Project Documentation

Project Name : Money_Matters

Team ID : Team-591008

TEAM MEMBERS: Suraj S, Varsha Posa, Dhatri Lekkala, Alan Gomes.

INTRODUCTION :

Project Overview :

The Money Matters project is all about learning how to manage your finances wisely. We'll cover topics like budgeting, saving, and investing. This project basically helps in keep track of the expenses and accounts and provides an overview of their financial status. Users can set a budget for various expenses and view their progress towards it.

Purpose:

The purpose of the Money Matters project is to help you understand and improve your financial literacy. It's all about empowering you to make smart money decisions and achieve your financial goals. In the Money Matters project, we'll dive into topics like creating a budget, managing debt, and making smart investments. You'll gain practical knowledge and skills to take control of your finances and build a secure financial future. It's all about making your money work for you.

LITERATURE SURVEY:

Existing Problem:

One common problem in the Money Matters project is understanding how to create a realistic budget that aligns with your financial goals. It can be challenging to track expenses and prioritize spending. But, we'll provide tips and tools to help you create a budget that works for you and helps you achieve your financial goals. When it comes to creating a budget, it's important to start by tracking your income and expenses. This will give you a clear picture of where your money is going. Then, you can identify areas where you can cut back or save more. We'll also explore different budgeting methods and strategies to help you stay on track. Budgeting is all about finding a balance that works for you and your financial goals.

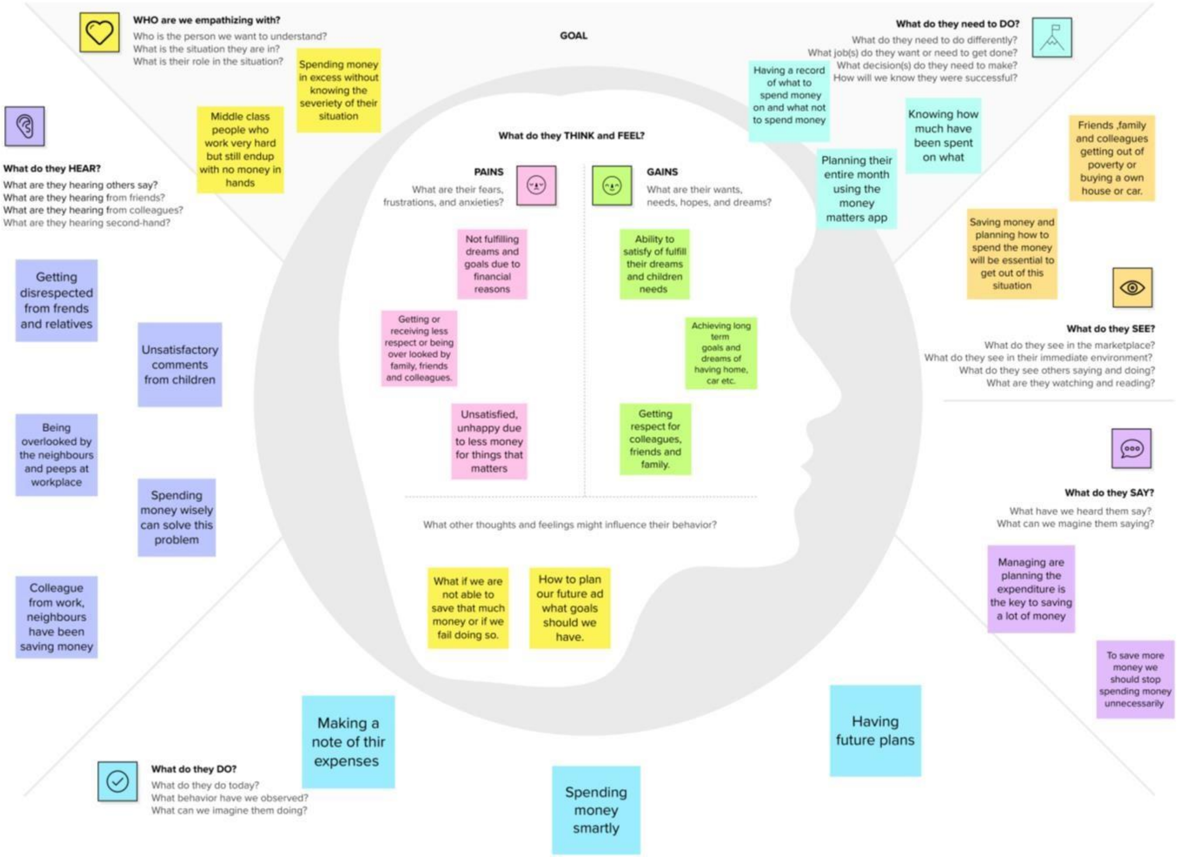
References :

For creating this app first of all we need the creating tool and by the help of websites and YNAB we got our references in creating the app. The knowledge sessions and pdfs and popular websites are the major references. Lastly we created the app using android studio.

Problem Statement Definition:
Many individuals struggle with managing their finances effectively, lacking the knowledge and skills to make informed financial decisions. This project aims to address this issue by providing accessible and practical resources to improve financial literacy and empower individuals to take control of their money and achieve their financial goals

IDEATION & PROPOSED SOLUTION:

Empathy Map Canvas :



WHO are we empathizing with?

Who is the person we want to understand?
What is the situation they are in?
What is their role in the situation?

Spending money in excess without knowing the severiety of their situation

Middle class people who work very hard but still endup with no money in hands

GOAL

What do they need to DO?

What do they need to do differently?
What job(s) do they want or need to get done?
What decision(s) do they need to make?
How will we know they were successful?

Having a record of what to spend money on and what not to spend money

Knowing how much have been spent on what

Planning their entire month using the money matters app

What do they HEAR?

What are they hearing others say?
What are they hearing from friends?
What are they hearing from colleagues?
What are they hearing second-hand?

Getting disrespected from frends and relatives

Unsatisfactory comments from children

Being overlooked by the neighbours and peeps at workplace

Spending money wisely can solve this problem

Colleague from work, neighbours have been saving money

What do they THINK and FEEL?

PAINS

What are their fears, frustrations, and anxieties?

Not fulfilling dreams and goals due to financial reasons

Getting or receiving less respect or being over looked by family, friends and colleagues.

Unsatisfied, unhappy due to less money for things that matters

GAINS

What are their wants, needs, hopes, and dreams?

Ability to satisfy of fulfill their dreams and children needs

Achieving long term goals and dreams of having home, car etc.

Getting respect for colleagues, friends and family.

What other thoughts and feelings might influence their behavior?

What if we are not able to save that much money or if we fail doing so.

How to plan our future ad what goals should we have.

What do they SEE?

What do they see in the marketplace?
What do they see in their immediate environment?
What do they see others saying and doing?
What are they watching and reading?

Friends ,family and colleagues getting out of poverty or buying a own house or car.

Saving money and planning how to spend the money will be essential to get out of this situation

What do they SAY?

What have we heard them say?
What can we magine them saying?

Managing are planning the expenditure is the key to saving a lot of money

To save more money we should stop spending money unnecessarily

What do they DO?

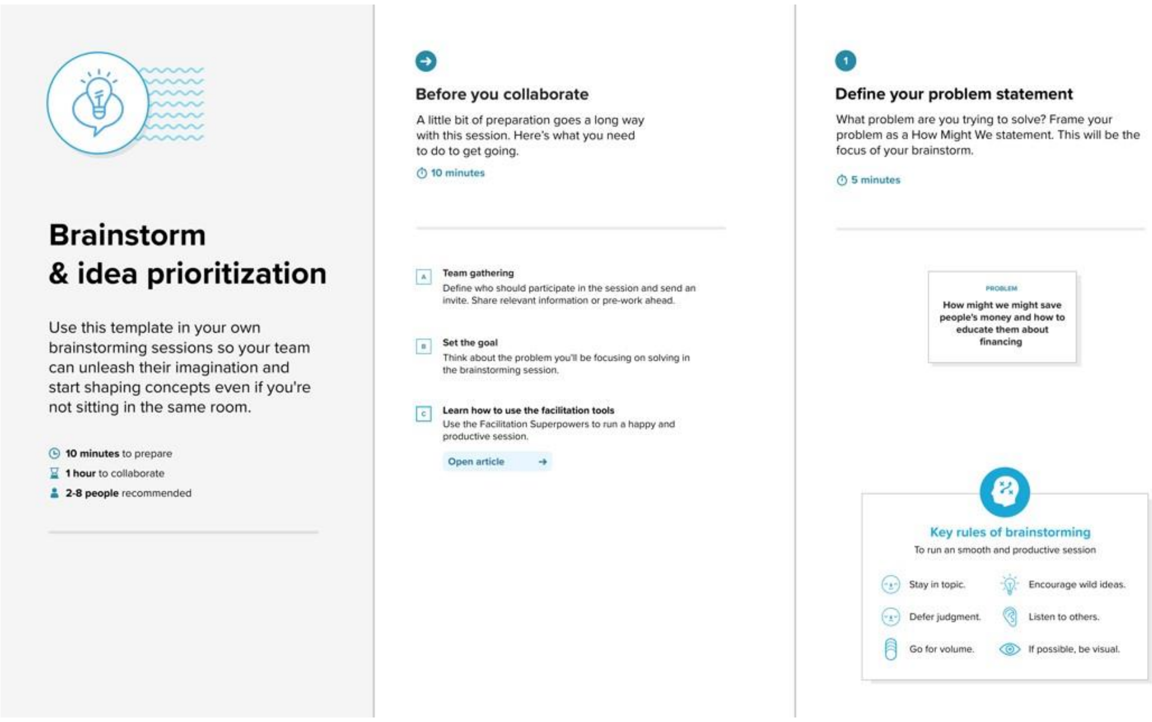
What do they do today?
What behavior have we observed?
What can we imagine them doing?

Making a note of thir expenses

Spending money smartly

Having future plans

Ideation & Brainstorming:



Brainstorm & idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

- **10 minutes** to prepare
- **1 hour** to collaborate
- **2-8 people** recommended

Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

A **Team gathering**
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B **Set the goal**
Think about the problem you'll be focusing on solving in the brainstorming session.

C **Learn how to use the facilitation tools**
Use the Facilitation Superpowers to run a happy and productive session.

1 Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

5 minutes

PROBLEM

How might we might save people's money and how to educate them about financing

Key rules of brainstorming

To run an smooth and productive session

- Stay in topic.
- Encourage wild ideas.
- Defer judgment.
- Listen to others.
- Go for volume.
- If possible, be visual.

2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

TIP

You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

Person 1

- Expense Tracking
- Setting budgets
- Showing remaining money
- Login page
- Ads for revenue
- Providing history of indian rupee

Person 2

- Adding max amount to spend for month
- Adding expenses daily
- Having profiles
- Making the app more attractive
- Allowing users to see the remaining money

Person 3

- Login page
- Different profiles for different users
- Ads for revenue
- Making the app more colourful

Person 4

- Tracking money
- Providing information about financing
- Get alearts
- Having profiles
- Calculator

Expense Tracking

Expense tracking is important for a number of reasons. First, it can help you to understand where your money is going. Once you know where you're spending your money, you can start to make changes to your budget to save more money.

Second, expense tracking can help you to avoid overspending. When you know how much you're spending each month, you can set limits for yourself and avoid going into debt.

Third, expense tracking can help you to reach your financial goals. Whether you're saving for a down payment on a house, retirement, or a vacation, expense tracking can help you to stay on track.

Login page

Security: The login page is the first line of defense against unauthorized access to your app. By using strong security measures, such as two-factor authentication, you can help to protect your users' data and prevent fraud.

Usability: The login page should be easy to use and understand. Users should be able to log in quickly and easily, without having to spend time trying to figure out how to use the page.

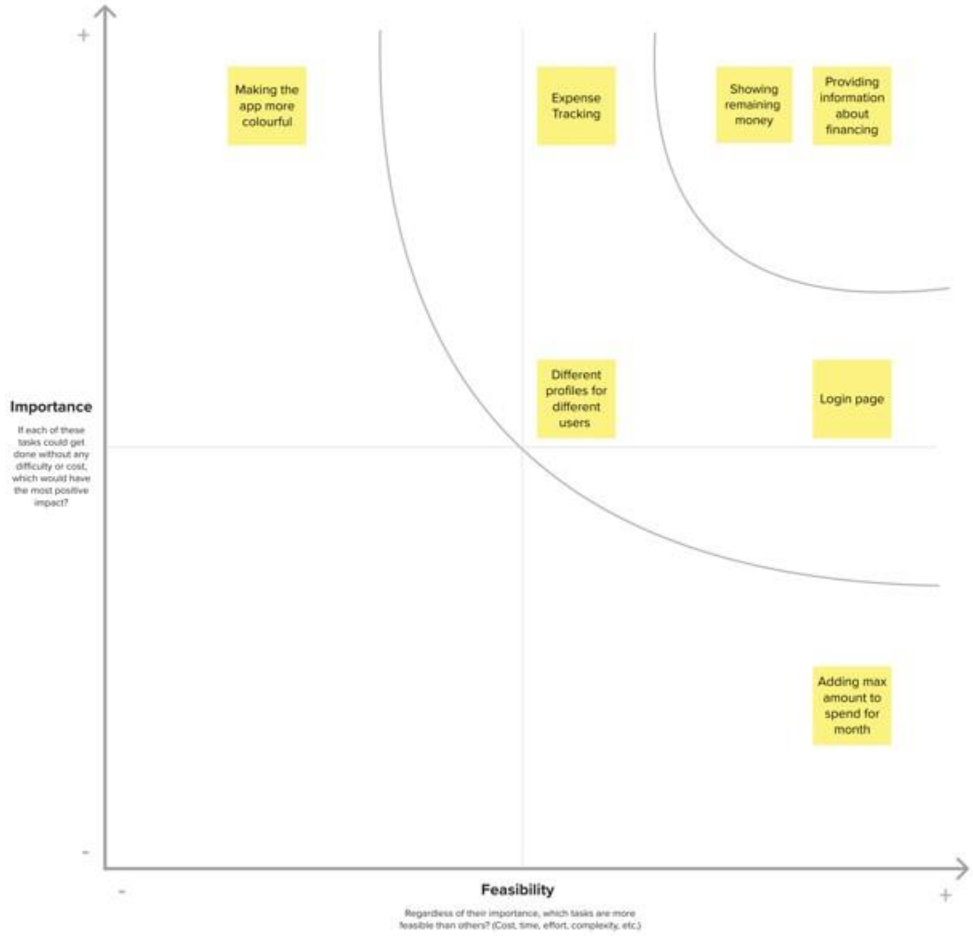
Branding: The login page is a great opportunity to promote your brand and create a positive user experience. By using a consistent design and messaging, you can help users to feel comfortable and confident using your app.

Providing information about financing

Helps them to make informed financial decisions: By understanding the different types of financing available and the pros and cons of each option, users can choose the best financing solution for their needs. This can help them to avoid making costly financial mistakes.

Helps them to achieve their financial goals: Financing can help users to achieve a variety of financial goals, such as saving for a down payment on a house, investing in retirement, or paying off debt. By providing information about financing, businesses and organizations can help users to reach their financial goals faster.

Helps them to save money: By comparing interest rates and fees, users can find the most affordable financing option. This can help them to save money on their monthly payments and overall financing costs.


+
Making the app more colourful
Expense Tracking
Showing remaining money
Providing information about financing
Different profiles for different users
Login page
Importance
If each of these tasks could get done without any difficulty or cost, which would have the most positive impact?
Adding max amount to spend for month
-
-
Feasibility
Regardless of their importance, which tasks are more feasible than others? (Cost, time, effort, complexity, etc.)
+

REQUIREMENT ANALYSIS:

Functional requirement :

Functional requirement is to develop a user-friendly budgeting tool that allows individuals to track their income and expenses, set financial goals, and receive personalized recommendations for saving and spending. The tool should also provide visualizations and reports to help users understand their financial health and progress. Additionally, it should have the ability to sync with bank accounts and categorize transactions automatically for convenience. Some of the important requirements will be

1.Integration with financial institutions: The app should have the capability to securely connect with users' bank accounts, credit cards, and other financial institutions to automatically import transactions and account balances.

2. Goal tracking and reminders: The app should allow users to set financial goals, such as saving for a vacation or paying off debt, and provide reminders and progress tracking to help them stay on track.

3. Expense categorization and analysis: The app should automatically categorize expenses based on transaction data and provide insights and analysis on spending patterns to help users identify areas where they can cut back or save more.

4. Bill payment reminders: The app should have a feature that reminds users of upcoming bill payments, helping them avoid late fees and stay organized.

5. Financial education resources: The app should provide educational content, such as articles, videos, or interactive modules, to help users improve their financial literacy and make informed financial decisions.

These are the functional requirements of the project.

Non-functional requirements:

1. Security: The app should prioritize the security and privacy of user data, implementing robust encryption and authentication measures to protect sensitive financial information.

2. Performance: The app should be responsive and perform efficiently, even when handling large amounts of financial data. It should load quickly, process transactions seamlessly, and provide a smooth user experience.

3. Accessibility: The app should be designed to be accessible to users with disabilities, following accessibility guidelines and providing features such as screen reader compatibility and adjustable font sizes.

4. Cross-platform compatibility: The app should be compatible with multiple platforms, such as iOS and Android, allowing users to access their financial information from various devices.

5. Scalability: The app should be designed to handle a growing user base and increasing data volume, ensuring that it can scale effectively without compromising performance or security.

These are the non-functional requirements.

PROJECT DESIGN:

Data Flow Diagrams & User Stories:

DFD for a financial management app Money Matters is :

Level 0 DFD:

This is the highest-level diagram, showing the overall system.

It consists of external entities, processes, data stores, and data flows.

It provides an overview of how data moves within the system.

Level 0 DFD:

Description of Components:

External Entities:

User: Represents the app's users who interact with the system through the user interface.

External Services: Represents any external services or APIs that the app interacts with, such as currency exchange rate services or financial news feeds.

Processes:

Input Data: Represents the process of collecting financial data from the user.

Business Logic: Represents the core logic responsible for financial calculations, expense tracking, and budget management.

Reporting: Generates financial reports and insights for the user.

Authentication: Manages user authentication and authorization.

Data Storage: Stores financial data securely.

Data Stores:

Database: Represents the data store where financial data like transactions, accounts, and expenses are stored.

Data Flows:

Arrows represent the flow of data between components.

Data flows from the "User" to "Input Data" when the user enters financial information.

"Input Data" sends data to "Business Logic" for processing.

"Business Logic" interacts with the "Database" for data storage and retrieval.

"Business Logic" generates data for "Reporting."

"User" can also request data from "Reporting."

"User" interacts with "Authentication" for secure access to the system.

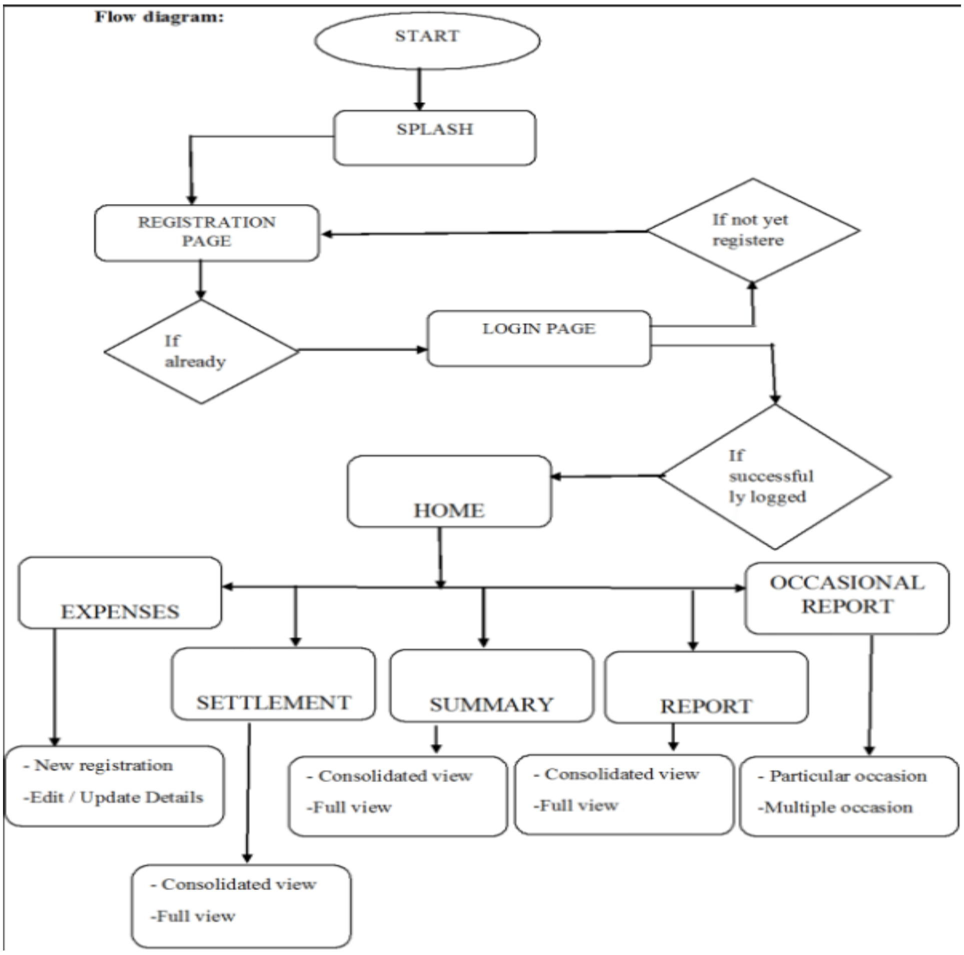
"External Services" can provide data to "Business Logic."

Level 1 DFD:

You can create detailed DFDs for each of the processes in the Level 0 diagram.

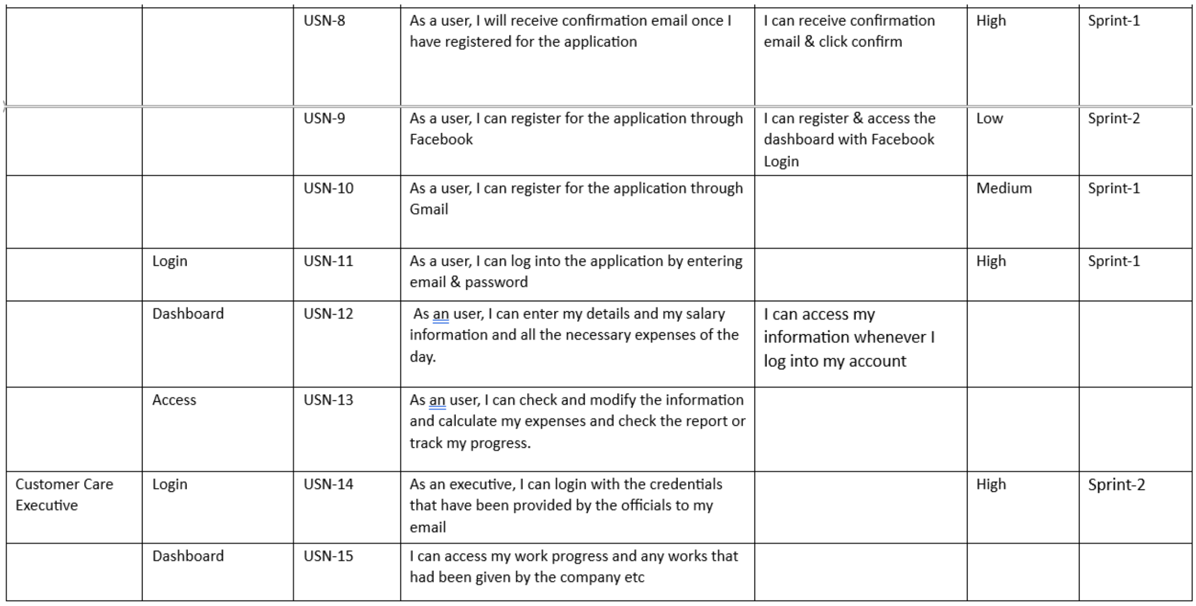
These diagrams provide a more in-depth view of how data flows within specific components.

Diagram :

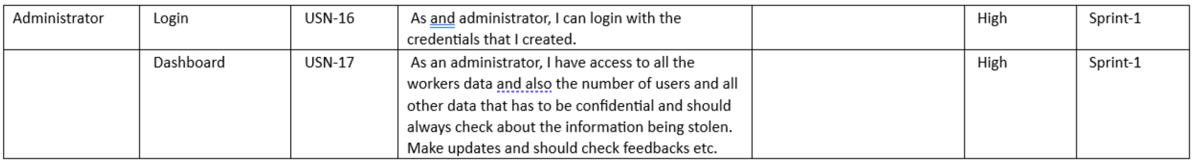


User Stories:

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Mobile user	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail		Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email & password		High	Sprint-1
	Dashboard	USN-6	As an user, I can enter my details and my salary information and all the necessary expenses of the day.	I can access my information whenever I log into my account		
	Access	USN-7	As an user, I can check and modify the information and calculate my expenses and check the report or track my progress.			
Customer (Web user)	Registration	USN-7	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1



		USN-8	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-9	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-10	As a user, I can register for the application through Gmail		Medium	Sprint-1
	Login	USN-11	As a user, I can log into the application by entering email & password		High	Sprint-1
	Dashboard	USN-12	As an user, I can enter my details and my salary information and all the necessary expenses of the day.	I can access my information whenever I log into my account		
	Access	USN-13	As an user, I can check and modify the information and calculate my expenses and check the report or track my progress.			
Customer Care Executive	Login	USN-14	As an executive, I can login with the credentials that have been provided by the officials to my email		High	Sprint-2
	Dashboard	USN-15	I can access my work progress and any works that had been given by the company etc			



Administrator	Login	USN-16	As and administrator, I can login with the credentials that I created.		High	Sprint-1
	Dashboard	USN-17	As an administrator, I have access to all the workers data and also the number of users and all other data that has to be confidential and should always check about the information being stolen. Make updates and should check feedbacks etc.		High	Sprint-1

Solution Architecture:

Solution architecture for the "Money Matters" project is:

(User Interface) Frontend

The main tool for creating Android applications is Android Studio.

Activities and Fragments: These parts are in charge of the navigation and user interface.

Define the visual organization of the app's screens using XML layouts.

Buttons, text boxes, and other widgets for user interaction make up UI components.

Server-side backend: Consider using a backend component if you wish to synchronize data between several devices, offer user accounts, or back up your data. This might be created using server-side frameworks like Node.js, Python, or other technologies.

Database: Use the SQLite database to store local financial information, such as transactions, account balances, and user preferences.

Firebase Realtime Database or Cloud Firestore: Firebase can be a fantastic option if you require real-time updates and cloud syncing.

User Identification: For managing user accounts, including sign-up, login, and password-reset features, utilize Firebase Authentication.

Commercial Logic: To handle financial calculations, transaction categorization, budget management, and other essential app functionality, create Java or Kotlin classes.

To represent financial entities such as transactions, accounts, and categories, implement data models.

Layer for Data Access: In order to communicate with the nearby SQLite database, create a data access layer. Methods for data retrieval, storage, and change will be included in this layer.

Network communication: If your application has a backend component, you'll need to create APIs and use tools like REST or GraphQL to communicate with the server from the mobile app.

Security: Encrypt critical information, including user credentials and financial transactions.

Securely manage user authentication.

Interaction between user interface and business logic:

To isolate UI from business logic, use the Model-View-Controller (MVC) or Model-View-ViewModel (MVVM) architectural patterns.

For a more seamless connection between UI components and underlying data, use data binding.

Testing: Perform unit testing on crucial elements like data access and financial calculations.

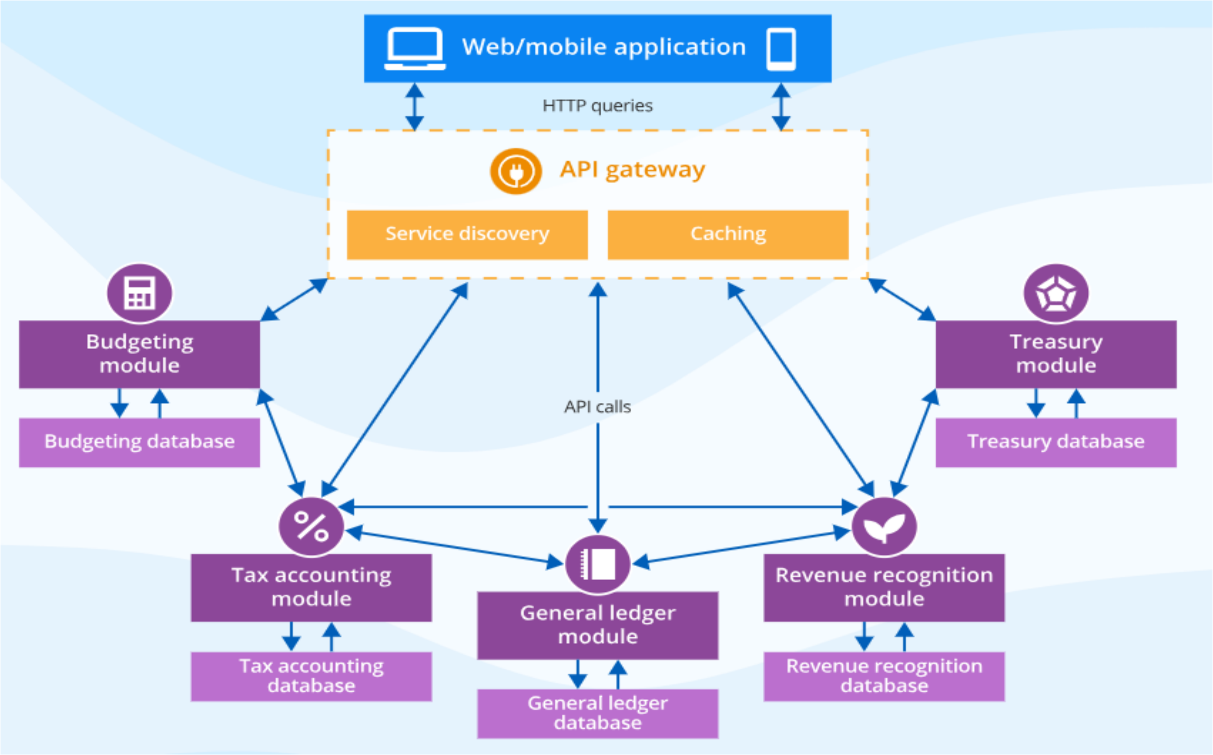
Test the user interface to make sure the app performs as planned.

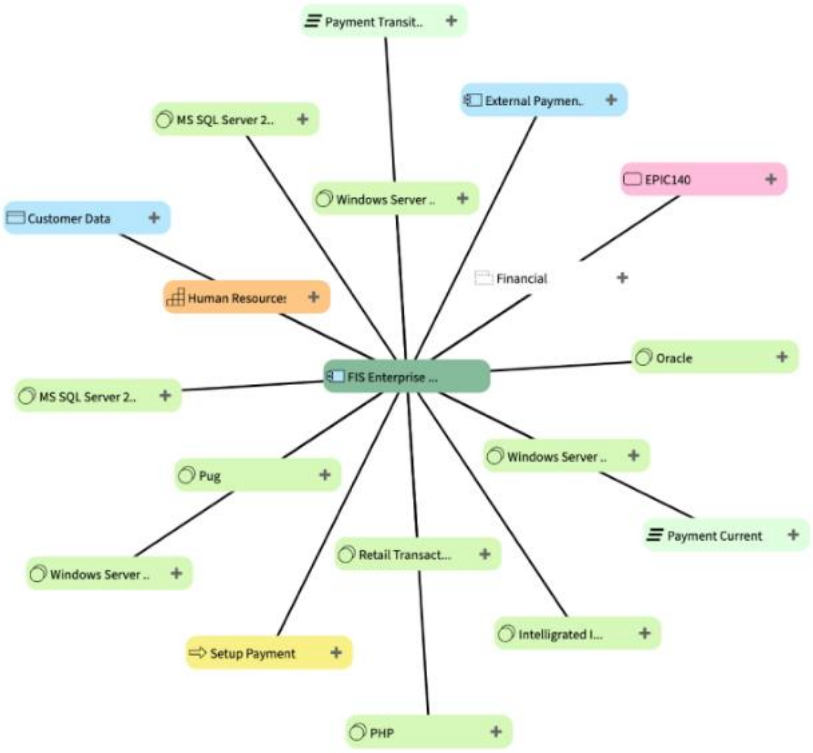
Think about frameworks for automated testing like Espresso.

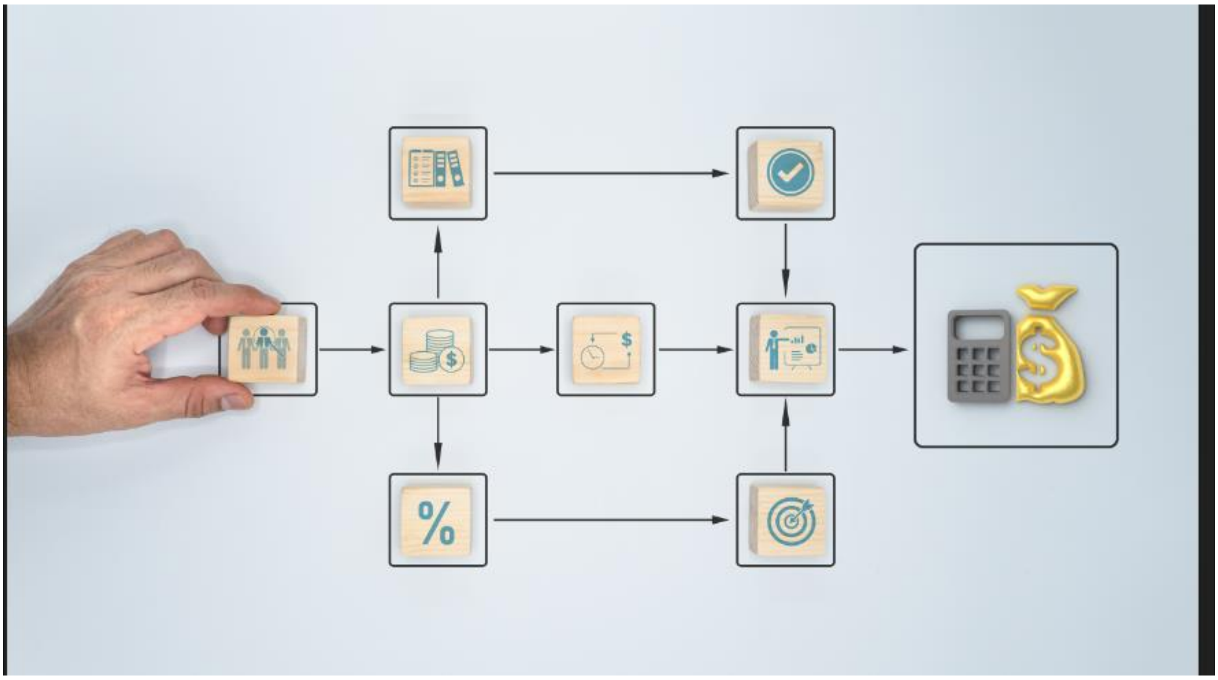
Documentation: Code, data models, and architecture documentation should be made for future collaboration and reference.

Deployment: Publish the Android app to the Google Play Store.

Solution Architecture Diagram:



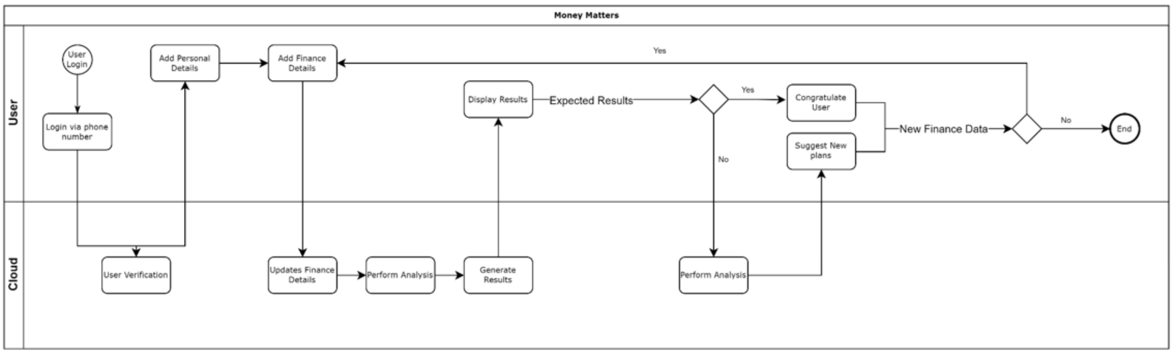




PROJECT PLANNING & SCHEDULING :

Technical Architecture:

The Deliverable shall include the architectural diagram as below and the information as per the table1 & table 2



Guidelines:

1. Include all the processes (As an application logic / Technology Block)
2. Provide infrastructural demarcation (Local / Cloud)
3. Indicate external interfaces (third party API's etc.)
4. Indicate Data Storage components / services
5. Indicate interface to machine learning models (if applicable)

Table-1 : Components & Technologies:

Sr. No.	Component	Description	Technology
1	User Interface	Mobile App	Kotlin Programming Language, Android Studio, Gradle, Android Jetpack, Firebase, Retrofit, Glide/Picasso, Koin/Dagger 2, Coroutines, JUnit and Mockito
2	Encryption Algorithm	Securing sensitive Data	RSA, AES
3	Data Mining Algorithm	Discover patterns, insights from collected Data	Apriori, K-Means
4	Graph Algorithm	Model relationships between different financial entities, such as transactions, expenses and income sources	Dijkstra's Algorithm, Breadth-First Search
5	Hashing Algorithm	Data integrity and security by generating unique hash codes for data sets	SHA-256, MD5
6	Database	Data type, Configurations, etc.	RDBMS like MySQL, PostgreSQL, or SQLite
7	Cloud Database	Database service on cloud	Amazon RDS, Google Cloud SQL, or Azure SQL Database
8	File Storage	File storage requirements	IBM Block Storage
9	External API - 1	Payment Gateway API (for transactions and payments)	Stripe API
10	External API - 2	SMS and Email API (for notifications and alerts)	Twilio API

11	Machine Learning Model	For learning spending, saving and target analysis and preparing plans	Regression Models, Classification Models, Clustering, etc
12	Infrastructure (Server/Cloud)	Application Deployment on Local System / Cloud Local Server Configuration, Cloud Server Configuration	Local, Cloud Foundry, Kubernetes, etc.

Table-2: Application Characteristics:

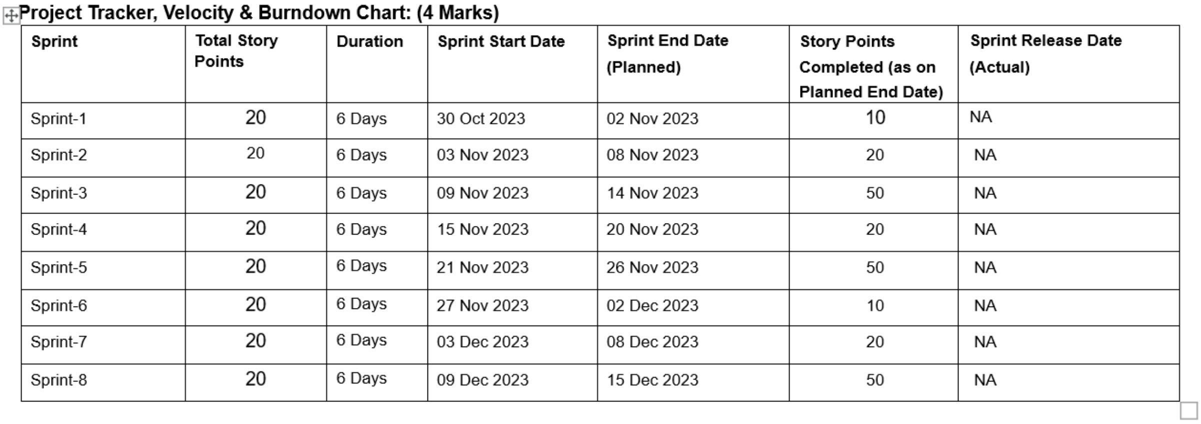
Sr. No.	Characteristics	Description	Technology
1	Open-Source Frameworks	Open-source frameworks allow for the development of the Money Matters app using community-supported tools, libraries, and frameworks, promoting transparency and collaboration within the development process	Kotlin programming language, Retrofit for networking, Koin or Dagger 2 for dependency injection, and Android Jetpack libraries
2	Security Implementations	Security implementations are crucial for protecting sensitive user financial data, ensuring data integrity, and preventing unauthorized access to user accounts and transaction details within the Money Matters app	Encryption algorithms like AES for data encryption, SSL/TLS for secure communication, Firebase Authentication for user authentication, and secure storage options such as encrypted databases and secure cloud storage services
3	Scalable Architecture	enables handling varying user loads and data processing requirements effectively, ensuring a seamless user experience even during periods of high traffic and increased data processing demands	Cloud-based infrastructure (e.g., AWS, GCP, Azure) for scalable compute resources, auto-scaling mechanisms for dynamic resource allocation, and cloud-based databases (e.g., DynamoDB, Firestore) for scalable and flexible data storage.
4	Availability	Availability is essential to ensure that the Money Matters app remains accessible to users without interruptions, minimizing downtime and service disruptions that could impact the user experience and financial management processes	oad balancing for distributing traffic, fault-tolerant design for ensuring continuous operation, redundant systems for high availability, and automated monitoring and alerting systems for proactive issue detection and resolution
5	Performance	Performance optimizations are crucial for ensuring that the Money Matters app delivers fast response times, smooth user interactions, and efficient data processing, enhancing the overall user experience and satisfaction	Caching mechanisms (e.g., Redis, Memcached) for improving data retrieval speeds, optimized database queries for efficient data retrieval, and use of asynchronous programming (e.g., Coroutines) for handling long-running tasks without blocking the main application thread

Sprint planning &Estimation:

Product Backlog, Sprint Schedule, and Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint - 1	User Authentication	USN-1	User can log in to the app using email and password	1	High	3
Sprint - 2	Dashboard Design	USN-2	Design a user-friendly dashboard displaying the user's financial overview	2	High	3
Sprint - 3	Expense Tracking	USN-3	Implement a feature to track and categorize expenses	5	High	3
Sprint - 4	Income Management	USN-4	Develop a module to manage user income sources and track earnings	2	Medium	2
Sprint - 5	Budget Planning	USN-5	Create a tool for setting and managing budgets for different categories	5	High	3
Sprint - 6	Transaction History	USN-6	Enable users to view detailed transaction history and filter transactions	1	Medium	2
Sprint - 7	Goal Setting	USN-7	Allow users to set financial goals and track progress	2	Medium	2
Sprint - 8	Data Security	USN-8	Implement enhanced security measures to protect user data	5	High	3

Sprint Delivery Schedule:



Project Tracker, Velocity & Burndown Chart: (4 Marks)

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	30 Oct 2023	02 Nov 2023	10	NA
Sprint-2	20	6 Days	03 Nov 2023	08 Nov 2023	20	NA
Sprint-3	20	6 Days	09 Nov 2023	14 Nov 2023	50	NA
Sprint-4	20	6 Days	15 Nov 2023	20 Nov 2023	20	NA
Sprint-5	20	6 Days	21 Nov 2023	26 Nov 2023	50	NA
Sprint-6	20	6 Days	27 Nov 2023	02 Dec 2023	10	NA
Sprint-7	20	6 Days	03 Dec 2023	08 Dec 2023	20	NA
Sprint-8	20	6 Days	09 Dec 2023	15 Dec 2023	50	NA

Velocity:

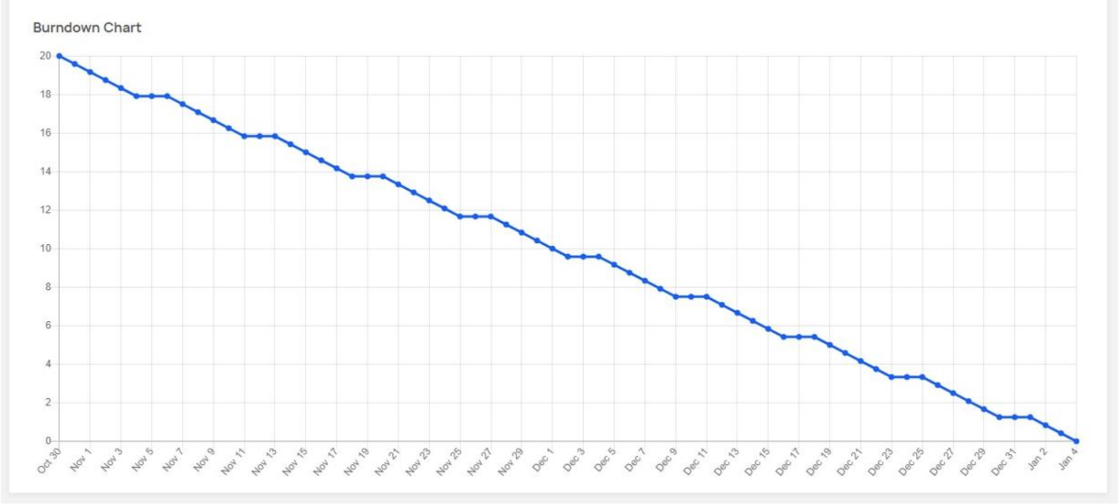
Average Story Points per sprint = 20

Average Sprint Duration = 6 We have a 8.25 sprint duration, and the velocity of the team is 35 (points per sprint). Let's calculate the team's average velocity (AV) per iteration unit (story points per day)

$$AV = \frac{Sprint\ Duration}{Velocity} = \frac{20}{6} = 3.33$$

Burndown Chart:

A burn down chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.



CODING & SOLUTIONING:

Features:
User credentials storage: User registration and authentication with secure features and storage.
Code:

```
class RegisterActivity : ComponentActivity() {
    private lateinit var databaseHelper: UserDatabaseHelper
    @OptIn(ExperimentalMaterial3Api::class)
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        databaseHelper = UserDatabaseHelper(this)
        setContent {
            MyApplicationTheme {
                    // A surface container using the 'background' color from
the theme
                    Surface(
                        modifier = Modifier.fillMaxSize(),
                        color = Color.White
                    ) {

                        RegistrationScreen(this, databaseHelper)
                    }
            }
        }
    }
}
@ExperimentalMaterial3Api
@Composable
fun RegistrationScreen(context: Context, databaseHelper:
UserDatabaseHelper) {

    Image(
        painterResource(id = R.drawable.img_1), contentDescription =
"",
        alpha =0.3F,
        contentScale = ContentScale.FillHeight,

        )

    var username by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var email by remember { mutableStateOf("") }
    var error by remember { mutableStateOf("") }

    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {

        Text(
            fontSize = 36.sp,
            fontWeight = FontWeight.ExtraBold,
            color = Color.White,
            text = "Register"
        )

        Spacer(modifier = Modifier.height(10.dp))
        TextField(
            value = username,
            onValueChange = { username = it },
            label = { Text("Username") },
            modifier = Modifier
                .padding(10.dp)
                .width(280.dp)

        )

        TextField(
            value = email,
            onValueChange = { email = it },
            label = { Text("Email") },
            modifier = Modifier
                .padding(10.dp)
                .width(280.dp)
        )

        TextField(
            value = password,
            onValueChange = { password = it },
            label = { Text("Password") },
            modifier = Modifier
                .padding(10.dp)
                .width(280.dp),
            visualTransformation = PasswordVisualTransformation()
        )


        if (error.isNotEmpty()) {
            Text(
                text = error,
                color = Color.Red,
                modifier = Modifier.padding(vertical = 16.dp)
            )
        }
```

```
        Button(
            onClick = {
                if (username.isNotEmpty() && password.isNotEmpty() &&
email.isNotEmpty()) {
                    val user = User(
                        id = null,
                        firstName = username,
                        lastName = null,
                        email = email,
                        password = password
                    )
                    databaseHelper.insertUser(user)
                    error = "User registered successfully"
                    // Start LoginActivity using the current context
                    context.startActivity(
                        Intent(
                            context,
                            LoginActivity::class.java
                        )
                    )

                } else {
                    error = "Please fill all fields"
                }
            },
            modifier = Modifier.padding(top = 16.dp)
        ) {
            Text(text = "Register")
        }
        Spacer(modifier = Modifier.width(10.dp))
        Spacer(modifier = Modifier.height(10.dp))

        Row() {
            Text(
                modifier = Modifier.padding(top = 14.dp), text = "Have
an account?"
            )
            TextButton(onClick = {
                context.startActivity(
                    Intent(
                        context,
                        LoginActivity::class.java
                    )
                )
            })

            {
                Spacer(modifier = Modifier.width(10.dp))
                Text(text = "Log in")
            }
        }
    }
}
private fun startLoginActivity(context: Context) {
    val intent = Intent(context, LoginActivity::class.java)
    ContextCompat.startActivity(context, intent, null)
}
```

2 . Expense Tracking: Add, edit, and categorize expenses.View spending trends and patterns.

Code:

```
class MainActivity : ComponentActivity() {
    @OptIn(ExperimentalMaterial3Api::class)
    @SuppressLint("UnusedMaterialScaffoldPaddingParameter",
        "UnusedMaterial3ScaffoldPaddingParameter"
    )
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Scaffold(
                // in scaffold we are specifying top bar.
                bottomBar = {
                    // inside top bar we are specifying
                    // background color.
                    BottomAppBar(
                        modifier = Modifier.height(80.dp),
                        // along with that we are specifying
                        // title for our top bar.
                        content = {

                            Spacer(modifier = Modifier.width(15.dp))

                            Button(
                                onClick =
{startActivity(Intent(applicationContext,AddExpensesActivity::class.java
))},
                                modifier = Modifier.size(height = 55.dp, width
= 110.dp).background(
                                    Color.White)
                            )
                            {
                                Text(
                                    text = "Add Expenses", color = Color.Black,
fontSize = 14.sp,
                                    textAlign = TextAlign.Center
                                )
                            }

                            Spacer(modifier = Modifier.width(15.dp))

                            Button(
                                onClick = {
                                    startActivity(
                                        Intent(
                                            applicationContext,
                                            SetLimitActivity::class.java
                                        )
                                    )
                                },
                                modifier = Modifier.size(height = 55.dp, width
= 110.dp).background(
                                    Color.White)
                            )
                            {
                                Text(
                                    text = "Set Limit", color = Color.Black,
fontSize = 14.sp,
                                    textAlign = TextAlign.Center
                                )
                            }

                            Spacer(modifier = Modifier.width(15.dp))

                            Button(
                                onClick = {
                                    startActivity(
                                        Intent(
                                            applicationContext,
                                            ViewRecordsActivity::class.java
                                        )
                                    )
                                },
                                modifier = Modifier.size(height = 55.dp, width
= 110.dp).background(
                                    Color.White
                                )
                            )
                            {
                                Text(
                                    text = "View Records", color = Color.Black,
fontSize = 14.sp,
                                    textAlign = TextAlign.Center
                                )
                            }

                        }
                    )
                }
```

```
            ) {
                MainPage()
            }
        }
    }
}

@Composable
fun MainPage() {
    Column(
        modifier = Modifier.padding(20.dp).fillMaxSize(),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {

        Text(text = "Welcome To Expense Tracker", fontSize = 42.sp,
fontWeight = FontWeight.Bold,
            textAlign = TextAlign.Center)
    }
}
```

3. modifying expenses : The expenses and budgets will be calculated if there is any additional submission of money.
Code:

```
class SetLimitActivity : ComponentActivity() {
    private lateinit var expenseDatabaseHelper:
ExpenseDatabaseHelper
    @OptIn(ExperimentalMaterial3Api::class)
    @SuppressLint("UnusedMaterialScaffoldPaddingParameter",
        "UnusedMaterial3ScaffoldPaddingParameter"
    )
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        expenseDatabaseHelper = ExpenseDatabaseHelper(this)
        setContent {
            Scaffold(
                // in scaffold we are specifying top bar.
                bottomBar = {

                    // inside top bar we are specifying
                    // background color.
                    BottomAppBar(
                        modifier = Modifier.height(80.dp),
                        // along with that we are specifying
                        // title for our top bar.
                        content = {

                            Spacer(modifier = Modifier.width(15.dp))

                            Button(
                                onClick = {
                                    startActivity(
                                        Intent(
                                            applicationContext,
                                            AddExpensesActivity::class.java
                                        )
                                    )
                                },
                                modifier = Modifier.size(height = 55.dp, width
= 110.dp).background(
                                    Color.White)
                            )
                            {
                                Text(
                                    text = "Add Expenses", color = Color.Black,
fontSize = 14.sp,
                                    textAlign = TextAlign.Center
                                )
                            }

                            Spacer(modifier = Modifier.width(15.dp))

                            Button(
                                onClick = {
                                    startActivity(
                                        Intent(
                                            applicationContext,
                                            SetLimitActivity::class.java
                                        )
                                    )
                                },
                                modifier = Modifier.size(height = 55.dp, width
= 110.dp).background(
                                    Color.White
                                )
                            )
                            {
                                Text(
                                    text = "Set Limit", color = Color.Black,
fontSize = 14.sp,
                                    textAlign = TextAlign.Center
                                )
                            }

                            Spacer(modifier = Modifier.width(15.dp))

                            Button(
                                onClick = {
                                    startActivity(
                                        Intent(
                                            applicationContext,
                                            ViewRecordsActivity::class.java
                                        )
                                    )
                                },
                                modifier = Modifier.size(height = 55.dp, width
= 110.dp).background(
                                    Color.White)
                            )
                            {
                                Text(
                                    text = "View Records", color = Color.Black,
fontSize = 14.sp,
                                    textAlign = TextAlign.Center
                                )
                            }

                        }
                    )
                }
```

4. Balance tracker : We can be able to keep track on the expenses and the amount remaining and so on.

Code:

```
@ExperimentalMaterial3Api
@Composable
fun Limit(context: Context, expenseDatabaseHelper:
ExpenseDatabaseHelper, expense: List<Expense>) {
    Column(
        modifier = Modifier
            .padding(top = 100.dp, start = 30.dp)
            .fillMaxHeight()
            .fillMaxWidth(),
        horizontalAlignment = Alignment.Start
    ) {

        var amount by remember { mutableStateOf("") }
        var error by remember { mutableStateOf("") }

        Text(text = "Monthly Amount Limit", fontWeight =
FontWeight.Bold, fontSize = 20.sp)
        Spacer(modifier = Modifier.height(10.dp))
        TextField(value = amount, onValueChange = { amount = it },
            label = { Text(text = "Set Amount Limit ") })

        Spacer(modifier = Modifier.height(20.dp))
```

```
if (error.isNotEmpty()) {
    Text(
        text = error,
        color = Color.Red,
        modifier = Modifier.padding(vertical = 16.dp)
    )
}

Button(onClick = {
    if (amount.isNotEmpty()) {
        val expense = Expense(
            id = null,
            amount = amount
        )
        expenseDatabaseHelper.insertExpense(expense)
    }
}) {
    Text(text = "Set Limit")
}

Spacer(modifier = Modifier.height(10.dp))

LazyRow(
    modifier = Modifier
        .fillMaxSize()
        .padding(top = 0.dp),

    horizontalArrangement = Arrangement.Start
) {
    item {

        LazyColumn {
            items(expense) { expense ->
                Column(

                ) {
```

```
                            Text("Remaining Amount: ${expense.amount}",
fontWeight = FontWeight.Bold)
                        }
                    }
                }
            }
```

5. View Records : Detailed history of all financial transactions. Search and filter transactions.

Code:

```
@Composable
fun Records(items: List<Items>) {
    Text(text = "View Records", modifier = Modifier.padding(top =
24.dp, start = 106.dp, bottom = 24.dp ), fontSize = 30.sp,
fontWeight = FontWeight.Bold)
    Spacer(modifier = Modifier.height(30.dp))
    LazyRow(
        modifier = Modifier
            .fillMaxSize()
            .padding(top = 80.dp),

        horizontalArrangement = Arrangement.SpaceBetween
    ){
        item {

            LazyColumn {
                items(items) { items ->
                    Column(modifier = Modifier.padding(top = 16.dp,
start = 48.dp, bottom = 20.dp)) {
                        Text("Item_Name: ${items.itemName}")
                        Text("Quantity: ${items.quantity}")
                        Text("Cost: ${items.cost}")
                    }
                }
            }
        }
    }
}
```

6. Security Features: Encryption of sensitive financial data.Biometric authentication options

Code :

```
      TextField(
            value = username,
            onValueChange = { username = it },
            label = { Text("Username") },
            modifier = Modifier.padding(10.dp)
                .width(280.dp)
        )

        TextField(
            value = password,
            onValueChange = { password = it },
            label = { Text("Password") },
            modifier = Modifier.padding(10.dp)
                .width(280.dp),
            visualTransformation = PasswordVisualTransformation()

        )

        if (error.isNotEmpty()) {
            Text(
                text = error,
                color = Color.Red,
                modifier = Modifier.padding(vertical = 16.dp)
            )
        }

        Button(
            onClick = {
                if (username.isNotEmpty() && password.isNotEmpty()) {
                    val user =
databaseHelper.getUserByUsername(username)
```

```
            if (user != null && user.password == password) {
                error = "Successfully log in"
                context.startActivity(
                    Intent(
                        context,
                        MainActivity::class.java
                    )
                )
                //onLoginSuccess()
            }
            else {
                error =  "Invalid username or password"
            }

        } else {
            error = "Please fill all fields"
        }
    },
    modifier = Modifier.padding(top = 16.dp)
) {
    Text(text = "Login")
}
Row {
    TextButton(onClick = {context.startActivity(
        Intent(
            context,
            RegisterActivity::class.java
        )
    )}
    )
    { Text(color = Color.Black,text = "Sign up") }
    TextButton(onClick = {
    })

    {
        Spacer(modifier = Modifier.width(60.dp))
        Text(color = Color.Black,text = "Forget password?")
```

```
                }
            }
        }
    }
    private fun startMainPage(context: Context) {
        val intent = Intent(context, MainActivity::class.java)
        ContextCompat.startActivity(context, intent, null)
    }
```

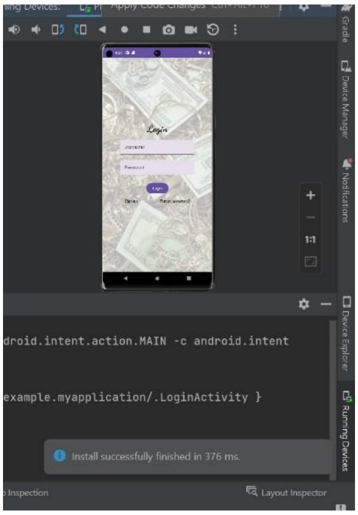
These are the features of the app that are included mainly and there are many more features that can help the user to have a good experience with the app.

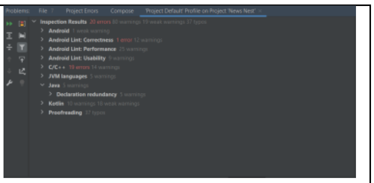
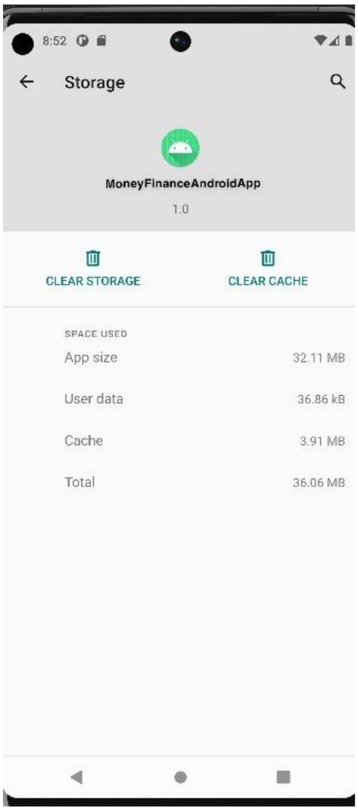
Database Scheme has been used for the monthly addition and for some security purposes.

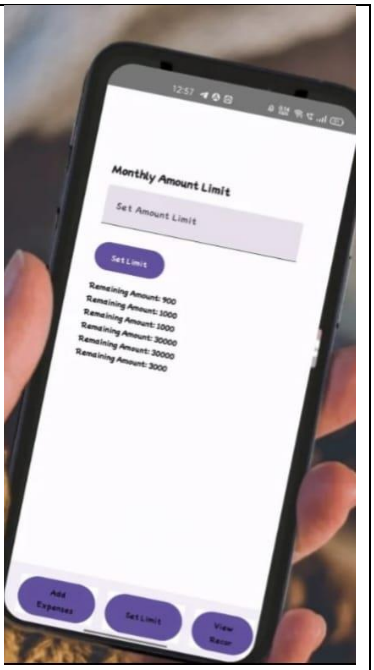
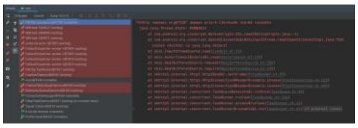
PERFORMANCE TESTING:

Performance Metrics:

Project team shall fill the following information in model performance testing template.

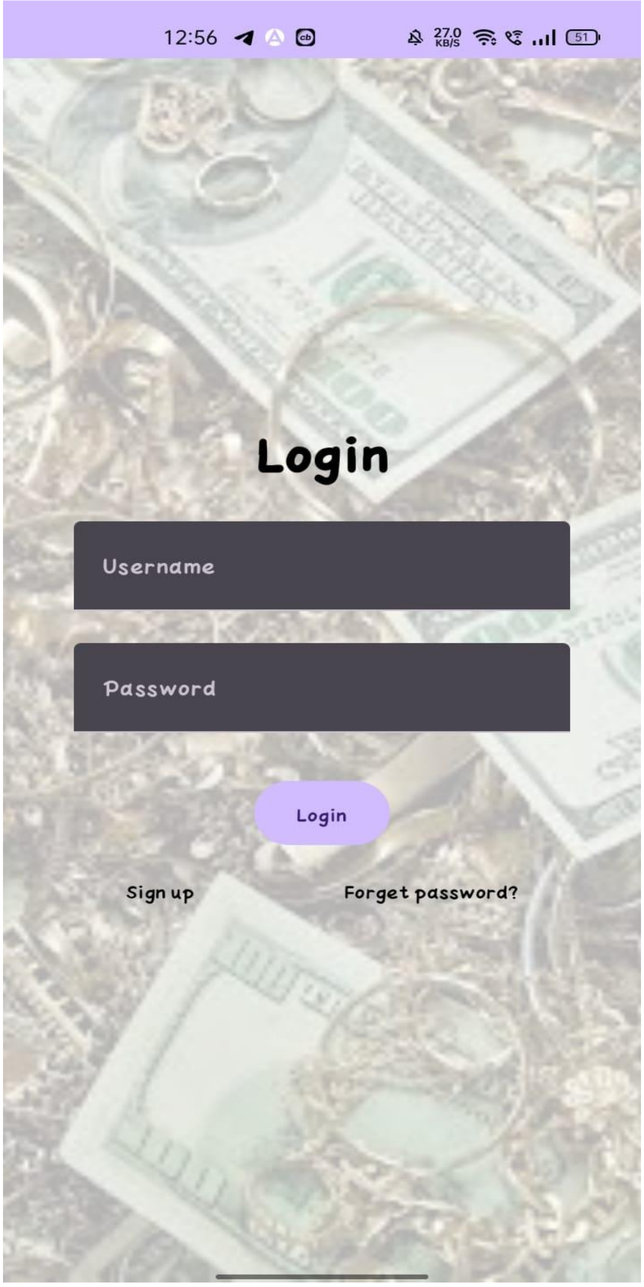
S.No.	Parameter	Values	Screenshot
1.	Metrics	App Launch Time- 375 ± 50 ms Screen Render Time- 100 ms Code Quality- 25 warnings	 droid.intent.action.MAIN -c android.intent example.myapplication/.LoginActivity } Install successfully finished in 376 ms. Login Gradle Device Manager Notifications Device Explorer Running Devices Layout Inspector Inspection

			 Problems: File 7 Project Errors Compose 'Project Default' Profile on Project 'News Nest' Inspection Results 20 errors 80 warnings 19 weak warnings 37 typos Android 1 weak warning Android Lint: Correctness 1 error 12 warnings Android Lint: Performance 25 warnings Android Lint: Usability 9 warnings C/C++ 19 errors 14 warnings JVM languages 5 warnings Java 5 warnings Declaration redundancy 5 warnings Kotlin 10 warnings 18 weak warnings Proofreading 37 typos
2.	Usage	App Size- 37.09 mb	 8:52 Storage MoneyFinanceAndroidApp 1.0 CLEAR STORAGE CLEAR CACHE SPACE USED App size 32.11 MB User data 36.86 kB Cache 3.91 MB Total 36.06 MB

		Customer Experience- 4.5/5	 Monthly Amount Limit Set Amount Limit Set Limit Remaining Amount: 900 Remaining Amount: 1000 Remaining Amount: 1000 Remaining Amount: 30000 Remaining Amount: 30000 Remaining Amount: 3000
3.	Performance	Error and Crash Rates- mid rate Database Query Performance- Good	

RESULTS:

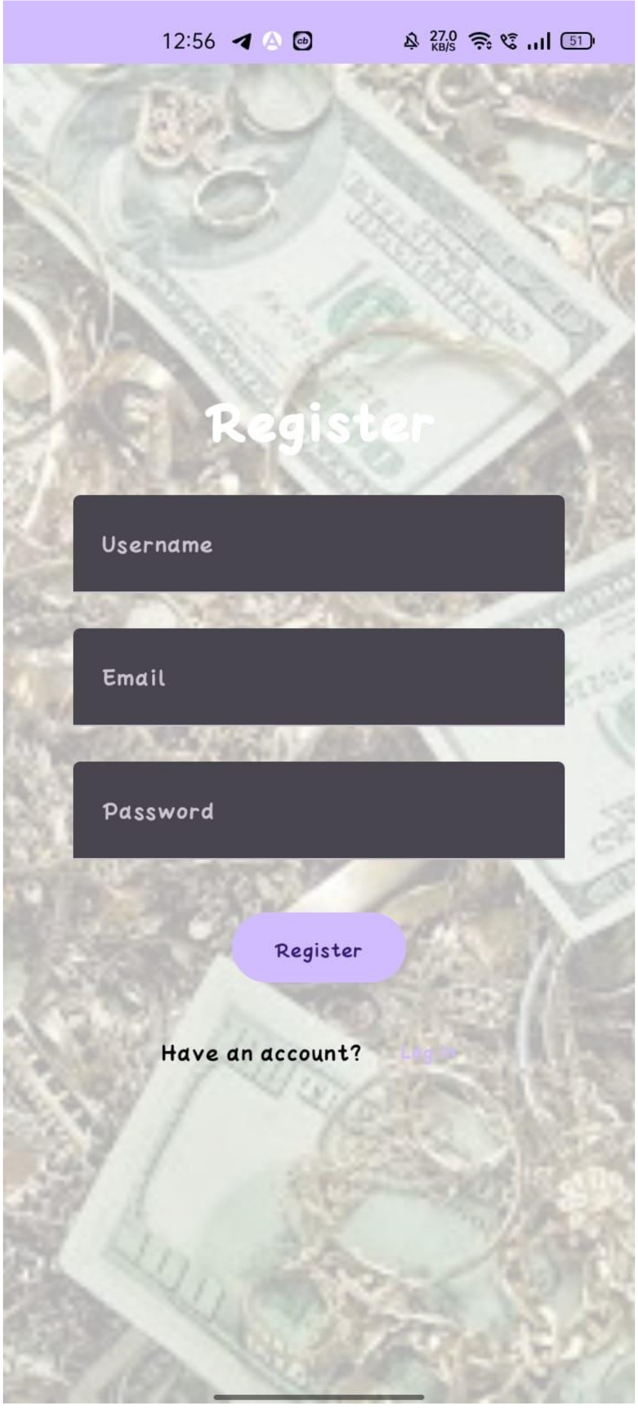
Output Screenshots:

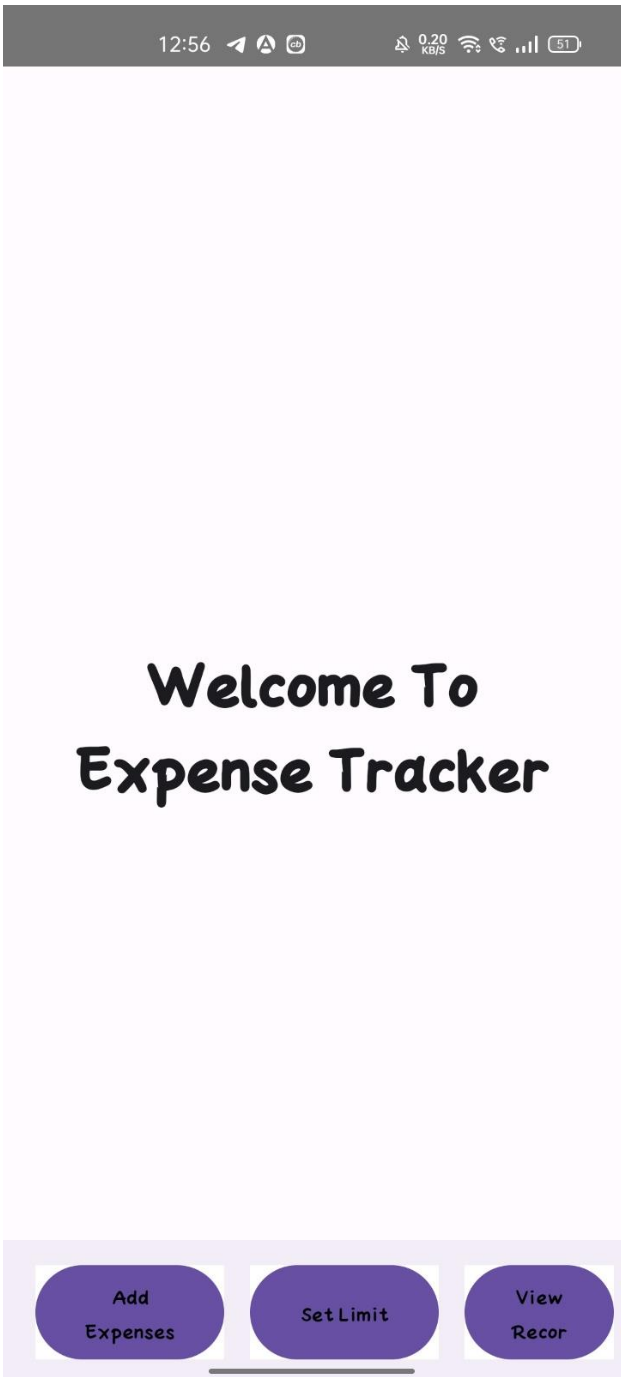


Login

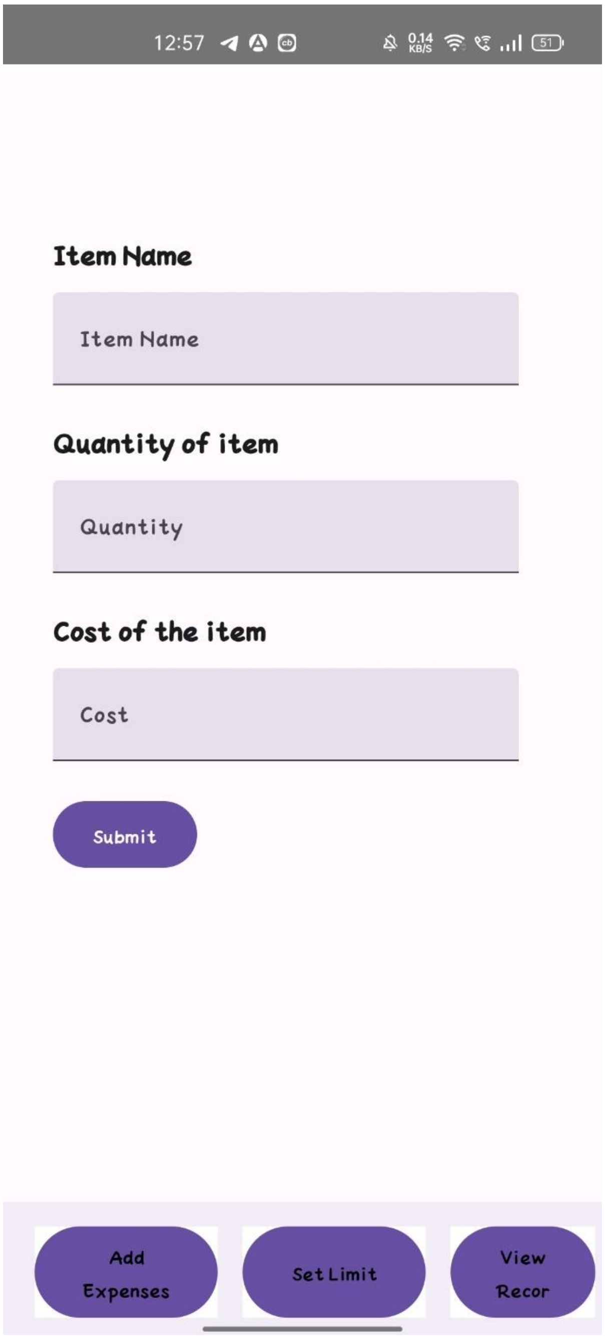
Username

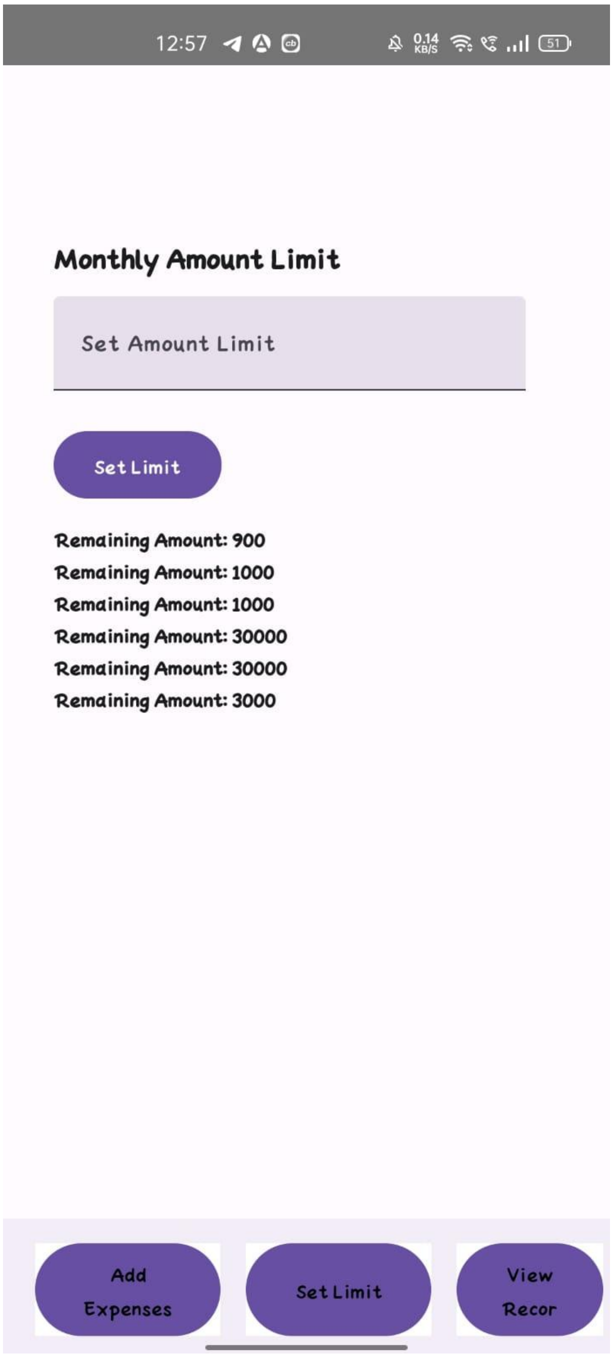
Password


12:56
27.0 KB/S
51
Register
Username
Email
Password
Register
Have an account?
Log in



Welcome To Expense Tracker

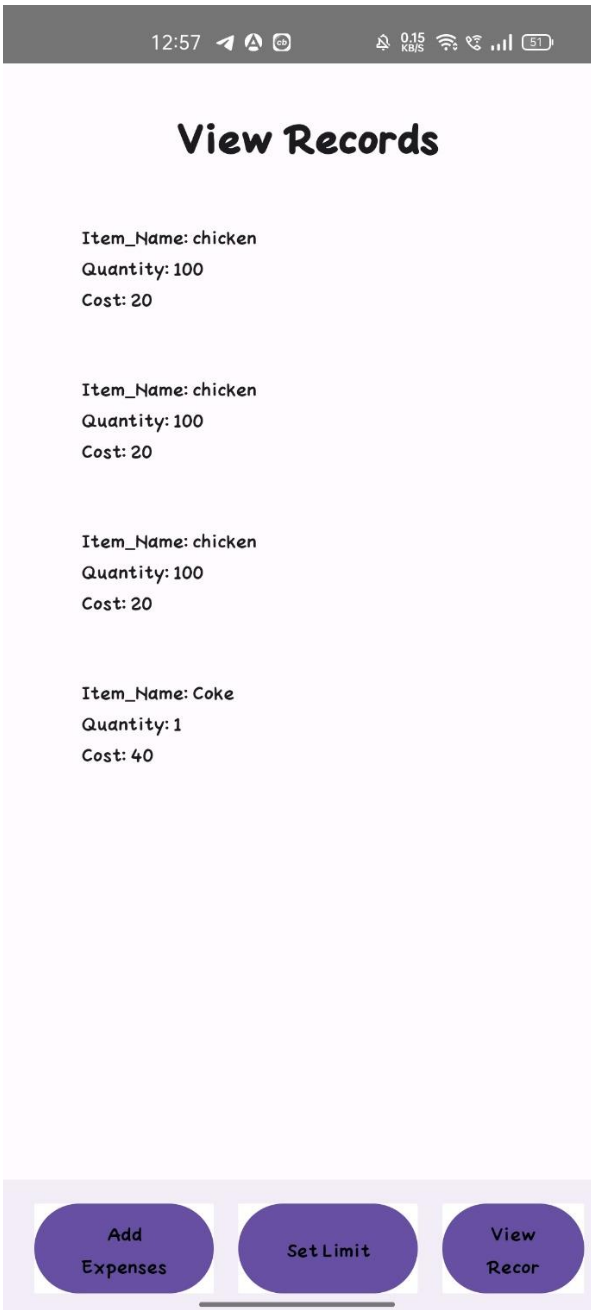

12:57
0.14 KB/S
51
Item Name
Item Name
Quantity of item
Quantity
Cost of the item
Cost
Submit
Add Expenses
Set Limit
View Recor



Monthly Amount Limit

Set Amount Limit

Remaining Amount: 900
Remaining Amount: 1000
Remaining Amount: 1000
Remaining Amount: 30000
Remaining Amount: 30000
Remaining Amount: 3000



View Records

Item_Name: chicken
Quantity: 100
Cost: 20

Item_Name: chicken
Quantity: 100
Cost: 20

Item_Name: chicken
Quantity: 100
Cost: 20

Item_Name: Coke
Quantity: 1
Cost: 40

ADVANTAGES & DISADVANTAGES :

Advantages:

1. Financial Awareness: The project helps users become more aware of their financial situation by providing tools to track income, expenses, and savings.
2. Budgeting and Planning: Users can set budgets and financial goals, helping them plan and manage their finances effectively.

3. Expense Tracking: The project allows users to track their expenses, categorize them, and identify areas where they can cut back or make adjustments.
4. Goal Achievement: With the goal tracker feature, users can set financial goals and track their progress, motivating them to achieve their targets.
5. Improved Saving Habits: By providing insights into spending patterns and offering budgeting tools, the project encourages users to develop better saving habits.
6. Financial Decision Making: Users can make informed financial decisions based on the data and insights provided by the project.
7. Importing Transactions: The ability to import transactions from various sources simplifies the process of managing and organizing financial data.
8. Accessibility and Convenience: The project can be accessed anytime, anywhere, making it convenient for users to stay on top of their finances.
9. Personalized Recommendations: The project can provide personalized recommendations based on users' financial data, helping them make smarter financial choices.
10. Financial Security: By promoting financial awareness and responsible money management, the project contributes to users' overall financial security and well-being.

Disadvantages:

1. Privacy Concerns: Users may have concerns about sharing their financial data with the project, especially if it's not adequately protected or if there's a risk of data breaches.
2. Technical Challenges: The project may face technical issues, such as bugs or glitches, which could impact the accuracy and reliability of the financial data and calculations.
3. Learning Curve: Some users may find it challenging to navigate and understand the features and functionalities of the project, especially if they are not familiar with financial management concepts
4. Dependency on Technology: The project relies on technology, and if there are any technical failures or disruptions, users may face

difficulties accessing their financial information or using the project's features.
5. Limited Customization: The project may not offer enough flexibility or customization options to cater to individual financial needs and preferences.
6. Data Accuracy: The accuracy of the financial data depends on the user's input and the integration with external sources, which may not always be completely accurate or up to date.
7. Cost: Depending on the project's pricing model, there may be costs associated with accessing certain features or services, which could be a disadvantage for some users.
8. Reliance on Internet Connectivity: Users need a stable internet connection to access and use the project, which could be a limitation in areas with poor connectivity.

CONCLUSION:

The Money Matters project offers numerous advantages such as financial awareness, budgeting tools, expense tracking, goal achievement, improved saving habits, personalized recommendations, and convenience. However, it's important to consider potential disadvantages like privacy concerns, technical challenges, a learning curve, dependency on technology, limited customization, data accuracy, cost, and reliance on internet connectivity. Overall, the project can be a valuable tool for managing finances, but users should weigh the pros and cons based on their individual needs and preferences.

FUTURE SCOPE:

The Money Matters has a bright future ahead! It could offer even more personalized recommendations based on individual financial goals and preferences. Integration with financial institutions would provide real-time updates on account balances and transactions. Gamification and rewards could make financial management more fun and motivating. Expanded educational resources would help users improve their financial literacy. Social features would allow users to connect and share tips. Integration with smart devices would provide easy access to financial information.

APPENDIX:

Source Code:
https://github.com/Suraj041203/MoneyFinanceAndroidApp/tree/main/app/src/main/java/com/example/myapplication

Github Link:
https://github.com/smartinternz02/SI-GuidedProject-587227-1696941582

Output video:
https://drive.google.com/file/d/1aA_3sNo7SkFsv9ymBYYIWyioV9sNBc9V/view?usp=drivesdk