Faculty of Engineering & Information Technology

32998 .Net Application Development – Spring 2025

Chi Yui Steve Chak 25952906

Jungho Jang 25628518

Assignment 2

Prof. Avinash Singh

17 Oct 2025

Table of content

Project Idea

Kipi is a contemporary, cross-platform desktop program that was designed with MudBlazor and developed with.NET 8 Blazor Hybrid. It is intended to tackle common household issues including unreported expenses, unequal financial accountability, and a lack of openness in day-to-day financial matters. Kipi is a stand-alone offline program that offers complete control over privacy, data, and stability, in contrast to cloud-based solutions.

Kipi, which draws inspiration from actual family experiences, establishes a common area where users may assign home chores, manage budgets, record transactions, and monitor wellness practices. It promotes cooperative budgeting and financial knowledge among all adult family members rather of depending on one person to handle everything.

The software has sections for goal-setting, tracking category expenses, budget planning, and wellness reminders. SQLite is used for local data storage, while Entity Framework Core provides safe access to the data. People of all ages can easily utilize the user interface because to its adaptable layouts, color-coded sections, and huge clickable elements.

The goal of Kipi is to transform home money management from a source of anxiety into a forum for collaboration. It encourages candid financial communication and boosts self-assurance in handling joint duties. In addition to addressing practical problems, the research shows how careful software design can support both technical and scholarly objectives.

Motivation

Stress, uncertainty, and imbalance are frequently the results of managing home finances. Uncertain spending, poor communication, and one person shouldering the whole financial load are problems for many families. Problems like excessive expenditure are sometimes overlooked until it is too late in the absence of shared visibility or adequate tracking.

Households require flexibility, offline access, and collaborative design, while the majority of current financial products concentrate on individual users or banking functions.

Kipi was created to address these issues. It provides a straightforward, offline-first interface where family members can track spending, manage budgets, and maintain financial literacy. Kipi assists households in developing trust, averting conflict, and gaining long-term financial confidence by encouraging shared responsibility and improved communication.

Vision

Kipi was founded on the idea that handling money in the home should be a team effort rather than a cause of contention. Its goal is very clear:

"To change financial management from conflict to cooperation."

Kipi allows everyone to manage, monitor, and comprehend pooled funds rather than depending on just one person. It helps people make confident, well-informed decisions together and promotes open discussions about spending and saving.

Kipi is a platform for communication, trust, and long-term financial well-being that goes beyond a budgeting tool.

Real-World Challenges and Solutions

Challenge	Solution
Untracked spending causes arguments between couples.	Shared expense views for user-specific profiles.
All money is handled by one person, which breeds distrust.	Multi account for multi family member in one app.
Budgeting is difficult for parents to teach.	Visual budget planner shows actual vs. target with alerts
Monthly budgets are not adhered to by families.	Centralised planner with indicators of actual vs live progress.
Stress related to money is not recognised until it is too late.	Spendings progress for overbudget categories.
Multiple payment methods obscure the complete financial picture.	Unified local tracking for all payment methods.
International family members contribute but lack visibility.	Transparency through shared views and exportable reports.

These solutions are designed to address not only technical gaps but also the social and emotional dynamics that affect financial harmony within families.

Key Features

Tech Stack

Feature	Implementation
GUI Framework	Blazor Hybrid (.NET 8) with MudBlazor
.NET Platform	C# (.NET 8) with Visual Studio 2022
Database and ORM	SQLite with Entity Framework

LINQ & Lambda	Filtering, summaries, and queries
Generics/Collections	List<T> for managing budgets and expenses
Interfaces/Polymorphism	Role-based logic and report generation
Unit Testing	NUnit for validating budget limits and logic

The application will be compiled and demonstrated using Visual Studio 2022 with .NET 8, ensuring compatibility with course requirements.

GUI Design and Interfaces

The frontend will be built using Blazor MudBlazor, which guarantees responsiveness and adherence to contemporary UI/UX standards. Blazor Hybrid seamlessly integrates.NET backend logic with desktop and web environments.

Key GUI features include:

Interfaces	Description
Home Page	The Home page serves as the main dashboard, giving customers a quick summary of the financial situation of their home. It shows the budget for the current month, the amount left over, the percentage of the budget that has been utilized, and a history of recent transactions. Users can easily see patterns or irregularities and evaluate the overall health of their expenditure.
Finance Page	Complete control over transaction management is available on this site. Financial entries can be seen, added, edited, and deleted by users. Filtering transactions by date range, category, and kind (revenue or spending) is possible. The results are shown in an interactive table with integrated pagination, sorting, and search. Financial data is constantly computed and updated in real time, including net balance, total revenue, and total spending.
The Wellbeing Page	Users may set and monitor their savings goals with the aid of the Wellbeing page. It facilitates financial goal setting, progress monitoring, and goal status or category-based filtering. Status chips and progress bars are examples of visual elements that give consumers instant feedback on how near they are to achieving their financial goals.
Reports Page	This module provides financial information and sophisticated analytics. Charts like line graphs, donut charts, and category breakdowns allow users to see their data. Users can examine reports by week, month, or year using filters. Households are able to comprehend individual

	spending patterns thanks to the support for member-specific information.
Settings Page	Options for managing user profiles and administering household members are available in the Settings interface. General profile options and family-specific customizations are separated by a tabbed structure. Changing the display name, PIN numbers, and controlling member visibility in shared views are among the features.
Login Page	The login page uses a PIN code approach for authentication. To access their data, users input their security code after choosing their profile from a dropdown list of family members.
Transaction Details Page	This detail view offers a thorough analysis of a particular transaction. Transaction ID, amount, category, date, description, and related member are among the fields it contains. Individual submissions may be quickly reviewed or audited using this site.

Blazor components and MANI ensure smooth data flow, unified user experience, and effective engagement across roles.

Data Management and flow

In order to guarantee privacy, dependability, and complete offline functionality, Kipi uses a local-first data management approach. SQLite, a lightweight embedded database, is used to store all application data, and Entity Framework Core (EF Core) is used for code-first access and manipulation.

When a user interacts with the application—for instance, by adding a new expenditure or making a monthly budget—the data flow starts. The in-memory models are updated as a result of these interactions, and EF Core is used to save the modifications to the local database. Key elements like Transaction, Budget, Goal, and User are represented by the structured tables in which the data is kept.

A simplified data flow is as follows:

User Input → ViewModel Binding → EF SaveChanges() → SQLite Storage → UI Refresh

Entities and Services Overview

Type	Name	Description / Responsibility
Entity	**User**	Represents a household member with basic identity and display role.

Entity	**Transaction**	Records income or expenses, including amount, date, category, and notes.
Entity	**Budget**	Stores target spending limits and tracks remaining balance per category.
Entity	**Goal**	Represents savings or wellness goals with progress tracking.
Entity	**Category**	Classifies types of expenses and goals (e.g., groceries, rent).
Entity	**Settings**	Stores application-wide preferences such as theme, currency, etc.
Service	**TransactionService**	Manages add/edit/delete of transaction data and summary calculations.
Service	**BudgetService**	Handles budget tracking, overspending detection, and updates.
Service	**GoalService**	Processes goal contributions and monitors completion status.
Service	**ReportService**	Aggregates and prepares data for reports and exports.
Service	**StorageService**	Manages database context, backup, and restore functions.

Testing and Validation

To ensure reliability and correctness, unit tests will be implemented using NUnit, focusing on the core functionalities of the application.

Key test cases include:

- Saving goal service tests: Verify the correct creation, update, and tracking of saving goals.
- Transaction service tests: Ensure that transactions are accurately recorded, categorized, and attributed to the correct user.

Early in the development cycle, testing will be incorporated to quickly identify problems and guarantee reliable application behaviour.

Team Contribution

Name	Role	Key contributions
Chi Yui Steve Chak	Frontend , Product Development, Documentation	-Built all GUI pages with Blazor Hybrid -Integrated MudBlazor components and visual design -Designed UI flow and interactions -Authored proposal, documentation, and final report
Jang Jungho	Backend , Testing in windows platform, Data Layer	- Implemented Entity Framework Core models and services - Wrote business logic for budgeting, transactions, and goals - Created NUnit test cases and handled backup/restore - Managed database structure and validation routines
Both	Shared Planning, Debugging, Integration	- Jointly tested and integrated backend with frontend - Prepared demonstration and final polish

Two team members, Jang Jungho and Chi Yui Steve Chak, worked together to build the Kipi application. The group adopted a complimentary skill-based division of labor, with each member's abilities being taken into consideration when assigning tasks. This guaranteed effective job completion, few bottlenecks, and a burden that was balanced throughout the project's duration.

Chi Yui Steve Chak was principally in charge of the entire product design and frontend development. Using Blazor Hybrid and MudBlazor, he oversaw the development of the user interface, making sure that all of the main pages—Home, Finance, Wellbeing, Reports, Settings, and Login—were responsive, usable, and functional. Steve was also in charge of user interaction flow, navigation routing, and the incorporation of interactive user interface components such tabbed controls, charts, and progress indicators. Beyond only coding, he also wrote the project proposal, kept track of documentation, and organized the finished report to meet academic standards.

Furthermore, Steve was in charge of preparing the space for the demonstration. This involved creating sample data to mimic real-world usage and scripting scripts to install the.NET SDK on lab computers. His focus on design details made the software user-friendly and accessible for users with varying degrees of technical expertise.

The testing head and backend engineer was Jang Jungho. He used Entity Framework fundamental to construct the application's fundamental data architecture, putting important entities like Transaction, Budget, Goal, User, and Category into place. Additionally, he developed service classes that include business logic, like tracking objective progress, enforcing budget limitations, and computing net balance. To ensure modularity and testability, all backend logic was created using SOLID principles and clean architecture practices.

Jang was also in charge of utilizing NUnit to develop unit tests, making sure that essential features like goal computations, spending classification, and budget enforcement operated as intended.

Together, the two members worked on debugging, testing, and integration. Project alignment and code quality were maintained through paired reviews, shared Trello boards, and weekly check-ins.

The project's timely completion depended on this division of labor. It enabled each participant to be informed and actively involved in the overall project progress while concentrating intently on their primary areas of expertise. A feature-rich, reliable, and thoroughly documented home financial application that demonstrates excellent teamwork, technical proficiency, and practical problem-solving was the end result.

Conclusion

Beyond software requirements, the Family & Household Financial Tracker tackles practical issues that families encounter when working together to manage their finances. In addition to making precise budgeting and transaction tracking easier, the tool promotes open communication, accountability, and shared decision-making among all members of the household. It promotes financial literacy, establishes trust, and guarantees transparency in household money management by fusing cutting-edge technology with well-considered design.

This project goes beyond standard financial apps by offering a platform that is accessible to users of all ages and technical skill levels, allowing kids and teenagers to learn about financial responsibility in a fun and safe setting. Clear visual representations of spending, savings objectives, and budget progress help parents and partners avoid financial stress and misunderstandings.

Technically speaking, the application exhibits expertise in secure data management, responsive interface design, GUI development, and thorough testing. Performance and

usability are guaranteed in desktop and web environments using Blazor Hybrid, and local-first database architecture.

In the end, the final product is prepared to satisfy academic requirements and provide a noticeable, beneficial influence on regular family life. It serves as an example of how thoughtfully designed software can improve financial literacy, foster family cooperation, and promote social well-being in significant ways.

Acknowledgements and AI usage

Examples of artificial intelligence (AI) tools that were used sparingly during the project's inception and development include Copilot and ChatGPT from OpenAI. These tools were not used to generate core source code or to actually implement assignment logic. Instead, they supported the development process in the following ways:

- Creating demonstration data: To test UI elements, charts, and reports, AI was utilized to create sample datasets (such as budget records and transaction samples). Neither pre-built financial datasets nor actual user data were used.
- Creating installation and setup scripts for the.NET SDK: In order to ensure that development environments could be readily set up on lab or home computers, AI was contacted while creating and testing Windows-compatible installation instructions for the.NET 8 SDK.

The student team wrote all of the code for the main project implementation, including the database structure, business rules, component integration, and GUI logic, without using or duplicating any code produced by AI models. The fundamental reasoning of the assignment closely complies with UTS's standards for originality, academic integrity, and individual contribution.