ASSIGNMENT GROUP WORK

Qualification	**Pearson BTEC Level 5 Higher National Diploma in Computing**		
Unit number and title	**Unit 22: Application Development**		
Submission date		**Date Received 1st submission**	
Re-submission Date		**Date Received 2nd submission**	
Group number:	**Student names & codes**	**Final scores**	**Signatures**
	VU MINH HIEU – BH01068		
	NGUYEN NHU PHUC – BH01072		
	BUI VIET HOANG – BH01048		
Class	SE06302	**Assessor name**	BUI NGOC LINH

Plagiarism

Plagiarism is a particular form of cheating. Plagiarism must be avoided at all costs and students who break the rules, however innocently, may be penalised. It is your responsibility to ensure that you understand correct referencing practices. As a university level student, you are expected to use appropriate references throughout and keep carefully detailed notes of all your sources of materials for material you have used in your work, including any material downloaded from the Internet. Please consult the relevant unit lecturer or your course tutor if you need any further advice.

Student Declaration

I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I declare that the work submitted for assessment has been carried out without assistance other than that which is acceptable according to the rules of the specification. I certify I have clearly referenced any sources and any artificial intelligence (AI) tools used in the work. I understand that making a false declaration is a form of malpractice.

P1	P2	P3	P4	P5	P6	M1	M2	M3	M4	M5	D1	D2

OBSERVATION RECORD

Student			
Description of activity undertaken			
Assessment & grading criteria			
How the activity meets the requirements of the criteria			
Student signature:		**Date:**	
Assessor signature:		**Date:**	
Assessor name:			

❒ Summative Feedback:	❒ Resubmission Feedback:

Grade:	Assessor Signature:	Date:

Internal Verifier's Comments:

Signature & Date:

Table of Contents

A. INTRODUCTION

We are BudgetWise Solutions, a dedicated and driven development team comprising Nguyen Nhu Phuc (Lead Developer), Vu Minh Hieu (UI/UX Designer), and Bui Viet Hoang (Backend Developer). Although our experience in mobile app development is limited, we view this project as a valuable opportunity for growth and innovation. Our initiative, CampusExpense Manager, aims to serve as an effective tool for university students to efficiently track and manage their daily expenses. Recognizing the financial hurdles that students encounter, we are focused on creating an intuitive and user-friendly application that allows them to establish budgets, monitor their spending, and gain better control over their finances, regardless of whether they reside on or off-campus. By harnessing our collective expertise in software development, design, and backend systems, we are dedicated to delivering a seamless and practical financial management solution. Through our commitment and collaboration, we aspire to make CampusExpense Manager a trustworthy resource for students seeking to cultivate smart financial habits and maintain their budgets.

B. DESIGN ANALYSIS AND EVALUATION

I. Identify and write detailed problem, user and system requirements

1. Scenario

Problem: The CampusExpense Manager application was designed to assist students in efficiently monitoring and managing their personal finances. Many students struggle with budget control, particularly when faced with numerous fixed and recurring monthly costs. This application offers features for expense tracking, budgeting, and financial analysis, enabling students to sustain a stable financial condition during their academic journey.

Main requirements of the application:

- User Registration and Authentication
  - Users can create an account with a username and password.
  - The authentication system is secure and allows login to access personal data.
- Track Expenses
  - Users can add, edit, and categorize expenses (e.g. rent, food, transportation).
  - Each expense includes a description, date, amount, and corresponding category.
- Set Budget
  - Allows users to set a monthly budget for each spending category (e.g. food, entertainment, education).
  - Budget limits can be adjusted as needed.
- Spending Overview
  - Shows total monthly spending, remaining budget balance, and category breakdown.
  - Provides visual charts to help users track spending trends.
- Recurring Expense Management
  - Supports adding recurring expenses (e.g. monthly rent).
  - Automatically updates to monthly budget.
- Spending Report
  - Allows users to generate spending reports for specific time periods (monthly, yearly).
  - Detailed reports help users evaluate and adjust their financial plans.
- Spending Alerts
  - Warns you when you're about to exceed your set budget.
  - Reminds you when recurring expenses are due.

Non-functional requirements:

- Performance
  - work smoothly, even with a lot of spending data.
- User-Friendly Interface
  - The interface design is simple and easy to use, helping students enter and track spending quickly.

- Cross-Platform Compatibility
    - Supports both Android and iOS operating systems to reach more users.
- Data Security
    - User data is encrypted and protected to ensure privacy.
    - Complies with personal data protection regulations.
- Support and Feedback
    - Provide a feedback form for users to send feedback and report errors.
    - The development team monitors and updates the app regularly.
- Offline Support
    - The app can work without an internet connection to support students in areas with weak networks.

The CampusExpense Manager application is designed to assist students in managing their personal finances efficiently and effectively, enabling them to make informed financial choices during their academic journey.

2. Analyze the calculation requirements

Functional Requirements:

- Registration and Authentication:
    - Users have the ability to securely create accounts, sign in, and sign out.
    - Protect user information while facilitating the management of personal data and spending.
- Track your spending:
    - Users have the ability to add, modify, remove expenses, and organize them by category.
    - Assists students in effortlessly monitoring their daily expenditures.
- Set a budget:
    - Establish a budget categorized by areas such as food, education, and entertainment.
    - Assist users in gaining improved management of their financial resources.
- Spending Overview:
    - Display the total expenditure, the remaining budget, and a chart illustrating the spending trend.
    - Offers a user-friendly interface to assist individuals in overseeing essential assets.
- Recurring expenses:
    - Automatically incorporate monthly expenses such as rent and utilities.
    - Assist users in reducing the time spent on data entry.
- Expense report:
    - Generate export reports based on timeframes (monthly, yearly) and evaluate expenditures by category.
    - Facilitate individual financial planning.

- Notification:
    - Notifications when approaching or exceeding budget limits.
    - Assist students in gaining improved management of their financial resources.

Non-funtional requirements:

- Performance:
    - The application must operate efficiently, even when handling large volumes of data.
    - Ensure a positive user experience by minimizing lag.
- Friendly interface:
    - User-friendly design that is simple to navigate.
    - Assist users in navigating efficiently while minimizing access obstacles.
- Platform Compatibility:
    - Operates on both Android and iOS platforms.
    - Increase your audience engagement.
- Data Security:
    - Safeguard user data and secure financial details.
    - Guarantee confidentiality and adhere to security standards.
- Feedback and Support:
    - Facilitates the submission of feedback and support from the development team.
    - Enhance the application by incorporating user feedback.

System Requirements:

- Database:
    - Data storage for users.
    - Maintain the integrity and security of data.
- Technology development: Select from Flutter, React Native, or Native.
- User Interface:
    - Utilize Material Design principles for Android or adhere to the Human Interface Guidelines for iOS.
    - Create an attractive and user-friendly interface.
- API and Support Services:
    - Firebase Authentication, along with Google Cloud Functions, provides a robust framework for managing user authentication and backend processes.
    - Enhance authentication processes and improve data synchronization.

Requirements

Criteria	Functional Requirements	Non-functional Requirements	System Requirements

Target	Determine particular functionalities of the application.	Guaranteeing efficiency, safety, and user satisfaction	Identify the necessary hardware, software, and platform specifications.
Project	- Register, login, verify account - Add, edit, delete transactions - Set budget - View spending reports	- Smooth performance with large data - Simple, easy-to-use interface - The application must work offline	- Runs on Android & iOS - Secure data storage - Integrated notification system
Level of importance	Essential for the operation of the application	Influence on user experience and security.	Determine the approach for application deployment and scalability
Impact if not met	The application is unable to execute its primary function	User dissatisfaction may lead to a decline in application usage	Challenging to implement or hard to sustain
Ability to change	This can be further developed and enhanced in the future	Typically established, it requires optimization from the outset	The outcome is contingent upon the initial platform and technology used

Solutions for Functional requirements

Functional requirements	**Proposed solution**
Sign up & verify account	Utilize Firebase Authentication or OAuth2 to ensure secure user registration and login processes
Expense management	Utilize SQLite for offline storage in conjunction with Firebase Firestore for cloud-based data management. This setup enables users to add, modify, and remove expense entries, which include fields for description, amount, date, and category.
Set a budget	- Enables budget allocation by category - Provides alerts when expenditures surpass the allocated budget.
View spending reports	Utilize visual representations such as pie charts and bar charts to present data effectively. Implement filters that allow users to generate reports based on weekly, monthly, or yearly intervals.
Spending reminder	Utilize Firebase Cloud Messaging (FCM) or Local Notifications to issue alerts when the budget is running low.
Support recurring transactions	Facilitates the automatic configuration of transactions, such as monthly rent payments. Employs cron jobs or background services to seamlessly incorporate transactions without manual intervention.

Solutions for Non-functional Requirements

Non-functional Requirements	Proposed solution
High performance	- Enhance data retrieval processes through the implementation of pagination and caching techniques. - Restrict the volume of data loaded initially to prevent performance lags.
Friendly interface	- Create the user interface in alignment with Material Design principles for Android and Human Interface Guidelines for iOS. - Implement Flutter or React Native to maintain a uniform interface across platforms.
High security	- Secure sensitive information using AES-256 encryption. - Implement JWT tokens for authenticating user login sessions.
Feedback and support	Incorporate feedback forms into your application. Utilize Google Forms or Firebase Crashlytics for monitoring errors.

Solutions for System Requirements

System requirements	Proposed solution
Multi-platform support	Utilize Flutter (Dart) or React Native for development on both Android and iOS platforms.
Database	Firebase Firestore (cloud) combined with SQLite (offline) for the storage of expenditure data.
System Notifications	Firebase Cloud Messaging (FCM) is utilized for delivering notifications, while Local Notifications serve as reminders when offline.
Third-party API integration	- Utilize Google Authentication and Apple Sign-In for streamlined login processes. - Implement a currency conversion API to enable support for various currencies.
Scalability	Constructed using MVVM architecture to facilitate straightforward future feature enhancements.

3. Conclusion and analysis summary

Object-Oriented Analysis and Design, or OOAD, is a method in software engineering that focuses on analyzing, designing, and creating software systems through the use of objects. It applies key concepts from object-oriented programming (OOP), including encapsulation, inheritance, polymorphism, and abstraction, to create systems that are simple to expand and maintain.

The OOAD process consists of two main stages:

- Object-Oriented Analysis (OOA) involves figuring out the different objects, their characteristics, and how they relate to each other by looking at what the problem needs.
- Object-Oriented Design (OOD) takes the analysis model and turns it into a software structure that can be programmed by using various software design principles.

Why is the "CampusExpense Manager" problem suitable for the OOAD method?

The CampusExpense Manager application development problem is very suitable for the OOAD method:

- The issue at hand is that there are numerous entities that are closely interconnected.
    - The application has clear objects such as:
        - User: Has account information, sets up personal budget.
        - Expense: Each transaction has amount, date, spending type.
        - Budget: Set spending limit for each category.
        - Report: Summarizes information from multiple spending items.
    - Using object orientation allows us to arrange these entities in a way that feels natural and can easily be expanded.
- Support system reuse and expansion:
    - Object-Oriented Analysis and Design (OOAD) facilitates code reuse through the implementation of inheritance and polymorphism.
    - You can enhance income management capabilities or incorporate AI-driven spending forecasts by extending the current layers, ensuring that the overall system remains unaffected.
- Easy to maintain and manage source code
    - When an error occurs or a function requires updating, we can simply modify a class without impacting the entire application.
- Support effective teamwork
    - Every team member is able to focus on distinct classes independently, ensuring that the work of others remains unaffected.
- Easy modeling with UML
    - Object-Oriented Analysis and Design (OOAD) can effectively illustrate the system through UML diagrams, including Class Diagrams, Use Case Diagrams, and Sequence Diagrams, facilitating a clearer understanding and planning of the development process.

Main objects

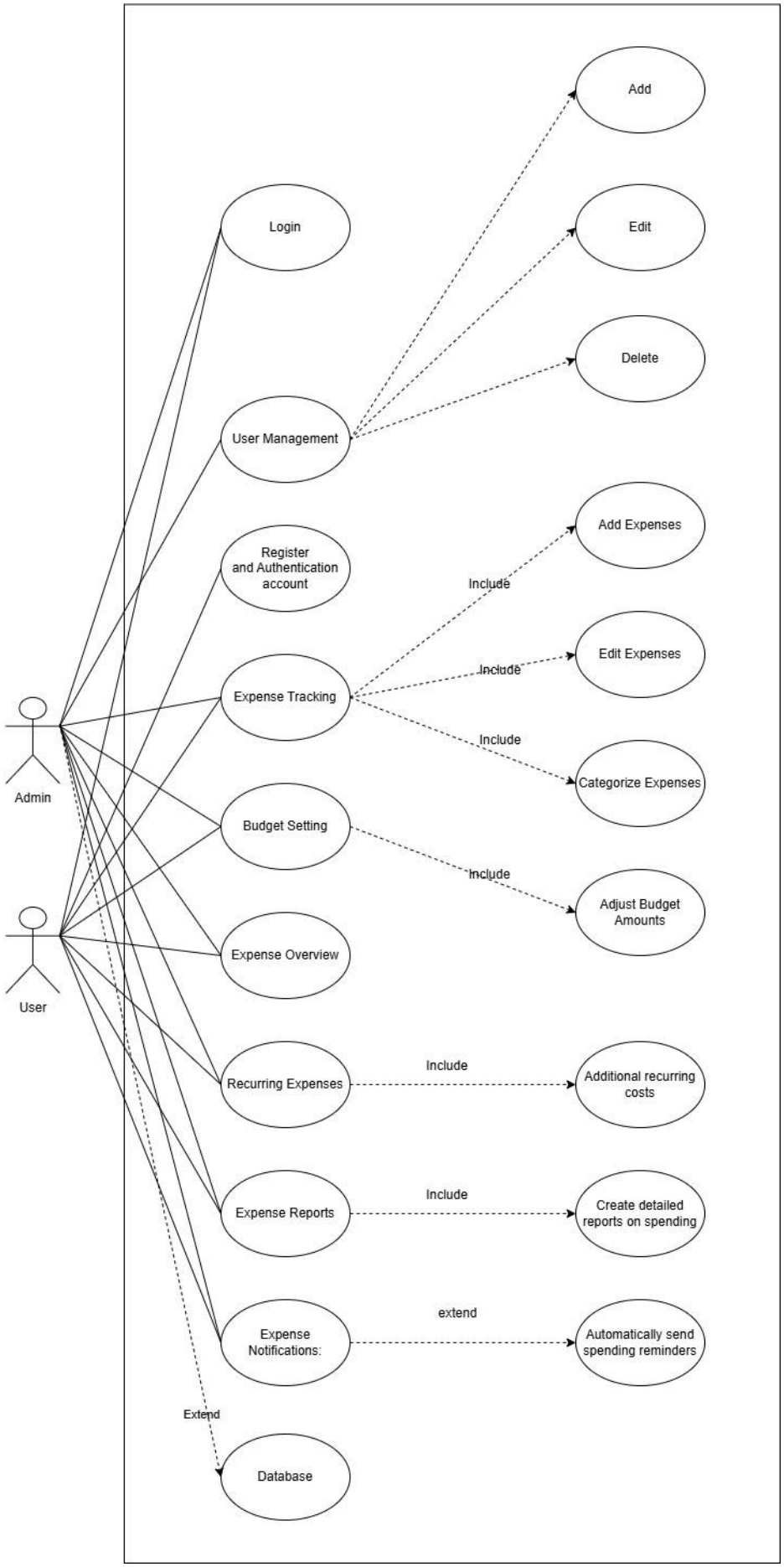
Object	Attributes	Methods/Funtions
User	userID, username, password, email, role	register(), login(), resetPassword(), updateProfile()
Expense	expenseID, userID, categoryID, amount, description, date	addExpense(), editExpense, deleteExpense(),

		getExpenseByMonth(month, year)
Budget	budgetID, userID, categoryID, amount, startDate, endDate	setBudget(categoryID, amount), updateBudget(), getRemainingBudget()
Report	reportID, userID, startDate, endDate	generateMonthlyReport(userID, month, year), generateAnnualReport(userID, year), exportReport(format)
Notification	notificationID, userID, message, dateSent	sendNotification(userID, message)
Category	categoryID, name, description	createCategory(name, description), updateCategory(), deleteCategory()
RecurringExpense	recurringID, userID, categoryID, amount, startDate, endDate, frequency	addRecurringExpense(), editRecurringExpense(), deleteRecurringExpense()

Object-Oriented Design (OOD)

Use-case diagram

Use Case

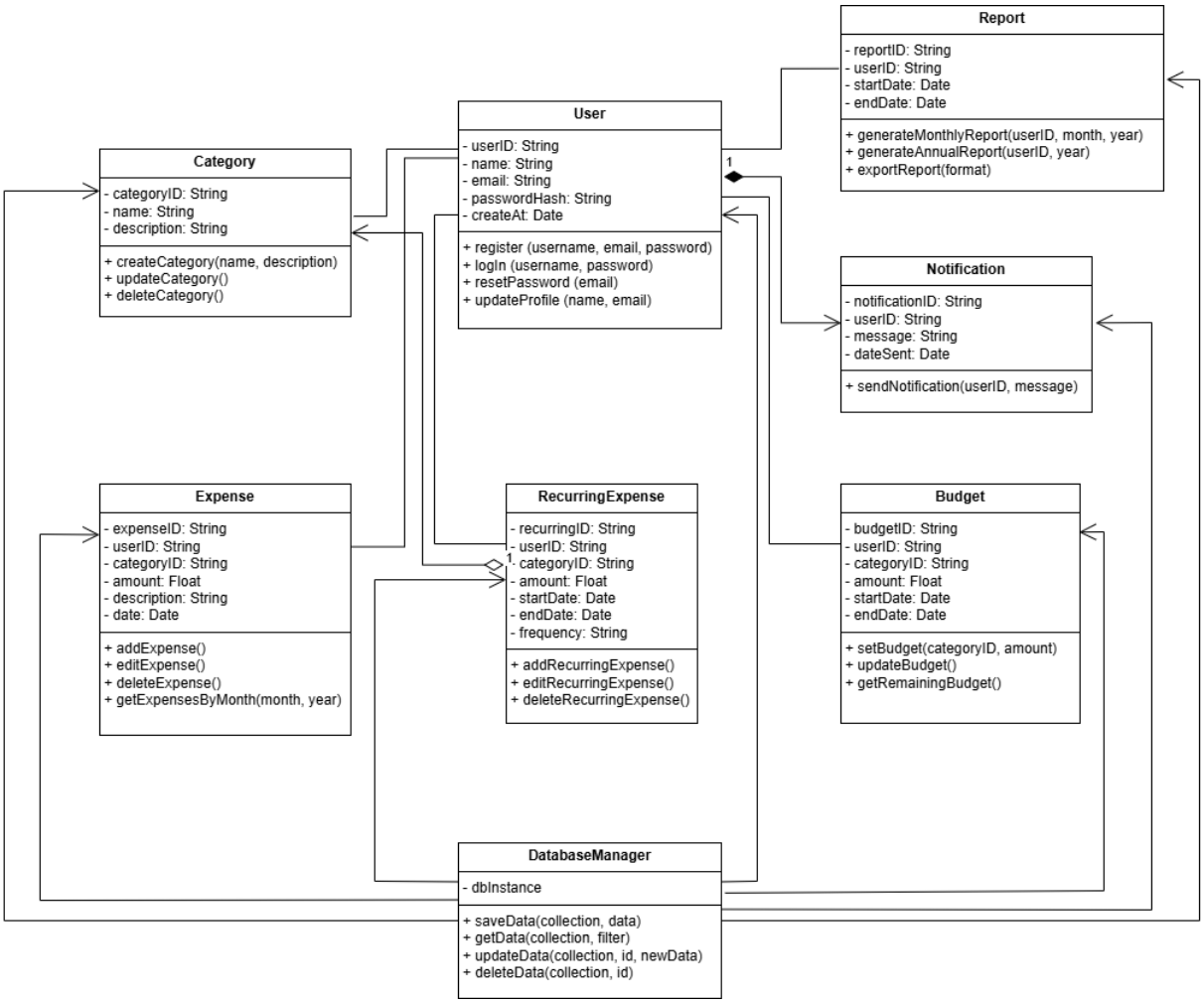

Add
Login
Edit
Delete
User Management
Add Expenses
Register and Authentication account
Include
Edit Expenses
Include
Expense Tracking
Include
Categorize Expenses
Admin
Budget Setting
Include
Adjust Budget Amounts
User
Expense Overview
Recurring Expenses
Include
Additional recurring costs
Expense Reports
Include
Create detailed reports on spending
extend
Expense Notifications:
Automatically send spending reminders
Extend
Database

The Use Case Diagram presented illustrates an expense management system featuring two primary entities: Admin and User.

- Actors:
    - Admin: Administrators possess the authority to oversee user management, expenditures, reports, budgets, notifications, and more.
    - User: Standard users are able to monitor their spending, establish budgets, access reports, and receive notifications, among other functionalities.
- Use cases:
    - User management
        - Login: Log in to the system.
        - Register and Authentication Account: Register and authenticate account.
        - User Management: Manage user information, including:
            - Add: Add user.
            - Edit: Edit user information.
            - Delete: Delete user.
    - Expense Tracking
        - Add Expenses: Add expenses.
        - Edit Expenses: Edit expenses.
        - Categorize Expenses: Categorize expenses.
    - Budget Setting
        - Adjust Budget Amounts: Adjust budget amounts.
    - View Expense Overview
    - Recurring Expenses
        - Additional Recurring Costs: Add recurring expenses.
    - Expense Reports
        - Create Detailed Reports on Spending.
    - Expense Notifications
        - Automatically Send Spending Reminders.
    - Database
        - Extend system with Database to store data.
- Relationships
    - Include:
        - Some sub-features of a larger feature. For example:
        - Expense Tracking includes Add Expenses, Edit Expenses, Categorize Expenses.
        - Budget Setting includes Adjust Budget Amounts.
        - Recurring Expenses includes Additional Recurring Costs.
        - Expense Reports includes Create Detailed Reports on Spending.
    - Extend:

- Expense Notifications can be extended with Automatically Send Spending Reminders.
- Database extends the system to store data.

Class diagram



The diagram presented above is a Class Diagram illustrating the object model for an expense management system. A comprehensive analysis of the diagram follows below

The diagram includes the following main class:

- User Class
  - Properties:
    - userID: Unique identifier of the user.
    - name: Username.

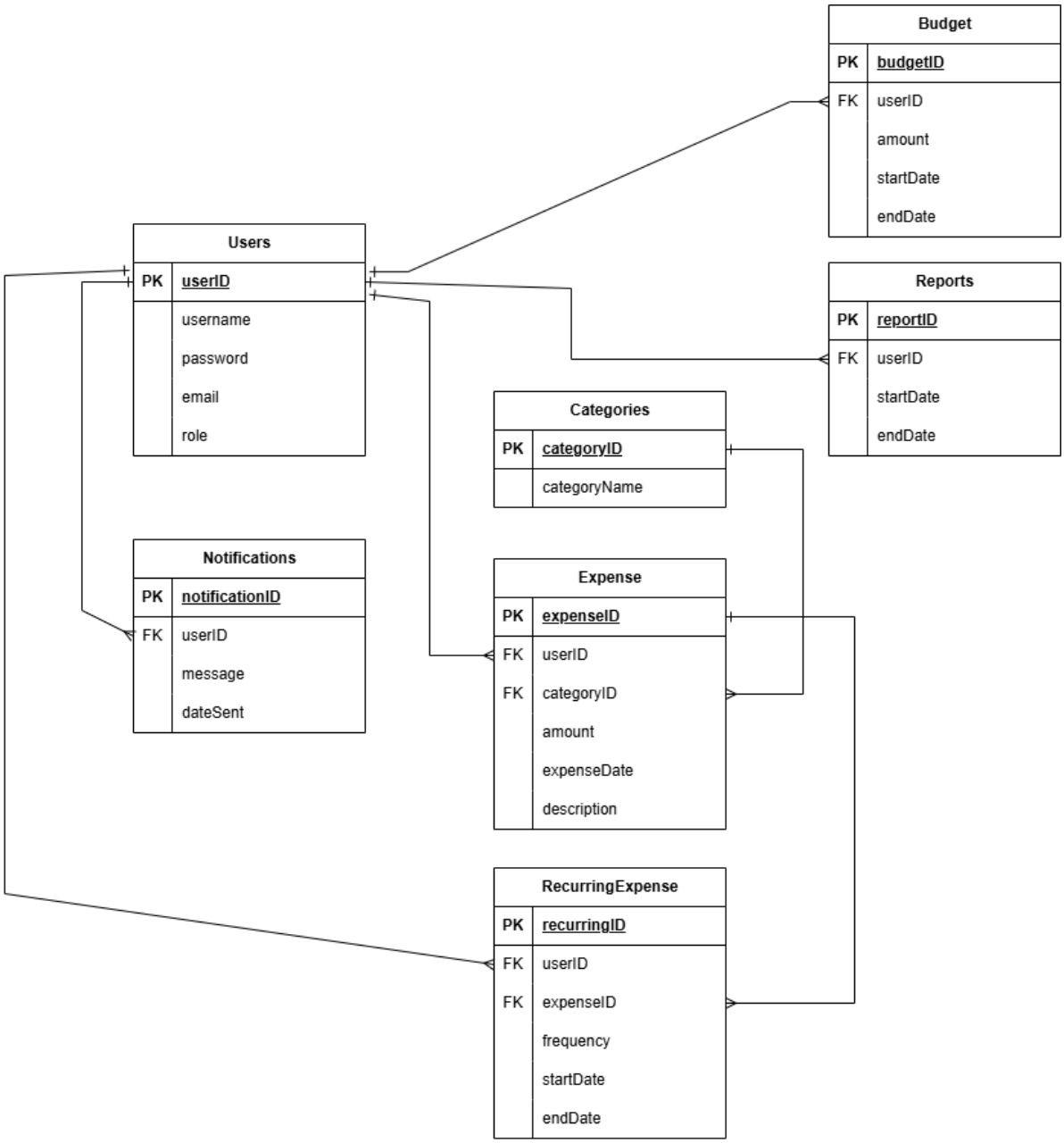
        - email: Email address.
        - passwordHash: Encrypted password.
        - createAt: Date the account was created.
    - Methods:
        - register(username, email, password): Register an account.
        - login(username, password): Log in.
        - resetPassword(email): Reset password.
        - updateProfile(name, email): Update profile information.
- Expense Class
    - Properties:
        - expenseID: Expense ID.
        - userID: User ID (linked to User).
        - categoryID: Category ID (linked to Category).
        - amount: Amount spent.
        - description: Expense description.
        - date: Expense date.
    - Methods:
        - addExpense(): Add Expense.
        - editExpense(): Edit Expense.
        - deleteExpense(): Delete Expense.
        - getExpensesByMonth(month, year): Get a list of expenses by month.
- Category Class
    - Properties:
        - categoryID: Category ID.
        - name: Category name.
        - description: Category description.
    - Methods:
        - createCategory(name, description): Create a new category.
        - updateCategory(): Update the category.
        - deleteCategory(): Delete the category.
- RecurringExpense Class
    - Properties:
        - recurringID: ID of the recurring expense.
        - userID: User ID.
        - categoryID: Category ID.
        - amount: Amount.
        - startDate: Start date.
        - endDate: End date.
        - frequency: Frequency (weekly, monthly, etc.).
    - Methods:
        - addRecurringExpense(): Add a recurring expense.

        - editRecurringExpense(): Edit a recurring expense.
        - deleteRecurringExpense(): Delete a recurring expense.
- Budget Class
    - Properties:
        - budgetID: Budget ID.
        - userID: User ID.
        - categoryID: Category ID.
        - amount: Budget amount.
        - startDate: Budget start date.
        - endDate: Budget end date.
    - Methods:
        - setBudget(categoryID, amount): Set budget for category.
        - updateBudget(): Update budget.
        - getRemainingBudget(): Get remaining budget amount.
- Report Class
    - Properties:
        - reportID: Report ID.
        - userID: User ID.
        - startDate: Report start date.
        - endDate: Report end date.
    - Methods:
        - generateMonthlyReport(userID, month, year): Generate monthly report.
        - generateAnnualReport(userID, year): Generate annual report.
        - exportReport(format): Export report in different formats.
- Notification Class
    - Properties:
        - notificationID: Notification ID.
        - userID: User ID.
        - message: Notification content.
        - dateSent: Date the notification was sent.
    - Methods:
        - sendNotification(userID, message): Sends notification to user.
- DatabaseManager Class
    - Properties:
        - dbInstance: Instance of database.
    - Methods:
        - saveData(collection, data): Save data to database.
        - getData(collection, filter): Query data.
        - updateData(collection, id, newData): Update data.
        - deleteData(collection, id): Delete data.

Relationship between classes

- The user maintains a one-to-many relationship with Expense, RecurringExpense, Budget, Report, and Notification. Both Expense and RecurringExpense are associated with Category, and Budget is similarly connected to Category. The DatabaseManager has the capability to access all objects within the system.

ERD



The diagram presented above illustrates the data model for a personal or corporate expense management system. Below is a comprehensive analysis of the diagram

Main entities:

- Users
    - Primary Key (PK): userID
    - Attributes:
        - username: Username.
        - password: Password (needs to be encrypted in practice).
        - email: User email address.
        - role: User role (can be admin, user,...).
    - Relationship:
        - 1-N relationship with Expense, RecurringExpense, Budget, Reports, Notifications.
- Categories
    - Primary Key (PK): categoryID
    - Attribute:
        - categoryName: The name of the expense category (e.g. Food, Travel, Bills...).
    - Relationship:
        - 1-N relationship with Expense (each expense belongs to one category).
- Expense
    - Primary Key (PK): expenseID
    - Foreign Key (FK):
        - userID (linked to Users).
        - categoryID (linked to Categories).
    - Attributes:
        - amount: Amount spent.
        - expenseDate: Date of the expense.
        - description: Description of the expense.
    - Relationship:
        - 1-N relationship with RecurringExpense (an expense that can repeat periodically).
- RecurringExpense
    - Primary Key (PK): recurringID
    - Foreign Key (FK):
        - userID (linked to Users).
        - expenseID (linked to Expense).
    - Attributes:
        - frequency: Frequency of recurring expenses (daily, weekly, monthly...).
        - startDate: Start date.
        - endDate: End date.

- Budget
    - Primary Key (PK): budgetID
    - Foreign Key (FK):
        - userID (linked to Users).
    - Attributes:
        - amount: Budget amount.
        - startDate: Budget start date.
        - endDate: Budget end date.
- Reports
    - Primary Key (PK): reportID
    - Foreign Key (FK):
        - userID (linked to Users).
    - Attributes:
        - startDate: Report start date.
        - endDate: Report end date.
    - Function:
        - Stores spending reports over time.
- Notifications
    - Primary Key (PK): notificationID
    - Foreign Key (FK):
        - userID (linked to Users).
    - Properties:
        - message: Notification content.
        - dateSent: Date the notification was sent.
    - Function:
        - Notify users of events such as budget overspending, recurring spending reminders, etc.

Relationship between entities

- Users maintain a one-to-many relationship with Expenses, Budgets, Reports, and Notifications, meaning that each user can have multiple instances of these elements.
- An Expense can transition into a Recurring Expense, establishing a one-to-many relationship between them.
- Each Expense is linked to a specific Category, indicating a one-to-many relationship with Categories.
- Budgets are exclusively tied to Users and are not currently linked to Categories, although there is potential for future enhancements to incorporate category-based budgeting.
- Reports are connected to Users but do not have a direct relationship with Expenses or Budgets; they primarily consist of aggregated data.

II. Consider the risks in application development (risks)

1. List the risks and describe and analyze the reasons why these risks occur

Potential challenges associated with the development of the CampusExpense Manager application.

- Technical risks
    - Security Vulnerabilities and Data Leaks
        - Description: The application handles sensitive information such as user accounts and financial data. If not properly secured, hackers can steal data or take control of the account.
        - Cause:
            - Inadequate data encryption.
            - Lack of two-factor authentication (2FA).
            - SQL Injection, XSS, or CSRF vulnerabilities due to failure to validate user input.
- Poor application performance when data is large
    - Description: As the number of transactions and financial reports increases, the application may become slow, affecting the user experience.
    - Cause:
        - The database design is not optimized (indexing, partitioning is not used).
        - Data queries are not optimized.
        - Caching is not used to reduce the load on the server.
- The app doesn't work well on both Android and iOS
    - Description: If not tested carefully, the app may work well on one platform but fail on another.
    - Causes:
        - Using incompatible libraries between the two operating systems.
        - Lack of testing on multiple devices.
        - Not optimizing the interface according to the UI/UX standards of each platform.
- Offline Mode Not Supported Well
    - Description: Some students may use the app in areas without internet. If the app does not support offline well, they will not be able to access the data.
    - Cause:
        - Data is not stored locally on the device.
        - There is no mechanism to synchronize data between offline and online modes.

Risks associated with the development process

- Development Team Capacity Limitations

    - Description: BudgetWise Solutions development team has limited experience in mobile application development, which can lead to design and implementation errors.
    - Causes:
        - Lack of expertise in mobile programming (Android, iOS).
        - Lack of experience in optimizing performance and security.
        - Insufficient resources to learn and improve skills in a short time.
- Development Time Delayed (Deadline Missed)
    - Description: The project has a development time of only 12 weeks, if not managed well, it may not be completed on time.
    - Cause:
        - Lack of detailed planning and reasonable division of work.
        - Encountering complex technical errors but no quick solution.
        - Delay in feedback between team members.
- Not meeting user needs
    - Description: If the application is not really convenient or does not solve the spending management needs of students, they will not use it.
    - Causes:
        - Not thoroughly surveying the actual needs of students before designing.
        - Not testing with real users before deploying.
        - Not listening to user feedback to adjust the product.

Risks related to business and finance

- Development Budget Constraints
    - Description: The project has a limited budget, if the cost is not managed properly, it may have features cut or be stopped halfway.
    - Causes:
        - Incorrect initial cost estimation.
        - Increased development costs due to constant changes in requirements.
        - Lack of highly qualified personnel, having to outsource at high cost.
- No Monetization Strategy
    - Description: If an app doesn't have a clear monetization model, it will be difficult to maintain and grow in the long term.
    - Causes:
        - Not properly integrating advertising or paid features.
        - Users are not willing to pay for the app.
        - Not finding the right sponsor or investment source.

Legal and data security risks

- Non-compliance with data protection regulations (GDPR, PDPA, …)

- Description: The application processes personal information of students, if it does not comply with data protection regulations, it may be sued or fined.
- Cause:
  - No clear data protection policy.
  - Storing sensitive information without encryption.
  - Sharing user data without consent.

2. Proposed solutions for risks

Type of risk	Risk Name	Risk Description	Level of impact	Likelihood	Risk prevention solutions
Technique	Security vulnerabilities and data leaks	Cybercriminals have the capability to infiltrate systems and extract users' financial information	High	Medium	- Data encryption (AES, SHA-256) - Implement two-factor authentication (2FA) - Regular security testing and patching
	Poor performance when data is large	The application experiences reduced performance when users have a high volume of transactions or reports	High	High	- Use indexing and caching - Optimize SQL queries - Offload using cloud platforms
	Doesn't work well on Android and iOS	The application might experience crashes on certain devices or operating systems	Medium	High	- Use cross-platform framework (Flutter, React Native) - Test on different devices
	Does not support offline mode well	Users are unable to process transactions without an Internet connection	Medium	Medium	- Use local database (SQLite, Room Database) - Data synchronization mechanism when there is a network connection

Development process	Lack of experience of development team	The team's limited experience in mobile programming increases the likelihood of errors.	High	High	- Short-term training on mobile programming - Cooperate with experts or mentors
	Late project deadline	Projects can exceed a duration of 12 weeks as a result of several technical or managerial considerations	High	High	- Agile detailed planning - Weekly progress checks - Break down functions for easy control
	Not meeting user needs	The application lacks user-friendliness and fails to address the issues faced by students	High	Medium	- Survey users before designing - Beta test with a small group of students - Collect feedback and improve
Finance & Business	Limited development budget	Insufficient funding can hinder both the advancement and the quality of the product	High	Medium	- Prioritize core features first - Consider raising capital from investors or university funding
Legal & Security	Non-compliance with data protection laws (GDPR, PDPA)	If breached, one may face legal action or administrative penalties	High	Medium	- Establish a clear privacy policy - Do not store unnecessary personal information - Ensure user consent when collecting data

III. Evaluate technology solutions and compare solution options

1. Propose technological solutions that respond to the above analysis

I have researched various technology solutions via the Internet and social media platforms, as well as through my academic studies. The following tools are among the most frequently utilized

Solution Type	Solution name	Defination	Applicable cases	Strengths	Weaknesses	Popularity
Programming language	Java	Java is a widely-used object-oriented programming language, particularly favored for developing Android applications	Mobile application development on Android.	Good security, high performance, large support community.	Writing code is more verbose than some other languages like Kotlin.	Very popular in Android app development.
	Kotlin	Kotlin is a contemporary programming language that Google endorses for Android development	Android application development, performance optimization.	The code is concise, easy to read, and integrates well with Java.	Learning takes time if you are familiar with Java.	Growing strong, increasingly popular.
	Dart	The programming language utilized in Flutter for developing cross-platform applications	When you want to develop Android & iOS apps at the same time.	Create fast, beautiful, high performance apps.	New community, few libraries compared to Java.	Increasingly popular, especially with Flutter.

Development tools	Android Studio	The official integrated development environment (IDE) for Android.	Develop Android apps with Java or Kotlin.	Strong support, many built-in tools.	Configuration requires powerful computer.	Very popular, officially supported by Google.
	Flutter	Google UI Toolkit helps develop cross-platform apps.	When you want your app to run on both Android and iOS.	Code once, run on multiple platforms.	Learning takes time with Dart.	Popular in cross-platform development.
	React Native	Mobile application development using the Facebook framework	When you want your app to run on both Android and iOS.	Cross-platform support, easy to learn for JavaScript knowers.	Performance is not as high as native.	Very popular with the JavaScript community.
Data storage	Firebase	Google's cloud database platform, supporting real-time database.	When you need fast data synchronization, easy integration.	Easy to use, powerful, supports authentication.	Limited free, high fees when expanding.	Very popular for mobile applications.
	SQLite	Lightweight database, suitable for offline storage.	When you want to store data on the device.	Light, fast, no server needed.	Cloud sync is not supported.	Popular in mobile applications.
	Mongo DB Atlats	NoSQL database on cloud platform.	When you want the application to have great scalability.	Flexible, supports unstructured data.	More complex than Firebase.	Widely used in cloud applications.

2. Compare to choose the solution that meets the requirements and addresses the risks

Solution Type	Solution name	Solution characterization	Describe the solution's response to requirements and risks
Programming language	Java	Object-oriented language, popular for Android.	Good for Android application development, high security.

Development tools	Android Studio	Official IDE for Android, full feature support.	Well integrated with Java, ensuring performance and stability.
Data storage	Firebase	Cloud database supports real-time and authentication.	Easy to store, fast data synchronization, high security.

3. Conclusion on choosing technology solutions

The comparative analysis indicates that the most suitable technology solution for the CampusExpense Manager application is:

- Programming language: Java - ensures security, performance and good compatibility with Android.
- Development tools: Android Studio - is an official tool that fully supports features for Android application development.
- Data storage: Firebase - supports real-time data storage and synchronization, helping users track spending conveniently.

This solution comprehensively satisfies the system's criteria for functionality, performance, security, and scalability.

IV. Evaluation provides project development methodology

1. Proposed development methodologies

Method name	Defination	Applicable cases	Strengths	Weaknesses
Waterfall	In a linear model, each phase must be finalized before progressing to the subsequent phase	The project has clear requirements and changes little during development.	Easy to manage, detailed documentation.	Inflexible, difficult to adapt to change.
Agile	The Agile development model involves breaking the project down into numerous small iterations, commonly referred to as sprints	Projects often encounter changing requirements and necessitate prompt feedback from users	Flexible, responsive, increased product quality.	Requires high teamwork skills, difficult to apply if the team has no experience.
Scrum	Agile methodology encompasses defined roles and processes, including positions like	The project involves a compact development team, necessitating ongoing	Transparency, increase team performance.	Requires experienced Scrum Master to manage effectively.

	Product Owner and Scrum Master	communication with the client		

2. Compare to choose the right method

Method name	Characteristic	Describe the level of project suitability
Waterfall	Linear model, little change during development.	Not suitable because the project may change requirements based on student feedback.
Agile	Agile development, breaking down work into sprints.	Suitable because the project needs quick feedback from users, improving gradually in each stage.
Scrum	Agile processes are more structured, requiring specific roles.	Good if the team has enough experience with Scrum, helps with progress tracking and quick feedback.

3. Conclusion on method selection

The CampusExpense Manager project is ideally aligned with an Agile methodology due to its inherent flexibility and adaptability:

- Adaptable and ideal for projects with evolving requirements driven by user input.
- Facilitates quicker development in each sprint, guaranteeing project completion within a 12-week timeframe.
- Enhances product quality through testing and modifications based on actual user feedback.

V. Summary conclusion and overall evaluation of options

- A comprehensive analysis of functional, non-functional, and system requirements has been conducted. The primary risks identified encompass changes in requirements, potential performance challenges within the system, data security concerns, and the skill levels of the development team.
- Development Methodology: Agile was selected for its flexibility and rapid responsiveness to changing requirements.
- Technology Explanation:
    - Programming Language: Java, known for its popularity and the team's existing expertise.
    - Development Tool: Android Studio, which provides robust support for Android applications.
    - Database: Firebase, ensuring effective data synchronization and strong security.
- Evaluating and contrasting various solutions guarantees the optimal selection for the project.
- Agile methodology promotes adaptability and fosters ongoing enhancement.

- Leveraging Firebase for data processing facilitates effective performance over time.
- The development team might need to enhance their Scrum expertise to better structure their workflow.

C. CONCLUSION