CHAPTER 2. THEORETICAL BASIS

2.1. Fundamental Theory

2.1.1. Money management

Money management encompasses the processes of budgeting, saving, investing, and spending money wisely to achieve financial goals. It is a crucial aspect of overall financial behavior that significantly impacts an individual's financial well-being [8]. Effective money management is not merely about earning and spending, money management also involves strategically planning and managing resources to ensure financial stability and growth.

Budgeting is fundamental, as it involves tracking income and expenses to ensure alignment with financial objectives. A well-structured budget aids in prioritizing expenses, identifying potential savings, and preventing overspending. It serves as a roadmap for financial decisions, helping individuals allocate resources efficiently and avoid unnecessary expenditures. Saving is equally vital for financial security, as it entails reserving a portion of income for future needs or emergencies. Regular savings can help individuals establish an emergency fund, save for specific goals such as purchasing a house or expanding a business, and create a cushion for financial stability. This practice ensures that unexpected expenses do not derail long-term financial plans.

Investing involves allocating money into assets with the expectation of generating a return, facilitating wealth accumulation over time, and assisting in achieving long-term financial goals. It requires understanding various investment options, assessing risk tolerance, and formulating investment strategies that align with financial objectives. Effective investing is crucial for building a robust financial portfolio that can support future aspirations.

Debt management is critical to maintaining financial health, involving the understanding and effective handling of debt obligations such as credit card debt, student loans, or mortgages. Formulating a plan to pay off debt, avoiding high-interest debt, and maintaining a good credit score are essential practices. Effective debt management prevents financial strain and ensures that debt levels remain manageable.

Cash management refers to the effective handling of cash flow, which includes monitoring income, expenses, and liquidity to optimize the use of cash for daily expenses, savings, and investments. Maintaining financial stability requires a careful balance between income and expenditures, ensuring that sufficient cash is available to meet immediate needs while supporting long-term financial goals.

Credit management involves using credit responsibly and maintaining a good credit score, which includes understanding credit terms, avoiding excessive debt, and making timely payments. Responsible credit management ensures access to credit when needed and prevents the accumulation of burdensome debt.

Collectively, these elements form the foundation of effective money management. By comprehensively understanding and applying these principles, individuals and businesses can make informed decisions, mitigate financial risks, and work towards their financial goals. Effective money management is multifaceted and requires a thorough grasp of various financial practices [9]. For MSMEs, mastering these principles is particularly crucial as it significantly influences their financial well-being and overall success.

This thesis focuses on enhancing the efficiency of cash management, a critical area often overlooked by other applications that tend to emphasize different aspects of financial management. By concentrating specifically on improving the effectiveness of cash management practices, this research aims to address a significant gap and provide MSMEs with a robust tool for better financial oversight.

2.1.2. Micro, Small, and Medium Enterprises (MSME)

MSMEs are integral components of the economic fabric of many countries, particularly those in the developing world. These entities, which vary in size based on employee count, assets, or revenue, are renowned for their contributions to employment creation, production, value addition, GDP growth, and regional development.

According to [10] In Indonesia, MSMEs play a pivotal role in the economic landscape, with a particular emphasis on their recent evolution and the challenges they face. It is important to note that MSMEs are not a monolithic group; rather, they exhibit diversity across various subcategories. Micro Enterprises (MIEs) and Small Enterprises (SEs), which are prevalent in rural areas, serve as crucial contributors to the local economy. These enterprises often emerge from individuals facing limited employment opportunities, thereby reflecting the country’s unemployment and poverty challenges.

MIEs are typically informal operations, frequently established by less educated households or individuals as their primary income source [11]. These enterprises may employ inappropriate technologies and unskilled labor, including unpaid family members. On the other hand, SEs may operate in both informal and formal sectors, while Medium Enterprises (MEs) are more modern, well-organized production units that utilize advanced technologies and employ highly skilled workers.

In Indonesia, MSMEs represent more than 98% of all firms across sectors and provide employment for over 90% of the country’s workforce [1]. Despite their numerical significance and employment generation capacity, Indonesian MSMEs face challenges such as lower productivity compared to Large Enterprises (LEs) and constraints related to access to finance, technology, and markets. Initiatives to support MSMEs are viewed as a means to stimulate economic development, create employment opportunities, generate income, and alleviate poverty.

2.2. Technology Overview

In the dynamic and complex domain of Android application development, the selection of technologies is a crucial factor that influences the efficiency, functionality, and user experience of the final application. The strategic choice of tools can not only streamline the development process but also enhance the application's performance and facilitate its maintenance and updates. A profound understanding of these technologies and their capabilities is essential for their effective application in Android app development.

2.2.1. Jetpack Compose

According to [12] Jetpack Compose, as introduced by the Google Development Team, is a contemporary UI toolkit designed for crafting user interfaces for the Android Operating System using the Kotlin programming language. This toolkit signifies a paradigm shift from the conventional approach of constructing user interfaces using XML layout files in tandem with Kotlin code. Jetpack Compose empowers developers to make user interfaces in a more efficient and expedited manner, fostering a more streamlined and integrated development process.

A salient feature of Jetpack Compose is its capacity to facilitate developers to write UI code in a more declarative style. This implies that developers can articulate the desired UI state and allow the toolkit to manage the rendering and updates contingent on alterations to that state. This declarative methodology can result in more succinct and comprehensible UI code, enhancing its understandability and maintainability. This information underscores the innovative nature of Jetpack Compose and its potential to revolutionize Android application development.

2.2.2. Kotlin

Kotlin, a statically typed programming language developed by JetBrains, has emerged as a powerful tool in the realm of software development. It operates on the Java Virtual Machine (JVM) and can be compiled into JavaScript or native code, offering a high degree of flexibility. Since 2017, Kotlin has been officially supported by Google for Android app development, marking a significant milestone in its adoption [13].

One of the key features of Kotlin is its Conciseness. Kotlin is designed to minimize boilerplate code, which refers to sections of code that must be included in many places with little or no alteration. Type inference in Kotlin allows the compiler to deduce the type of a variable from its initializer expression, reducing verbosity and enhancing readability. Furthermore, data classes and lambda expressions in Kotlin allow for more expressive and concise code, improving maintainability.

Another unique feature of Kotlin’s type system is Null Safety, designed to eliminate the risk of null references, a common pitfall in Java. In Kotlin, types are non-nullable by default, and variables need to be explicitly declared as nullable if they are to hold a null value. This feature helps prevent the occurrence of Null Pointer Exceptions, a common runtime error, thereby enhancing the robustness of the code.

Kotlin also introduces the concept of Extension Functions, which allows developers to add new functions to existing classes without altering their source code. This feature enhances the modularity and flexibility of the code, enabling developers to extend the functionality of classes from external libraries or the standard library without inheriting from them.

Coroutines are a standout feature of Kotlin that simplifies asynchronous programming by enabling suspending and resuming of computations. They allow developers to write asynchronous code in a sequential style, making the code more readable and easier to understand. Coroutines are especially useful in managing tasks that are network or disk I/O-bound, such as database operations, network requests, and file reads/writes.

Kotlin's Smart Casts feature reduces the verbosity of the code by eliminating the need for explicit type checks and type casting. If the compiler can determine through a 'is' check or an 'if' statement that the variable is of a certain type, it automatically casts it to that type, reducing the need for explicit casting and enhancing code readability.

Data Classes in Kotlin provide a concise way to define classes that hold data. When a class is declared as a data class, Kotlin automatically generates boilerplate code like equals(), hashCode(), and toString() methods based on their properties. This leads to cleaner, more readable code, and promotes the use of immutability, which is a key principle in functional programming.

Finally, Interoperability is a key feature of Kotlin. It is fully interoperable with Java, which means all existing Java libraries and frameworks can be used in Kotlin, and Kotlin code can also be called from Java. This interoperability enables a smooth transition for developers already familiar with Java, as they can gradually introduce Kotlin features into their codebase without having to rewrite the entire codebase in Kotlin.

In essence, Kotlin offers a modern, expressive, and pragmatic approach to software development. It is particularly well-suited for Android app development due to its seamless integration with existing Java codebases and robust tooling support. This aligns well with the objectives of this thesis, as the proposed solutions aim to leverage advanced technologies like Kotlin to address the challenges faced by Indonesian MSMEs, thereby contributing to their growth and development.

2.2.3. Firebase

According to [14] Firebase is presented as a NoSQL-based platform renowned for its real-time database and backend services. These services facilitate data synchronization across clients and storage on Firebase’s cloud. The platform also provides a suite of features including a cloud-based infrastructure for Android app testing, comprehensive crash reporting, and targeted user notifications. Firebase is lauded for its user-friendly interface, scalability, and cost-effectiveness, making it a preferred choice for mobile app development. The platform offers an extensive array of services specifically designed for mobile app development, which include:

1. Firebase Analytics: This service provides critical insights into app usage and user engagement, enabling developers to comprehend how users interact with their applications.
2. Firebase Cloud Messaging (FCM): Formerly known as Google Cloud Messaging (GCM), FCM is a cross-platform solution for sending messages and notifications to Android, web applications, and iOS devices.
3. Firebase Auth: This service supports social login providers such as Facebook, Google, GitHub, and Twitter, facilitating user authentication using client-side code. It includes user management features for email and password login stored with Firebase.
4. Real-time Database (Firestore): This service offers a real-time database and backend services that synchronize application data across clients and store it on Firebase’s cloud. Client libraries are provided for seamless integration with Android, iOS, and JavaScript applications.
5. Firebase Storage: This service facilitates secure file transfer for Firebase apps, backed by Google Cloud Storage. Developers can efficiently store images, audio, video, and other user-generated content.
6. Firebase Test Lab for Android: This service provides a cloud-based infrastructure for testing Android apps across a variety of devices and configurations, offering test results and insights in the Firebase console.
7. Firebase Crash Reporting: This service generates detailed error reports within the app, grouping errors into clusters for easier triaging based on severity. Developers can log custom events to capture the steps leading up to a crash.
8. Firebase Notifications: This service enables targeted user notifications for mobile app developers, enhancing user engagement and interaction.

In this thesis, the author utilized Firebase Storage, Real-time Database (Firestore), and Firebase Auth for this project, demonstrating the practical application of these services in a real-world scenario. This underscores the versatility and comprehensive nature of Firebase as a platform for mobile app development. The decision to employ Firestore was also influenced by its nature as a Database-as-a-Service (DBaaS) [15].

DBaaS offers several advantages that align with the needs of this thesis. Firestore provides scalability on demand, crucial for handling varying workloads. It offers operational agility, allowing for rapid deployment and configuration changes with minimal effort. Firestore ensures enhanced security measures, a critical aspect of any data-driven application.

2.2.4. Text Recognition

Text recognition, as delineated in the study by [16], involves the identification and comprehension of text within natural scene images. This research specifically discusses the utilization of a Deep Convolutional Neural Network (CNN) [17] to perform word recognition on the whole proposal region at the same time. This involves taking the entire cropped region of the word as input to the network and gradually pooling evidence from across the image to perform the classification of the word across a large dictionary of words.

Moreover, the research [18] introduces the Practical Plus Optical Character Recognition (PP-OCR) system, which is an ultra-lightweight OCR solution designed to address the challenges of text recognition in various application scenarios. PP-OCR boasts a remarkably small overall model size, with specific sizes allocated for recognizing Chinese characters and alphanumeric symbols. The system comprises text detection, detected box rectification, and text recognition components, each optimized for efficiency and effectiveness.

The text recognition system within PP-OCR begins by receiving a cropped region of a word within a natural scene image as input. This region encapsulates the text that is to be recognized. The system then applies a CNN to process the entire cropped region of the word as input. This means that the CNN processes the entire word image in one go, rather than focusing on individual characters. The CNN, trained on synthetic data for realistic and varied word image samples, incrementally pools evidence from across the image to perform classification of the word across a comprehensive dictionary of words [19].

As a result, the CNN produces a probability distribution over all the words in the dictionary. The recognition result is then determined based on this probability distribution. The word with the highest probability is deemed the recognition result, meaning that the word that the CNN identifies as the most probable match for the input word image is chosen as the recognized word. This approach ensures efficient and accurate text recognition within natural scene images, showcasing the capabilities of the PP-OCR system in handling diverse text appearances and scenarios.

In this thesis, the author employs text recognition technology to facilitate the capture of receipts in the form of images and convert it into text. This technology subsequently enables the extraction and conversion of the textual content present on the receipt into a digital note within the application. This process significantly enhances the efficiency and accuracy of data capture and storage.

2.2.5. API Services

An Application Programming Interface, or API, is a set of rules and protocols that software applications use to communicate with each other [20]. It’s like a user interface, but instead of enabling human-computer interaction, it facilitates machine-to-machine communication. APIs are used to connect, extend, and integrate software systems. They allow different software systems to share data and functionality, making them essential for building distributed, loosely coupled systems. Web APIs, a type of API that operates over the internet, are commonly used in web applications, mobile apps, and cloud apps. They allow software applications to communicate and share data over the internet. Despite their crucial role, APIs are typically invisible to end users and operate “under the hood”. The only people who interact with APIs directly are developers, who use them to build applications or solutions.

APIs are not just about connecting software systems, they also connect businesses, services, and products. By providing a means for integrating IT systems within a company and with external business partners, APIs enable businesses to expand and innovate. They allow company assets to be accessed and used in new ways, both internally and by external third-party developers. A good API should be valuable, fit the needs of its users, be simple to understand, easy to integrate and monitor, and be secure, reliable, and meet performance requirements. APIs that fulfill these conditions are more likely to be adopted by developers and can become a means for retaining existing partners and obtaining new ones.

In this thesis, several APIs are employed to enhance the functionality and user experience of the application. The Exchangerates API, which provides exchange rate data for over 170 global currencies, is used to provide real-time currency values. The Rest Countries API complements this by providing detailed information about each country associated with the currency, allowing the application to present a comprehensive overview of each currency. Additionally, the Tab Scanner API is utilized to scan images and convert them into a line-by-line string format, effectively extracting receipt information and enhancing the application's text-to-object capabilities.

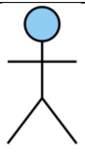
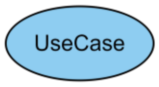
2.3. UML Diagram Overview

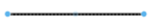
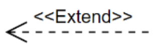
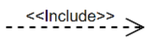
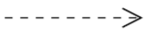
According to [21] The Unified Modelling Language (UML) is a visual language used in software engineering to define and document a system. It provides a way to visualize a system's architecture, design, and requirements through various types of diagrams. These diagrams are graphical illustrations that encapsulate various facets of a system, including its structure, behavior, and interactions. Here, the author will focus solely on the specific diagrams employed within the study.

2.3.1. Use-case Diagram

According to [22] A Use Case Diagram, a specific type of UML diagram, serves as a graphical representation of the interactions between a system and its users, or actors, in the context of use cases. These diagrams hold a pivotal role in the realm of software development, as they encapsulate the functional requirements of a system from the vantage point of the user. The symbols utilized in this study, along with their corresponding meanings, are comprehensively detailed in *Table 2.1* below:

Table 2.1: Use-Case Diagram Symbols

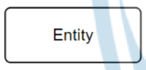
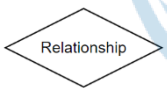
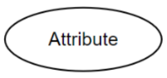
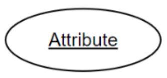
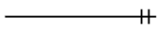
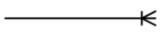
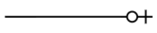
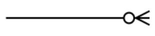
No.	**Symbol**	**Name**	**Description**
1.		Actor	An actor represents a user or an external system that interacts with the system being modeled. They are outside the system but have specific roles or goals within the system.
2.		Use Case	A use case represents a specific functionality or a discrete unit of work that the system performs to achieve a specific goal for an actor.

3.		Association	A line connecting an actor to a use case indicates that the actor is involved in that particular use case.
4.		Extend	Indicates that one use case can extend another use case under certain conditions.
5.		Include	Shows that one use case includes the functionality of another use case.
6.		Generalization	In some cases, actors or use cases can be generalized to show commonalities.

2.3.2. Entity Relationship Diagram

According to [23] An Entity-Relationship Diagram (ERD) is a visual representation of the entities within a system or domain, and the relationships between those entities. ERDs are commonly used in database design to model the structure of a database and illustrate how data is organized and related to each other. ERDs consist of three main components: entities, attributes, and relationships as explained in the Table 2.2.

Table 2.2: ERD Symbols

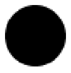
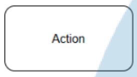
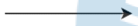
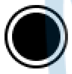
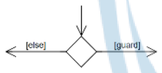
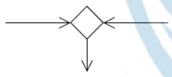
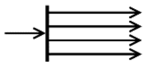
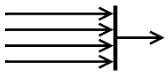
No.	Symbol	Name	Description
1.		Entity	Entities represent real-world objects or concepts within the system being modeled.
2.		Relationship	Relationships describe how entities are related to each other. They illustrate the connections and interactions between different entities in the database.
3.		Attribute	Attributes are the properties or characteristics of entities. They provide more detailed information about the entities.
4.		Primary Attribute	Primary Attributes are the unique key that differentiates one entity from another.
5.		To one Relationship	One Relationship is one of the relationships that shows the entity side will contain only one.
6.		To many Relationship	To Many Relationship is one of the relationships that shows the entity side will contain one or more.
7.		Optional To One Relationship	Optional To One Relationship is one of the relationships that shows the entity side can contain one or zero.
8.		Optional To Many	An optional To Many Relationship is one of the

		Relationship	relationships that shows the entity side can contain zero or many.

2.3.3. Activity Diagram

According to [24] activity diagram is a graphical representations used to describe internal processing and action-object flow within a system. They are part of the UML and offer a way to visualize workflows, especially the flow of control and data. Here's a detailed explanation of the symbols used in activity diagrams.

Table 2.3: Activity diagram symbols

No.	Symbol	Name	Description
1.		Start/ Initial Node	Used to represent the starting point or the initial state of an activity
2.		Action	Used to represent the executable sub-areas of an activity
3.		Control Flow / Edge	Used to represent the flow of control from one action to the other
4.		Activity Final Node	Used to mark the end of all control flows within the activity
5.		Decision Node	Used to represent a conditional branch point with one input and multiple outputs
6.		Merge Node	Used to represent the merging of flows. It has several inputs, but one output.
7.		Fork	Used to represent a flow that may branch into two or more parallel flows
8.		Merge	Used to represent two inputs that merge into one output