University of South Wales
Prifysgol De Cymru

Bachelor Final Year Project Report Public ver
University of South Wales
Connecting and extend Real World FOSS finance and budget app: CashZ
------ WEI

Table of Contents

Background

There are a lot of finance and budgeting app, however a lot of those apps are made by companies that wants to earn profit and do extra analytics. For example, emma app[1] their mobile app not only contain basic crash reporting but also includes facebook(now called META) and google analytics.

While (FOSS)free and open source applications exist which do not have analytics. They often has less features and unmaintained after the main developer found new job or work on other projects. For example, budgetzero[2] is inactive since 2022 for 2 years without any update, dependencies were not updated and no improvements were made.

Database Migration

Database migration can be categorized into two types:

1. **Homogeneous Migration:** This involves migrating data between databases of the same type, such as SQL to SQL. It is often simpler, as the underlying database structures and query languages are consistent.

2. **Heterogeneous Migration:** This involves migrating data between different types of databases, such as SQL to NoSQL. This process is more complex due to differences in data models and query languages.

There are also many challenges in SQL Database Migration

Migrating SQL databases presents several challenges, including:

- Ensuring data integrity during the transfer process.
- Addressing compatibility issues between source and target schemas.
- Managing large volumes of data efficiently

Existing Research Contributions

Recent studies have explored techniques to optimize database migration. For example, Bhandari and Chitrakar (2024) proposed an enhanced transformation algorithm for migrating SQL databases to NoSQL graph databases, addressing issues such as data loss and performance degradation

Similarly, Saadouni et al. (2023) conducted a comparative study of migration approaches, highlighting the benefits and limitations of various techniques.

Insecure visualization problem

Budgeting involves planning and quantifying expected revenues and expenses for a future period, while actuals analysis focuses on analyzing actual spending against the budget. Spending analysis is also commonly used as a layman term for actuals analysis. In this report we use the term spending analysis more often as there are more similar terms used later from other references.

Budgeting/Spending analytics provides users with valuable insights into their spending habits which allows user make informed financial decisions.
There are few fundamental benefits:

1. Budgeting and Expense Tracking
 - Helps users create and stick to budgets by providing an overview of their spending patterns.
 - Tracks expenses across different categories, allowing users to identify areas where they can cut back.

2. Financial Awareness
 - Increases users' awareness of their spending habits, helping them avoid unnecessary expenses.
 - Provides a clear picture of where the money is going, making it easier to manage finances.

3. Goal Setting and Achievement
 - Assists users in setting financial goals, such as saving for a vacation or paying off debt.
 - Tracks progress towards these goals, motivating users to stay on track.

4. Fraud Detection using spending analysis
 - Identifies unusual spending patterns that may indicate fraudulent activity.
 - Alerts users about potential security issues allowing user take immediate action in that moment.

Above points shows that some spending analysis&budgeting are useful for user. But there are multiple ways to do budgeting, we could hire a professional accountant do it, we could let Large Language Model(LLM) do it, some would just do it themselves. The main difference here is the privacy and the concept of ***locality***. Be it paying remote online freelancer or hiring a friend to do it. It is not *local* even though your friend could be staying in the same apartment block but the data/information is not processed locally in users control but instead by others.

In this report the focus will be on local or online analysis as it is a software project, focusing on hiring a trusted worker or debating about doing analysis on a personally owned property is out of scope.
So, what are the difference and importance of doing the analysis locally with software?

1. Data Security
   - Performing analytics locally ensures that financial data is not shared with third-party services.
   - Reduces the risk of data breaches and unauthorized access.

2. User Control
   - Users have full control over their data, deciding what information to analyze and store.
   - Eliminates concerns about data misuse or exploitation by external party.

These points shows that there are huge security and privacy benefits for doing spending analysis locally.

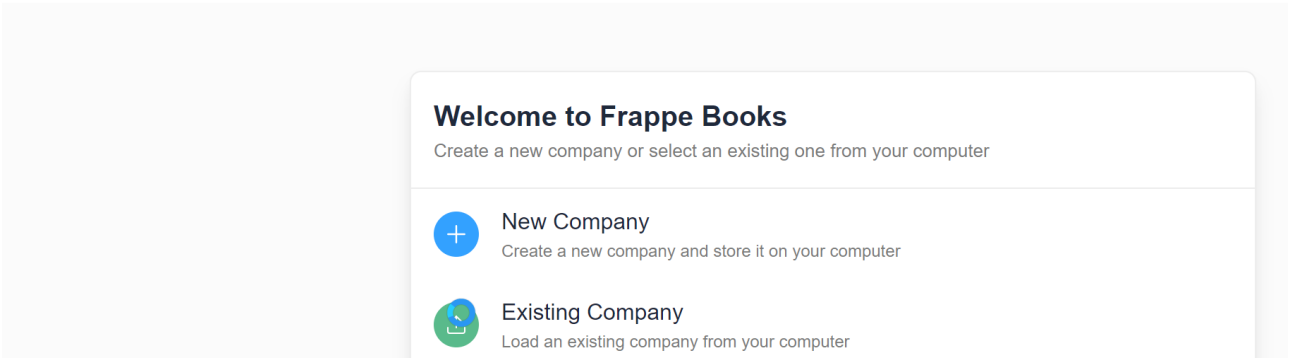
Rationale and Justification

Based on the background of the ecosystem, it is clear that there are a lot of foss finance and budgeting app ranging from average to high quality while enterprise software generally has high feature but low privacy guarantee. Thus, as a student with zero financial backing and appreciates privacy it is clear the most suitable choice for this project would be using FLOSS(free, libre and open source) software as a data source.

1st Problem

FOSS software however also has its own problem as most FOSS software has low integration and compatibility with each other unlike existing enterprise solutions. Which leads to incompatibility between software and in some case all of the transactions needs to be manually re-inserted into the new software user wanted to use. For example, both cashew[3] and Frappe Books[4] app uses a sql based database, however if we were to try import cashew database into Frappe Books it would cause the whole app to not respond as seen here.

Frappe Books (Not Responding)



Therefore, if we want to make frappe possible to import it we would need to import it with format that it understands, requires and expected.

This research and invention will allow a smoother transition between foss budget app. The 1st problem I noticed is in order to use FOSS budgeting apps that are made for different platform or sharing information between them we generally need to import from one to another.

2nd Problem discovered during development

This problem is discovered during development of this project. The lack of private spending analysis&budgeting in spending by the major banks.

The notification when user spent is mainly meant for security and compliance and most bank do not have any other feedback during spending, which is good for privacy and prevent bias in spending especially considering not all legal banking systems are neutral to economics.

Some banks do have statistics on spending/spending analysis.
For example, in HSBC it is called Spending Insights however as stated before they tend to have bad privacy implementations even though there are GDPR to disclose the risk, GDPR however does not force the banking vendors to minimize interaction with third party and allows companies to offload the work to other companies.

As written on the page to toggle spending insights ‘When spending insights are turned on, we'll share your transaction details securely with our partner company to help turn this data into useful spending insights.’(HSBC UK,2025)

This statement is literally informing users personal transactions data will be sent to third party other than HSBC to process. Which leads to some questions. Is this necessary? Who are the partnered company? When will the third party delete the processed data? How will they process&delete the data? What are the total risk?

Should we use Online Data Analytics

This is frankly not necessary. MATLAB (software) is 1st released at 1984, which proves that 41 years ago people had done statistics and visualization locally despite having far weaker hardware processing power. However, let’s give HSBC and other banking vendors the benefit of doubt and analyze it logically.

Pros and Cons of Local Data Analysis

Pros

1. Control and Customization:

- Users have complete control over the analysis environment and can customize the setup according to specific needs.

- The customization of the analysis ultimately depends on the app implementation from the bank. For example, a bad implementation would be a simple sum of transaction per month which is barely useful, pie chart of the type of transaction would be much more useful as it shows the major and minor transactions that could be eliminated. The main factor is dependent on banks app implementation which is not controllable by user making this a weak argument.

2. Reduced Latency:

- Data processing is faster as it does not rely on internet connectivity.
- Example: Real-time analysis of high-frequency transaction data.

- It not only reduce the latency. It also remove from being dependent on internet which in itself is a benefit. The reduce in latency is generally insignificant to most users as it is fundamentally not mission critical. It however is extremely important for business owner/operators such as investor or professional stock traders or entrepreneur. These scenario shows that although not a wide population will use it, it does have its use case. Similar to how HSBC app has investment section in app which concludes this as a strong argument

3. Data Security:

- Physical control over data storage reduces the risk of external breaches.
- This is obviously a strong argument. For example, “34,000,000 customer and employee emails and personally identifiable information belonging to Cylance customers, partners, and employees” data are leaked by hackers.(Sergiu G., 2024) In this case, it is also hacked through a third-party platform which proves the fundamental security problem off source the processing to third party especially financial data is extremely problematic and should be taken seriously.

Cons

1. Maintenance and Updates:

- Requires manual maintenance and updates to hardware and software dependent on user.

-It realistically however doesn’t matter as most mobile banking app especially in UK has minimum hardware requirements and it needs decent phone to run the app smoothly. The software update is also not a problem as most banking app forces the user to update to the latest version. These shows that this is a weak argument.

Pros and Cons of Online Data Analysis

Pros

1. Scalability:
   - Easily scalable with cloud services, allowing for the analysis of large datasets without significant upfront investment in hardware.
   - Example: Using cloud platforms like AWS or Google Cloud for big data analytics.

-Note: Most users do not have more than 1000 transaction within a month. A personal budget dataset is extremely small just a few Mb size of text file. Scalability in personal spending insights almost have no effect, thus making it a weak argument.

2. Access to Advanced Tools:
 - Provides access to a wide range of advanced analytics tools and machine learning frameworks.
 - Example: Utilizing Google BigQuery for large-scale data analysis.

- This is a weak argument. It is theoretically possible, however as of the time of writing this report, no banking system provides spending insights or analytics tailored to each individual user through the use of machine learning. There are no complex machine learning nor simple predicted budgeting with gradient descent machine learning deployed to give a personalized experience for every user. It is very obvious that huge computing power will be needed which indirectly incurs huge expenses for the banking vendor and thus is a conflict of their interest.

3. Automatic Updates and Maintenance:
 - Cloud service providers handle updates and maintenance, reducing the burden on users.

- This is also a weak argument. It is a theoretical possible benefit but extremely not practical. Let’s assume that the normal banking app update is once every 3 month. The point where it would benefit from having a cloud analytics is when the update for analytics logic occurs at least once every 2 months. On surface it sounds logical that a statistic component needs to update or improve once every 2 months, however in real life this scenario rarely happens for example there are rarely any changes to the functions of a stock market viewing website, not only there are fundamentally lack of development and changes on the statistics logic, it is also fundamentally required to not change often for ease of use which goes opposite of this argument.

For example, if there are 1000 elderly people using the app spending statistics to help them plan and spend their money during retirement. A UI update once every 1 or 2 month would cause a lot of them to panic, especially those that are less tech savvy. Let’s assume 10% of them are not good with electronics with a sample size of 1000 it would cause 100 elderly people try to seek help for understanding this new change every 1 or 2 month. If we then assume the banking vendor does update the statistics logic once every 2 month, it would be 6 update per year.

The amount of panics occurred could be calculated as: 6*100=600 panics.

In a real life scenario of assuming 6 updates per year and 10% of UK elderly people are not tech savvy. Based on (Office of National Statistics, 2023) there are 11million elderly people

It would be: 11 million *0.1*6=6.6million panics expected to occur per year due to the change.

This simple calculation easily prove that such theoretical rapid update extremely difficulty for most user to adapt, even if the United Kingdom banking vendor were to theoretically heavily invest on improving spending analysis for users which didn't happen at recent years.

Cons

1. Latency and Reliability:
   - Dependent on internet connectivity, which can introduce latency and reliability issues.
   - Example: Network outages can disrupt data analysis processes.
      - It is the opposite of local analytics, which makes it indirectly a strong argument.

2. Cost:
      - Can become expensive with high usage, especially for large datasets and extensive processing.
      - It is mainly on the banking vendor perspective that has this problem. It is a strong argument on the banking vendor side.

3. Data Transfer Bottlenecks:
   - Transferring large datasets to and from the cloud can be time-consuming and costly.
   - Example: Uploading terabytes of data to cloud storage may incur significant bandwidth costs.
      - This is also a banking vendor side problem that they need to plan and resolve, however it does not need to have that much bandwidth if the implementation does not save any result in the server side. A weak argument.

Sort&Mind map information of both method

With above analysis we can make a simple diagram to show the amount of benefits and harm it would bring in both case to reveal which is better.

From the diagram above, we can see that there are no strong arguments for online analytics. However it may still look confusing and not obvious to some people so the diagrams needs to be processed more.

If we remove the duplicated similar content across implementations we will get:

From the analysis it is extremely obvious that there are no significant benefits for users to do their spending analysis online, it even incurs cost for the banking vendor. So why do they still do it until this day in 2025?
This is a good question for another paper as it is not the main focus of this paper to further debate and analyze.

A logical conclusion based on analysis

A repeat on the questions to be answered for conclusion.

1)Is this necessary?

2)Who are the partnered company?

3)When will the third party delete the processed data?

4)How will they process&delete the data?

5)What are the total risk of processing analysis online?

-1st No, as there are no significant benefits for users to do their spending analysis online.
-2nd No idea, it is also not stated in the privacy policy leaving user on their own.
-3rd No idea, users are only informed about HSBC data retention duration here is a quote from HSBC mobile app privacy policy “we’ll normally save your main banking information for a period of 7 years from the time our relationship ends with you.” (HSBC, no date)[11]. This leads to a valid possibility of being kept in their database forever if users didn’t close the bank account after passing away.
-4th How do the third party process or remove the data is also not mentioned at all. Due to the lack of information about that third party *partnered company* it is impossible to figure out how is the personal information processed by third party.
-5th High potential in leak of financial data, high latency when generating statistics and low transparency in the statistics.

Summary of the background and rationale:

Why this project?

1) Although current app market have similar or better functional budget app, most are not implemented with maximum privacy for users in mind including current mobile banking app provided by bank vendors which is a long term high risk privacy threat. EG: online ads, analytics service, online budget analysis

2) Cross platform conversion problem and low integration compatibility of Free and Open Source(FOSS) apps.

3) Me(the paper author) being poor and having problems in finding out which product to cut and trend of my purchases.

4) Learn and implement essential skills for android development, sql to nosql both way conversion, Create, Read, Update, Delete (CRUD) of structured data, data analytics, data visualization, encrypted database data at rest, direct paypal donation, dependency injection, implement Model-View ViewModel with jetpack compose(MVVM) .

5) Potential increased usage as wealth inequality and potential global economic recession increase over time.

6) Offering an alternative open source statistics budget app to prevent users taking the less privacy friendly way.

These two problems are what this research attempts to solve.

Who?

1) There are multiple indirect contributors in this project.

Friends, lecturers, parents, and international community.

2) Beneficiaries of this project.

Android users, users of actual app in windows, new users that are finding budget app.

What quality?

- Parsing of at least 2 foss budget app data is completed.
- The selected 2 foss budget app can export and import passing through the conversion application with minimal loss of data for record transfers between them.
- Have good financial analytics visualization or statistics.
- Have High security implemented.
- Extreme level of privacy (be in top percentile of app market, **0 online analytics**).

Objectives

-1st, Make android app that can parse and understand the database of 2 high quality foss budget app into a standardized format/data.
-2nd, convert them to the standard format in order to allow easy transfer of personal financial and budgeting records.
-3rd, export and make sure the transfer of data is lossless or at least try to minimize loss.
-4th, Put as much effort as possible to make conversion process smoothly with good UI design.
-5th, Implement good user feedback in case of error or conflicts of data.
-6th
Initial Objective: Learn and utilize the advantages of vue framework that does not exist in pure js and html implementation.
Experimentation and Adaptation: During the early stages of experimentation, Vue framework was selected for its modularity and ease of integration. However, as development progressed into the second month, it became apparent that an alternative framework and target OS would better suit the requirements of the project due to user mobility and software accessibility after re-evaluation of target platform by lecturer. For more details on this decision, refer to the **Development Changes section**. As a result, the focus shifted toward android development framework to improve the final implementation usability.
Outcome: This iterative approach enabled a more effective exploration of tools and methodologies, ensuring that the final implementation aligns with the project objectives while maintaining flexibility in adapting to new insights.
-7th, do analysis with the standardized format/data.

Overall Aim

In summary, the objective is to invent a proof-of-concept database converter app that can support transfer between selected foss budget app while learning and making sure it is user friendly. For example, it will allow software B import records that are exported from software A lossless without data corruption and software crashing. The application also should have ability to import records from other apps and analyze the budget or spendings.

Workload justification

- difficulty

although difficulty in the database conversion has been reduced from the initial proposal.

It is still sufficiently difficult. No database lecturer are able to offer guidance or responded to my emails proving its complexity, **at least 50** unique technical errors are met during development which proves the complexity and difficulty.

Besides that, additional functions are introduced including these 5 most significant ones:
1) Encrypted Database at Rest: Protects personal data by preventing data leaks in the event of a security breach.
2) Local Budget Analysis: Enables data analysis without relying on online services, ensuring privacy and minimizing external dependencies.
3) Direct Donation to developer: Provides a mechanism for users to directly support the developer, promoting project sustainability.
4) Application Security Lock: Enhances security by preventing unauthorized access or misuse of personal data.
5) Heterogeneous Database Migration with NoSQL Data Format: To achieve faster local data analytics, further increasing the value and enhancing the impact of the local budget analysis feature (Function 2).

These are implemented to make it a complete application that is similar to a full fledged application in app store with long term usage in mind.

- uniqueness

The absence of online tutorials on migrating financial databases necessitated the project's implementation without a similar known reference, resulting in its initial direction as a proof of concept. However, after discussions with peers, the project's aim was shifted towards developing an application adhering to industry standards for long-term humanitarian benefits . This transition was guided by a personal vision of creating a tool that not only serves immediate research objectives but also contributes meaningfully to the global community. By aligning the application's design and functionality with established standards, the project ensures compatibility with existing systems, facilitates collaborative improvement, and fosters widespread adoption.
This focus on industry standards positions the application as a sustainable and adaptable solution, capable of supporting diverse use cases.

Ultimately, the shift reflects a commitment to advancing technology in ways that empower individuals, enhance efficiency, and promote innovation for the collective benefit of humanity.

Overall Methodology

- Analyze the format of each foss budget app and their internal data structure.
- Process data which Dr Emlyn Everitt(appendix 1) suggested to use regex but later decided to use existing sql parsers during development for example [6]
- Finding a way to store or resolve the additional data from various budget app that are more advanced or has additional features that are not related to budgeting.
- Implement standardized format and structure in codebase.
- Follow the V model development workflow until it is stable and tested. Then proceed to add new feature.
- Start to test import and export from 1 app to another,following the V model workflow until it is completed.
- After finishing all objective,do final compatibility check then fix accordingly if there are any edge case.
- Finalize packaging,optimize and setup basic demo on university web server on campus due to network limitation.

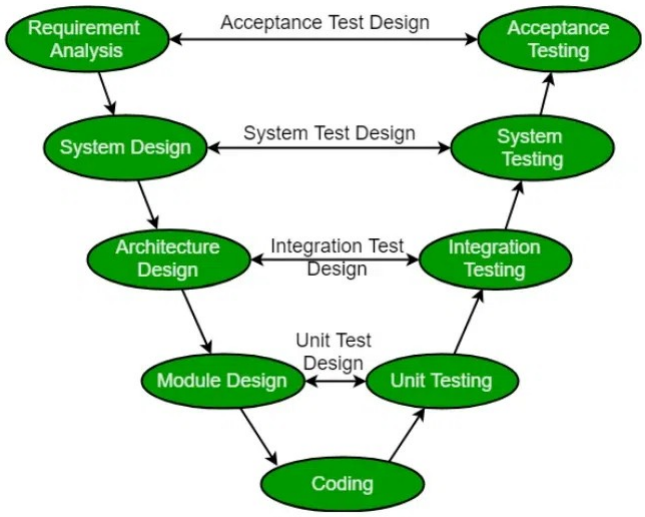
Methodologies

1) Agile-modular V Software Development Lifecycle(SDLC)

Planned SDLC Approach: V-Model

Initially, the project was structured around the V-Model Software development lifecycle (SDLC), which follows a sequential development process emphasizing early-stage requirement validation and rigorous testing phases.
Under this model, each feature was expected to take approximately two weeks to implement, ensuring strict validation at each stage before progressing to the next. The approach is well-suited for projects where requirements are well-defined upfront, minimizing errors at later stages through a structured testing methodology.

Below is a standard V model SDLC diagram[5].



Challenges Leading to SDLC Modification

Delay in Commencement of Effective Development

The initial phase of development utilized HTML and vue.js to prototype the application's functionality and user interface. However, upon reassessment, it was determined that transitioning to an Android framework would better align with the project's objectives, particularly in terms of privacy and accessibility for the intended user base. For more details on this decision, refer to the **Development Changes section**.

To further accelerate development, a rapid prototyping approach called “Agile-Modular V SDLC” was adopted during the initial transition phase. New features were implemented with minimal testing overhead to allow for quick validation and iterative improvements. While this temporarily reduced the structured verification processes, it significantly enhanced development speed and provided early-stage user feedback.

The modification to the development lifecycle enabled rapid prototyping during the initial stages, with minimal testing implementation. Once features reached stability at a module level, the V-Model-based SDLC was reactivated for rigorous validation and verification processes.

Hybrid Development Methodology: Agile-Modular V SDLC

Transitioning Towards Agile SDLC

As project timelines tightened, development practices evolved to embrace Agile SDLC, emphasizing incremental progress, iterative testing, and continuous user feedback. This shift fostered flexibility in feature adjustments, enabling dynamic refinement of security protocols and modular components to meet evolving requirements.

Hybrid SDLC Approach: Integrating Agile with V-Model

Rather than abandoning the V-Model entirely, the revised methodology selectively applies its structured validation at stability checkpoints within modular development. This hybrid strategy preserves the rigorous testing benefits of V-Model while incorporating Agile’s adaptability in earlier time constrained phases, ensuring responsiveness without compromising long-term system integrity.

Full database compatibility test

(O for pass, X for fail)

Import=import from target to CashZ

Export=export from CashZ from target

SQL Table	NoSQL	Import	Export	statistics accuracy
cashew	CashZ	O	O	O
actual	CashZ	O	X	O

2)Software that is supported

The following software tools are utilized to support the implementation and analysis framework:

- **Cashew**: A feature-rich budgeting application built with Flutter, offering tools for budget management, transaction tracking, and financial goal setting.
https://github.com/jameskokoska/Cashew
- **Actual**: A local-first personal finance tool designed for envelope budgeting, providing synchronization across devices and self-hosting options.
https://github.com/actualbudget/actual

3)Standardized Generated artificial data

Defining Role of Standardized Artificial Data in Modern Analytics

The use of standardized artificial data, or synthetic data, has emerged as a transformative approach in data-driven research and applications.

Applications and Supporting Research

The utility of synthetic data spans various domains:

- Healthcare: Synthetic patient records have been used to develop predictive models for disease diagnosis and treatment, ensuring compliance with privacy regulations
- Finance: Synthetic transaction data aids in fraud detection and risk assessment without exposing sensitive customer information
- Machine Learning: Generative models like GANs (Generative Adversarial Networks) and VAEs (Variational Autoencoders) are widely used to create synthetic datasets for training and testing algorithms

Challenges and Future Directions

While synthetic data offers numerous advantages, challenges such as computational requirements, training stability, and ensuring data fidelity remain. Addressing these issues will be crucial for broader adoption and standardization

Current Implementation

To ensure the validity and applicability of the proposed framework, the minimal test dataset is structured as follows:

The minimal test data should consist of:

- **Base Budget:** A fixed GBP per month as demonstration in the foundational allocation of resources across spending categories.
- **Buying, Cost, and Usage Data:** Detailed records reflecting spending habits and cost distributions.

- **Time Span:** Data samples spanning across two months to capture short-term trends and variations.

Simplified Simulation

The following example is used to demonstrate the exact implementation of the framework:

- **Budget:** £100 allocated for a single month.
- **Weekly Cost:** £20 spent per week.
- **default app settings**: minimal change.

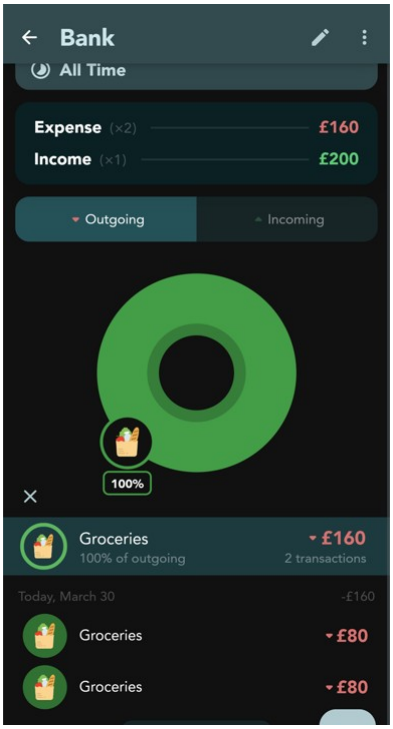
This simulation illustrates how budget and cost data are processed within the framework, offering a clear and concise representation of the methodology.

Illustrative Example with Visual Reference

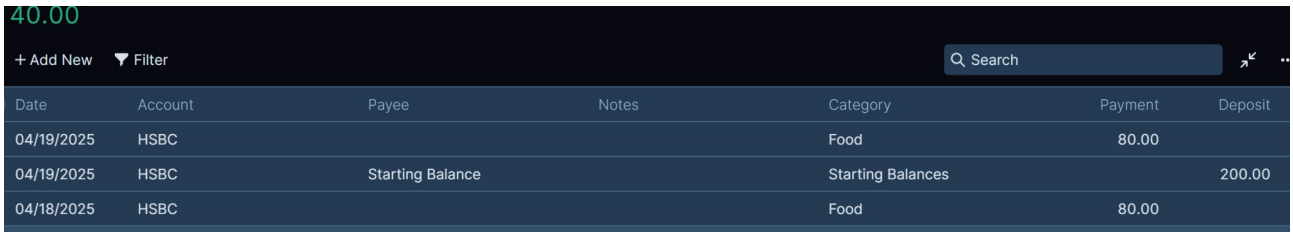
In a scaled test it is 200GBP income, -80 expense twice as a 2 month simulation.

Each expense should be shown clearly.

For example in cashew when manually entry the data is the statistics page shows:



While in Actual app it looks like:



4)Selective incremental format support

Prioritizing the order in which database tables are supported is essential to ensuring smooth development with minimal conflicts.

For example, in the initial phase of cashew export processing, the only required table is the Transactions table, with additional tables integrated at later stages as needed.

Subsequent testing revealed that creation of additional tables were necessary to establish a valid database structure for the cashew application. While the initial setup relied solely on the Transactions table, further validation demonstrated that supplementary tables needed to be created to support comprehensive data integrity and functionality.

5)NOSQL as the internal database for analytics

During research, I happened to stumble upon M. A. Rifqi, A. Y. Husodo and H. Wijayanto, (2024) works which highlights the speed gains of NOSQL for simple queries with quote “This difference in response time is influenced by the complexity of the features developed in the application” M. A. Rifqi, A. Y. Husodo and H. Wijayanto, (2024) in their paper.

Given that this project involves minimal data modification during analysis, it is likely to experience similar performance benefits to those observed in their experiments.

6)Encrypted transaction data in rest & secure access

Based on my current knowledge, there are multiple ways to implement this effectively.

- Manually encrypt then save into shared preference.

- Save into encrypted preference.

- Auto encrypt&save as file.

However we need pick 1 from these choices instead of implementing all 3. So let’s review each option carefully.

Option 1: Manually Encrypt Then Save into Shared Preferences

Pros

- Provides complete control over encryption algorithms and key management.
- Allows developers to customize security measures based on specific application needs.

Cons

- Requires additional development effort to implement and maintain encryption manually.
- Increased risk of human error in handling encryption keys, potentially leading to security vulnerabilities.

- Complexity in ensuring proper encryption and decryption processes across different parts of the application.

Option 2: Save into Encrypted Preferences

Pros

- Built-in encryption eliminates the need for manual implementation, reducing development complexity.
- Seamless integration with existing preference storage mechanisms, improving efficiency.
- Optimized for storing key-value data securely with minimal overhead.
- Reduces maintenance burden by relying on established security frameworks.

Cons

- Limited flexibility in encryption customization compared to manual encryption.
- Dependence on the security strength of the underlying encrypted preference system.

Option 3: Auto Encrypt & Save as File

Pros

- Suitable for larger data storage where structured data needs encryption.
- Automates encryption processes, ensuring consistent security practices.

Cons

- File-based encryption introduces additional storage complexity compared to encrypted preferences.
- Not ideal for small data items like authentication tokens or simple settings.
- Requires careful management of file access permissions and storage locations.

Why Option 2 is the Best Choice

Option 2 provides a balance between security, efficiency, and ease of implementation, making it the most suitable choice for development. Unlike manual encryption, it reduces development effort while maintaining strong encryption standards. Compared to file-based storage, encrypted preferences are lighter, more manageable, and optimized for key-value data, which aligns with this application's needs. By leveraging built-in security mechanisms, it promotes focus on application functionality rather than encryption complexities, ensuring a more streamlined and maintainable approach.

7) Holistic Analysis of Budget and Spending Patterns

To empower users in understanding and optimizing their financial habits, this framework combines **multi-chart visualizations** with **AI-assisted extrapolation techniques**, seamlessly integrating static representation, dynamic trends, and predictive insights.

9) Sustainable donation to developer

Directly donate to developer using paypal instead of using ko-fi or patreon which have a cut in the payment.

10) Centralized state management for app security

Summary: Instead of having multiple viewModel/ models in the app, most security view model are centralized around 2 script with modular design.

Comparing Android Compose ViewModel vs Traditional Activity-Based Design Flow in Modular Security Implementation

Traditional Activity-Based Design Flow

In legacy Android development, UI components such as Activities and Fragments were tightly coupled with business logic, often relying on distinct ViewModel instances for each screen. This approach led to fragmented security implementations, where encryption, authentication, and access control were distributed across multiple ViewModels. While this design allowed per-screen customization, it introduced the following challenges:

- **Redundant ViewModel Instances**: Each Activity or Fragment maintained its own ViewModel, causing excessive code duplication and inconsistencies in security logic.
- **Complex Lifecycle Management**: UI state had to be manually managed across lifecycle transitions, increasing the risk of data loss and security vulnerabilities.
- **Inconsistent Security Implementation**: Developers needed to replicate security mechanisms across multiple ViewModels, which could result in inconsistencies and oversight.

This decentralized approach complicated security maintenance, particularly in applications handling sensitive user data.

Android Compose ViewModel Approach

Jetpack Compose fundamentally shifts UI management to a **declarative state-driven** model. ViewModels are no longer tied directly to Activities or Fragments but instead using a singleton code structure can provide centralized state management that is independent of UI lifecycles. This enables:

- **Decoupled ViewModels**: Security logic can be maintained independently from UI components, ensuring consistent authentication and encryption mechanisms.

- **Reactive UI Updates**: ViewModels persist across recompositions, eliminating the need for manual state retention, simplifying management of sensitive security states.
- **Scalability and Maintainability**: Modular security scripts allow for reusable security functions across different screens without requiring multiple ViewModels.

This declarative model not only improves UI flexibility but also enhances the security structure by providing a centralized security control mechanism.

Decision to use Centralized Security ViewModel

Instead of distributing security logic across multiple ViewModel instances, consolidating it into **two modular scripts** presents several advantages:

- **Uniform Security Policies**: Centralized encryption and authentication reduce inconsistencies across different application components.
- **Minimized Redundancy**: Security logic is maintained in a singular location, avoiding duplicated code across various ViewModels.
- **Simplified Debugging**: Isolating security mechanisms within specific modules improves traceability and simplifies compliance audits.
- **Optimized Memory Usage**: Reducing the number of ViewModels dedicated to security results in efficient memory allocation and streamlined performance.
- **Scalable Security Architecture**: A modular approach ensures that security logic can be expanded without restructuring the entire application.

By applying a modular security implementation within Compose's ViewModel architecture, the application benefits from improved performance, maintainability, and security integrity.

Significant development changes

1) Transition from Regex-Based SQL Parsing to Library-Based Parsing

During the initial stages of development, SQL data parsing was implemented using a regex-based approach. This method aimed to identify and extract relevant patterns from SQL queries for further processing. While effective in simple cases, this approach presented limitations in handling complex SQL structures, especially those involving nested queries and advanced syntax variations.

To address these challenges, the development transitioned to utilizing an existing SQL parsing library. This change was motivated by the need for increased accuracy, efficiency, and compatibility with diverse SQL syntax. By leveraging a specialized library, the parsing process directly and correctly interprets SQL queries without relying on pattern recognition, ensuring a more robust and scalable solution.

The transition was further supported by the use of a SQL browser to navigate database layouts. This allowed for a deeper understanding of table relationships, query structures, and data dependencies, aiding the integration of the new parsing library and optimizing its functionality within the project.

2) Development Progression: From Python Prototype to Android Framework

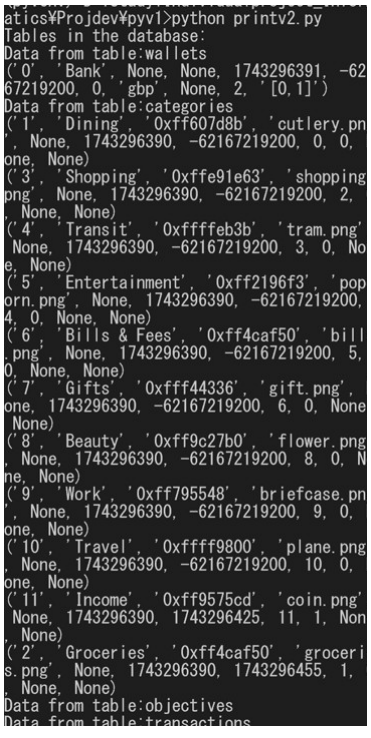
It started with python as sql prototype, node& vue.js as the main version 1, then transitioned to android development after re-evaluation of objective.

Prototype Phase: Python SQL and Node & Vue.js

At this phase, the main focus is to figure out how to inspect and analyze SQL database to implement the conversion.

Python was utilized to prototype SQL parsing and processing mechanisms.

The image shows a detailed dump of the database data.



Node.js and Vue.js were then used to create a web interface for visualizing database structures and make a stable user friendly conversion workflows.

This exploratory phase provided critical insights into the complexities of database analysis and served as the foundation for developing a more robust and scalable solution in later stages.

The transition to Android Framework

In the first month, I am still not quite certain that android or web app would be a better fit for the final application. After some discussion and encouragement by supervisor.

I undertook the task of evaluating and selecting the appropriate framework for the project.

Framework options:

- python/other cli tools

- web app with vue.js

- android app

Decision metrics

There are also a few metric that are crucial and used to decide.

- security

- privacy

- Offline accessibility

- Cross-platform accessibility

To understand why they are important we need to understand it’s benefits.

Security Benefits

1. **Protection from Data Breaches:** Ensures sensitive information is secure against unauthorized access.
2. **User Trust:** Promotes confidence in the application’s ability to safeguard user data.
3. **Safe Transactions:** Enables secure handling of financial or personal transactions within the app.

Privacy Benefits

1. **Data Control:** Allows users to manage who has access to their information.
2. **Confidentiality:** Ensures private communications and sensitive data remain secure.
3. **Identity Protection:** Shields user identities from exposure or misuse.

Offline usage first Benefits(Accessibility)

1. **Offline Access:** The application remains fully functional without an internet connection, ensuring users can rely on it even in areas with limited or no connectivity.
2. **Inclusivity:** It increases accessibility for users in rural or underprivileged areas where Wi-Fi or mobile data may not be consistently available.
3. **Cost Efficiency:** Offline functionality eliminates the need for users to consume mobile data, reducing their expenses.
4. **Reliability in Critical Scenarios:** Users can access essential tools or data during emergencies, travel, or in environments where internet access is restricted.

Cross-Platform Accessibility Benefits

1. **Device Independence:** Empowers users to access the application across multiple devices, improving convenience.
2. **Enhanced Reach:** Expands the application's user base by accommodating diverse platforms and ecosystems.
3. **Consistency:** Maintains uniform functionality and experience, regardless of the device or operating system.

Comparison Based on Key Metrics

With the benefits of each metrics in mind we can then compare each option accordingly.

CLI/Desktop Apps

Strengths:

1. **Performance:** Generally faster than Android/web apps since they run directly on the user's machine and can leverage hardware resources effectively.
2. **Robust Features:** Can offer complex functionalities that are resource-intensive or dependent on local processing power, such as high-speed data processing or intricate algorithms.
3. **Offline Capability:** Typically, desktop apps and CLI tools don't require internet access for core functionalities.
4. **Power Users:** CLI tools appeal to developers and advanced users for tasks that require precision, automation, and scripting capabilities.

Limitations (Compared to Android/Web Apps):

1. **Accessibility:** CLI/desktop apps are platform-specific and require installation, making them less accessible across different devices.
2. **User Experience:** The lack of a graphical interface in CLI tools can be intimidating for non-technical users.

3. **Portability:** Desktop apps don’t allow seamless usage on mobile devices, unlike Android/web apps.

Web Apps Strengths:

1. **Cross-Platform Accessibility:** Web apps work across multiple platforms (PC, Mac, Android, iOS) with just a browser—far better reach than CLI/desktop or Android apps.
2. **Instant Access:** No installation required; users can access the app from any device with internet connectivity.
3. **Ease of Updates:** Updates are implemented server-side, meaning users always interact with the latest version without needing manual updates.

Benefits of an Android App vs. a Web App (Beyond Offline Use)

1. **Native Performance:** Android apps are optimized for the device, leveraging native APIs and system resources to provide faster and smoother performance compared to web apps.
2. **Enhanced User Experience:** Android apps can utilize platform-specific design guidelines (e.g., Material Design) to offer a more intuitive and engaging user experience.
3. **Device Integration:** Android apps can integrate seamlessly with device features such as GPS, camera, notifications, contacts, and storage, enhancing functionality.
4. **Persistent Notifications:** Unlike web apps, Android apps can send push notifications, keeping users informed and engaged without requiring them to open the app or browser.
5. **Customizations and Personalization:** Android apps can store user preferences locally and provide a personalized experience tailored to individual needs.
6. **Performance in Diverse Environments:** Android apps often have better handling for limited hardware or older devices compared to resource-heavy web apps reliant on browsers.
7. **Security:** Data handled within an Android app can be better secured using platform-specific encryption techniques and permissions, compared to the browser's sandboxed environment.
8. **App Store Accessibility:** Being available on app stores like Google Play increases the app’s visibility, credibility, and ease of access for a broad audience.

3) Incorporating Online AI and Manual Code Adjustments in Development

Throughout the development process, online AI tools were employed to assist in generating code snippets, resolving complex programming challenges, and streamlining repetitive tasks. These tools provided valuable recommendations, optimizations, and solutions to enhance the efficiency of the development workflow. For instance, AI-based suggestions were leveraged to identify potential improvements in algorithm design and generate templates for specific functionalities.

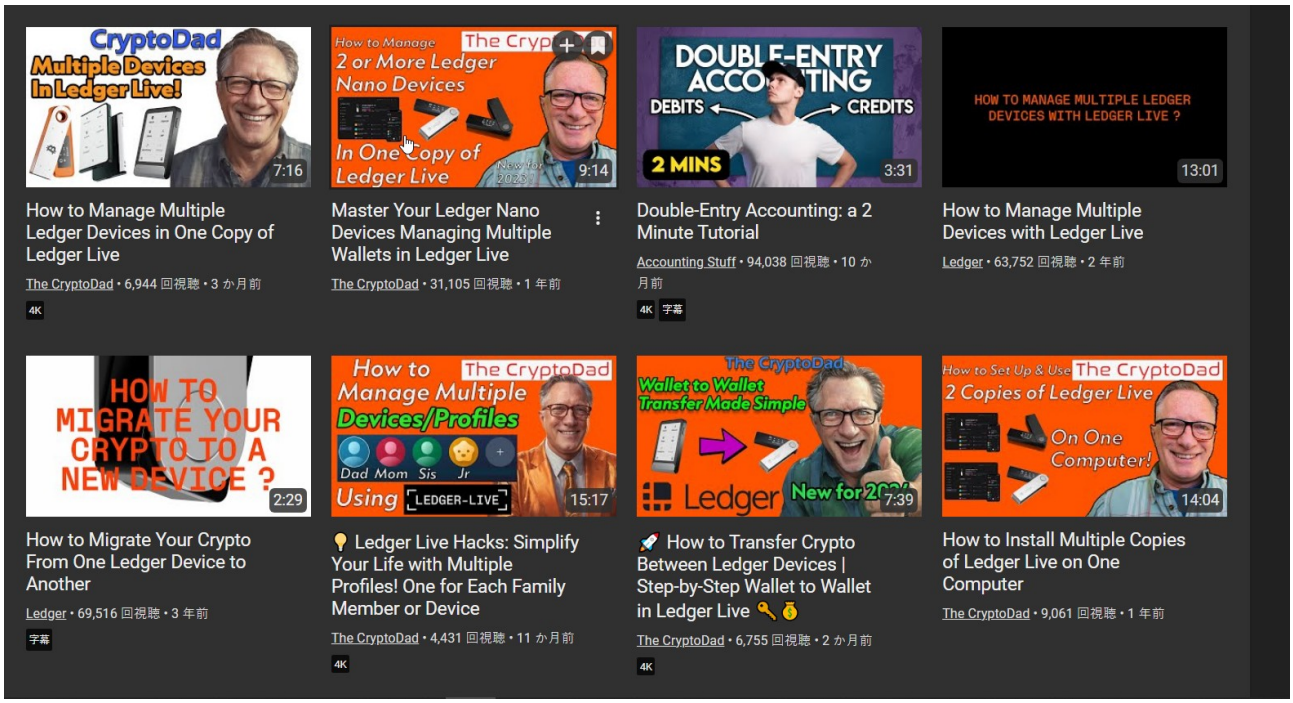
However, manual code modification remained integral to the process, ensuring that the generated outputs aligned with project requirements and adhered to predefined standards. This included refining AI-generated code to incorporate specific logic, address edge cases, and integrate seamlessly with the existing framework. As requirements evolved, manual interventions were critical in adapting the codebase to meet unique project demands and ensuring functionality across diverse scenarios.

The combined use of online AI and manual modifications exemplifies a hybrid approach to software development, balancing automation with human oversight for optimal outcomes.

4) Omission of Double Ledger to Single Ledger Conversion Feature

After some research into the conversion of database from Frappe books to Cashew. It is discovered that if a lossless transfer to Cashew app were to be implemented if would need about 5-9 weeks. Especially with the lack of understanding on double ledger structure and its flow in the software implementation. Furthermore, there are a lack of information or help about double ledger to single ledger conversion for example there are no existing videos to learn and develop in sql.

Screenshot of searching “double ledger to single ledger” in youtube.



After consultations with professionals and lecturers, it became evident through their advice and discussions that the complexity of implementing a double ledger to single ledger conversion is extremely high. Without a comprehensive analysis of database properties and cash flow dynamics, it is impractical to proceed with such implementation.

Appedix
[1] https://uk.linkedin.com/in/dr-emlyn-everitt-a4109825
[2] https://github.com/buyvtuygJW/CashZ

References

[1]Nmma(no date) Emma. Available at: https://emma-app.com/ (Accessed: 1Nov2024)
[2]Budgetzero(no date). Available at: https://github.com/budgetzero/budgetzero (Accessed: 1Nov2024)
[3]cashew(no date). Available at: https://github.com/jameskokoska/Cashew (Accessed: 1Nov2024)
[4]frappe books(no date). Available at: https://github.com/frappe/books/releases/tag/v0.24.1 (Accessed: 1Nov2024)
[5]geeks for geeks (06 Mar, 2024)SDLC V-Model – Software Engineering. Available at: https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/ (Accessed: 1Nov2024)
[6]https://hackernoon.com/14-open-source-sql-parsers
[7]M. A. Rifqi, A. Y. Husodo and H. Wijayanto, (2024) "*Development of SQL and NoSQL Database Integration Using Query Direct Access and Change Data Capture Approaches in an Android Application,*" *2024 IEEE International Conference on Communication, Networks and Satellite (COMNETSAT)*, Mataram, Indonesia, 2024, pp. 671-679. Available at: doi: 10.1109/COMNETSAT63286.2024.10862087. (Accessed: 13 Feb 2025)
[8]HSBC (2025) HSBC UK (Version3.53.9)
[Computer programme]. Available at: Google Play store (Accessed: 23 Feb 2023)
[9]Sergiu G. (2024) *Cylance confirms data breach linked to 'third-party' platform*. Available at: https://www.bleepingcomputer.com/news/security/cylance-confirms-data-breach-linked-to-third-party-platform/ (Accessed: 24 February 2025).
[10]Office of National Statistics (2023) *Profile of the older population living in England and Wales in 2021 and changes since 2011*. Available at: https://www.ons.gov.uk/peoplepopulationandcommunity/birthsdeathsandmarriages/ageing/articles/profileoftheolderpopulationlivinginenglandandwalesin2021andchangessince2011/2023-04-03 (Accessed: 24 February 2025).
[11]HSBC (no date) *App Privacy Notice*. Available at: https://www.hsbc.co.uk/mobile-privacy-notice/ (Accessed: 24 February 2025).
[12]UK Data Service (2024) Evaluating the benefits, costs and utility of synthetic data. Available at: https://ukdataservice.ac.uk/about/research-and-development/past-projects/evaluating-the-benefits-costs-and-utility-of-synthetic-data/ (Accessed: 19 April 2025).
[13]Goyal, M. (2024) 'A Systematic Review of Synthetic Data Generation Techniques Using Generative AI ', Recent Advances in Technologies and Optimization for Intelligent Data Processing, 13(17), pp. 3509. doi: https://doi.org/10.3390/electronics13173509 .
[14]Saadouni, C., El Bouchti, K., Reda, O.M., Ziti, S. (2023). Data Migration from Relational to NoSQL Database: Review and Comparative Study. In: Kacprzyk, J., Ezziyyani, M., Balas, V.E. (eds) International Conference on Advanced Intelligent Systems for Sustainable Development. AI2SD 2022. Lecture Notes in Networks and Systems, vol 637. Springer, Cham. https://doi.org/10.1007/978-3-031-26384-2_22
[15]Bhandari, H.L. and Chitrakar, R. (2024) ‘Enhancement of a Transformation Algorithm to Migrate SQL Database into NoSQL Graph Database’, Data Science Journal, 23(1), p. 35. Available at: https://doi.org/10.5334/dsj-2024-035.