FINAL PROJECT

PENGEMBANGAN APLIKASI BERGERAK

APLIKASI *MONEY TRACKER*

DISUSUN OLEH KELOMPOK 4 :

1. Clementine Dwayani Danitasari (L0123041)
2. Diva Muthi'ah Sholihah (L0123044)
3. Kezia Manuella Tambunan (L0123074)

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI DAN SAINS DATA

UNIVERSITAS SEBELAS MARET

2025

BAB I
PENDAHULUAN

Aplikasi *Money Tracker* merupakan sebuah platform inovatif berbasis aplikasi Android yang dirancang khusus untuk mempermudah pengelolaan uang, baik itu pemasukan maupun pengeluaran. Platform ini hadir sebagai solusi terintegrasi untuk mendukung efektivitas dan efisiensi dalam mengelola serta memonitor berbagai transaksi keuangan pengguna secara *real-time*. Aplikasi ini memungkinkan pengguna untuk mencatat setiap transaksi harian, mingguan, bahkan tahunan, serta mengelompokkan pengeluaran berdasarkan dan melacak perkembangan tabungan dan pengeluaran pribadi.

Dengan adanya fitur-fitur Aplikasi *Money Tracker* ini, diharapkan pengguna bisa menjadi lebih produktif dan lebih mudah dalam mengatur keuangan yang mereka miliki.

1. Membantu mencatat transaksi harian

   Aplikasi *Money Tracker* dapat memudahkan pengguna dalam mencatat pemasukan dan pengeluaran secara rinci dan terstruktur. Semua transaksi dapat dikategorikan, seperti makanan, transportasi, belanja, dan lain-lain, sehingga lebih mudah dianalisis. Aplikasi ini juga menyediakan laporan dan riwayat transaksi untuk evaluasi keuangan secara menyeluruh.
2. Statistik dan grafik pengeluaran

   Aplikasi ini menyajikan visualisasi pengeluaran berdasarkan waktu (mingguan, bulanan, dan tahunan) untuk memantau pola keuangan pengguna.
3. Fitur tabungan

   Dalam aplikasi ini juga tersedia fitur tabungan yang dapat digunakan untuk membantu pengguna menetapkan tujuan menabung, menghitung jumlah yang harus dikumpulkan, dan memantau progress tabungna.
4. Pengingat pembayaran

   Aplikasi ini membantu mencegah keterlambatan pembayaran dengan notifikasi otomatis sesuai jadwal yang ditentukan.

Desain antarmuka yang *user-friendly* dan fungsional membuat platform ini tidak hanya mudah digunakan, tetapi juga memberikan transparansi yang lebih baik dalam pengelolaan keuangan. Pengguna dari berbagai kalangan, baik itu siswa, mahasiswa, maupun orang tua, dapat merasakan manfaat dari pengelolaan keuangan yang cepat dan akurat. Dengan demikian, *Money Tracker* menjadi alat yang vital dalam mendukung kerapihan dan keteraturan dalam mengatur keuangan.

BAB II
ANALISIS *SOURCE CODE*

A. *Splash Screen*

```
class SplashScreen : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setContent {
            MoneyTrackerTheme {
                Splash()
            }
        }

        // Pindah ke MainActivity setelah 5 detik
        Handler(Looper.getMainLooper()).postDelayed({
            val intent = Intent( packageContext: this, MainActivity::class.java)
            startActivity(intent)
            finish()
        }, delayMillis: 5000)
    }
}
```

Tampilan *Splash Screen* diatur melalui file **SplashScreen.kt** dengan menggunakan fungsi **Handler(looper.getMainLooper()).postDelayed**. Tampilan Splash Screen akan ditampilkan di layar selama 5 detik, kemudian secara otomatis akan dialihkan ke halaman *Home*.

B. Tombol Navigasi

```
@Composable
fun ButtonNav(navController: NavController) {
    val currentDestination = navController.currentBackStackEntryAsState().value?.destination?.route

    val items = listOf("home", "charts", "saving", "payment")
    val labels = listOf("Home", "Charts", "Saving", "Payment")
    val icons = listOf(
        R.drawable.home,
        R.drawable.chart,
        R.drawable.saving,
        R.drawable.payment
    )
```

Tampilan tombol navigasi yang berada di bawah layar diatur di dalam fungsi **fun ButtonNav**. Tombol navigasi ini mencakup tombol *home*, *charts*, *saving*, dan *payment*. Masing-masing tombol memiliki *icon* nya masing-masing yang diatur didalam variabel **val icons**. Sebagai alur navigasinya, diatur didalam variabel **val currentDestination**.

C. Menu *Home*

```
@Composable
fun Home(navController: NavController, name: String, modifier: Modifier = Modifier, viewModel: MainViewModel = viewModel()) {
    // Menyimpan transaksi pada halaman Home
    val transactions = viewModel.transactions?.filter { it.type == "Income" || it.type == "Expense" } ?: emptyList()
    Log.d( tag: "Home",  msg: "Transactions in Home: ${transactions.size}")
    Log.d( tag: "HomeVM",  msg: "Hash: ${viewModel.hashCode()}, Size: ${viewModel.transactions.size}")

    // Menyimpan transaksi per-weekly, monthly, dan yearly
    var selectedPeriod by remember { mutableStateOf( value: "Monthly") }
    Log.d( tag: "DEBUG",  msg: "All transactions: ${transactions.map { it.date }}")
```

Tampilan halaman *home* diatur dalam file **Home.kt** dengan segala fiturnya yang diatur pula di dalam file **Activity.kt**. Fungsi ini memiliki empat parameter, salah satunya adalah **viewModel: MainViewModel = viewModel()** yang berfungsi untuk menambahkan data transaksi di dalamm *home* yang telah dicatat pengguna melalui Activity.kt.

```
// fungsi filter transaksi sesuai periode (weekly, monthly, dan yearly)
fun filterTransactionsByPeriod(period: String, allTransactions: List<Transaction>): List<Transaction> {
    val calendar = Calendar.getInstance()
    val nowTime = calendar.time.time
    return when (period) {
        "Weekly" -> {
            calendar.time = Date(nowTime)
            calendar.set(Calendar.HOUR_OF_DAY, 0)
            calendar.set(Calendar.MINUTE, 0)
            calendar.set(Calendar.SECOND, 0)
            calendar.set(Calendar.MILLISECOND, 0)
            calendar.add(Calendar.DAY_OF_YEAR,  amount: -7)
            val startOfWeek = calendar.time.time

            allTransactions.filter { it.date >= startOfWeek && it.date < nowTime }
        }
        "Monthly" -> {
            calendar.time = Date(nowTime)
            calendar.set(Calendar.DAY_OF_MONTH, 1)
            calendar.set(Calendar.HOUR_OF_DAY, 0)
            calendar.set(Calendar.MINUTE, 0)
            calendar.set(Calendar.SECOND, 0)
            calendar.set(Calendar.MILLISECOND, 0)
            val startOfMonth = calendar.time.time

            calendar.add(Calendar.MONTH,  amount: 1)
            val startOfNextMonth = calendar.time.time

            allTransactions.filter { it.date >= startOfMonth && it.date < startOfNextMonth }
        }
        "Yearly" -> {
            calendar.time = Date(nowTime)
            calendar.set(Calendar.MONTH, 0)
            calendar.set(Calendar.DAY_OF_MONTH, 1)
            calendar.set(Calendar.HOUR_OF_DAY, 0)
            calendar.set(Calendar.MINUTE, 0)
            calendar.set(Calendar.SECOND, 0)
            calendar.set(Calendar.MILLISECOND, 0)
            val startOfYear = calendar.time.time
            calendar.add(Calendar.YEAR,  amount: 1)
            val startOfNextYear = calendar.time.time
            allTransactions.filter { it.date >= startOfYear && it.date < startOfNextYear }
        }
        else -> allTransactions
    }
}
```

Fitur rentang waktu (*Weekly*, *Monthly*, dan *Yearly*) diatur dalam fungsi **filterTransactionsByPeriod**. Ketika pengguna menekan tombol *Weekly*, maka aplikasi akan menampilkan segala transaksi yang berada di minggu tersebut. Apabila pengguna menekan tombol *Monthly*, maka aplikasi akan menampilkan segala transaksi yang berada di bulan tersebut. Apabila pengguna menekan tombol

Yearly, maka aplikasi akan menampilkan segala transaksi yang berada di tahun tersebut.

```
// Transaction List
Spacer(modifier = Modifier.height(10.dp))
val filteredTransactions = filterTransactionsByPeriod(selectedPeriod, transactions)
val groupedTransactions = filteredTransactions.groupBy {
    SimpleDateFormat( pattern: "EEE, dd/MM", Locale.getDefault()).format(Date(it.date))
}
```

Untuk mengatur munculnya hasil dari pemilihan *Weekly*, *Monthly*, maupun *Yearly*, diatur dalam variabel **val filteredTransactions** dan **val groupedTransactions**.

D. Activity.kt

```
@Dao
interface TransactionDao {

    @Insert
    suspend fun insertTransaction(transaction: Transaction)

    @Update
    suspend fun updateTransaction(transaction: Transaction)

    @Delete
    suspend fun deleteTransaction(transaction: Transaction)

    @Query("SELECT * FROM transactions ORDER BY date DESC")
    suspend fun getAllTransactions(): List<Transaction>
}
```

TransactionDao merupakan antarmuka DAO yang berfungsi untuk mengelola data pada tabel **transactions** di *database*. Aplikasi menyediakan fungsi untuk menyisipkan (**insertTransaction**), memperbarui (**updateTransaction**), dan menghapus (**deleteTransaction**) data transaksi. Selain itu, fungsi **getAllTransactions()** digunakan untuk mengambil seluruh data transaksi dan mengurutkannya berdasarkan tanggal dari yang terbaru.

```
@Composable
fun Activity(navController: NavController, viewModel: MainViewModel = viewModel()) {
    var selectedTab by remember { mutableStateOf( value: 0) }
    var calculatorInput by remember { mutableStateOf( value: "") }
    var categoryEmoji by remember { mutableStateOf( value: "") }
    var categoryText by remember { mutableStateOf( value: "") }
    var categoryName by remember { mutableStateOf( value: "") }
    var showEmojiPicker by remember { mutableStateOf( value: false) }
    var selectedDate by remember { mutableStateOf(Calendar.getInstance()) }
    var showDeleteConfirmation by remember { mutableStateOf( value: false) }
    var categoryToDelete by remember { mutableStateOf<CategoryData?>( value: null) }

    // Goal dialog states
    var showGoalDialog by remember { mutableStateOf( value: false) }
    var goalAmount by remember { mutableStateOf( value: "") }

    // Separate state for each tab's selected category
    var selectedExpenseCategory by remember { mutableStateOf( value: "") }
    var selectedIncomeCategory by remember { mutableStateOf( value: "") }
    var selectedSavingCategory by remember { mutableStateOf( value: "") }

    // State to store transactions
    var transactions by remember { mutableStateOf<List<Transaction>>(emptyList()) }

    val context = LocalContext.current
```

Fungsi di atas merupakan fungsi **fun Activity** yang merupakan bagian dari Jetpack Compose. Fungsi ini menyimpan infromasi tentang tab yang sedang aktif (**selectedTab**), *input* kalkulator (**calculatorInput**), kategori transaksi beserta emoji dan nama kategori (**categoryEmoji**, **categoryText**, **categoryName**), serta tanggal yang dipilih (**selectedDate**). Fungsi ini juga mengelola *state* terpisah untuk masing-masing kategori transaksi (pengeluaran, pemasukan, dan tabungan) dengan variabel **selectedExpenseCategory**, **selectedIncomeCategory**, dan **selectedSavingCategory**. Terakhir, daftar semua transaksi yang dicatat oleh pengguna disimpan dalam **transactions**, yang akan digunakan untuk ditampilkan atau dimodifikasi selama aplikasi dijalankan.

```
// Date picker dialog setup
fun showDatePicker() {
    val dialog = DatePickerDialog(
        context,
        { _: DatePicker, year: Int, month: Int, day: Int ->
            val newCal = Calendar.getInstance()
            newCal.set(year, month, day)
            selectedDate = newCal
        },
        selectedDate.get(Calendar.YEAR),
        selectedDate.get(Calendar.MONTH),
        selectedDate.get(Calendar.DAY_OF_MONTH)
    )
    dialog.show()
}
```

Di dalam fungsi Activity terdapat fungsi **fun showDatePicker()** yang digunakan untuk memilih tanggal ketika pengguna akan memasukkan jenis transaksinya.

```
// Function to save transaction
fun saveTransaction() {
    if (calculatorInput.isEmpty()) {
        Toast.makeText(context, text: "Please enter an amount", Toast.LENGTH_LONG).show()
        return
    }
    if (getCurrentSelectedCategory().isEmpty()) {
        Toast.makeText(context, text: "Please select a category", Toast.LENGTH_LONG).show()
        return
    }
```

Di dalam fungsi Activity pula terdapat fungsi **fun saveTransaction()** yang berguna untuk menyimpan segala transaksi yang dilakukan pengguna di dalam Activity.kt.

```
// Function to save transaction with goal for Saving tab
fun saveTransactionWithGoal() {
    try {
        val amount = calculatorInput.toDoubleOrNull()
        val goal = goalAmount.toDoubleOrNull()

        if (amount == null || amount <= 0) {
            Toast.makeText(context, text: "Please enter a valid amount", Toast.LENGTH_LONG).show()
            return
        }

        val newTransaction = Transaction(
            type = "Saving",
            category = getCurrentSelectedCategory(),
            amount = amount,
            date = selectedDate.timeInMillis,
            goalAmount = goal // Include goalAmount in the Transaction
        )

        // Add to ViewModel
        viewModel.addTransaction(newTransaction)

        // Reset form
        calculatorInput = ""
        updateSelectedCategory("")
        categoryEmoji = ""
        categoryText = ""
        goalAmount = ""
        showGoalDialog = false

        Toast.makeText(context, text: "Saving goal created!", Toast.LENGTH_SHORT).show()

        // Navigate to saving after saving
        navController.navigate( route: "saving") {
            popUpTo( route: "saving") { inclusive = true }
        }

    } catch (e: Exception) {
        Toast.makeText(context, text: "Error saving goal: ${e.message}", Toast.LENGTH_LONG).show()
    }
}
```

Terdapat pula fungsi **fun saveTransactionWithGoal** di dalam fungsi Activity. Fungsi ini bertujuan untuk menyimpan data tabungan atau *saving* pengguna, sekaligus memperbarui target tabungan yang telah ditentukan. Saat pengguna memasukkan jumlah uang yang ingin ditabung, data tersebut akan direkam sebagai transaksi dan disimpan ke dalam daftar **transactions**. Selain itu, fungsi ini juga menghitung selisih antara target tabungan (**goalAmount**) dan jumlah yang telah ditabung, agar pengguna dapat mengetahui berapa nominal yang masih perlu dikumpulkan untuk mencapai tujuan finansial mereka.

```
// Function to add new category
fun addNewCategory() {
    if (categoryName.isNotEmpty() && categoryEmoji.isNotEmpty()) {
        val newCategory = CategoryData( name: "$categoryEmoji $categoryName", R.drawable.ic_launcher_foreground)
        when (selectedTab) {
            0 -> expenseCategories.add(newCategory)
            1 -> incomeCategories.add(newCategory)
            2 -> savingCategories.add(newCategory)
        }
        categoryName = ""
        categoryEmoji = ""
        showEmojiPicker = false
    }
}
```

Fungsi **addNewCategory** bertujuan untuk menambahkan kategori baru di dalam aktivitas yang dilakukan pengguna. Dalam kasus ini, pengguna juga dapat memilih emoji yang diinginkan, yang diatur di dalam **$categoryEmoji**.

E. Hapus Transaksi

```
@Composable
fun TransactionItem(transaction: Transaction, viewModel: MainViewModel) {
    var showDeleteDialog by remember { mutableStateOf( value: false) }

    val splitCategory = transaction.category.split( ...delimiters: " ", limit = 2)
    val emoji = splitCategory.getOrNull( index: 0) ?: "❓"
    val categoryName = splitCategory.getOrNull( index: 1) ?: transaction.category
```

Gambar di atas merupakan kode pada fungsi **TransactionItem** yang mengatur tampilan satu data transaksi dalam daftar, termasuk pemisahan emoji dan nama kategori dari data **transaction.category**. Fungsi ini juga mengelola *state* untuk menampilkan dialog konfirmasi saat pengguna ingin menghapus transaksi.

```
    if (showDeleteDialog) {
        androidx.compose.material3.AlertDialog(
            onDismissRequest = { showDeleteDialog = false },
            title = { Text( text: "Delete Transaction?") },
            text = { Text( text: "Are you sure you want to delete this transaction?") },
            confirmButton = {
                androidx.compose.material3.TextButton(onClick = {
                    viewModel.deleteTransaction(transaction)
                    showDeleteDialog = false
                }) {
                    Text( text: "Delete", color = Color.Red)
                }
            },
            dismissButton =  {
                androidx.compose.material3.TextButton(onClick = { showDeleteDialog = false}) {
                    Text( text: "Cancel")
                }
            }
        )
    }
}
```

Aplikasi juga mengaktifkan fitur *delete* atau hapus transaksi apabila nantinya terdapat kesalahan ketika pengguna mencatat transaksi. Fitur *delete* ini berada di dalam fungsi **TransactionItem** yang masih berada di dalam file **Home.kt**.

F. Menu *Saving*

```
@Composable
fun Saving(navController: NavController, viewModel: MainViewModel = viewModel()) {
```

Fungsi **fun Saving** yang disusun dalam file **Saving.kt** memiliki 2 parameter, yaitu **navController: NavController** dan **viewModel: MainViewModel = viewModel()**. Fungsi **navController** digunakan sebagai perubahan tombol navigasi dari satu halaman ke halaman berikutnya. Fungsi **viewModel** digunakan sebagai tampilan yang menyimpan data yang bersifat dinamis dan dapat berubah seiring waktu.

G. Menu *Chart*

```
@SuppressLint("UnrememberedMutableState")
@Composable
fun Chart(navController: NavController, viewModel: MainViewModel = viewModel()) {
    var selectedPeriod by remember { mutableStateOf( value: "Monthly") }

    val transactions by rememberUpdatedState(newValue = viewModel.transactions.filter { it.type == "Expense" })
    val filteredTransactions by derivedStateOf { filterTransactionsByPeriod(selectedPeriod, transactions) }

    val total = filteredTransactions.sumOf { it.amount }
    val grouped = filteredTransactions.groupBy { it.category }.mapValues { it.value.sumOf { it.amount } }

    // Ambil warna dari ViewModel
    val categoryColors = grouped.keys.associateWith { viewModel.getCategoryColor(it) }
```

Menu Chart disusun di dalam file **Chart.kt**. Fungsi ini memiliki dua parameter, yaitu **navController:NavController** dan **viewModel: MainViewModel = viewModel()**, yang memungkinkan akses terhadap navigasi dan data aplikasi. Di dalam fungsi ini, terdapat variabel **val categoryColors** yang berfungsi untuk menghasilkan warna secara acak yang nantinya digunakan untuk membedakan setiap kategori dalam tampilan diagram *chart*.

```
fun filterTransactionsByPeriod(period: String, allTransactions: List<Transaction>): List<Transaction> {
    val calendar = Calendar.getInstance()
    val nowTime = calendar.time.time
    return when (period) {
```

Terdapat pula fungsi **fun filterTransactionByPeriod** yang digunakan untuk mengatur data dalam halaman *weekly*, *monthly*, dan *yearly*.

```
@Composable
fun PieChartWithLegend(data: Map<String, Double>, total: Double, colors: Map<String, Color>) {
    if (data.isEmpty()) {
        Text( text: "No expense data", color = Color.Gray, fontSize = 16.sp)
        return
    }
    Canvas(modifier = Modifier.size(200.dp)) {
        var startAngle = 0f
        data.entries.forEach { entry ->
            val sweepAngle = (entry.value / total * 360).toFloat()
            drawArc(
                color = colors[entry.key] ?: Color.Gray,
                startAngle = startAngle,
                sweepAngle = sweepAngle,
                useCenter = true
            )
            startAngle += sweepAngle
        }
    }
}
```

Fungsi **PieChartWithLegend** digunakan untuk menampilkan diagram *chart* berdasarkan data keuangan yang diberikan. Fungsi ini menerika tiga parameter, yaitu data, total, dan warna. Jika data kosong, akan ditampilkan teks "*No expense data*". Jika ada data, komponen **Canvas** akan menggambar *pie chart* dengan menghitung sudut tiap bagian berdasarkan proporsi nilai terhadap total dan memberi warna sesuai kategori.

```
@Composable
fun ChartDetailRow(category: String, amount: Double, total: Double, color: Color) {
    val percentage = if (total == 0.0) 0 else (amount / total * 100).roundToInt()
```

Fungsi **ChartDetailRow** digunakan untuk menghitung dan menampilkan persentase kontribusi suatu kategori terhadap total keseluruhan data keuangan.

Fungsi ini menerima parameter berupa nama kategori (**category**), jumlah nominal dalam kategori tersebut (**amount**), total seluruh jumlah (**total**), serta warna (**color**) yangn digunakan untuk elemen visual. Di dalamnya, nilai persentase dihitung dengan membagi *amount* terhadap total, dikalikan 100, dan dibulatkan ke bilangan bulat terdekat. Jika total bernilai nol, maka persentasenya otomatis diset menjadi nol untuk menghindari pembagian dengan nol.

H. Menu *Payment*

```
@Entity(tableName = "payments")
data class PaymentEntity(
    @PrimaryKey
    val id: String = UUID.randomUUID().toString(),
    val icon: String,
    val title: String,
    val reminder: String,
    val totalPayment: String,
    val date: String
)
```

PaymentEntity merupakan kelas data yang merepresentasikan entitas tabel **payments** dalam *database Room*. Setiap objek menyimpan informasi pembayaran seperti **id** unik secara otomatis (**UUID**), **icon** (emoji), **title** (judul pembayaran), **reminder** (pengulangan pembayaran), **totalPayment** (jumlah yang harus dibayar), dan **date** (tanggal pembayaran).

```kotlin
@Dao
interface PaymentDao {
    @Query("SELECT * FROM payments")
    suspend fun getAllPayments(): List<PaymentEntity>

    @Insert
    suspend fun insertPayment(payment: PaymentEntity)

    @Delete
    suspend fun deletePayment(payment: PaymentEntity)
}
```

PaymentDao merupakan antarmuka *Data Access Object* (DAO) yang menyediakan metode untuk mengakses dan mengelola data pada tabel **payments**. Terdapat tiga fungsi utama, **getAllPayments()** untuk mengambil seluruh data pembayaran, **insertPayment()** untuk menambahkan data pembayaran baru, dan **deletePayment()** untuk menghapus data pembayaran tertentu.

```kotlin
@Composable
fun Payment(navController: NavController, viewModel: MainViewModel = viewModel()) {
```

Fungsi **Payment** menerima dua parameter yaitu **navController: NavController** dan **viewModel: MainViewModel = viewModel()**.

```kotlin
@Composable
fun PaymentPage(viewModel: MainViewModel) {
    var paymentToDelete by remember { mutableStateOf<PaymentEntity?>( value: null) }
```

Fungsi **PaymentPage** menampilkan halaman pembayaran dalam aplikasi. Di dalamnya terdapat variabel **paymentToDelete** yang menggunakan **remember** untuk menyimpan data pembayaran dari **PaymentEntity** yang akan dihapus. Variabel ini bersifat *mutable state*, sehingga perubahan nilainya akan secara otomatis memperbarui tampilan UI.

```kotlin
@Composable
fun InputPayment(navController: NavController, viewModel: MainViewModel = viewModel()) {
```

Dalam meng-*input payment* menggunakan fungsi **InputPayment** yang terdapat dalam file **InputPayment.kt**. Fungsi ini menerima **navController** untuk navigasi antar layar dan **viewModel** sebagai sumber data dan logika bisnis.

```
@Composable
fun InputPaymentPage(navController: NavController, viewModel: MainViewModel) {
    val titleState = remember { mutableStateOf( value: "") }
    val selectedEmoji = remember { mutableStateOf( value: "💳") }
    val paymentState = remember { mutableStateOf( value: "") }
    val dateState = remember { mutableStateOf( value: "") }
    val context = LocalContext.current
    val selectedDate = remember { mutableStateOf(Calendar.getInstance()) }
    var showDatePicker by remember { mutableStateOf( value: false) }
    val reminderState = remember { mutableStateOf( value: "Every Day") }
    var expanded by remember { mutableStateOf( value: false) }
    var isSelected by remember { mutableStateOf( value: false) }
```

Fungsi **InputPaymentPage** digunakan untuk menyimpan *input* pengguna seperti judul pembayaran (**titleState**), emoji kategori (**selectedEmoji**), nominal pembayaran (**paymentState**), tanggal (**dateState** dan **selectedDate**), pengulangan pengingat (**reminderState**), dan kontrol tampilan *dropdown* serta *date picker*.

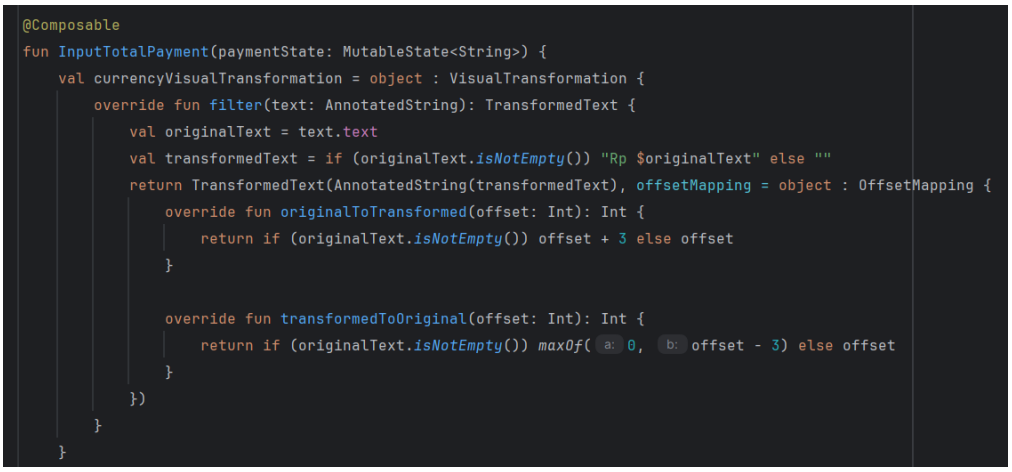
```
@Composable
fun InputTitle(titleState: MutableState<String>) {
```

Fungsi **InputTitle** digunakan untuk menerima *input* dari pengguna ketika memasukkan judul yang akan akan dimasukkan ke dalam halaman *Payment*.

```
@Composable
fun InputIcon(selectedEmoji: MutableState<String>) {
    val emojiList = listOf(
        "💳", "🏫", "⚡", "💧", "🏠", "🚗", "🐕", "💊"
    ).filter { it.isNotEmpty() }

    var showDialog by remember { mutableStateOf( value: false) }
```

Fungsi **InputIcon** digunakan untuk menerima *input* yang dimasukkan oleh pengguna ketika memasukkan data ke dalam halaman *Payment*.



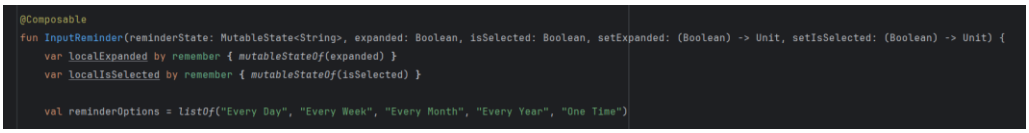
```
@Composable
fun InputTotalPayment(paymentState: MutableState<String>) {
    val currencyVisualTransformation = object : VisualTransformation {
        override fun filter(text: AnnotatedString): TransformedText {
            val originalText = text.text
            val transformedText = if (originalText.isNotEmpty()) "Rp $originalText" else ""
            return TransformedText(AnnotatedString(transformedText), offsetMapping = object : OffsetMapping {
                override fun originalToTransformed(offset: Int): Int {
                    return if (originalText.isNotEmpty()) offset + 3 else offset
                }

                override fun transformedToOriginal(offset: Int): Int {
                    return if (originalText.isNotEmpty()) maxOf( a: 0,  b: offset - 3) else offset
                }
            })
        }
    }
```

Fungsi **InputTotalPayment** digunakan untuk memformat *input* nominal pembayaran agar tampil dengan awalan "Rp" di depan angka. Fungsi ini memakai **VisualTransformation** untuk mengubah tampilan teks secara visual tanpa mengubah data aslinya, serta mengatur posisi kursor dengan **OffsetMapping** agar tetap sinkron saat pengguna mengetik.

```
@Composable
fun InputDate(dateState: MutableState<String>, context: Context, selectedDate: MutableState<Calendar>, showDatePicker: Boolean, setShowDatePicker: (Boolean) -> Unit) {
    var localShowDatePicker by remember { mutableStateOf(showDatePicker) }
```

Fungsi **InputData** digunakan untuk menampilkan dan mengatur *input* tanggal pada aplikasi. Fungsi ini menerima beberapa parameter, seperti **dateState** untuk menyimpan tanggal yang dipilih, **context** untuk akses konteks Android, serta **selectedDate** dan **showDatePicker** untuk mengatur pemilihan tanggal. Variabel **localShowDatePicker** menyimpan status apakah *date picker* sedang ditampilkan, dan dikontrol dengan **remember** agar UI bisa merespons perubahan secara otomatis.



```
@Composable
fun InputReminder(reminderState: MutableState<String>, expanded: Boolean, isSelected: Boolean, setExpanded: (Boolean) -> Unit, setIsSelected: (Boolean) -> Unit) {
    var localExpanded by remember { mutableStateOf(expanded) }
    var localIsSelected by remember { mutableStateOf(isSelected) }

    val reminderOptions = listOf("Every Day", "Every Week", "Every Month", "Every Year", "One Time")
```

Fungsi **InputReminder** digunakan untuk menampilkan dan mengatur opsi pengingat pembayaran dalam bentuk *dropdown* menu. Di dalamnya terdapat **localExpanded** dan **localIsSelected** untuk mengontrol apakah menu pengingat sedang terbuka dan apakah opsi telah terpilih daftar **reminderOptions** berisi pilihan pengingat seperti "*Every Day*", "*Every Week*", hingga "*One Time*", yang akan ditampilkan kepada pengguna.

I. AppDatabase.kt

```
@Database(entities = [Transaction::class, PaymentEntity::class], version = 2, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {
    abstract fun transactionDao(): TransactionDao
    abstract fun paymentDao(): PaymentDao

    companion object {
        @Volatile
        private var INSTANCE: AppDatabase? = null

        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized( lock: this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java,
                     name: "money_tracker_db"
                )
                    .fallbackToDestructiveMigration()
                    .build()
                INSTANCE = instance
                instance ^synchronized
            }
        }
    }
}
```

Kelas **AppDatabase** merupakan konfigurasi *database* utama untuk aplikasi *Money Tracker* menggunakan *Room*. Kelas ini mendefinisikan dua DAO, yaitu **transactionDao()** dan **paymentDao()**, yang digunakan untuk mengakses data transaksi dan pembayaran. *Database* ini bersifat *singleton* melalui fungsi **getDatabase()**, yang memastikan hanya ada satu *instance database* selama aplikasi berjalan. Metode **fallbackToDestructiveMigration()** digunakan agar *database* tetap bisa dibangun ulang jika ada perubahan versi skema.

BAB III

ARSITEKTUR APLIKASI

Aplikasi *Money Tracker* ini terdiri dari beberapa komponen arsitektur yang saling mendukung dalam menjalankan fungsinya. Cara penggunaannya cukup mudah, yaitu:

1. **Menyalakan *Handphone* Android**

   Pengguna dapat menyalakan *handophone* Android yang dimiliki.

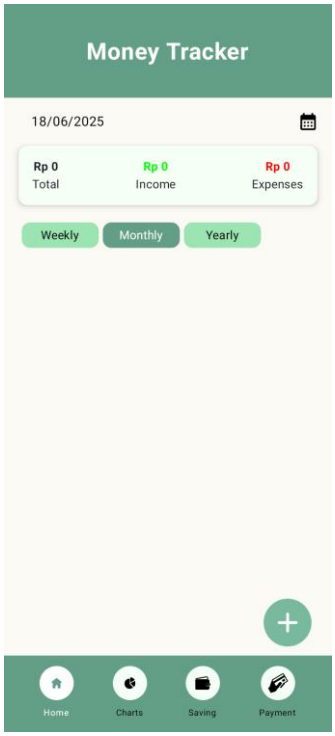
2. **Buka aplikasi *Money Tracker***

   Pengguna dapat membuka aplikasi *Money Tracker* melalui perangkat Android. Saat dijalankan, aplikasi akan menampilkan *Splash Screen* sebelum masuk ke halaman *Home*.

Gambar tampilan splash screen

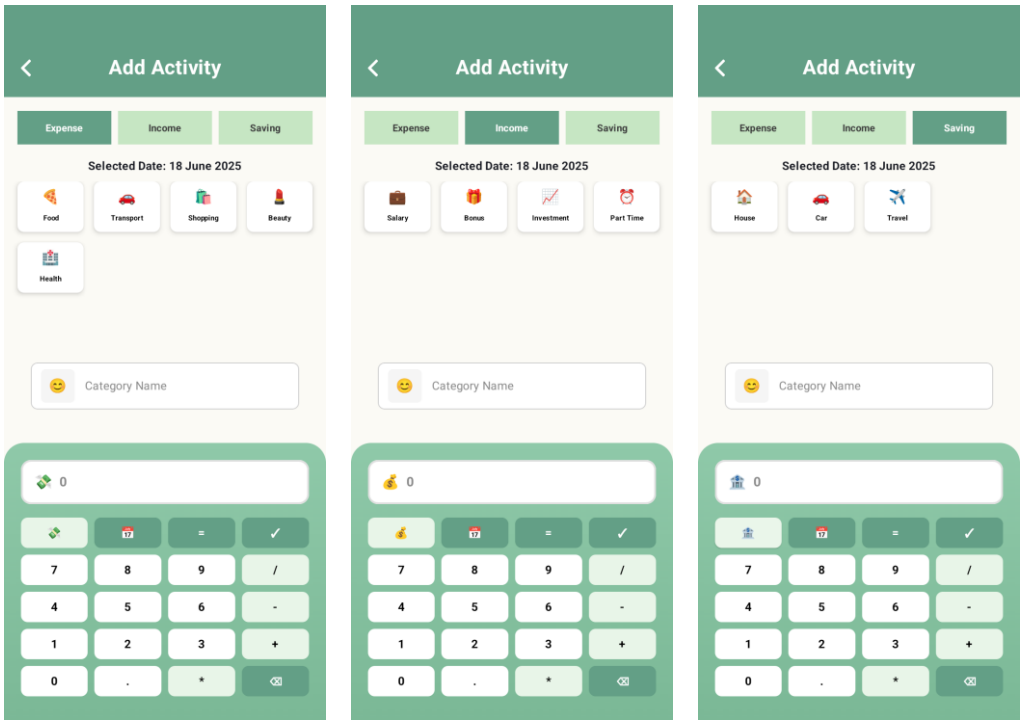
3. **Tampilan awal aplikasi**

Setelah itu pengguna akan otomatis masuk ke dalam tampilan awal aplikasi yaitu ***Home***.



Gambar tampilan home money tracker

Pada halaman ini, pengguna dapat memanfaatkan segala fiturnya melalui melalui tombol plus yang berada di pojok kanan bawah layar.

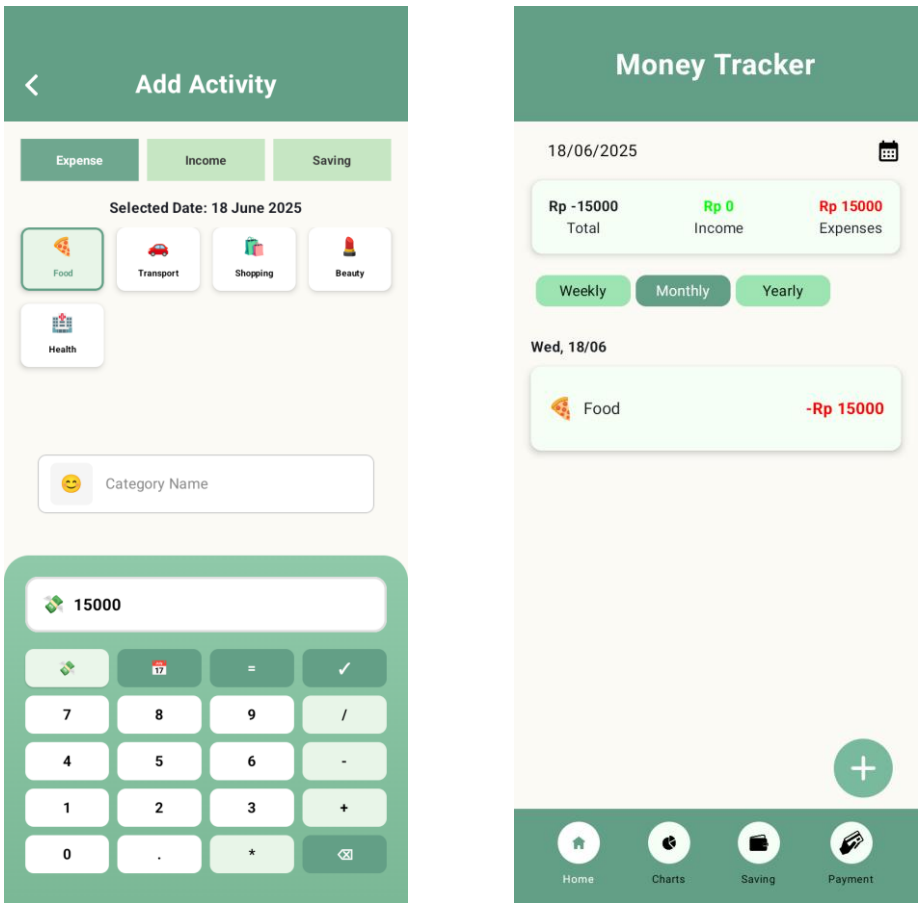


Gambar tampilan add activity

Berdasarkan gambar di atas, pengguna dapat mencatat pengeluaran pada bagian ***Expense***, mencatat pendapatan pada bagian ***Income***, dan mencatat tabungan pada bagian ***Saving***.

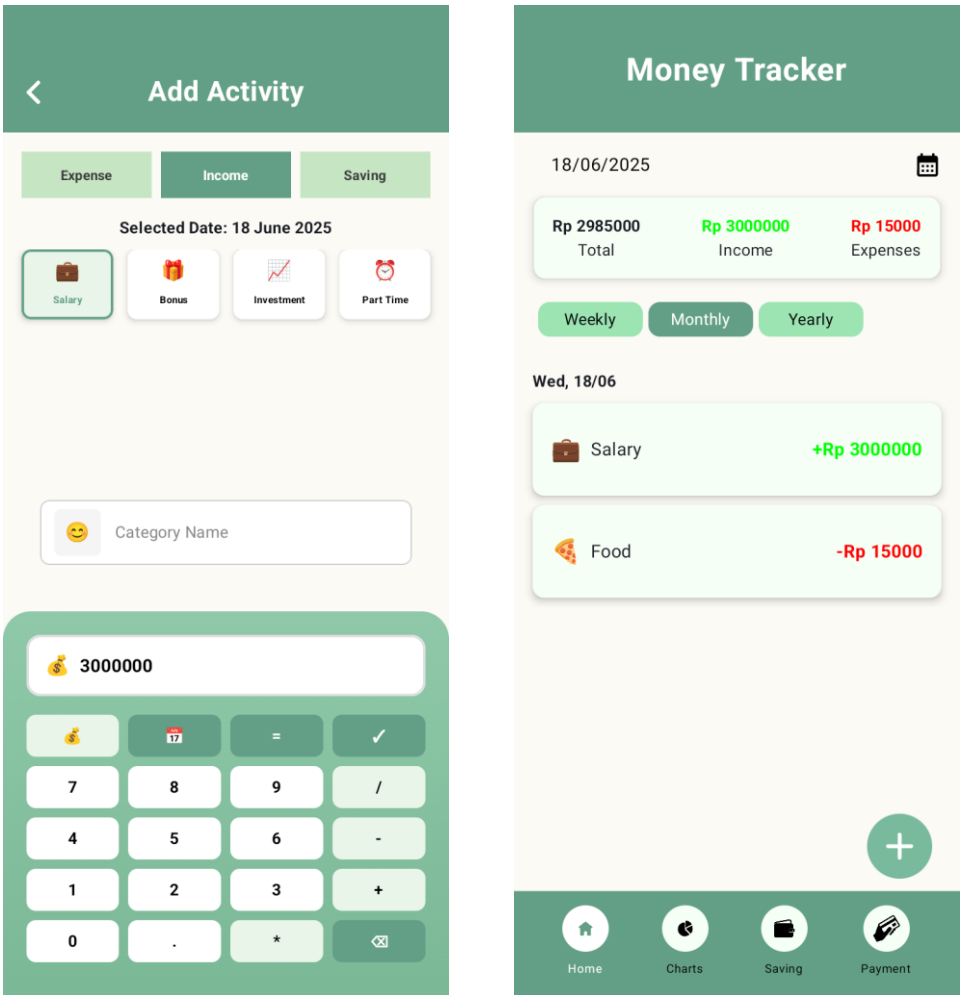
Pada bagian *Expense*, pengguna dapat memilih kategori pengeluaran yang telah disediakan, seperti *Food* (makanan), *Transport* (transportasi), *Shopping* (belanja), *Beauty* (kecantikan), dan *Health* (kesehatan). Selain kategori tersebut, pengguna juga dapat menambahkan kategori baru dengan mengisi kolom "*Category Name*" dan menyisipkan emoji ebagai ikon kategori. Setelah itu, pengguna dapat memasukkan nominal pengeluaran pada kolom angka yang tersedia di bawahnya.

Hal yang sama juga berlaku untuk bagian *Income* dan *Saving*. Pada bagian *Income*, pengguna dapat memilih kategori pendapatan seperti *Salary* (gaji), Bonus, *Investment* (investasi), dan *Part Time* (kerja sampingan). Sementara itu, pada bagian *Saving*, tersedia kategori tabungan seperti *Home* (rumah), *Car* (mobil), dan *Travel* (liburan). Pengguna juga dapat menambahkan kategori baru sesuai kebutuhan, lengkap dengan nama dan emoji sebagai identitas kategori.



Gambar tampilan Expense

Gambar di atas merupakan contoh visualisasi bagaimana pengguna mencatat pengeluaran dan melihat ringkasan keuangan dalam aplikasi *Money Tracker*. Setelah pengguna menyimpan transaksi, data tersebut akan langsung tercatat dan ditampilkan pada layar *Home* sesuai dengan tanggal yang dipilih. Informasi transaksi yang tersimpan mencakup kategori, jumlah pengeluaran, serta ikon yang mewakili kategori tersebut, sehingga memudahkan pengguna dalam memantau aktivitas keuangan secara cepat dan efisien.



Gambar tampilan Income

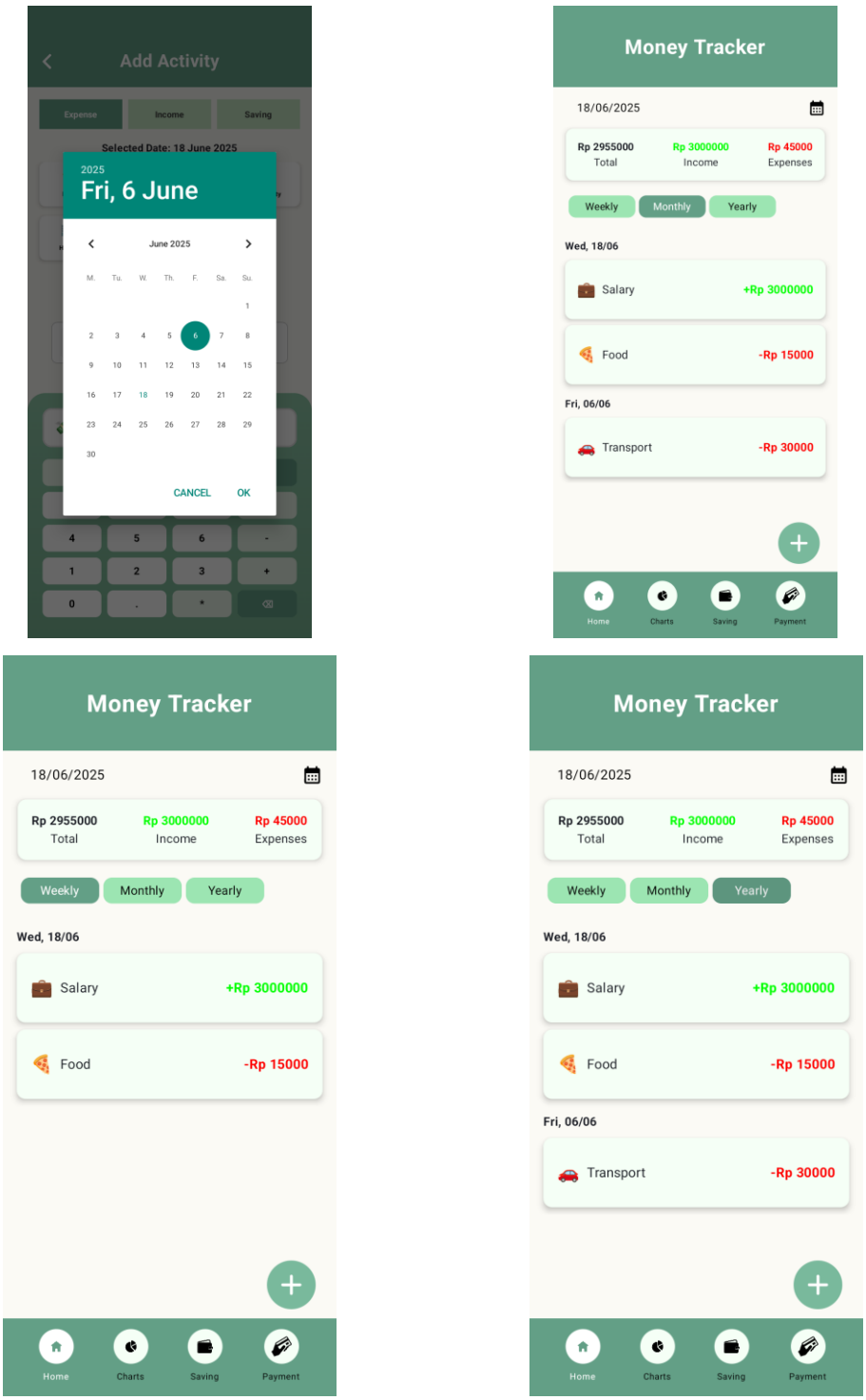
Gambar di atas merupakan contoh visualisasi bagaimana pengguna mencatat pendapatan dan melihat ringkasan keuangan dalam aplikasi *Money Tracker*. Setelah pengguna menyimpan transaksi, data tersebut akan langsung tercatat dan ditampilkan pada layar *Home* sesuai dengan tanggal yang dipilih.

Selain itu, box paling atas di halaman *Home* juga menampilkan ringkasan keuangan yang terdiri dari tiga komponen utama, yaitu:

- Total : menunjukkan saldo akhir yang dimiliki pengguna, yaitu sebesar Rp 2.985.000. Nilai ini diperoleh dari pengurangan total *Income* dan *Expenses*.

- *Income* : menunjukkan total pemasukan yang telah dicatat pengguna, yaitu Rp 3.000.000. Angka ini ditampilkan dalam warna hijau sebagai penanda positif.
- *Expense* : menunjukkan total pengeluaran pengguna, yaitu Rp 15.000, yang ditampilkan dalam warna merah sebagai penanda pengurangan saldo.

4. Fitur rentang waktu

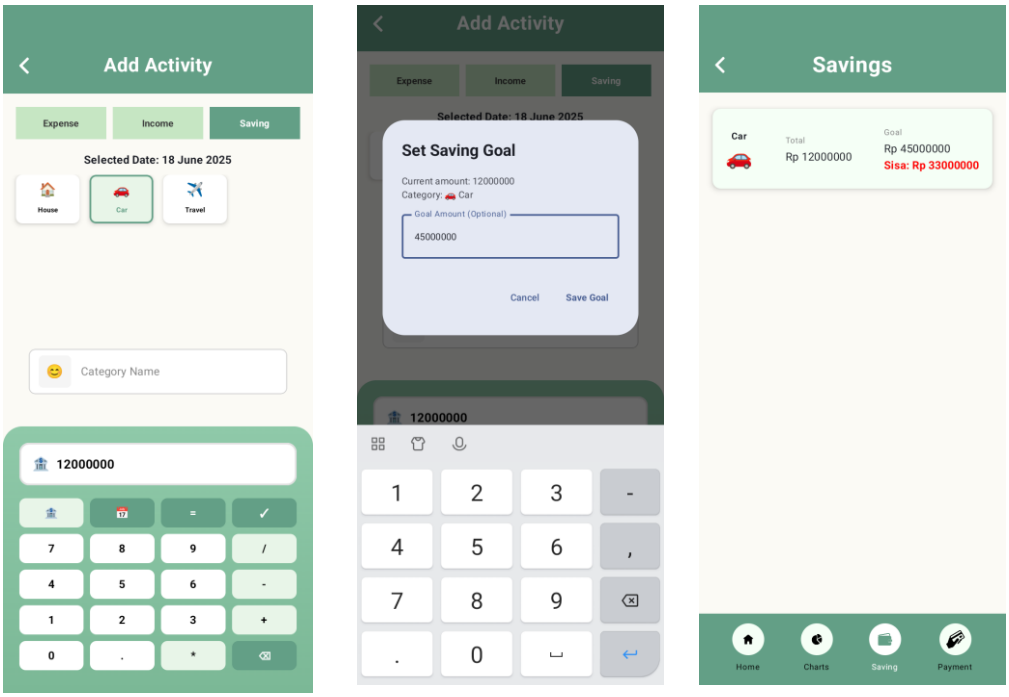


Gambar tampilan fitur rentang waktu

Gambar di atas merupakan contoh visualisasi proses pencatatan transaksi dan tampilan ringkasan keuangan dalam aplikasi *Money Tracker*. Gambar pertama menunjukkan fitur pemilihan tanggal saat pengguna menambahkan aktivitas keuangan (Fri, 6 June). Gambar-gambar berikutnya menampilkan halaman *Home* yang memperlihatkan detail transaksi serta filter waktu (*Weekly*, *Monthly*, *Yearly*) untuk menyesuaikan rentang tampilan data.

5. Menu *Saving*

Menu *Saving* masih berhubungan erat dengan tampilan di *activity* sebelumnya. Menu *Saving* bertujuan untuk membantu pengguna dalam merencanakan dan mengelola tabungan secara lebih terstruktur. Melalui menu ini, pengguna dapat menetapkan tujuan tabungan, seperti untuk membeli rumah, mobil, atau merencanakan liburan. Selain itu, pengguna juga dapat menentukan target jumlah uang yang ingin dikumpulkan dan memantau perkembangan tabungan dari waktu ke waktu. Dengan adanya fitur ini, pengguna dapat lebih focus dan disiplin dalam mencapai tujuan finansialnya.



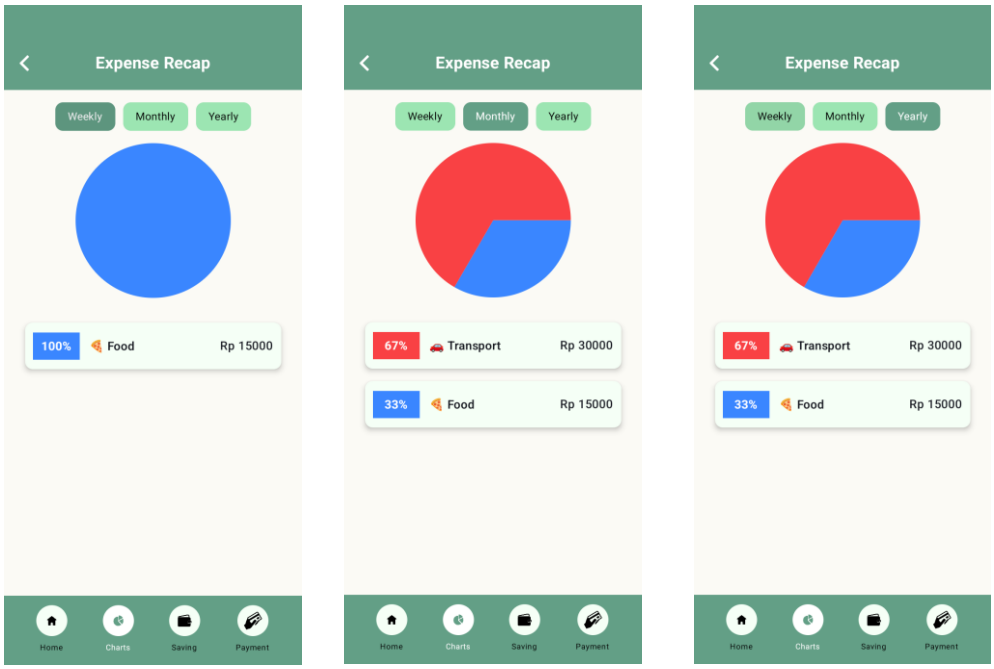
Gambar tampilan Saving

Gambar di atas merupakan contoh visualisasi bagaimana pengguna mencatat tabungan dan memantau progresnya dalam aplikasi *Money Tracker*. Setelah pengguna menyimpan data tabungan, informasi tersebut akan langsung tercatat dan ditampilkan pada layar *Saving*, lengkap dengan tujuan tabungan (*Car*), jumlah yang sudah terkumpul (Rp 12.000.000), serta target yang ingin dicapai (Rp

45.000.000). Aplikasi juga secara otomatis menghitung dan menampilkan sisa nominal yang perlu ditabung, yaitu sebesar Rp 33.000.000, agar tujuan pembelian mobil dapat tercapai.

6. Menu *Chart*

Menu *Chart* bertujuan untuk menampilkan visualisasi data keuangan pengguna dalam bentuk grafik, sehingga memudahkan pengguna dalam memahami pola pengeluaran mereka. Melalui menu ini, pengguna juga dapat melihat perbandingan berbagai kategori pengeluaran dalam rentang waktu tertentu (mingguan, bulanan, atau tahunan), sehingga membantu dalam pengambilan keputusan dan perencanaan keuangan yang lebih bijak.



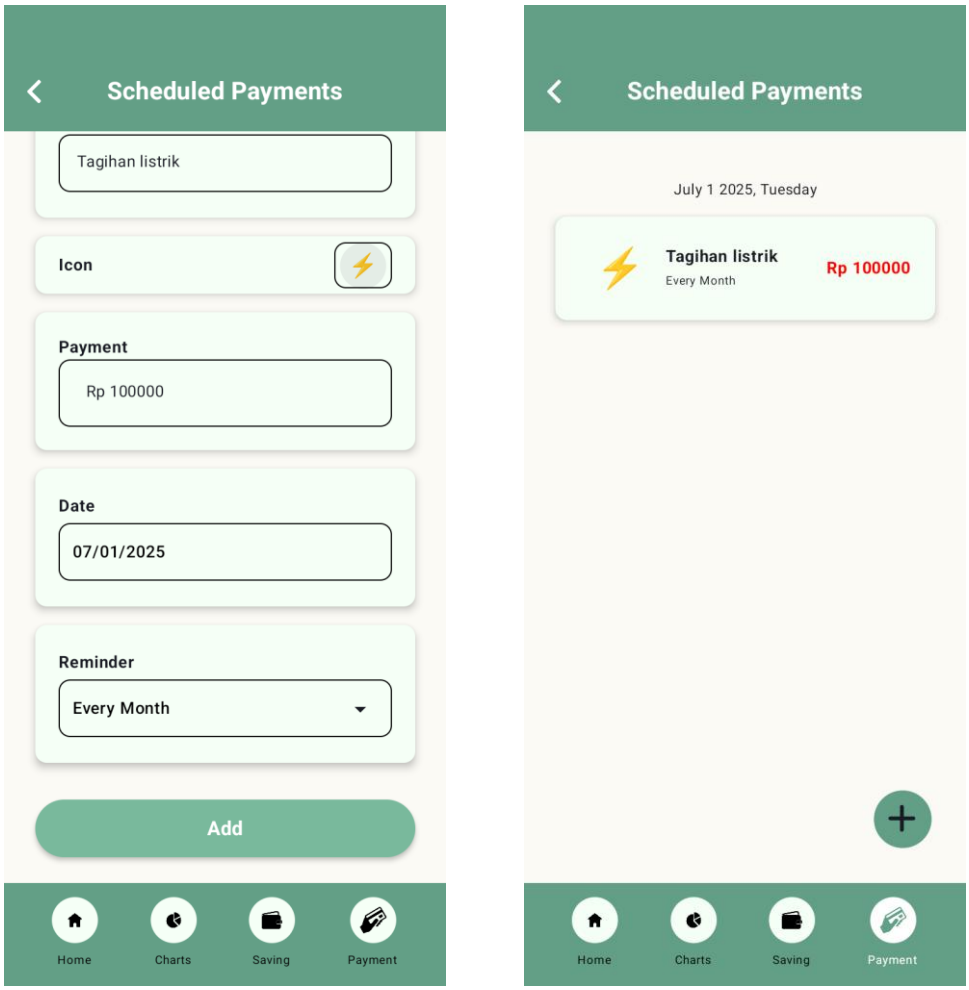
Gambar tampilan chart

Daftar transaksi keuangan yang berada di halaman *home* diolah menjadi grafik atau diagram pada menu *Chart* seperti gambar di atas ini. Gambar di atas menunjukkan visualisasi data pengeluaran selama 1 minggu, 1 bulan, dan 1 tahun dalam bentuk *chart* dan yang direkap di dalam 1 halaman.

7. Menu *Payment*

Menu *Payment* bertujuan untuk membantu pengguna mencatat dan mengelola pengingat pembayaran yang perlu dilakukan di waktu tertentu. Melalui fitur ini, pengguna dapat menyimpan informasi pembayaran rutin seperti tagihan listrik, cicilan, atau langganan, lalu memilih frekuensi pengingat, baik itu satu kali, harian, mingguan, bulanan, ataupun tahunan. Dengan begitu, pengguna tidak akan

melewatkan pembayaran penting dan dapat lebih disiplin dalam mengatur keuangan.



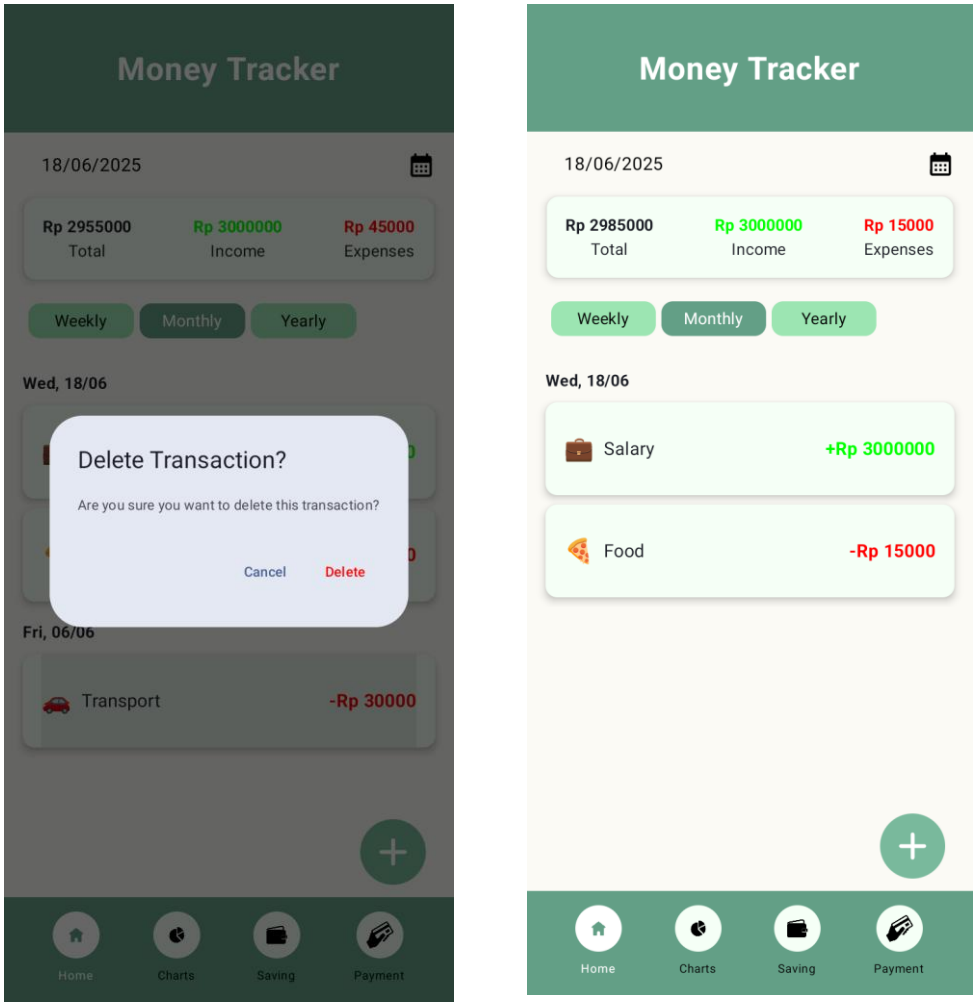
Gambar tampilan payment

Gambar di atas merupakan contoh visualisasi

Gambar di atas merupakan contoh visualisasi fitur *Payment* dalam aplikasi *Money Tracker*. Melalui tampilan ini, pengguna dapat mencatat jenis pembayaran yang perlu dilakukan, menetapkan nominal, serta menentukan jadwal pengingat berdasarkan frekuensi tertentu. Fitur ini memudahkan pengguna untuk mengelola kewajiban pembayaran agar tidak terlewat.

8. Hapus transaksi

Pengguna dapat menghapus transaksi yang sudah dicatat apabila terdapat kesalahan dalam pencatatan.



Gambar tampilan hapus transaksi

Gambar di atas merupakan visualisasi dari penghapusan data transaksi pada halaman *home*. Apabila ada kesalahan dalam pencatatan transaksi, pengguna dapat menghapus transaksi tersebut.

BAB IV
KESIMPULAN

Aplikasi *Money Tracker* hadir dan dikembangkan untuk menghadapi permasalahan umum yang sering dihadapi banyak orang, yaitu kurangnya kebiasaan mencatan dan mengelola keuangan pribadi secara rutin dan terstruktur. Banyak pengguna kesulitan dalam melacak pemasukan dan pengeluaran harian, tidak memiliki Gambaran yang jelas tentang kondisi finansialmereka, serta sering melewatkan pembayaran penting karena tidak adanya pengingat yangn sistematis.

Untuk mengatasi masalah tersebut, hadirnya aplikasi *Money Tracker*, sebuah platform berbasis Android yang menyediakan fitur-fitur lengkap seperti pencatatan transaksi harian, pelacakan pengeluaran dan pendapatan, pengelolaan tabungan, pengingat pembayaran berkala, serta visualisasi data keuangan dalam bentuk grafik. Fitur-fitur ini dirancang dengan antarmuka yang sederhana dan *user-friendly* agar mudah digunakan oleh berbagai kalangan.

Secara teknis, aplikasi ini dibangun menggunakan Jetpack Compose untuk UI yang reaktif dan modern, serta *Room Database* sebagai sistem penyimpanan lokal yang efisien. Data keuangan dikelola melalui ViewModel dan DAO agar tetap sinkron dan aman. Dengan adanya fitur seperti filter waktu (mingguan, bulanan, tahunan), *pie chart*, dan pengelompokan transaksi berdasarkan kategori, pengguna dapat dengan mudah memantau dan mengevaluasi kondisi keuangan mereka secara visual dan *real-time*.

Dengan demikian, *Money Tracker* tidak hanya menjadi alat pencatatan keuangan, tetapi juga menjadi solusi praktis dan terintegrasi untuk membangun kebiasaan finansial yang lebih sehat dan terstruktur.