PostgreSQL. I also improved programming skills in TypeScript, SQL and C#.
I look forward to completing the next iterations, adding new features and continually improving it.

From Bradley Erskine