Project Report Format

# INTRODUCTION

* 1. Project Overview:

The personal finance management app aims to provide users with a comprehensive solution for tracking expenses, managing budgets and setting savings goals. The app targets individuals seeking efficient financial management tools to achieve their financial goals.

* 1. Purpose:

The purpose of this project is to empower users with a user-friendly and feature-rich app that enhances their financial well-being through effective budgeting, expense tracking, and informed investment decisions.

# LITERATURE SURVEY

* 1. Existing problem:

Many individuals face challenges in managing their finances effectively, leading to overspending, lack of savings, and suboptimal investment decisions. Existing financial management apps often lack comprehensive features or user-friendly interfaces.

* 1. References:
* <https://moneyview.in/insights/best-personal-finance-management-apps-in-india>
* <https://www.moneycontrol.com/news/business/personal-finance/how-budgeting-apps-can-enhance-your-financial-management-11253761.html>
  1. Problem Statement Definition:

The lack of a user-friendly, all-in-one personal finance management app that seamlessly integrates expense tracking and budgeting poses a challenge for individuals striving for financial health.

# IDEATION & PROPOSED SOLUTION

* 1. Empathy Map Canvas:
* Users: Individuals seeking efficient financial management.
* Pain Points: Lack of a comprehensive solution, difficulty in tracking expenses.
* Gains: User-friendly interface, personalized financial insights.
  1. Ideation & Brainstorming:
* Comprehensive expense tracking.
* Budget management with real-time alerts.
* Savings goals customization.
* Integration with third-party financial APIs.
* Personalized investment advice.

Both Empathy map and brainstorming map is available in the git repository.

# REQUIREMENT ANALYSIS

* 1. Functional requirement:
* Expense tracking and categorization.
* Budget setting and monitoring.
* Savings goals creation and tracking.
  1. Non-Functional requirements:
* Secure user authentication (SHA-256, Firebase Authentication).
* Responsive and user-friendly UI.
* Scalable architecture (Microservices, Cloud Foundry).
* High availability (Load balancing, Redundant servers).
* Optimal performance (Caching, CDN integration).

# PROJECT DESIGN

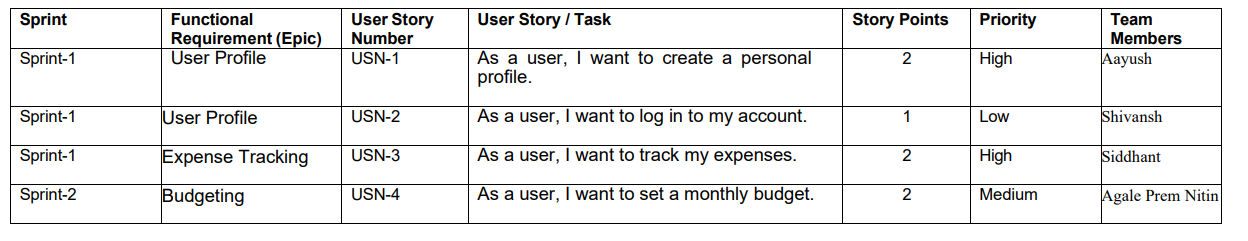
* 1. Data Flow Diagrams & User Stories:

Data flow diagram and user stories present in the git repository

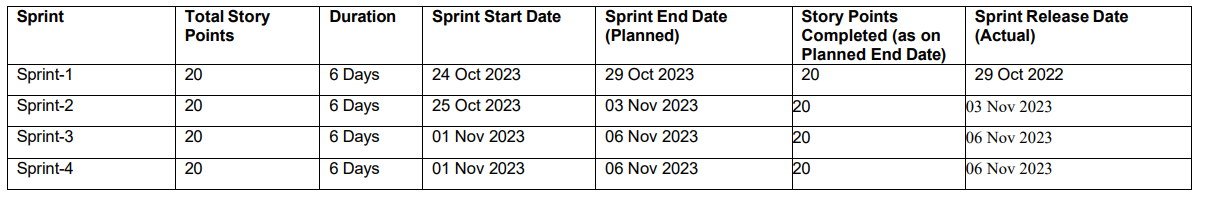
* 1. Solution Architecture:
* Client-side (Mobile App): Kotlin, XML (Android Studio).
* Server-side (Backend): Kotlin, Firebase, Firestore.
* Database: SQLite (Local Storage), Firebase Realtime Database.

# PROJECT PLANNING & SCHEDULING

* 1. Technical Architecture:
* Utilization of open-source frameworks (Kotlin, Retrofit, OkHttp).
* Security implementations (HTTPS, Firebase Authentication, Encryption).
* Scalable architecture with microservices.
  1. Sprint Planning & Estimation:



* 1. Sprint Delivery Schedule:



# CODING & SOLUTIONING (Explain the features added in the project along with code)

* 1. Feature 1: Expense Tracking

Code:

package com.babacode.walletexpensetracker.ui  
  
import android.Manifest  
import android.app.Activity  
import android.content.Intent  
import android.content.pm.PackageManager  
import android.net.Uri  
import android.os.Bundle  
import android.provider.Settings  
import androidx.activity.result.contract.ActivityResultContracts  
import androidx.appcompat.app.AppCompatActivity  
import androidx.core.content.ContextCompat  
import androidx.navigation.NavController  
import androidx.navigation.fragment.NavHostFragment  
  
import androidx.navigation.ui.setupActionBarWithNavController  
import com.babacode.walletexpensetracker.R  
import com.babacode.walletexpensetracker.databinding.ActivityMainBinding  
import com.babacode.walletexpensetracker.ui.setting.notification.AlarmUtils  
import com.babacode.walletexpensetracker.utiles.SettingUtils  
import com.google.android.material.snackbar.Snackbar  
import dagger.hilt.android.AndroidEntryPoint  
import java.util.\*  
  
  
@AndroidEntryPoint  
class MainActivity : AppCompatActivity() {  
  
 private lateinit var binding: ActivityMainBinding  
  
 private lateinit var navController: NavController  
  
 private val settingUtilsForNotification by lazy **{** SettingUtils(this)  
 **}** private val alarmUtils by lazy **{** AlarmUtils(applicationContext)  
 **}** override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 binding = ActivityMainBinding.inflate(layoutInflater)  
 val view = binding.root  
 setContentView(view)  
  
 val navHostFragment =  
 supportFragmentManager.findFragmentById(R.id.nav\_host\_fragment\_container) as NavHostFragment  
  
 navController = navHostFragment.navController  
  
  
 setupActionBarWithNavController(navController)  
  
  
 val toSetAlarm = settingUtilsForNotification.notificationForAlarm()  
  
 if (toSetAlarm) {  
 val calendar = Calendar.getInstance()  
 alarmUtils.initAlarmForNotification(calendar)  
 } else {  
 alarmUtils.cancelNotificationAlarm()  
 }  
  
 checkForNotificationPermission()  
  
 }  
  
 private fun checkForNotificationPermission() {  
 if (ContextCompat.checkSelfPermission(  
 this,  
 Manifest.permission.POST\_NOTIFICATIONS  
 ) == PackageManager.PERMISSION\_DENIED  
 ) {  
 notificationPermissionLauncher.launch(Manifest.permission.POST\_NOTIFICATIONS)  
 }  
 }  
  
 val notificationPermissionLauncher =  
 registerForActivityResult(ActivityResultContracts.RequestPermission()) **{** isGranted **->** if (!isGranted) {  
 showSettingSnackBar()  
 }  
 **}** private fun showSettingSnackBar() {  
 Snackbar.make(  
 binding.root,  
 R.string.notification\_permission\_text,  
 Snackbar.LENGTH\_LONG  
 ).setAction(R.string.open) **{** val intent = Intent(Settings.ACTION\_APPLICATION\_DETAILS\_SETTINGS)  
 intent.data = Uri.parse("package:$packageName")  
 startActivity(intent)  
 **}**.show()  
 }  
  
 override fun onSupportNavigateUp(): Boolean {  
 return navController.navigateUp() || super.onSupportNavigateUp()  
 }  
  
  
}  
  
  
const val ADD\_TRANSACTION\_RESULT\_OK = Activity.RESULT\_FIRST\_USER  
const val EDIT\_TRANSACTION\_RESULT\_OK = Activity.RESULT\_FIRST\_USER + 1

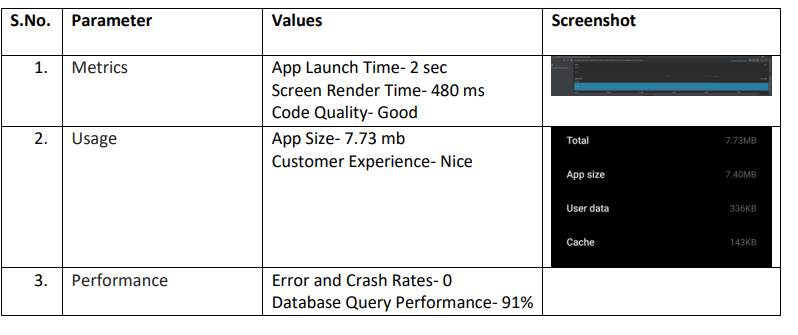
* 1. Feature 2: Calendar wise sorting

Code:

package com.babacode.walletexpensetracker.ui.calender  
  
import android.os.Bundle  
import android.view.View  
import androidx.fragment.app.Fragment  
import androidx.fragment.app.setFragmentResultListener  
import androidx.fragment.app.viewModels  
import androidx.navigation.fragment.findNavController  
import androidx.recyclerview.widget.LinearLayoutManager  
import com.babacode.walletexpensetracker.R  
import com.babacode.walletexpensetracker.data.model.Transaction  
import com.babacode.walletexpensetracker.databinding.FragmentTransactionCalenderViewBinding  
import com.babacode.walletexpensetracker.ui.*ADD\_TRANSACTION\_RESULT\_OK*import com.babacode.walletexpensetracker.ui.*EDIT\_TRANSACTION\_RESULT\_OK*import com.babacode.walletexpensetracker.ui.home.HomeAdepter  
import com.babacode.walletexpensetracker.utiles.Extra.*REQUEST\_KEY\_FOR\_ADD\_EDIT*import com.babacode.walletexpensetracker.utiles.Extra.convertCalenderDateToLong  
import com.babacode.walletexpensetracker.utiles.hide  
import com.babacode.walletexpensetracker.utiles.show  
import com.babacode.walletexpensetracker.utiles.showSnackBar  
import dagger.hilt.android.AndroidEntryPoint  
import java.util.\*  
  
  
@AndroidEntryPoint  
class TransactionCalenderViewFragment : Fragment(R.layout.*fragment\_transaction\_calender\_view*),  
 HomeAdepter.OnItemClick {  
  
  
 private var \_binding: FragmentTransactionCalenderViewBinding? = null  
 private val binding get() = \_binding!!  
 private lateinit var mAdepter: HomeAdepter  
 private val viewModel: CalenderViewViewModel by *viewModels*()  
 private var currentDayDate = Calendar.getInstance().*time* override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 \_binding = FragmentTransactionCalenderViewBinding.bind(view)  
 mAdepter = HomeAdepter(this)  
 setUpRecyclerView()  
 setUpObserver()  
  
 getTransactionForSelectedDate(currentDayDate)  
 binding.calendarView.*date* = convertCalenderDateToLong(currentDayDate)  
 binding.calendarView.setOnDateChangeListener **{** \_, year, month, dayOfMonth **->** val calender = Calendar.getInstance()  
 calender.set(Calendar.*YEAR*, year)  
 calender.set(Calendar.*MONTH*, month)  
 calender.set(Calendar.*DAY\_OF\_MONTH*, dayOfMonth)  
 currentDayDate = calender.*time* getTransactionForSelectedDate(currentDayDate)  
  
 **}** *setFragmentResultListener*(*REQUEST\_KEY\_FOR\_ADD\_EDIT*) **{** \_, bundle **->** when (bundle.getInt(*REQUEST\_KEY\_FOR\_ADD\_EDIT*)) {  
 *ADD\_TRANSACTION\_RESULT\_OK* -> binding.*root*.*showSnackBar*(R.string.*transaction\_added*)  
 *EDIT\_TRANSACTION\_RESULT\_OK* -> binding.*root*.*showSnackBar*(R.string.*transaction\_update*)  
 }  
 **}** }  
  
 private fun showViewInCalenderFrag(transactionList: List<Transaction>) {  
 binding.*apply* **{** noTransactionTv.*hide*()  
 calenderRecyclerView.*show*()  
 **}** mAdepter.submitList(transactionList)  
  
 }  
  
 private fun hideViewInCalender() {  
 binding.noTransactionTv.*show*()  
 binding.calenderRecyclerView.*hide*()  
 }  
  
  
 private fun getTransactionForSelectedDate(date: Date) {  
 val newDate = convertCalenderDateToLong(date)  
 viewModel.getSelectedDateFromCalender(newDate)  
 }  
  
 override fun onDestroyView() {  
 binding.calenderRecyclerView.*adapter*= null  
 super.onDestroyView()  
 \_binding = null  
  
  
 }  
  
 private fun setUpObserver() {  
 viewModel.selectedDateTransaction.observe(*viewLifecycleOwner*) **{** transactionList **->** if (transactionList.*isNotEmpty*()) {  
  
 showViewInCalenderFrag(transactionList)  
  
 } else {  
  
 hideViewInCalender()  
 }  
  
 **}** }  
  
 private fun setUpRecyclerView() {  
 binding.calenderRecyclerView.*apply* **{** *layoutManager* = LinearLayoutManager(*context*)  
 *adapter* = mAdepter  
 **}** }  
  
  
 override fun onTransactionClick(transaction: Transaction) {  
 val action =  
 TransactionCalenderViewFragmentDirections.actionCalenderViewFragmentToAddTransactionFragment(  
 transaction,  
 getString(  
 R.string.*edit\_transaction\_title* )  
 )  
  
 *findNavController*().navigate(action)  
 }  
  
 override fun onLongPress(transaction: Transaction) {  
 val action =  
 TransactionCalenderViewFragmentDirections.actionGlobalDeleteTransaction(  
 transaction  
 )  
 *findNavController*().navigate(action)  
 }  
  
  
}

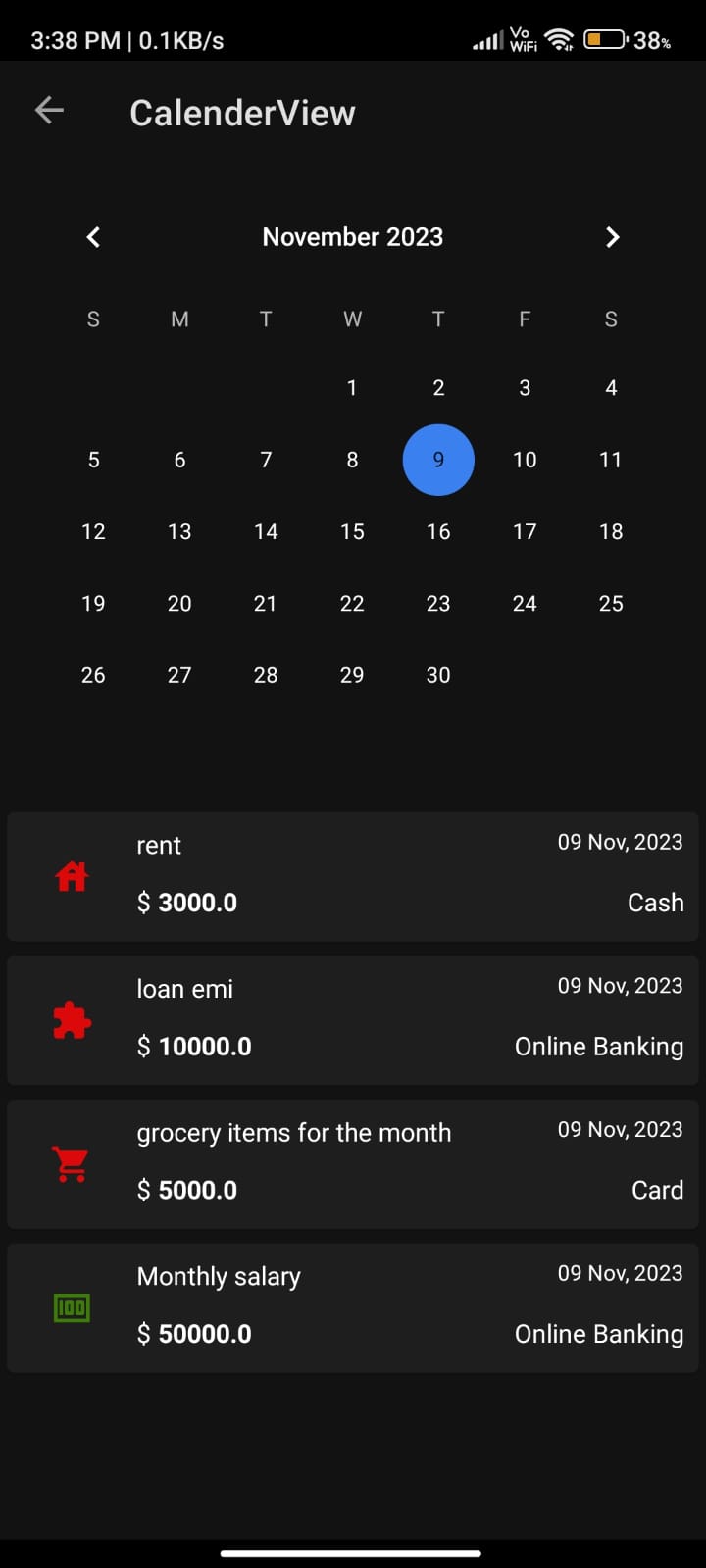
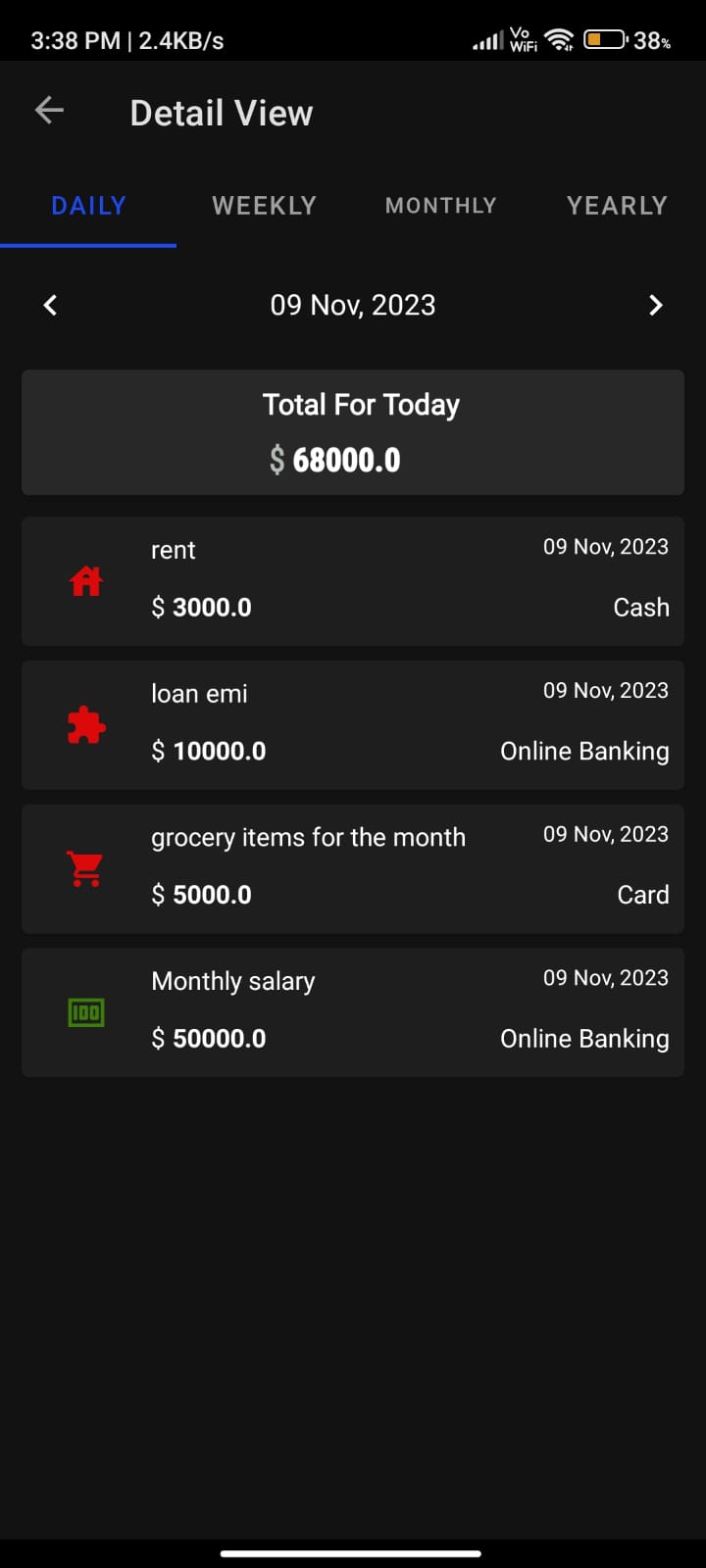
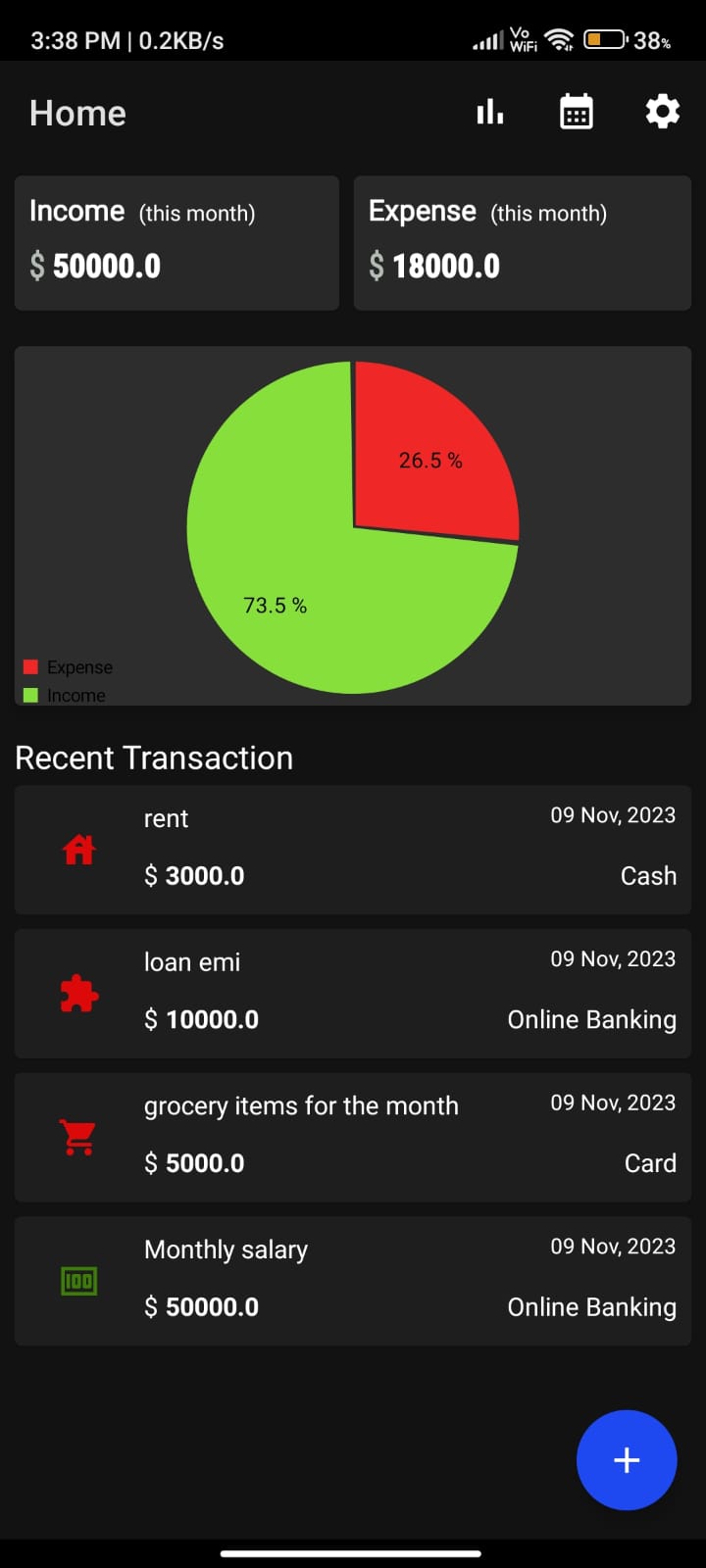
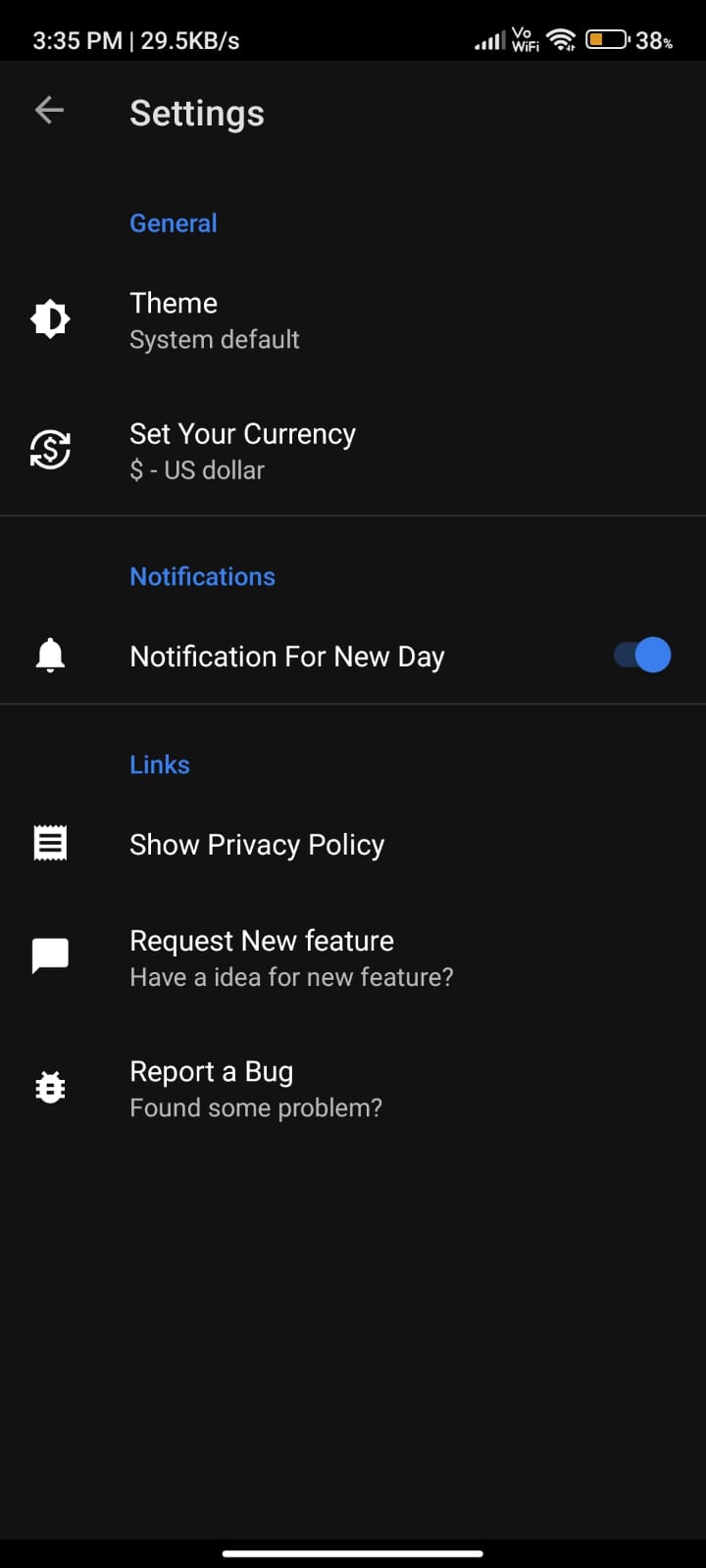
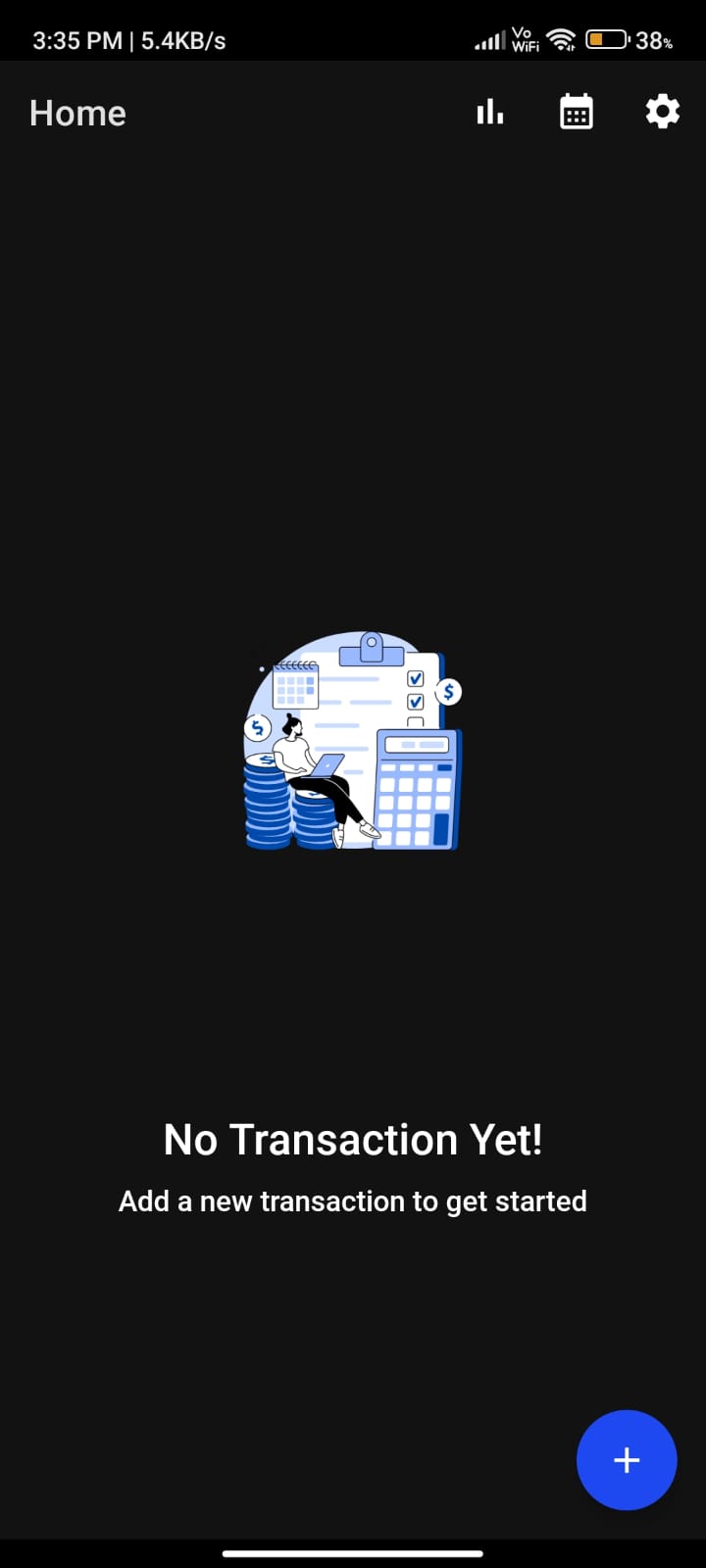
# PERFORMANCE TESTING

* 1. Performance Metrics:



# RESULTS

* 1. Output Screenshots:



# ADVANTAGES & DISADVANTAGES

# Advantages: Comprehensive financial management, user-friendly interface personalized advice.

# Disadvantages: Dependency on third-party APIs, potential security concerns (addressed through encryption and secure practices).

1. **CONCLUSION**

The personal finance management app addresses the existing challenges in financial management by providing a holistic solution. The app's features and robust architecture contribute to a user-friendly experience and effective financial planning.

# FUTURE SCOPE

# Integration of additional financial services.

# Enhanced machine learning models for more accurate investment advice.

# Multi-platform support (iOS, Web).

1. **APPENDIX**

Source Code: <https://github.com/smartinternz02/SI-GuidedProject-588234-1697653184/tree/main/Project%20Files>

GitHub Link: <https://github.com/smartinternz02/SI-GuidedProject-588234-1697653184>

Project Demo Link: <https://drive.google.com/file/d/1WxnqdehZ5-MBaFrBSWIPRalltEGKbAqw/view?usp=drivesdk>