Infinity - Money Manager

Entrega Final - CSI-28

Equipe:

Gabriel Telles Missailidis
João Lucas Rocha Rolim
Moisés Moreira
Othon Daiki Ishiyi
Rafael Camargo
Samir Nunes da Silva

Professores:

Johnny Cardoso Marques
Karla Donato Fook

Infinity - Money Manager

Protótipo de aplicativo Android gerenciador de finanças.

Infinity - Money Manager

Objetivos:

- Gerenciamento de:
  - Gastos
  - Ganhos
  - Metas

Usuários:

- Gustavo Gomes (Gago, T25), um iteano que necessita organizar suas contas
- Jovens profissionais
- Famílias
- Estudantes

Como?

- Aplicativo Android
- Linguagem Kotlin
- IDE Android Studio
- Princípios e Padrões de Engenharia de Software
- Figma (design)

Requisitos Funcionais

[RF001] Tela de Gestão de Finanças

[RF002] Criação de gastos e ganhos únicos

[RF003] Adição de gastos e ganhos periódicos

[RF004] Visualização dos ganhos e gastos

[RF006] Criação de metas financeiras

Requisitos Não Funcionais

[NF001] Botões de fácil entendimento

[NF002] Uso natural de cores e símbolos

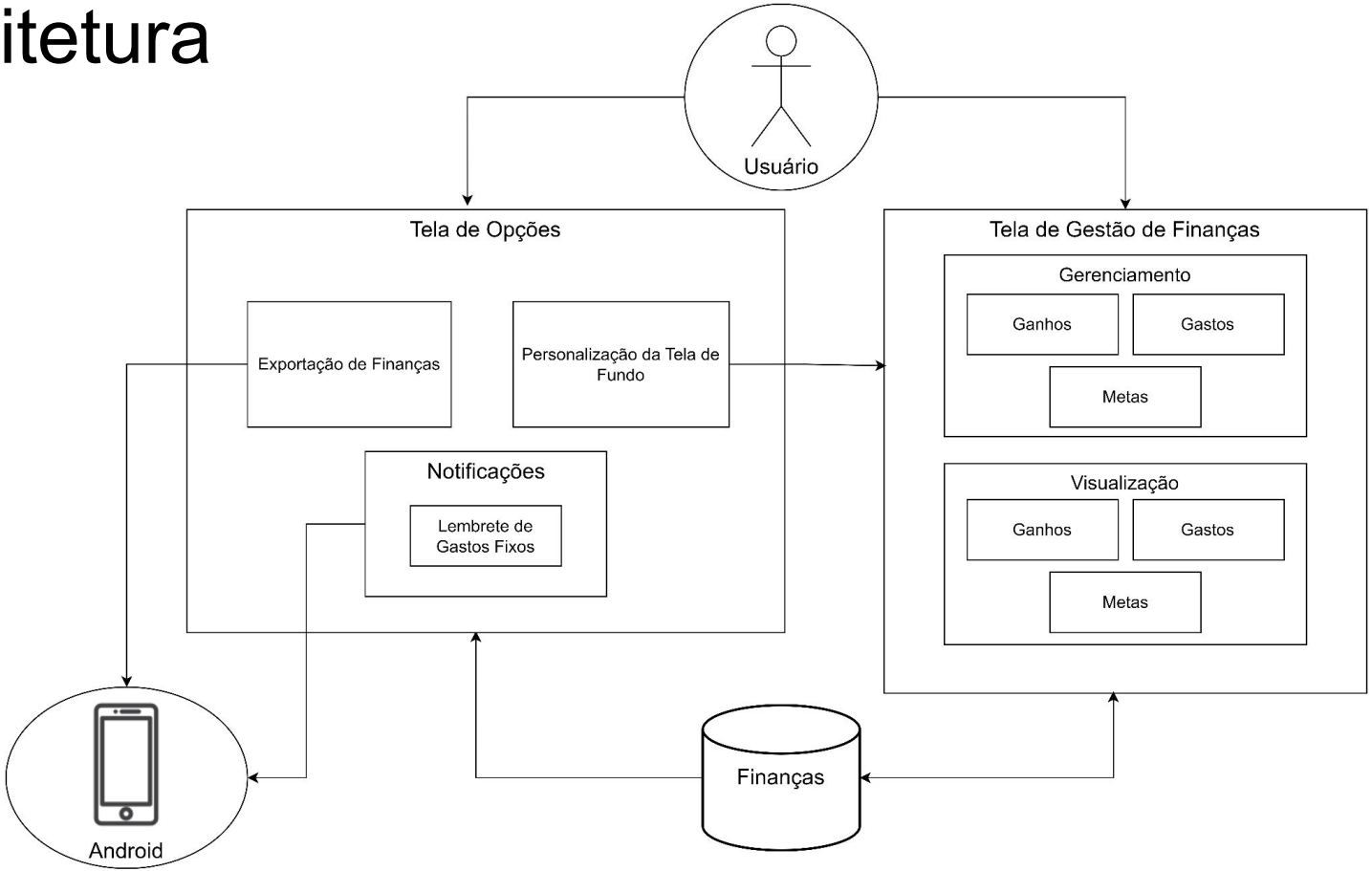
[NF003] Robustez a falhas de memória

[NF004] Rápida recuperação

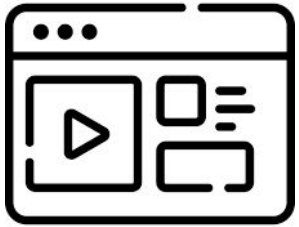
[NF005] Alta responsividade ao toque

[NF007] Restrição ao armazenamento de dados

Arquitetura

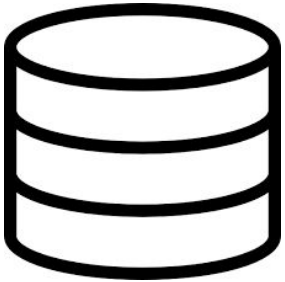


Implementação



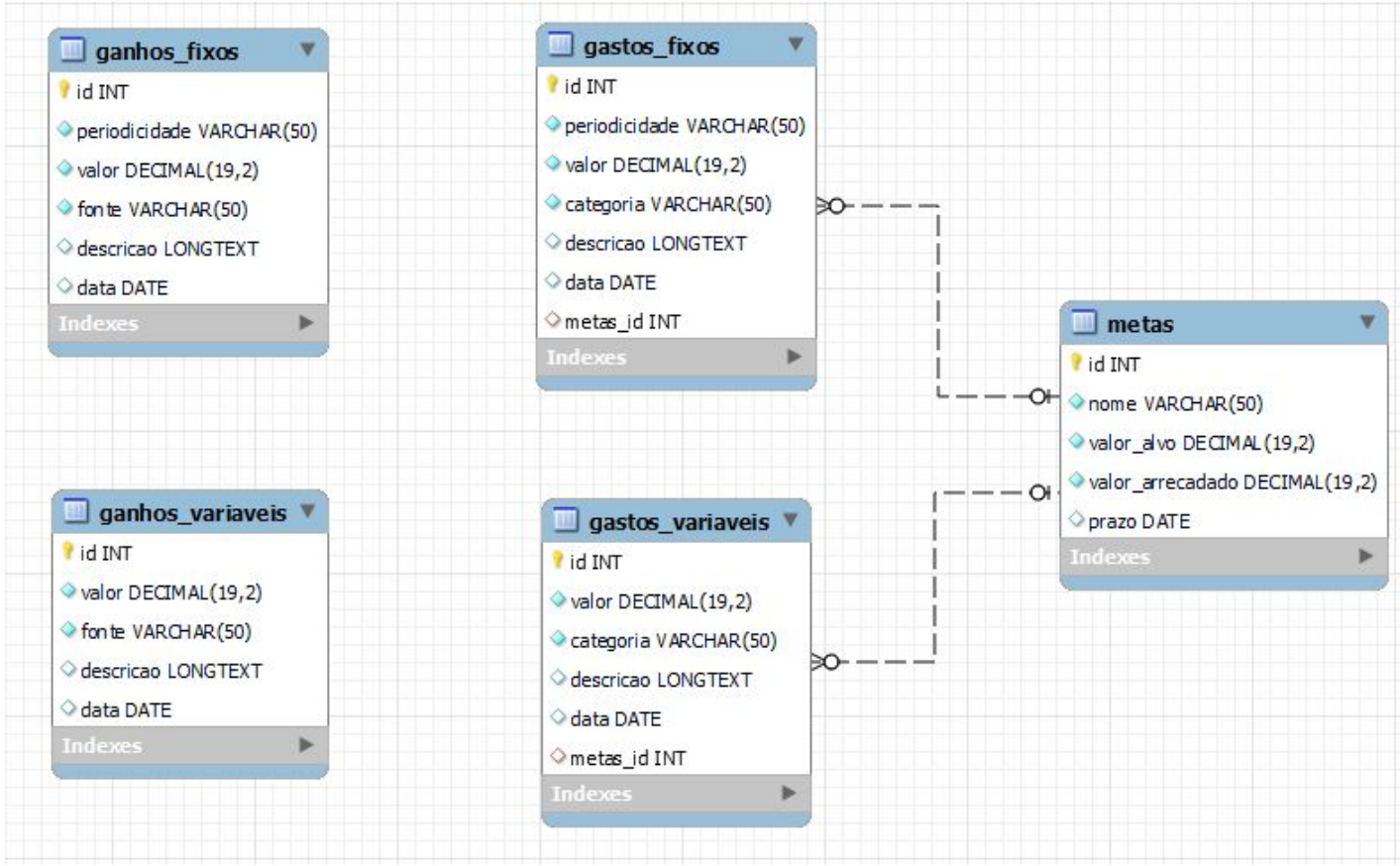
Frontend

Backend

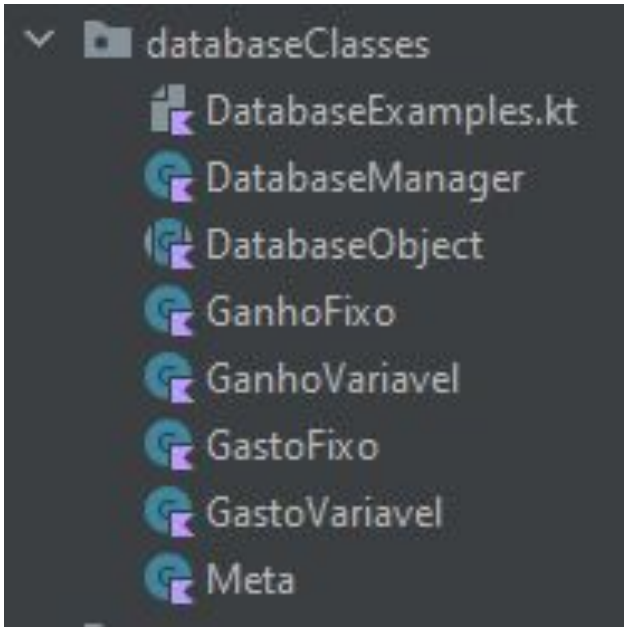


Banco de Dados

Banco de Dados



Banco de Dados




```kotlin
package com.example.infinitymoneymanager.databaseClasses

import java.sql.PreparedStatement

Samir Nunes da Silva
abstract class DatabaseObject{
    protected abstract val name: String
    protected abstract val sqlTable: String
    protected abstract val sqlColumns: String

    Samir Nunes da Silva
    abstract fun setQueryVariables(query: PreparedStatement)

    Samir Nunes da Silva
    fun getObjectName(): String {return name}
    Samir Nunes da Silva
    fun getSqlTableName(): String {return sqlTable}
    Samir Nunes da Silva
    fun getSqlColumnsNames(): String {return sqlColumns}
}
```

Banco de Dados

Operações:

- Insert
- Delete
- Select

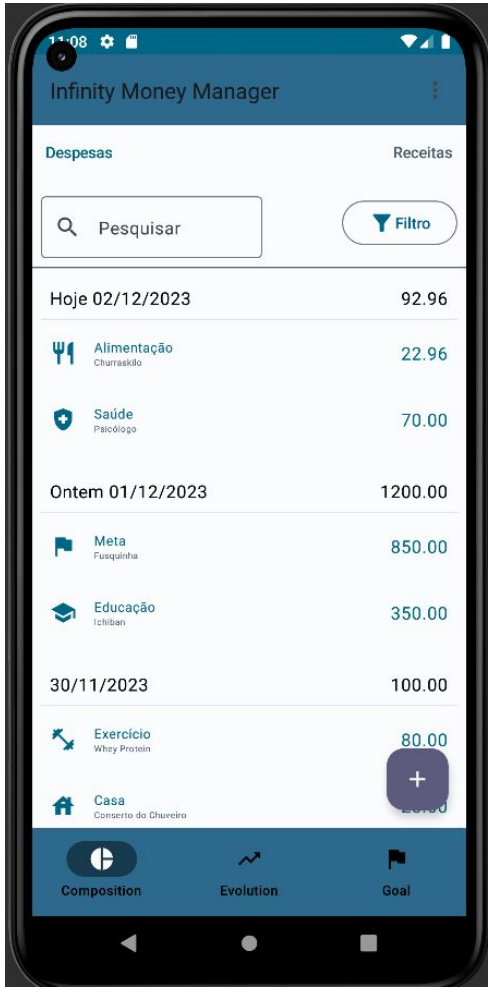
```
package com.example.infinitymoneymanager.databaseClasses

import java.sql.Connection
import java.sql.DriverManager
import java.sql.PreparedStatement
import java.sql.ResultSet

Samir Nunes da Silva
class DatabaseManager{
    Samir Nunes da Silva
    companion object {
        private var connection: Connection? = null
        Samir Nunes da Silva
        @JvmStatic
        fun openConnection() {
            connection = DriverManager.getConnection(
                url: "jdbc:mysql://localhost/infinity",
                user: "root",
                password: "infinity"
            )
            println("Connection with database opened successfully.")
        }
```

Frontend

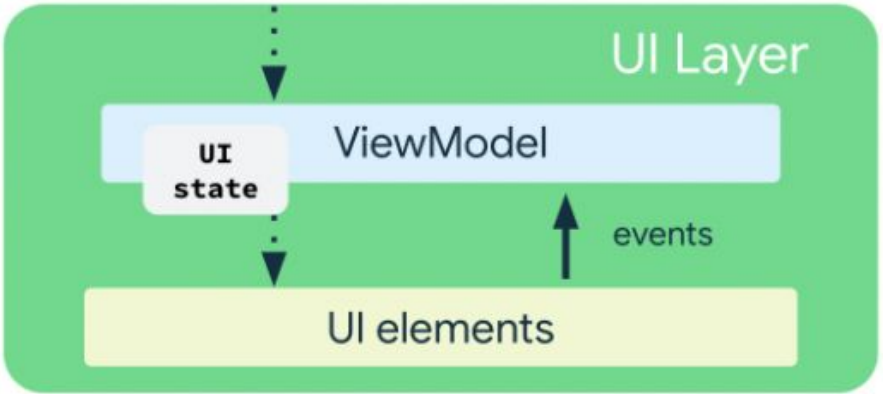
- Responsável pela Interface com o Usuário (UI)



Frontend

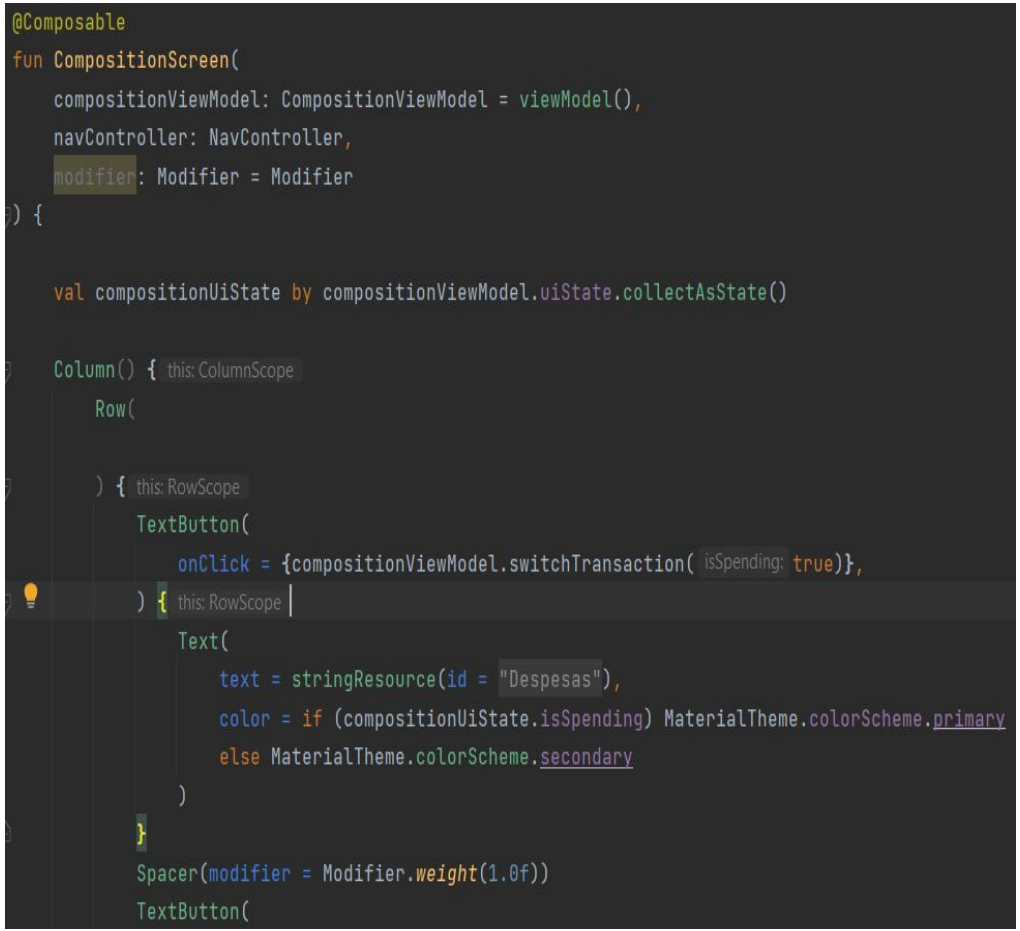
- Adotou-se o framework Jetpack Compose para o desenvolvimento da UI
  - Ferramenta desenvolvida pela Google e lançada em 2019
  - Sua utilização é recomendada em vez do framework por XML

- Arquitetura do frontend se divide entre as UI Components e as ViewModels



UI Elements

- Principalmente constituídos de funções composable (combináveis)
  - Criam desde textos até telas.



ViewModel

- Camada de manipulação dos estados e de integração com o backend


```kotlin
data class CompositionUiState(
    val isSpending: Boolean = true,
    val currentSearch: String = ""
)

 othon-ishiyi
class CompositionViewModel : ViewModel(){
    private val _uiState = MutableStateFlow(CompositionUiState())
    val uiState: StateFlow<CompositionUiState> = _uiState.asStateFlow()

    private val _transactions = MutableLiveData<List<Transaction>>()
     othon-ishiyi
    val transactions: List<Transaction>? get() = _transactions.value


     othon-ishiyi
    fun onSearchChange(value: String) {
        _uiState.update{currentState ->
            currentState.copy(
                currentSearch = value
            )
        }
        getTransactions()
    }
```

Backend

- API de gerenciamento financeiro em Kotlin com Spring Boot.

```kotlin
@RequestMapping("/finance")
class FinanceController(private val financeService: FinanceService) {

    @PostMapping("/gastos")
    fun createGasto(@RequestBody gastoDto: GastoDto): GastoFixo {
        println(gastoDto.categoria)
        return financeService.createGasto(gastoDto)
    }

    @PostMapping("/metas")
    fun createMeta(@RequestBody metaDto: MetaDto): Meta {
        return financeService.createMeta(metaDto)
    }

    @PostMapping("/gastos-variaveis")
    fun createGastoVariavel(@RequestBody dto: GastoVariavelDto): GastoVariavel {
        return financeService.createGastoVariavel(dto)
    }

    @PostMapping("/ganhos-fixos")
    fun createGanhoFixo(@RequestBody dto: GanhoFixoDto): GanhoFixo {
        return financeService.createGanhoFixo(dto)
    }

    @PostMapping("/ganhos-variaveis")
    fun createGanhoVariavel(@RequestBody dto: GanhoVariavelDto): GanhoVariavel {
        return financeService.createGanhoVariavel(dto)
    }

    @GetMapping("/gastos")
    fun getAllGastos(): ResponseEntity<List<Any>> {
        val gastos = financeService.getAllGastos()
        return ResponseEntity.ok(gastos)
    }

    @GetMapping("/ganhos")
    fun getAllGanhos(): ResponseEntity<List<Any>> {
        val ganhos = financeService.getAllGanhos()
        return ResponseEntity.ok(ganhos)
    }
```

Obrigado!