LAPORAN TUGAS BESAR
PEMROGRAMAN MOBILE
APLIKASI PENCATATAN KEUANGAN
“MyCoin”

DISUSUN OLEH :

AHMAD JIDAN FAHRIZAL

1809075018

PROGRAM STUDI TEKNIK ELEKTRO
FAKULTAS TEKNIK
UNIVERSITAS MULAWARMAN

SAMARINDA
2021

DAFTAR ISI

Daftar Gambar

BAB I
PENDAHULUAN

1. 1 Latar Belakang

Pada umumnya kebanyakan orang membuat sebuah daftar list catatan keuangan mandiri masih dibuat dengan cara manual, yaitu mencatatkan daftar keuangan yang akan kita lakukan pada sebuah kertas berupa bentuk fisik. Mengapa penting membuat catatan keuangan mandiri? sesuatu yang besar biasanya harus dimulai dari kebiasaan kecil yang terlihat sederhana namun berdampak cukup banyak jika dilakukan dengan baik. Salah satu contohnya adalah kebiasaan membuat catatan keuangan mandiri untuk mengatur keuangan. Secara teori mungkin kita sudah tahu dan pernah melakukan hal ini, namun karena merasa sibuk dan terburu kadang kita lupa dan tidak disiplin dalam melakukannya. Selain itu anggapan bahwa hal ini tidak terlalu penting menjadi faktor pendukung untuk tidak membuat catatan keuangan tersebut. Namun jika kita melihat manfaat yang didapat dengan membuat catatan keuangan yang baik, mungkin kita bisa mempertimbangkan kembali untuk lebih rajin dan sadar akan pentingnya membuat catatan keuangan. Manfaat utama dari membuat catatan keuangan adalah membantu kita berpikir dalam mengeluarkan uang kita. Kita harus ingat jika pengeluaran keuangan kita melebihi pemasukan dan melihat untuk apa pengeluaran tersebut memang diperlukan atau tidak maka kita dapat berpikir bahwa pengeluaran apa saja yang diperlukan. Pengecekan waktu juga dilakukan secara manual sehingga menjadi kurang efektif terutama untuk banyak aktivitas yang telah terjadi bila melihat dengan kertas/buku tulis. Cara lain adalah dengan menggunakan perangkat lunak pada PC. Namun cara ini masih mempunyai kekurangan karna kita harus tetap berada pada PC tempat kita menyimpan task list tersebut untuk mengeceknya. Oleh karena itu dibuatlah sebuah aplikasi yang dapat menyediakan kemudahan pengelolaan keuangan secara mandiri seperti kemudahan menuliskan pada sebuah kertas yang dapat di akses lewat smartphone kita tanpa takut kehilangan bukti fisik catatan kita.

1. 2 Rumusan Masalah Manfaat

Untuk Penelitian ini akan dibatasi pada hal-hal berikut:

a. Memaparkan pembuatan sistem aplikasi Android yang bisa berjalan di sistem operasi Android.
b. Versi Android yang digunakan yaitu Android versi 4.0 (KitKat) sampai Android 11.1(R).
c. Aplikasi dapat berjalan dalam kondisi offline maupun online

1. 3 Tujuan Pembuatan Aplikasi Android

Tujuan dari penelitian ini adalah untuk merancang dan membuat aplikasi pencatatan keuangan secara mandiri yang dapat berjalan pada sistem operasi Android sehingga memberikan kemudahan bagi pemakainya dalam mengatur keuangan.

1. 4 Manfaat

a. Mempermudah user dalam mengakses aplikasi karena menggunakan system operasi android
b. Membantu user dalam membuat pencatatan keuangan secara mandiri.

BAB II
LANDASAN TEORI

2. 1 RecyclerView

RecyclerView adalah tampilan yang menggunakan arsitektur yang disederhanakan dengan UI Controller, Viewodel, dan LiveData. Menampilkan list atay grid data adalah salah satu tugas UI paling umum di Android. Daftar bervariasi dari yang sederhana hingga yang sangat kompleks. Daftar tampilan teks mungkin menampilkan data sederhana, seperti daftar belanja. Daftar yang kompleks, seperti daftar tujuan liburan yang beranotasi, dapat menunjukkan kepada pengguna banyak detail di dalam scrolling grid dengan header. Untuk mendukung semua kasus penggunaan ini, Android menyediakan widget RecyclerView. Manfaat terbesar dari RecyclerView adalah sangat efisien untuk daftar besar: Secara default, RecyclerView hanya berfungsi untuk memproses atau menggambar item yang saat ini terlihat di layer. Misalnya, jika list memiliki seribu elemen tetapi hanya 10 elemen yang terlihat, RecyclerView hanya berfungsi untuk menggambar 10 item di layer. Ketika pengguna melakukan scroll, RecyclerView mengetahui item baru apa yang seharusnya ada di layer dan tidak cukup berfungsi untuk menmpilkan item itu. Ketika suatu item scroll dari layer, tampilan item tersebut didaur ulang. Itu berarti item diisi dengan konten baru yng scroll ke layer. Perilaku RecyclerView ini menghemat banyak waktu pemrosesan dan membantu scroll list dengan lancar. Ketika suatu item berubah, alih-alih menggambar ulang seluruh daftar, RecyclerView dapat memperbarui satu item itu. Ini adalah keuntungan efisiensi yang sangat besar Ketika menampilkan daftar item kompleks.

2. 2 Floating Action Button

Floating Action Button (FAB) adalah salah satu komponen dari Material Desain. FAB merupakan sebuah tombol yang berbentuk lingkaran dan ditampilkan seperti melayang pada aplikasi android.

2. 3 SQL Database

Database SQLite adalah solusi penyimpanan yang baik jika anda memiliki data terstruktur yang perlu diakses dan disimpan secara persisten serta sering ditelusuri dan diubah. Anda juga bisa menggunakan SQLite sebagai media penyimpanan utama untuk data aplikasi atau pengguna, atau anda juga bisa menggunakannya untuk proses caching serta menyediakan data yang diambil dari cloud.

Jika anda bisa menyatakan data berupa baris dan kolom, pertimbangkan untuk memakai database SQLite. Jika anda menggunakan database SQLite, yang dinyatakan sebagai objek SQLiteDatabase adalah semua interaksi dengan database adalah melalui instance dari kelas SQLiteOpenHelper yang akan mengeksekusi permintaan dan pengelolaan database. Aplikasi anda hanya boleh berinteraksi dengan SQLiteOpenHelper, yang akan kita bahas bersama-sama dibawah ini.

Singktnya SQLite Database memiliki metode untuk membuat, menghapus, menjalankan perintah SQL, dan melakukan tugas manajemen database umum lainnya. seperti perintah CRUD (Create, Read, Update, Delete) data pada sebuah aplikasi catatan sederhana.

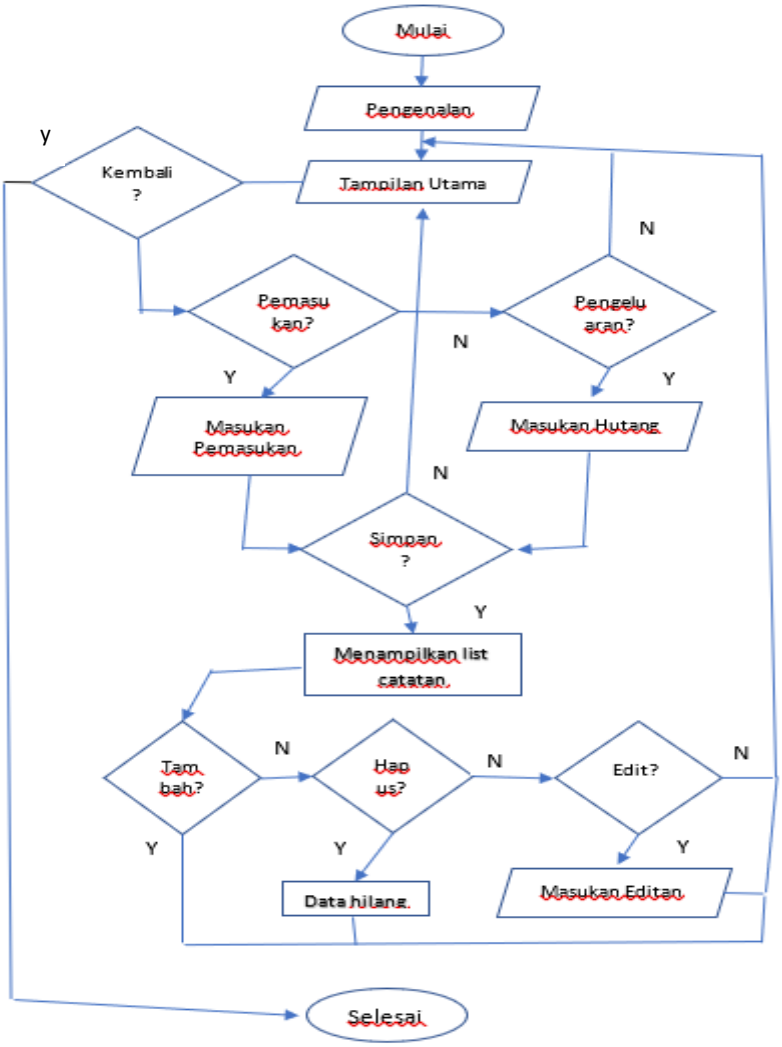
BAB III
PERANCANGAN APLIKASI

3. 1 Perancangan Sistem

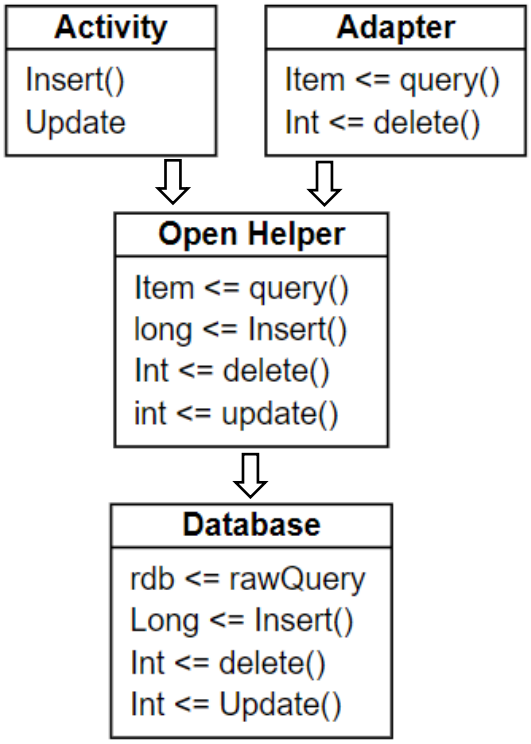
- Spesifikasi Minimum

  Aplikasi Pengingat Tugas ini dapat berjalan dengan spek minimum Android versi 4.0 (KitKat)

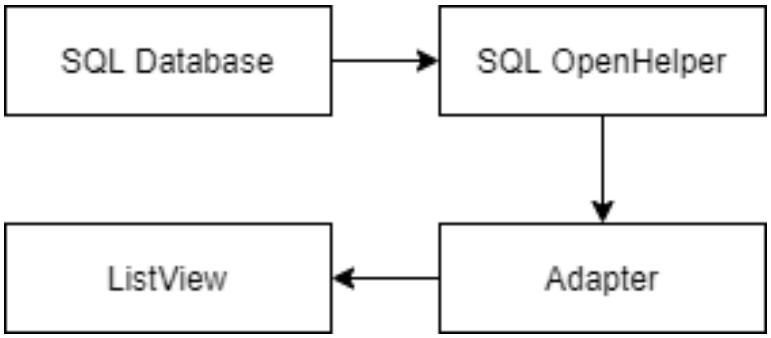
- Flowchart



- Class Diagram



- Entity Relationship Diagram



3. 2 Perancangan UI/UX

- Component

  Component yang digunakan dalam aplikasi ini ada recycleview, floating action button, spinner, text view, image button, progress bar, textview, imageview, check box.

- Layout

  Layout yang digunakan dalam aplikasi ini relative layout, constrain layout, linear layout, dan fragment.

- Style

  Style yang digunakan dalam aplikasi ini Theme.MaterialComponents.DayNight.NoActionBar .

3. 3 Coding

- activity_main.xml

```
import android.Manifest
import android.content.Intent
import android.content.pm.PackageManager
import android.os.Bundle
import android.os.Environment
import android.view.Menu
import android.view.MenuItem
import android.widget.Toast
import androidx.appcompat.app.ActionBarDrawerToggle

import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import androidx.lifecycle.ViewModelProvider
import com.ajts.androidmads.library.SQLiteToExcel
import com.pm.mycoin.R
import com.pm.mycoin.databinding.ActivityMainBinding
import com.pm.mycoin.databinding.SaldoDialogLayoutBinding
import com.pm.mycoin.utils.CurencyFormatter

import com.pm.mycoin.view.fragment.main.MainFragment
import com.pm.mycoin.view.fragment.main.MainViewModel

class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding
    private lateinit var viewModel: MainViewModel
    private lateinit var excelConverter: SQLiteToExcel

    private val directoryPath =
```

```
Environment.getExternalStoragePublicDirectory(Environment.DIRECTOR
Y_DOWNLOADS).path
    private val tableList = arrayListOf("record_table",
"debt_table")
    private var permissionGranted = false
    private var alreadyConverted = false

    private var totalExpenses = 0L
    private var totalIncome = 0L
    private var totalDebt = 0L

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        setSupportActionBar(binding.mainToolbar)
        supportActionBar?.title = null

        ActivityCompat.requestPermissions(
            this,
            arrayOf(Manifest.permission.WRITE_EXTERNAL_STORAGE),
            1
        )


        val toggle = ActionBarDrawerToggle(
            this, binding.drawerLayout,
            R.string.open,
            R.string.close
        )
        binding.drawerLayout.addDrawerListener(toggle)
        toggle.syncState()
        toggle.isDrawerIndicatorEnabled = true
        toggle.drawerArrowDrawable.color =
ContextCompat.getColor(this, android.R.color.white)




supportFragmentManager.beginTransaction().replace(R.id.fragment,
MainFragment()).commit()

        viewModel =
ViewModelProvider(this).get(MainViewModel::class.java)
        viewModel.getTotalExpenses()?.observe(this, {
            if (it != null) {
                totalExpenses = it.toLong()
            }
        })

        viewModel.getTotalIncome()?.observe(this, {
            if (it != null) {
                totalIncome = it.toLong()
            }
```

```
        })

        viewModel.getTotalDebt()?.observe(this, {
            if (it != null) {
                totalDebt = it.toLong()
            }
        })
    }

    override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<out String>,
        grantResults: IntArray
    ) {
        when (requestCode) {
            1 -> if (grantResults.isNotEmpty() && grantResults[0]
== PackageManager.PERMISSION_GRANTED) {
                permissionGranted = true
            }
        }
    }

    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        menuInflater.inflate(R.menu.main, menu)
        return true
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        when (item.itemId) {
            R.id.action_reset -> showDialog()
            R.id.action_saldo -> showBalanceDialog()

        }

        return true
    }



    private fun showBalanceDialog() {
        val dialog = AlertDialog.Builder(this)
        val dialogView =
SaldoDialogLayoutBinding.inflate(layoutInflater)

        dialogView.apply {
            dialogMainIncome.text =
CurencyFormatter.convertAndFormat(totalIncome)
            dialogMainExpenses.text =
CurencyFormatter.convertAndFormat(
                totalExpenses
            )
            dialogMainDebt.text =
CurencyFormatter.convertAndFormat(totalDebt)
            dialogMainSaldo.text =
                CurencyFormatter.convertAndFormat(totalIncome -
(totalExpenses + totalDebt))
```

```
        }

        dialog.setView(dialogView.root)
        dialog.setTitle(R.string.dialog_title_saldo)
        dialog.setCancelable(true)
        dialog.setPositiveButton("Close", null)
        dialog.show()
    }

    private fun showDialog() {
        val dialog = AlertDialog.Builder(this)
        dialog.setTitle(getString(R.string.perhatian))
        dialog.setMessage(R.string.dialog_message)
        dialog.setCancelable(true)
        dialog.setPositiveButton("Yes") { _, _ ->
            viewModel.deleteAllRecord()
            viewModel.deleteAllDebt()

            totalIncome = 0
            totalExpenses = 0
            totalDebt = 0
        }
        dialog.setNegativeButton("Cancel") { innerDialog, _ ->
            innerDialog.dismiss()
        }

        dialog.show()
    }

    fun reduceValue(key: String, amount: Long) {
        when (key) {
            "income" -> totalIncome -= amount
            "expenses" -> totalExpenses -= amount
            else -> totalDebt -= amount
        }
    }

}
```

- **Intro1activity**

```
import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.ImageButton
import com.pm.mycoin.R

class Intro1Activity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_intro1)
        val btnnext = findViewById<Button>(R.id.btnnext)

        btnnext?.setOnClickListener {
            startActivity(Intent(this, MainActivity::class.java))
```

```
        }

    }
}
```

- **SplashScreenActivity**

```
import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.os.Handler
import com.pm.mycoin.R

class SplashScreenActivity : AppCompatActivity() {
    private val SPLASH_TIME_OUT:Long = 3000
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_splash_screen)
        Handler().postDelayed({
            startActivity(Intent(this,Intro1Activity::class.java))
            finish()
        }, SPLASH_TIME_OUT)


    }
}
```

- **MainFragment**

```
import android.annotation.SuppressLint
import android.app.DatePickerDialog
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.appcompat.app.AlertDialog
import androidx.fragment.app.Fragment
import androidx.lifecycle.ViewModelProvider
import androidx.viewpager.widget.ViewPager
import com.pm.mycoin.R
import com.pm.mycoin.databinding.AddDialogLayoutBinding
import com.pm.mycoin.databinding.FragmentMainBinding
import com.pm.mycoin.db.Debt
import com.pm.mycoin.db.Record
import com.pm.mycoin.utils.CurencyFormatter
import com.pm.mycoin.utils.DateUtil
import java.util.*

class MainFragment : Fragment() {
    private lateinit var binding: FragmentMainBinding
    private lateinit var viewModel: MainViewModel
    private lateinit var adapter: PagerAdapter
```

```
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentMainBinding.inflate(inflater, container,
false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        viewModel =
ViewModelProvider(this).get(MainViewModel::class.java)
        viewModel.getTotalExpenses()?.observe(viewLifecycleOwner,
{
            if (it != null) {
                binding.mainTotalExpenses.text =
CurencyFormatter.convertAndFormat(it.toLong())
            } else {
                binding.mainTotalExpenses.text =
CurencyFormatter.convertAndFormat(0)
            }
        })

        viewModel.getTotalIncome()?.observe(viewLifecycleOwner, {
            if (it != null) {
                binding.mainTotalIncome.text =
CurencyFormatter.convertAndFormat(it.toLong())
            } else {
                binding.mainTotalIncome.text =
CurencyFormatter.convertAndFormat(0)
            }
        })

        adapter = PagerAdapter(childFragmentManager)
        binding.apply {
            mainViewPager.adapter = adapter
            mainViewPager.offscreenPageLimit = 3
            mainViewPager.addOnPageChangeListener(object :
ViewPager.OnPageChangeListener {
                override fun onPageScrollStateChanged(p0: Int) {}

                override fun onPageScrolled(p0: Int, p1: Float,
p2: Int) {
                    if (p0 == 1) {
                        historyFab.hide()
                        debtFab.show()
                    } else {
                        historyFab.show()
                        debtFab.hide()
                    }
                }
```

```
                override fun onPageSelected(p0: Int) {
                    if (p0 == 0) {
                        historyFab.hide()
                        debtFab.show()
                    } else {
                        historyFab.show()
                        debtFab.hide()
                    }
                }
            })


binding.mainTabLayout.setupWithViewPager(mainViewPager)

            historyFab.setOnClickListener {
showAddDataDialog("history") }
            debtFab.setOnClickListener {
showAddDataDialog("utang") }
        }
    }

    @SuppressLint("SetTextI18n")
    private fun showAddDataDialog(key: String) {
        val builder = context?.let { AlertDialog.Builder(it) }
        val dialogView =
AddDialogLayoutBinding.inflate(layoutInflater)

        var selectedDate = Date()
        val c = Calendar.getInstance()
        val year = c.get(Calendar.YEAR)
        val month = c.get(Calendar.MONTH)
        val day = c.get(Calendar.DAY_OF_MONTH)

        when (key) {
            "utang" -> dialogView.dialogCheckboxIncome.visibility
= View.GONE
        }

        builder?.setView(dialogView.root)
        dialogView.dialogShowDate.setOnClickListener {
            DatePickerDialog(requireContext(), { _, year,
monthOfYear, dayOfMonth ->
                val calendar = Calendar.getInstance()
                calendar.set(year, monthOfYear, dayOfMonth)
                dialogView.dialogDate.text =
                    "Transaction date:
${DateUtil.formatDate(calendar.time)}"
                selectedDate = calendar.time
            }, year, month, day).show()
        }

        builder?.setCancelable(true)
        builder?.setPositiveButton(R.string.dialog_simpan, null)
        val dialog = builder?.create()
        dialog?.show()
```

```
dialog?.getButton(AlertDialog.BUTTON_POSITIVE)?.setOnClickListener
{
            val isIncome = if
(dialogView.dialogCheckboxIncome.isChecked) {
                "income"
            } else {
                "expenses"
            }

            if (dialogView.dialogTitle.text.isNotBlank() &&
dialogView.dialogAmount.text.isNotBlank()
            ) {

                val totalIncome =
dialogView.dialogAmount.text.toString()

                if (key == "history") {
                    val record = Record(
                        0,
                        dialogView.dialogTitle.text.toString(),
                        totalIncome.toLong(),
                        selectedDate,
                        isIncome
                    )

                    viewModel.insertRecord(record)
                } else {
                    val debt = Debt(
                        0,
                        dialogView.dialogTitle.text.toString(),
                        totalIncome.toInt(),
                        selectedDate
                    )

                    viewModel.insertDebt(debt)
                }

                dialog.dismiss()
            } else {
                Toast.makeText(context, R.string.toast_isi_kolom,
Toast.LENGTH_SHORT).show()
            }
        }
    }
}
```

- **MainViewModel**

```
import android.app.Application
import androidx.lifecycle.AndroidViewModel
import androidx.lifecycle.LiveData
import com.pm.mycoin.db.Debt
import com.pm.mycoin.db.DebtRepo
import com.pm.mycoin.db.Record
import com.pm.mycoin.db.RecordRepo
import com.pm.mycoin.utils.DateUtil
```

```
import java.util.*

class MainViewModel(application: Application) :
AndroidViewModel(application) {
    private val recordRepo = RecordRepo(application)
    private val debtRepo = DebtRepo(application)

    fun getAllRecords(isNewest: Boolean): LiveData<List<Record>>?
{
        return if (isNewest) {
            recordRepo.getAllRecordsDesc()
        } else {
            recordRepo.getAllRecordsAsc()
        }
    }

    fun getFilteredRecord(
        startDate: Date,
        endDate: Date,
        isDesc: Boolean
    ): LiveData<List<Record>>? =

recordRepo.getFilteredRecord(DateUtil.subtractDays(startDate, 1),
endDate, isDesc)

    fun getAllDebts(isNewest: Boolean): LiveData<List<Debt>>? {
        return if (isNewest) {
            debtRepo.getAllDebtDesc()
        } else {
            debtRepo.getAllDebtAsc()
        }
    }

    fun getFilteredDebt(startDate: Date, endDate: Date, isDesc:
Boolean): LiveData<List<Debt>>? =

debtRepo.getFilteredDebtDesc(DateUtil.subtractDays(startDate, 1),
endDate, isDesc)

    fun getTotalExpenses(): LiveData<Int>? {
        return recordRepo.getTotalExpenses()
    }

    fun getTotalDebt(): LiveData<Int>? {
        return debtRepo.getTotalDebt()
    }

    fun getTotalIncome(): LiveData<Int>? {
        return recordRepo.getTotalIncome()
    }

    fun insertRecord(record: Record) {
        recordRepo.insertRecord(record)
    }

    fun updateRecord(record: Record) {
```

```
        recordRepo.updateRecord(record)
    }

    fun insertDebt(debt: Debt) {
        return debtRepo.insertDebt(debt)
    }

    fun deleteRecord(record: Record) {
        recordRepo.deleteRecord(record)
    }

    fun deleteDebt(debt: Debt) {
        debtRepo.deleteDebt(debt)
    }

    fun updateDebt(debt: Debt) {
        debtRepo.updateDebt(debt)
    }

    fun deleteAllRecord() {
        recordRepo.deleteAllRecord()
    }

    fun deleteAllDebt() {
        debtRepo.deleteAllDebt()
    }
```

- **PagerAdapter**

```
import androidx.fragment.app.Fragment
import androidx.fragment.app.FragmentManager
import androidx.fragment.app.FragmentStatePagerAdapter
import com.pm.mycoin.view.fragment.debt.DebtFragment
import com.pm.mycoin.view.fragment.history.HistoryFragment

class PagerAdapter(fragmentManager: FragmentManager) :
    FragmentStatePagerAdapter(fragmentManager,
BEHAVIOR_RESUME_ONLY_CURRENT_FRAGMENT) {

    private val fragments = listOf(HistoryFragment(),
DebtFragment())

    override fun getItem(p0: Int): Fragment = fragments[p0]

    override fun getCount(): Int = fragments.size

    override fun getPageTitle(position: Int): CharSequence? {
        return when (position) {
            0 -> "RIWAYAT"
            1 -> "HUTANG"
            else -> super.getPageTitle(position)
        }
    }
}
```

- **Debt**

```
import androidx.annotation.NonNull
import androidx.room.*
import com.pm.mycoin.db.converter.DateConverter
import java.util.*

@Entity(tableName = "record_table")
@TypeConverters(DateConverter::class)
class Record(
    @PrimaryKey(autoGenerate = true)
    @NonNull
    @ColumnInfo(name = "id")
    var id: Int = 0,
    @ColumnInfo(name = "judul")
    var judul: String = "None",
    @ColumnInfo(name = "total")
    var total: Long = 0,
    @ColumnInfo(name = "date")
    var date: Date? = null,
    @ColumnInfo(name = "description")
    var description: String = "expenses",
    @Ignore
    var type: Int = 0
)

@Entity(tableName = "debt_table")
@TypeConverters(DateConverter::class)
class Debt(
    @PrimaryKey(autoGenerate = true)
    @NonNull
    @ColumnInfo(name = "id")
    var id: Int = 0,
    @ColumnInfo(name = "judul")
    var judul: String = "None",
    @ColumnInfo(name = "total")
    var total: Int = 0,
    @ColumnInfo(name = "date")
    var date: Date? = null,
    @Ignore
    var type: Int = 0
)
```

- **DebtAdapter**

```
import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import androidx.viewbinding.ViewBinding
import com.pm.mycoin.databinding.ItemDateBinding
import com.pm.mycoin.databinding.ItemRowBinding
import com.pm.mycoin.db.Debt
import com.pm.mycoin.utils.CurencyFormatter.convertAndFormat
import com.pm.mycoin.utils.DateUtil
import java.util.*
```

```
class DebtAdapter(
    private var datas: MutableList<Debt>?,
    private val clickUtils: (Debt, String) -> Unit
) : RecyclerView.Adapter<RecyclerView.ViewHolder>() {
    var date: Date? = null
    override fun getItemViewType(position: Int): Int {
        if (datas != null) {
            return datas!![position].type
        }

        return 0
    }

    override fun onCreateViewHolder(p0: ViewGroup, p1: Int):
RecyclerView.ViewHolder {
        val inflater = LayoutInflater.from(p0.context)
        val binding: ViewBinding

        if (p1 == 0) {
            binding = ItemRowBinding.inflate(inflater, p0, false)
            return MainHolder(binding)
        }

        binding = ItemDateBinding.inflate(inflater, p0, false)
        return DateHolder(binding)
    }

    override fun getItemCount(): Int {
        return datas?.size ?: 0
    }

    override fun onBindViewHolder(p0: RecyclerView.ViewHolder, p1:
Int) {
        if (datas != null) {
            if (p0.itemViewType == 0) {
                p0 as DebtAdapter.MainHolder
                p0.bind(datas!![p1], clickUtils)
            } else {
                p0 as DebtAdapter.DateHolder
                p0.bind(datas!![p1].date!!)
            }
        }
    }

    fun setData(debt: MutableList<Debt>?) {
        if (debt == null || debt.isEmpty()) {
            datas?.clear()
        } else {
            var date = DateUtil.formatDate(debt[0].date!!)
            debt.add(0, Debt(type = 1, date = debt[0].date))

            var i = 0
            while (i <= debt.size - 1) {
                val formattedDate =
DateUtil.formatDate(debt[i].date!!)
```

```
                if (date != formattedDate) {
                    date = formattedDate
                    debt.add(i, Debt(type = 1, date =
debt[i].date))
                } else {
                    i++
                }
            }

            datas = debt
        }
        notifyDataSetChanged()
    }

    inner class MainHolder(private val binding: ItemRowBinding) :
        RecyclerView.ViewHolder(binding.root) {
        private lateinit var debt: Debt

        fun bind(debt: Debt, clickUtils: (Debt, String) -> Unit) {
            this.debt = debt

            binding.apply {
                itemTitle.text = debt.judul
                itemUang.text =
convertAndFormat(debt.total.toLong())
//                itemDate.text = DateUtil.formatDate(debt.date!!)
                itemDelete.setOnClickListener { clickUtils(debt,
"delete") }
                itemUpdate.setOnClickListener { clickUtils(debt,
"edit") }
                itemView.setOnClickListener { clickUtils(debt,
"edit") }
            }
        }
    }

    inner class DateHolder(private val binding: ItemDateBinding) :
        RecyclerView.ViewHolder(binding.root) {
        fun bind(date: Date) {
            binding.itemDate.text = DateUtil.formatDate(date)
        }
    }
}
```

- **DebtFragment**

```
import android.annotation.SuppressLint
import android.app.DatePickerDialog
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.appcompat.app.AlertDialog
import androidx.fragment.app.Fragment
import androidx.lifecycle.ViewModelProvider
```

```
import androidx.recyclerview.widget.LinearLayoutManager
import com.pm.mycoin.R
import com.pm.mycoin.databinding.AddDialogLayoutBinding
import com.pm.mycoin.databinding.FilterDialogLayoutBinding
import com.pm.mycoin.databinding.FragmentDebtBinding
import com.pm.mycoin.db.Debt
import com.pm.mycoin.utils.DateUtil
import com.pm.mycoin.view.activity.main.MainActivity
import com.pm.mycoin.view.fragment.main.MainViewModel
import java.util.*

class DebtFragment : Fragment() {
    private lateinit var binding: FragmentDebtBinding
    private var adapter: DebtAdapter? = null
    private var viewModel: MainViewModel? = null
    private var debts: List<Debt>? = null

    private var startDate: Date? = null
    private var endDate: Date? = null

    private var isNewest = true
    private var isFiltered = false

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentDebtBinding.inflate(inflater, container,
false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        viewModel = activity?.let {
ViewModelProvider(it).get(MainViewModel::class.java) }
        populateRecycler()
        getAllDebts()

        binding.debtSort.setOnClickListener {
            isNewest = !isNewest

            debts = debts?.reversed()
            adapter?.setData(debts?.toMutableList())

            if (!isNewest) binding.debtSortImage.rotationX =
180.0.toFloat()
            else binding.debtSortImage.rotationX = 0.toFloat()

            if (!isNewest) binding.debtSortText.text =
getString(R.string.sort_oldest)
            else binding.debtSortText.text =
getString(R.string.sort_newest)
        }
```

```
        binding.debtFilter.setOnClickListener {
            if (isFiltered) {
                binding.debtFilterText.text =
getString(R.string.filter)
                getAllDebts()
            } else {
                showFilterDialog()
            }

            isFiltered = !isFiltered
        }
    }

    private fun getAllDebts() {

viewModel?.getAllDebts(isNewest)?.observe(viewLifecycleOwner, {
            debts = it
            adapter?.setData(it?.toMutableList())
        })
    }

    private fun populateRecycler() {
        adapter = DebtAdapter(null) { it, it1 ->
            if (it1 == "delete") {
                (parentFragment?.activity as
MainActivity).reduceValue("", it.total.toLong())

                viewModel?.deleteDebt(it)
                Toast.makeText(context,
R.string.toast_hapus_berhasil, Toast.LENGTH_SHORT).show()
            } else {
                showAddDataDialog(it)
            }
        }

        binding.debtRecycler.apply {
            layoutManager = LinearLayoutManager(context)
            setHasFixedSize(true)
        }

        binding.debtRecycler.adapter = adapter
    }

    @SuppressLint("SetTextI18n")
    private fun showAddDataDialog(debt: Debt) {
        val builder = context?.let { AlertDialog.Builder(it) }
        val dialogView =
AddDialogLayoutBinding.inflate(layoutInflater)

        dialogView.apply {
            dialogTitle.setText(debt.judul)
            dialogAmount.setText(debt.total.toString())
            dialogDate.text = "Transaction date: ${debt.date?.let
{ DateUtil.formatDate(it) }}"
            dialogCheckboxIncome.visibility = View.GONE
```

```
        }

        var selectedDate = debt.date
        val c = Calendar.getInstance()
        val year = c.get(Calendar.YEAR)
        val month = c.get(Calendar.MONTH)
        val day = c.get(Calendar.DAY_OF_MONTH)

        dialogView.dialogShowDate.setOnClickListener {
            DatePickerDialog(requireContext(), { _, year,
monthOfYear, dayOfMonth ->
                val calendar = Calendar.getInstance()
                calendar.set(year, monthOfYear, dayOfMonth)
                dialogView.dialogDate.text =
                    "Transaction date:
${DateUtil.formatDate(calendar.time)}"
                selectedDate = calendar.time
            }, year, month, day).show()
        }

        builder?.setView(dialogView.root)
        builder?.setCancelable(true)
        builder?.setPositiveButton(R.string.dialog_simpan, null)

        val dialog = builder?.create()
        dialog?.show()

dialog?.getButton(AlertDialog.BUTTON_POSITIVE)?.setOnClickListener
{
            if (dialogView.dialogTitle.text.isNotBlank() &&
dialogView.dialogAmount.text.isNotBlank()
                && selectedDate != null
            ) {
                val innerDebt = Debt(
                    debt.id,
dialogView.dialogTitle.text.toString(),

dialogView.dialogAmount.text.toString().toInt(),
                    selectedDate
                )

                viewModel?.updateDebt(innerDebt)
                dialog.dismiss()
            } else {
                Toast.makeText(context, R.string.toast_isi_kolom,
Toast.LENGTH_SHORT).show()
            }
        }
    }

    private fun showFilterDialog() {
        val dialog = context?.let { AlertDialog.Builder(it) }
        val dialogView =
FilterDialogLayoutBinding.inflate(layoutInflater)

        val c = Calendar.getInstance()
```

```
        val year = c.get(Calendar.YEAR)
        val month = c.get(Calendar.MONTH)
        val day = c.get(Calendar.DAY_OF_MONTH)

        dialogView.filterStartDate.setOnClickListener {
            DatePickerDialog(requireContext(), { _, year,
monthOfYear, dayOfMonth ->
                val calendar = Calendar.getInstance()
                calendar.set(year, monthOfYear, dayOfMonth)
                startDate = calendar.time
                dialogView.filterStartDate.text =
DateUtil.formatDate(calendar.time)
            }, year, month, day).show()
        }

        dialogView.filterEndDate.setOnClickListener {
            DatePickerDialog(requireContext(), { _, year,
monthOfYear, dayOfMonth ->
                val calendar = Calendar.getInstance()
                calendar.set(year, monthOfYear, dayOfMonth)
                endDate = calendar.time
                dialogView.filterEndDate.text =
DateUtil.formatDate(calendar.time)
            }, year, month, day).show()
        }

        dialog?.setView(dialogView.root)
        dialog?.setCancelable(true)
        dialog?.setPositiveButton(R.string.dialog_simpan) { _, _ -
>
            if (startDate != null && endDate != null) {
                viewModel?.getFilteredDebt(startDate!!, endDate!!,
isNewest)
                    ?.observe(viewLifecycleOwner, {
                        debts = it
                        adapter?.setData(it?.toMutableList())

                        binding.debtFilterText.text =
getString(R.string.remove_filter)
                    })
            }
        }

        dialog?.show()
    }
}
```

- **HistoryAdapter**

```
import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.core.content.ContextCompat
import androidx.recyclerview.widget.RecyclerView
import androidx.viewbinding.ViewBinding
```

```
import com.pm.mycoin.R
import com.pm.mycoin.databinding.ItemDateBinding
import com.pm.mycoin.databinding.ItemRowBinding
import com.pm.mycoin.db.Record
import com.pm.mycoin.utils.CurencyFormatter.convertAndFormat
import com.pm.mycoin.utils.DateUtil
import java.util.*

class HistoryAdapter(
    private val context: Context,
    private var datas: MutableList<Record>?,
    private val fromGraph: Boolean,
    private val clickUtils: (Record, String) -> Unit,
) : RecyclerView.Adapter<RecyclerView.ViewHolder>() {
    var date: Date? = null
    override fun getItemViewType(position: Int): Int {
        if (datas != null) {
            return datas!![position].type
        }

        return 0
    }

    override fun onCreateViewHolder(p0: ViewGroup, p1: Int):
RecyclerView.ViewHolder {
        val inflater = LayoutInflater.from(context)
        val binding: ViewBinding

        if (p1 == 0) {
            binding = ItemRowBinding.inflate(inflater, p0, false)
            return MainHolder(binding)
        }

        binding = ItemDateBinding.inflate(inflater, p0, false)
        return DateHolder(binding)
    }

    override fun getItemCount(): Int {
        return datas?.size ?: 0
    }

    override fun onBindViewHolder(p0: RecyclerView.ViewHolder, p1:
Int) {
        if (datas != null) {
            if (p0.itemViewType == 0) {
                p0 as MainHolder
                p0.bind(datas!![p1], clickUtils)
            } else {
                p0 as DateHolder
                p0.bind(datas!![p1].date!!)
            }
        }
    }

    fun setData(records: MutableList<Record>?) {
        datas?.clear()
```

```
        if (!records.isNullOrEmpty()) {
            var date = DateUtil.formatDate(records[0].date!!)
            records.add(0, Record(type = 1, date =
records[0].date))

            var i = 0
            while (i <= records.size - 1) {
                val formattedDate =
DateUtil.formatDate(records[i].date!!)

                if (date != formattedDate) {
                    date = formattedDate
                    records.add(i, Record(type = 1, date =
records[i].date))
                } else {
                    i++
                }
            }

            datas = records
        }

        notifyDataSetChanged()
    }

    inner class MainHolder(private val binding: ItemRowBinding) :
        RecyclerView.ViewHolder(binding.root) {
        private lateinit var record: Record

        fun bind(record: Record, clickUtils: (Record, String) ->
Unit) {
            this.record = record

            binding.apply {
                itemTitle.text = record.judul
                itemUang.text =
convertAndFormat(record.total.toLong())
                itemDelete.setOnClickListener { clickUtils(record,
"delete") }
                itemUpdate.setOnClickListener { clickUtils(record,
"edit") }
                itemView.setOnClickListener { clickUtils(record,
"edit") }

                if (!fromGraph) {
                    itemDelete.setOnClickListener {
clickUtils(record, "delete") }
                    itemUpdate.setOnClickListener {
clickUtils(record, "edit") }
                } else {
                    itemDelete.visibility = View.GONE
                    itemUpdate.visibility = View.GONE
                }

                if (record.description == "income") {
```

```
                    itemColor.setBackgroundColor(
                        ContextCompat.getColor(
                            context,
                            R.color.colorAccent
                        )
                    )

itemUang.setTextColor(ContextCompat.getColor(context,
R.color.colorAccent))
                } else {
                    itemColor.setBackgroundColor(
                        ContextCompat.getColor(
                            context,
                            R.color.colorRed
                        )
                    )

itemUang.setTextColor(ContextCompat.getColor(context,
R.color.colorRed))
                }
            }
        }

    inner class DateHolder(private val binding: ItemDateBinding) :
        RecyclerView.ViewHolder(binding.root) {
        fun bind(date: Date) {
            binding.itemDate.text = DateUtil.formatDate(date)
        }
    }
}
```

- **HistoryFragent**

```
import android.annotation.SuppressLint
import android.app.DatePickerDialog
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.appcompat.app.AlertDialog
import androidx.fragment.app.Fragment
import androidx.lifecycle.ViewModelProvider
import androidx.recyclerview.widget.LinearLayoutManager
import com.pm.mycoin.R
import com.pm.mycoin.databinding.AddDialogLayoutBinding
import com.pm.mycoin.databinding.FilterDialogLayoutBinding
import com.pm.mycoin.databinding.FragmentHistoryBinding
import com.pm.mycoin.db.Record
import com.pm.mycoin.utils.DateUtil
import com.pm.mycoin.view.activity.main.MainActivity
import com.pm.mycoin.view.fragment.main.MainViewModel
import java.util.*
```

```
class HistoryFragment : Fragment() {
    private lateinit var binding: FragmentHistoryBinding

    private var viewModel: MainViewModel? = null
    private var adapter: HistoryAdapter? = null
    private var records: List<Record>? = null

    private var startDate: Date? = null
    private var endDate: Date? = null

    private var isNewest = true
    private var isFiltered = false

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentHistoryBinding.inflate(inflater,
container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        viewModel = activity?.let {
ViewModelProvider(it).get(MainViewModel::class.java) }
        populateRecycler()
        getAllRecords()

        binding.historySort.setOnClickListener {
            isNewest = !isNewest

            records = records?.reversed()
            adapter?.setData(records?.toMutableList())

            if (!isNewest) binding.historySortImage.rotationX =
180.0.toFloat()
            else binding.historySortImage.rotationX = 0.toFloat()

            if (!isNewest) binding.historySortText.text =
getString(R.string.sort_oldest)
            else binding.historySortText.text =
getString(R.string.sort_newest)
        }

        binding.historyFilter.setOnClickListener {
            if (isFiltered) {
                binding.historyFilterText.text =
getString(R.string.filter)
                getAllRecords()
            } else {
                showFilterDialog()
            }
```

```
            isFiltered = !isFiltered
        }
    }

    private fun getAllRecords() {

viewModel?.getAllRecords(isNewest)?.observe(viewLifecycleOwner, {
            records = it
            adapter?.setData(it?.toMutableList())
        })
    }

    private fun deleteRecords(record: Record) {
        (parentFragment?.activity as MainActivity).reduceValue(
            record.description,
            record.total
        )

        viewModel?.deleteRecord(record)
        Toast.makeText(context, R.string.toast_hapus_berhasil,
Toast.LENGTH_SHORT).show()
    }

    private fun populateRecycler() {
        adapter = context?.let {
            HistoryAdapter(it, null, false) { record, it1 ->
                if (it1 == "delete") {
                    deleteRecords(record)
                } else {
                    showAddDataDialog(record)
                }
            }
        }

        binding.historyRecycler.apply {
            layoutManager = LinearLayoutManager(context)
            setHasFixedSize(true)
        }

        binding.historyRecycler.adapter = adapter
    }

    @SuppressLint("SetTextI18n")
    private fun showAddDataDialog(record: Record) {
        val dialog = context?.let { AlertDialog.Builder(it) }
        val dialogView =
AddDialogLayoutBinding.inflate(layoutInflater)

        dialogView.apply {
            dialogTitle.setText(record.judul)
            dialogAmount.setText(record.total.toString())
            dialogDate.text = "Transaction date:
${record.date?.let { DateUtil.formatDate(it) }}"
            dialogCheckboxIncome.isEnabled = false
        }
```

```
        var selectedDate = record.date
        val c = Calendar.getInstance()
        val year = c.get(Calendar.YEAR)
        val month = c.get(Calendar.MONTH)
        val day = c.get(Calendar.DAY_OF_MONTH)

        dialogView.dialogShowDate.setOnClickListener {
            DatePickerDialog(requireContext(), { _, year,
monthOfYear, dayOfMonth ->
                val calendar = Calendar.getInstance()
                calendar.set(year, monthOfYear, dayOfMonth)
                dialogView.dialogDate.text =
                    "Transaction date:
${DateUtil.formatDate(calendar.time)}"
                selectedDate = calendar.time
            }, year, month, day).show()
        }

        dialog?.setView(dialogView.root)
        dialog?.setCancelable(true)
        dialog?.setPositiveButton(R.string.dialog_simpan) { _, _ -
>
            val innerRecord = Record(
                record.id, dialogView.dialogTitle.text.toString(),
                dialogView.dialogAmount.text.toString().toLong(),
                selectedDate,
                record.description
            )

            viewModel?.updateRecord(innerRecord)
        }

        dialog?.show()
    }

    private fun showFilterDialog() {
        val dialog = context?.let { AlertDialog.Builder(it) }
        val dialogView =
FilterDialogLayoutBinding.inflate(layoutInflater)

        val c = Calendar.getInstance()
        val year = c.get(Calendar.YEAR)
        val month = c.get(Calendar.MONTH)
        val day = c.get(Calendar.DAY_OF_MONTH)

        dialogView.filterStartDate.setOnClickListener {
            DatePickerDialog(requireContext(), { _, year,
monthOfYear, dayOfMonth ->
                val calendar = Calendar.getInstance()
                calendar.set(year, monthOfYear, dayOfMonth)
                startDate = calendar.time
                dialogView.filterStartDate.text =
DateUtil.formatDate(calendar.time)
            }, year, month, day).show()
        }
```

```
        dialogView.filterEndDate.setOnClickListener {
            DatePickerDialog(requireContext(), { _, year,
monthOfYear, dayOfMonth ->
                val calendar = Calendar.getInstance()
                calendar.set(year, monthOfYear, dayOfMonth)
                endDate = calendar.time
                dialogView.filterEndDate.text =
DateUtil.formatDate(calendar.time)
            }, year, month, day).show()
        }

        dialog?.setView(dialogView.root)
        dialog?.setCancelable(true)
        dialog?.setPositiveButton(R.string.dialog_simpan) { _, _ -
>
            if (startDate != null && endDate != null) {
                viewModel?.getFilteredRecord(startDate!!,
endDate!!, isNewest)?.observe(
                    viewLifecycleOwner,
                    {
                        records = it
                        adapter?.setData(it?.toMutableList())

                        binding.historyFilterText.text =
getString(R.string.remove_filter)
                    })
            }
        }

        dialog?.show()
    }
}
```

- **Record**

```
import androidx.annotation.NonNull
import androidx.room.*
import com.pm.mycoin.db.converter.DateConverter
import java.util.*

@Entity(tableName = "record_table")
@TypeConverters(DateConverter::class)
class Record(
    @PrimaryKey(autoGenerate = true)
    @NonNull
    @ColumnInfo(name = "id")
    var id: Int = 0,
    @ColumnInfo(name = "judul")
    var judul: String = "None",
    @ColumnInfo(name = "total")
    var total: Long = 0,
    @ColumnInfo(name = "date")
    var date: Date? = null,
    @ColumnInfo(name = "description")
    var description: String = "expenses",
    @Ignore
```

```
    var type: Int = 0
)

@Entity(tableName = "debt_table")
@TypeConverters(DateConverter::class)
class Debt(
    @PrimaryKey(autoGenerate = true)
    @NonNull
    @ColumnInfo(name = "id")
    var id: Int = 0,
    @ColumnInfo(name = "judul")
    var judul: String = "None",
    @ColumnInfo(name = "total")
    var total: Int = 0,
    @ColumnInfo(name = "date")
    var date: Date? = null,
    @Ignore
    var type: Int = 0
)
```

- **DebtRepo**

```
import android.app.Application
import androidx.lifecycle.LiveData
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.launch
import java.util.*

class DebtRepo(application: Application) {

    private val debtDb = RecordDb.getDb(application)
    private val debtDao = debtDb?.recordDao

    fun getAllDebtDesc(): LiveData<List<Debt>>? {
        return debtDao?.getAllDataDebtDesc()
    }

    fun getAllDebtAsc(): LiveData<List<Debt>>? {
        return debtDao?.getAllDataDebtAsc()
    }

    fun getFilteredDebtDesc(
        startDate: Date,
        endDate: Date,
        isDesc: Boolean
    ): LiveData<List<Debt>>? {
        return if (isDesc) {
            debtDao?.getFilteredDebtDesc(startDate, endDate)
        } else {
            debtDao?.getFilteredDebtAsc(startDate, endDate)
        }
    }

    fun getTotalDebt(): LiveData<Int>? {
        return debtDao?.getTotalDebt()
    }
```

```
    fun insertDebt(debt: Debt) {
        debtDao?.let {
            GlobalScope.launch {
                debtDao.insertDebt(debt)
            }
        }
    }

    fun updateDebt(debt: Debt) {
        debtDao?.let {
            GlobalScope.launch {
                debtDao.updateDebt(debt)
            }
        }
    }

    fun deleteAllDebt() {
        debtDao?.let {
            GlobalScope.launch {
                debtDao.deleteAllDebt()
            }
        }
    }

    fun deleteDebt(debt: Debt) {
        debtDao?.let {
            GlobalScope.launch {
                debtDao.deleteDebt(debt)
            }
        }
    }
```

- **RecordDAO**

```
import androidx.lifecycle.LiveData
import androidx.room.*
import com.pm.mycoin.db.converter.DateConverter
import java.util.*

@Dao
interface RecordDAO {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insert(record: Record)

    @Query("Delete from record_table")
    fun deleteAll()

    @Delete
    fun delete(record: Record)

    @Update
    fun update(record: Record)

    @Query("select * from record_table order by date desc")
```

```
    fun getAllDataDesc(): LiveData<List<Record>>

    @Query("select * from record_table order by date asc")
    fun getAllDataAsc(): LiveData<List<Record>>

    @TypeConverters(DateConverter::class)
    @Query("select * from record_table where date between
:startDate and :endDate order by date desc")
    fun getFilteredRecordDesc(startDate: Date, endDate: Date):
LiveData<List<Record>>

    @TypeConverters(DateConverter::class)
    @Query("select * from record_table where date between
:startDate and :endDate order by date asc")
    fun getFilteredRecordAsc(startDate: Date, endDate: Date):
LiveData<List<Record>>

    @Query("select * from record_table where description =
'expenses' order by date desc")
    fun getAllExpenses(): LiveData<List<Record>>

    @Query("select * from record_table where description =
'income' order by date desc")
    fun getAllIncome(): LiveData<List<Record>>

    @Query("select sum(total) from record_table where description
= 'expenses'")
    fun getTotalExpenses(): LiveData<Int>

    @Query("select sum(total) from record_table where description
= 'income'")
    fun getTotalIncome(): LiveData<Int>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertDebt(debt: Debt)

    @Query("Delete from debt_table")
    fun deleteAllDebt()

    @Delete
    fun deleteDebt(debt: Debt)

    @Update
    fun updateDebt(debt: Debt)

    @Query("select * from debt_table order by date desc")
    fun getAllDataDebtDesc(): LiveData<List<Debt>>

    @Query("select * from debt_table order by date asc")
    fun getAllDataDebtAsc(): LiveData<List<Debt>>

    @TypeConverters(DateConverter::class)
    @Query("select * from debt_table where date between :startDate
and :endDate order by date desc")
    fun getFilteredDebtDesc(startDate: Date, endDate: Date):
LiveData<List<Debt>>
```

```
    @TypeConverters(DateConverter::class)
    @Query("select * from debt_table where date between :startDate
and :endDate order by date asc")
    fun getFilteredDebtAsc(startDate: Date, endDate: Date):
LiveData<List<Debt>>

    @Query("select sum(total) from debt_table")
    fun getTotalDebt(): LiveData<Int>
}
```

- **RecordDb**

```
import android.app.Application
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase

@Database(entities = [Record::class, Debt::class], version = 3)
abstract class RecordDb : RoomDatabase() {
    abstract val recordDao: RecordDAO

    companion object {
        @Volatile
        private var db: RecordDb? = null

        fun getDb(application: Application): RecordDb? {
            if (db == null) {
                synchronized(RecordDb::class.java) {
                    if (db == null) {
                        db = Room.databaseBuilder(
                            application.applicationContext,
                            RecordDb::class.java, "record_db"
                        )
                            .fallbackToDestructiveMigration()
                            .build()
                    }
                }
            }

            return db
        }
    }
}
```

- **RecordRepo**

```
import android.app.Application
import androidx.lifecycle.LiveData
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.launch
import java.util.*

class RecordRepo(application: Application) {
    private val recordDb = RecordDb.getDb(application)
```

```
    private val recordDao = recordDb?.recordDao

    fun getAllRecordsDesc(): LiveData<List<Record>>? {
        return recordDao?.getAllDataDesc()
    }

    fun getAllRecordsAsc(): LiveData<List<Record>>? {
        return recordDao?.getAllDataAsc()
    }

    fun getFilteredRecord(
        startDate: Date,
        endDate: Date,
        isDesc: Boolean
    ): LiveData<List<Record>>? {
        return if (isDesc) {
            recordDao?.getFilteredRecordDesc(startDate, endDate)
        } else {
            recordDao?.getFilteredRecordAsc(startDate, endDate)
        }
    }

    fun getAllIncome(): LiveData<List<Record>>? {
        return recordDao?.getAllIncome()
    }

    fun getAllExpenses(): LiveData<List<Record>>? {
        return recordDao?.getAllExpenses()
    }

    fun getTotalExpenses(): LiveData<Int>? {
        return recordDao?.getTotalExpenses()
    }

    fun getTotalIncome(): LiveData<Int>? {
        return recordDao?.getTotalIncome()
    }

    fun insertRecord(record: Record) {
        recordDao?.let {
            GlobalScope.launch {
                recordDao.insert(record)
            }
        }
    }

    fun updateRecord(record: Record) {
        recordDao?.let {
            GlobalScope.launch {
                recordDao.update(record)
            }
        }
    }

    fun deleteAllRecord() {
        recordDao?.let {
```

```
            GlobalScope.launch {
                recordDao.deleteAll()
            }
        }
    }

    fun deleteRecord(record: Record) {
        recordDao?.let {
            GlobalScope.launch {
                recordDao.delete(record)
            }
        }
    }
}
```

- **DateConverter**

```
import androidx.room.TypeConverter
import java.util.*

class DateConverter {
    @TypeConverter
    fun toDate(dateLong: Long?): Date? {
        return dateLong?.let { Date(it) }
    }

    @TypeConverter
    fun fromDate(date: Date?): Long? {
        return date?.time
    }
}
```

- **CurencyFormatter**

```
import java.text.DecimalFormat;
import java.text.DecimalFormatSymbols;

public class CurencyFormatter {

    @SuppressWarnings("UnusedReturnValue")
    public static String convertAndFormat(long s) {
        DecimalFormat format = (DecimalFormat)
DecimalFormat.getCurrencyInstance();
        DecimalFormatSymbols formatRp = new
DecimalFormatSymbols();
        formatRp.setCurrencySymbol("Rp.");
        formatRp.setMonetaryDecimalSeparator(',');
        formatRp.setGroupingSeparator('.');

        format.setDecimalFormatSymbols(formatRp);
        return format.format(s);
    }
}
```

- **DateUtil**

```
import java.text.SimpleDateFormat
import java.util.*

class DateUtil {
    companion object {
        fun formatDate(input: Date): String {
            val result = SimpleDateFormat("EEE, dd MMMM yyyy",
Locale.getDefault())
            return result.format(input)
        }

        fun getCurrentDate(): String {
            val formatter = SimpleDateFormat("dd MM yyyy",
Locale.getDefault())
            return formatter.format(Date())
        }

        fun subtractDays(date: Date, days: Int): Date {
            val cal = GregorianCalendar()
            cal.time = date
            cal.add(Calendar.DATE, -days)
            return cal.time
        }
    }
}
```

- **PreUtil**

```
import android.content.Context
import android.content.SharedPreferences

class PrefUtil(val context: Context) {
    private var PRIVATE_MODE = 0
    private val PREF_NAME = "financial-records"

    private var sharedPref: SharedPreferences =
        context.getSharedPreferences(PREF_NAME, PRIVATE_MODE)

    fun saveToPref(key: String, value: Int) {
        sharedPref.edit().putInt(key, value).apply()
    }

    fun getFromPref(key: String) = sharedPref.getInt(key, 2)
}
```

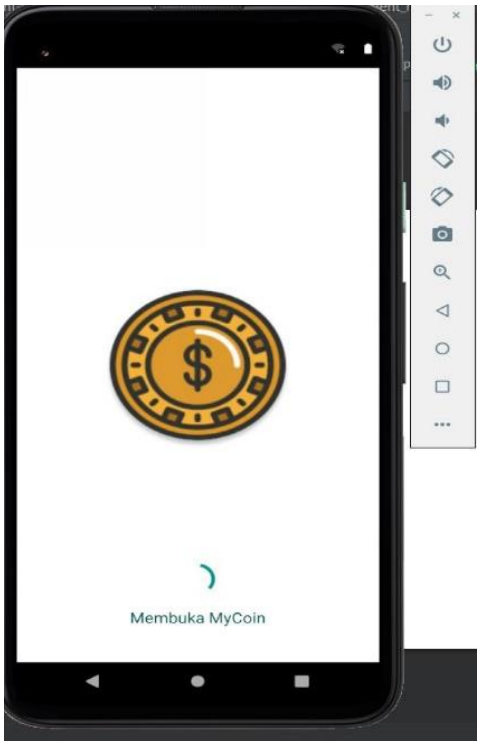
- **ReportFragment**

```
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ArrayAdapter
import androidx.fragment.app.Fragment
```

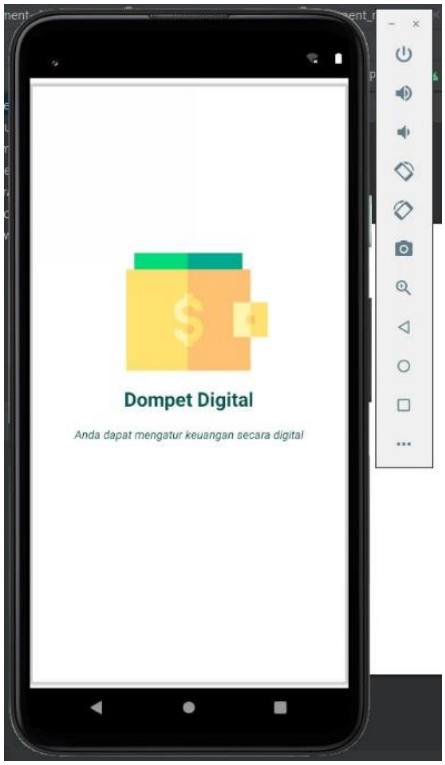
```
import androidx.lifecycle.ViewModelProvider
import com.pm.mycoin.R
import com.pm.mycoin.databinding.FragmentReportBinding
import com.pm.mycoin.view.fragment.main.MainViewModel

class ReportFragment : Fragment() {
    private lateinit var binding: FragmentReportBinding
    private var viewModel: MainViewModel? = null

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentReportBinding.inflate(inflater,
container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        val adapter = ArrayAdapter(
            requireContext(),
            android.R.layout.simple_spinner_item,
            resources.getStringArray(R.array.months)
        )
        binding.reportMonth.adapter = adapter

        viewModel = activity?.let {
ViewModelProvider(it).get(MainViewModel::class.java) }
    }
}
```

3. 4 Testing

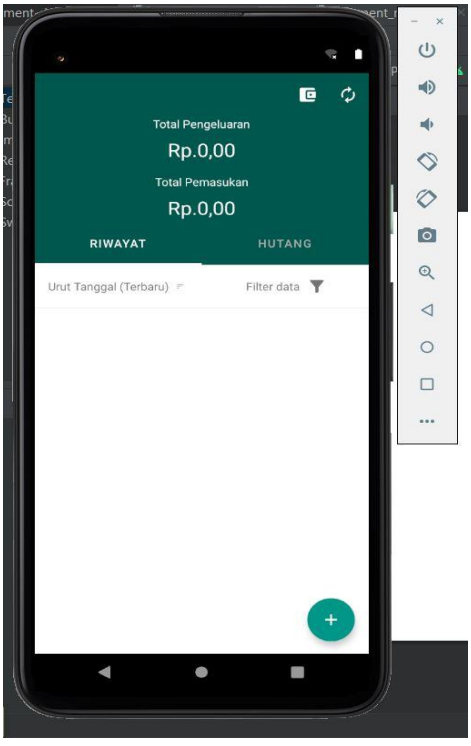
Testing berhasil decompile dan di jalankan pada emulator android studio berikut screenshot aplikasinya :



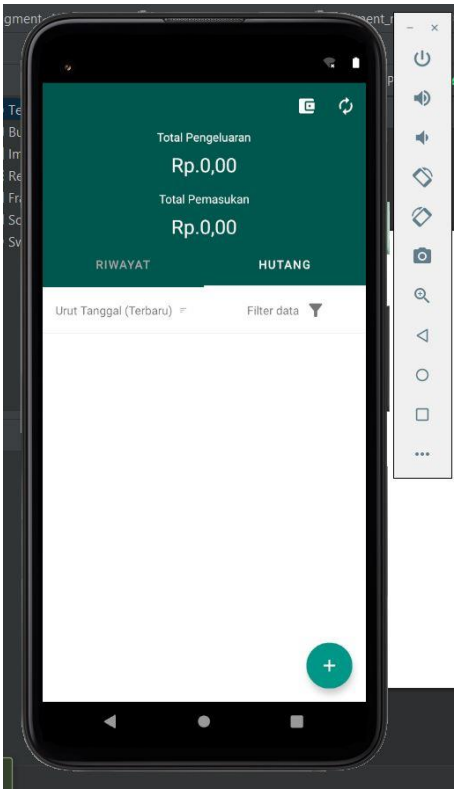
Gambar 3. 1 Tampilan splash screen program



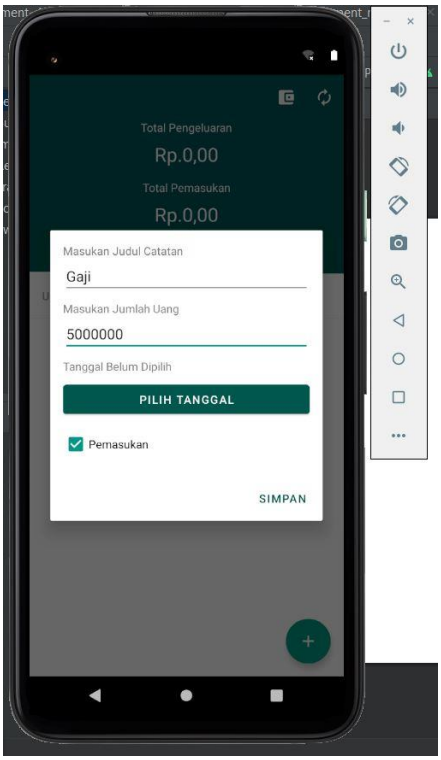
Gambar 3. 2 Tampilan pengenalan aplikasi



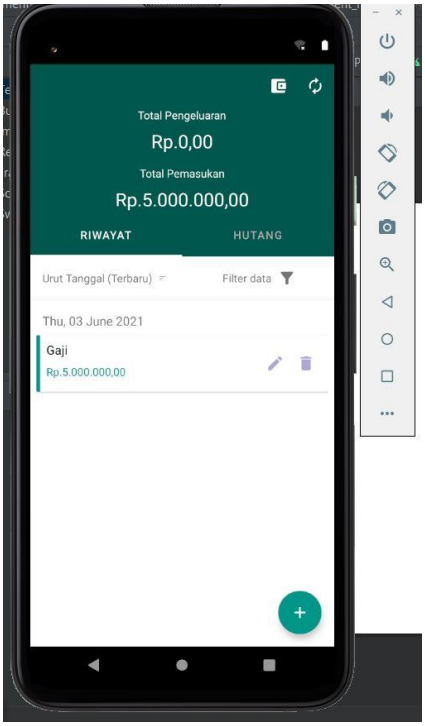
Gambar 3. 3 Tampilan utama halaman riwayat



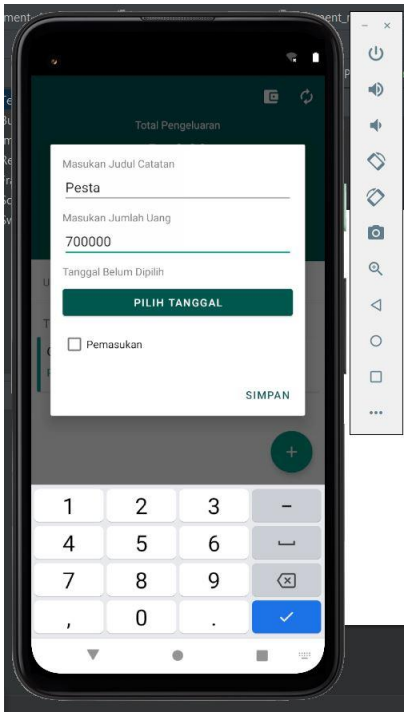
Gambar 3. 4 Tampilan utama halaman hutang



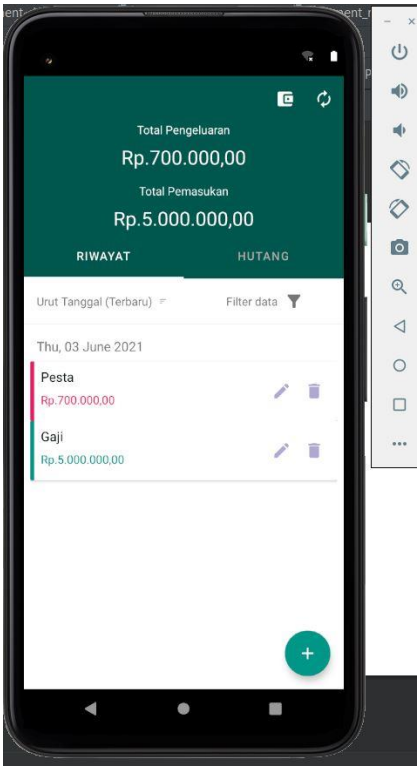
Gambar 3. 5 Tampilan tambah catatan pemasukan halaman riwayat



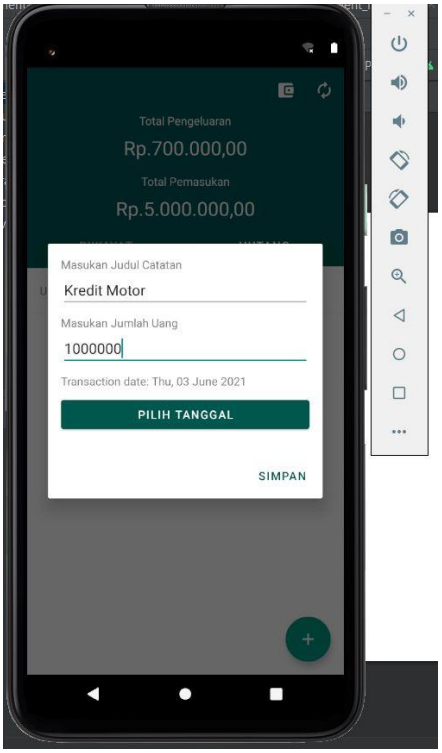
Gambar 3. 6 Tampilan setelah klik simpan



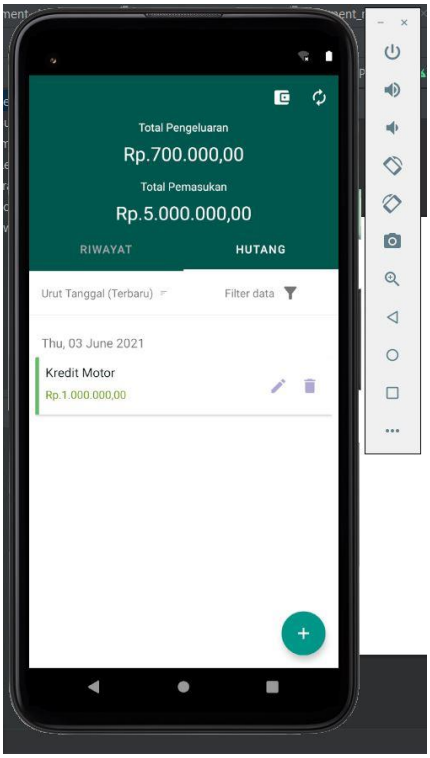
Gambar 3. 7 Tampilan tambah catatan pengeluaran halaman hutang



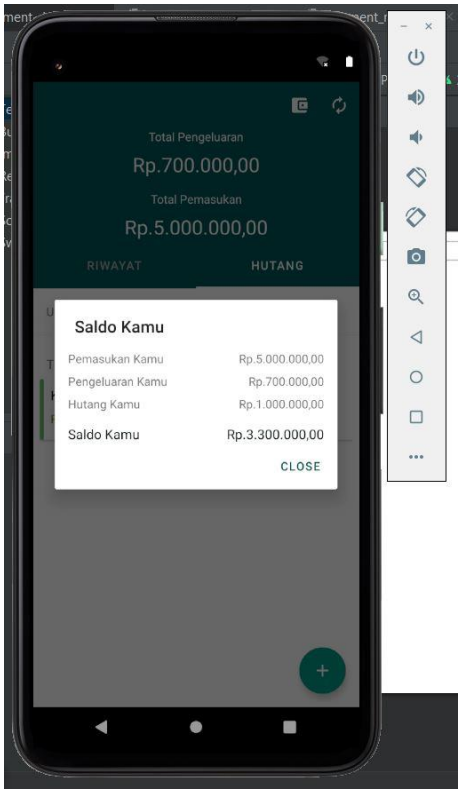
Gambar 3. 8 Tampilan setelah klik simpan



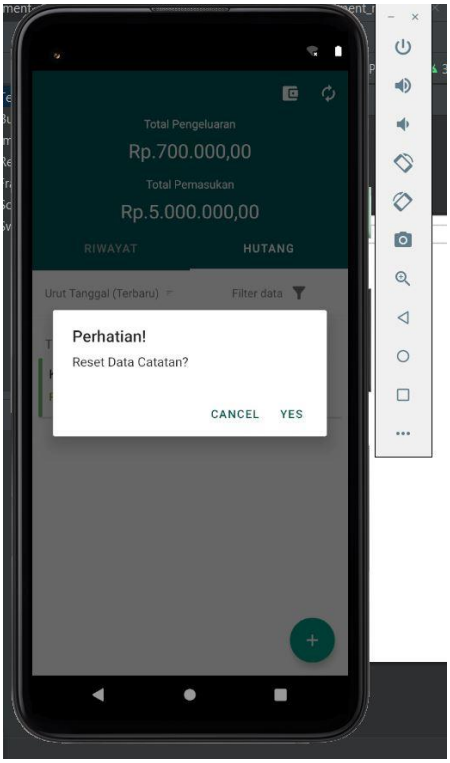
Gambar 3. 9 Tampilan tambah catatan hutang



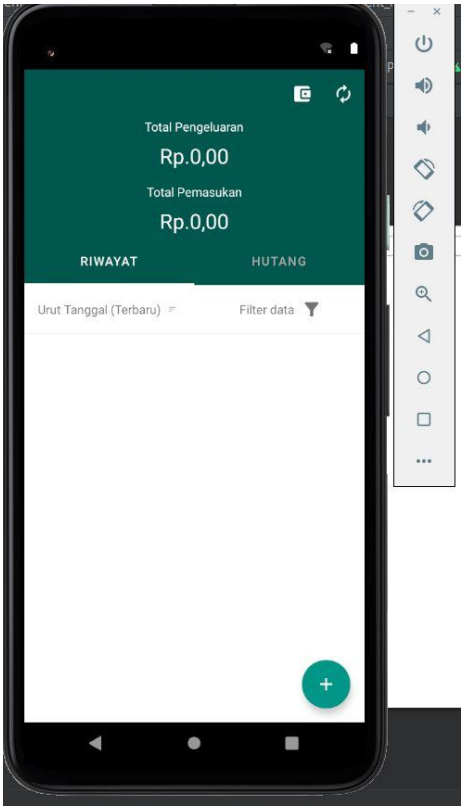
Gambar 3. 10 Tampilan setelah klik simpan



Gambar 3. 11 Tampilan setelah klik report



Gambar 3. 12 Tampilan setelah klik reset



Gambar 3. 13 Tampilan setelah klik yes

3. 5 Debuging

BAB IV
KESIMPULAN DAN SARAN

4. 1 Kesimpulan

Aplikasi ini merupakan aplikasi pencatatan keuangan mandiri dimana user dapat mengatur pemasukan dan pengeluaran yang dapat dicatat secara non fisik di smartphone dimana saja dan kapan saja. Aplikasi ini berisikan Riwayat pendapatan dan pengeluaran serta waktu aktivitasnya.

4. 2 Saran

Bisa di konfigurasikan dengan aplikasi pembayaran online seperti dana, link dan lain-lain