FINANCIAL TRACKER ASSISTANCE A SIMPLE ANDROID APPLICATION TO TRACK MONEY

Dissertation submitted in fulfilment of the requirements for the Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

SHAIK LATHEEF

12107544

CSE 225

ANDROID APP DEVELOPMENT

School of Computer Science and Engineering

Lovely Professional University

Phagwara, Punjab (India)

Month-May Year-2024

TABLE OF CONTENTS

Sl.No CONTENTS PAGE NO.

INTRODUCTION TO THE FINANCIAL TRACKER ASSISTANCE

ABOUT:

Welcome to Save App [financial tracker assistance], your personal finance companion designed to help you effortlessly manage your money and achieve your financial goals. Whether you're saving for a dream vacation, planning for retirement, or simply aiming to gain better control over your finances, our app provides the tools and insights you need to succeed.

In today's fast-paced world, keeping track of your spending, savings, and income can be challenging. That's where [Your App Name] comes in. With its user-friendly interface and powerful features, managing your finances has never been easier.

Key Features: -

Expense Tracking: Easily log your daily expenses and categorize them for better organization. Monitor your spending habits and identify areas where you can save more.

Income Management: Keep track of your income sources and set goals for increasing your earnings. Gain a comprehensive view of your financial inflows and outflows.

Savings Goals: Set achievable savings goals and track your progress in real-time. Whether you're saving for a big purchase or building an emergency fund, [Your App Name] keeps you motivated along the way.

Budgeting Tools: Create personalized budgets for different expense categories and time periods. Receive insights and recommendations to optimize your spending and achieve your financial objectives.

Reports and Analytics: Access detailed reports and analytics to gain valuable insights into your financial habits. Understand where your money is going and make informed decisions to improve your financial health.

Reminders and Notifications: Stay on top of your finances with timely reminders for bill payments, savings goals, and upcoming expenses. Never miss a deadline again.

Security and Privacy: Rest assured that your financial data is safe and secure with advanced encryption and authentication measures. Your privacy is our top priority.

Modules or Activity Explanation:-

1. Authentication Module: This module handles user authentication and registration. It includes activities such as:

- Login Activity: Allows existing users to log in to their accounts.

- Registration Activity: Enables new users to create accounts.

2. Dashboard Module: This module serves as the main interface of the app, providing an overview of the user's financial status. Activities in this module may include:

- Dashboard Activity: Displays summaries of savings, expenses, income, and progress towards goals.

- Graphs/Charts Activity: Visual representations of financial data, such as pie charts or line graphs.

3. Expense Tracking Module: This module helps users track their expenses and manage their spending habits. Activities may include:

- Expense List Activity: Displays a list of recorded expenses, allowing users to add, edit, or delete entries.

- Expense Details Activity: Provides detailed information about a specific expense, including category, date, and notes.

4. Income Tracking Module: Similar to the expense tracking module, this module allows users to track their sources of income. Activities may include:

- Income List Activity: Displays a list of recorded income sources, allowing users to add, edit, or delete entries.

- Income Details Activity: Provides detailed information about a specific income source, including type, frequency, and amount.

5. Savings Goals Module: This module enables users to set and track their savings goals. Activities may include:

- Goals List Activity: Displays a list of savings goals, showing progress and allowing users to add, edit, or delete goals.

- Goal Details Activity: Provides detailed information about a specific savings goal, including target amount and deadline.

6. Budgeting Module: This module helps users create and manage budgets for different expense categories. Activities may include:

- Budget List Activity: Displays a list of budget categories, allowing users to set limits and track spending.

- Budget Details Activity: Provides detailed information about a specific budget category, including allocated amount and actual spending.

7. Reports and Analytics Module: This module generates reports and analytics to help users gain insights into their financial habits. Activities may include:

- Reports Activity: Displays summary reports, charts, and graphs based on user data.

- Analytics Activity: Provides in-depth analysis of spending patterns, trends, and areas for improvement.

8. Settings Module: This module allows users to customize their app preferences and settings. Activities may include:

- Settings Activity: Provides options for configuring notifications, currency settings, and other preferences.

- Profile Activity: Allows users to view and update their profile information.

Activities:

- Main Activity: This is the entry point of the app. It may include navigation buttons or tabs to access different modules.
- Login/Register Activity: Handles user authentication and registration.
- Dashboard Activity: Displays an overview of the user's financial status.
- Expense Activity: Allows users to track their expenses.
- Income Activity: Allows users to track their income sources.
- Goals Activity: Allows users to set and track savings goals.
- Budget Activity: Helps users create and manage budgets.
- Reports Activity: Generates reports and analytics based on user data.
- Settings Activity: Allows users to customize app settings.

Each module and activity play a crucial role in providing users with a comprehensive financial tracking experience. By organizing your app in this manner, users can easily navigate between different functionalities and manage their finances effectively.

Code - Module wise or Activity Wise:-

I Budget: -

```
package com.ferrariofilippo.saveapp

import android.content.Intent
import android.os.Bundle
import android.util.Log
import android.view.MenuItem
import android.widget.Button
import androidx.activity.result.ActivityResult
import androidx.activity.result.contract.ActivityResultContracts
import androidx.activity.result.contract.ActivityResultContracts.GetContent
import androidx.activity.result.contract.ActivityResultContracts.CreateDocument
import androidx.appcompat.app.AppCompatActivity
import androidx.core.os.bundleOf
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.lifecycleScope
import androidx.navigation.findNavController
import androidx.work.WorkManager
import com.ferrariofilippo.saveapp.model.enums.Currencies
import com.ferrariofilippo.saveapp.util.BudgetUtil
import com.ferrariofilippo.saveapp.util.CloudStorageUtil
import com.ferrariofilippo.saveapp.util.CurrencyUtil
import com.ferrariofilippo.saveapp.util.ImportExportUtil
import com.ferrariofilippo.saveapp.util.SettingsUtil
import com.ferrariofilippo.saveapp.util.SpacingUtil
import com.ferrariofilippo.saveapp.util.StatsUtil
import com.ferrariofilippo.saveapp.util.SubscriptionUtil
import com.ferrariofilippo.saveapp.util.TagUtil
import com.google.android.gms.auth.api.identity.AuthorizationResult
import com.google.android.gms.auth.api.identity.Identity
import com.google.android.material.bottomappbar.BottomAppBar
import com.google.android.material.floatingactionbutton.ExtendedFloatingActionButton
import com.google.android.material.snackbar.Snackbar
import kotlinx.coroutines.launch
import java.io.FileInputStream
import java.io.FileOutputStream

class MainActivity : AppCompatActivity() {
  companion object {
    private var _restartFunction: () -> Unit = { }

    private var _checkpointFunction: () -> Unit = { }

    fun requireRestart() {
      _restartFunction()
    }

    fun requireCheckpoint() {
      _checkpointFunction()
    }
  }
```

```
private var lastFragmentId: Int = R.id.homeFragment

private var navControllerInitialized = false

private lateinit var rootDestinations: Set<Int>

private val _isUpdatingCurrencies: MutableLiveData<Boolean> = MutableLiveData(false)

val isUpdatingCurrencies: LiveData<Boolean> = _isUpdatingCurrencies

// IO Activities
val exportMovements = registerForActivityResult(CreateDocument("text/csv")) { uri ->
    if (uri != null) {
        ImportExportUtil.export(
            ImportExportUtil.CREATE_MOVEMENTS_FILE,
            contentResolver?.openOutputStream(uri) as FileOutputStream,
            application as SaveAppApplication
        )
    }
}
val exportSubscriptions = registerForActivityResult(CreateDocument("text/csv")) { uri ->
    if (uri != null) {
        ImportExportUtil.export(
            ImportExportUtil.CREATE_SUBSCRIPTIONS_FILE,
            contentResolver?.openOutputStream(uri) as FileOutputStream,
            application as SaveAppApplication
        )
    }
}
val exportBudgets = registerForActivityResult(CreateDocument("text/csv")) { uri ->
    if (uri != null) {
        ImportExportUtil.export(
            ImportExportUtil.CREATE_BUDGETS_FILE,
            contentResolver?.openOutputStream(uri) as FileOutputStream,
            application as SaveAppApplication
        )
    }
}

val importMovements = registerForActivityResult(GetContent()) { uri ->
    if (uri != null) {
        ImportExportUtil.import(
            ImportExportUtil.OPEN_MOVEMENTS_FILE,
            contentResolver.openInputStream(uri) as FileInputStream,
            application as SaveAppApplication
        )
    }
}
val importSubscriptions = registerForActivityResult(GetContent()) { uri ->
    if (uri != null) {
        ImportExportUtil.import(
            ImportExportUtil.OPEN_SUBSCRIPTIONS_FILE,
            contentResolver.openInputStream(uri) as FileInputStream,
            application as SaveAppApplication
        )
    }
}
val importBudgets = registerForActivityResult(GetContent()) { uri ->
    if (uri != null) {
        ImportExportUtil.import(
            ImportExportUtil.OPEN_BUDGETS_FILE,
            contentResolver.openInputStream(uri) as FileInputStream,
            application as SaveAppApplication
        )
    }
}

val createMovementsTemplate = registerForActivityResult(CreateDocument("text/csv")) { uri ->
    if (uri != null) {
        ImportExportUtil.writeTemplate(
            ImportExportUtil.CREATE_MOVEMENTS_TEMPLATE,
            contentResolver?.openOutputStream(uri) as FileOutputStream,
            application as SaveAppApplication
        )
    }
}
val createSubscriptionsTemplate = registerForActivityResult(CreateDocument("text/csv")) { uri ->
    if (uri != null) {
        ImportExportUtil.writeTemplate(
            ImportExportUtil.CREATE_SUBSCRIPTIONS_TEMPLATE,
            contentResolver?.openOutputStream(uri) as FileOutputStream,
            application as SaveAppApplication
        )
    }
}
val createBudgetsTemplate = registerForActivityResult(CreateDocument("text/csv")) { uri ->
    if (uri != null) {
        ImportExportUtil.writeTemplate(
            ImportExportUtil.CREATE_BUDGETS_TEMPLATE,
            contentResolver?.openOutputStream(uri) as FileOutputStream,
            application as SaveAppApplication
        )
    }
}

val uploadBackupToDrive =
        registerForActivityResult(ActivityResultContracts.StartIntentSenderForResult()) { result:
ActivityResult ->
            val authResult: AuthorizationResult = Identity.getAuthorizationClient(this)
                .getAuthorizationResultFromIntent(result.data)

            CloudStorageUtil.enqueueUpload(application as SaveAppApplication, authResult)
        }
    val downloadBackupFromDrive =
        registerForActivityResult(ActivityResultContracts.StartIntentSenderForResult()) { result:
ActivityResult ->
            val authResult: AuthorizationResult = Identity.getAuthorizationClient(this)
                .getAuthorizationResultFromIntent(result.data)

            CloudStorageUtil.enqueueDownload(application as SaveAppApplication, authResult)
        }

    // Overrides
    override fun onCreate(savedInstanceState: Bundle?) {
        val saveApp = application as SaveAppApplication
        SettingsUtil.setStore(saveApp)
        CurrencyUtil.setStore(saveApp)
        TagUtil.updateAll(saveApp)
        BudgetUtil.init(saveApp)
        StatsUtil.init(saveApp)
        SpacingUtil.init(saveApp)

        _restartFunction = { restartApplication() }
        _checkpointFunction = { saveApp.utilRepository.checkpoint() }

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        saveApp.setCurrentActivity(this)
        lifecycleScope.launch { SubscriptionUtil.validateSubscriptions(saveApp) }
        lifecycleScope.launch { CurrencyUtil.init() }

        setupButtons()
    }

    override fun onStart() {
        super.onStart()
        WorkManager.getInstance(this).cancelAllWork()
    }

    // Methods
    fun goToSettings() {
        ensureNavControllerInitialized()
findNavController(R.id.containerView).navigate(R.id.action_homeFragment_to_settingsFragment)
    }

    fun goToSubscriptions() {
        ensureNavControllerInitialized()
findNavController(R.id.containerView).navigate(R.id.action_homeFragment_to_subscriptionsFragmen
t)
    }

    fun goToNewBudget() {
        ensureNavControllerInitialized()
findNavController(R.id.containerView).navigate(R.id.action_budgetsFragment_to_newBudgetFragme
nt)
    }

    fun goToEditMovementOrSubscription(id: Int, isMovement: Boolean) {
        ensureNavControllerInitialized()
        val bundle = bundleOf("itemId" to id, "isMovement" to isMovement)
        findNavController(R.id.containerView).navigate(R.id.newMovementFragment, bundle)
    }

    fun goToEditBudget(id: Int) {
        ensureNavControllerInitialized()

        val bundle = bundleOf("itemId" to id)
        findNavController(R.id.containerView).navigate(
            R.id.action_budgetsFragment_to_newBudgetFragment,
            bundle
        )
    }

    fun goToManageTags() {
        ensureNavControllerInitialized()
findNavController(R.id.containerView).navigate(R.id.action_settingsFragment_to_manageTagsFragme
nt)
    }

    fun goToManageData() {
        ensureNavControllerInitialized()
findNavController(R.id.containerView).navigate(R.id.action_settingsFragment_to_manageDataFragme
nt)
    }

    fun gotToAddOrEditTag(id: Int) {
        ensureNavControllerInitialized()

        val bundle = bundleOf("tagId" to id)
        findNavController(R.id.containerView).navigate(
            R.id.action_manageTagsFragment_to_newTagFragment,
            bundle
        )
    }

    fun popLastView() {
        ensureNavControllerInitialized()
        findNavController(R.id.containerView).popBackStack()
    }

    fun updateAllToNewCurrency(value: Currencies) {
        lifecycleScope.launch {
            _isUpdatingCurrencies.value = true
            CurrencyUtil.updateAllToNewCurrency(application, value)
            _isUpdatingCurrencies.value = false

            Snackbar.make(
                findViewById(R.id.containerView),
                R.string.default_currency_updated,
                Snackbar.LENGTH_SHORT
            ).setAnchorView(findViewById(R.id.bottomAppBar)).show()
        }
    }

    private fun setupButtons() {
        val addMovementButton: Button = findViewById(R.id.addMovementFAB)
        addMovementButton.setOnClickListener {
            onAddMovementClick()
        }

        val appBar: BottomAppBar = findViewById(R.id.bottomAppBar)
        appBar.setOnMenuItemClickListener { menuItem: MenuItem ->
```

```
            onMenuItemClick(menuItem)
        }
    }

    private fun ensureNavControllerInitialized() {
        if (navControllerInitialized)
            return

        rootDestinations =
            setOf(R.id.homeFragment, R.id.historyFragment, R.id.budgetsFragment, R.id.statsFragment)

        findNavController(R.id.containerView).addOnDestinationChangedListener { _, destination, _ ->
            val fab = findViewById<ExtendedFloatingActionButton>(R.id.addMovementFAB)
            if (rootDestinations.contains(destination.id)) {
                fab.show()
            } else {
                fab.hide()
            }
        }

        lastFragmentId = R.id.homeFragment
        navControllerInitialized = true
    }

    private fun onAddMovementClick() {
        ensureNavControllerInitialized()

        try {
            when (lastFragmentId) {
                R.id.homeFragment ->

findNavController(R.id.containerView).navigate(R.id.action_homeFragment_to_newMovementFragm
ent)
```

```
                R.id.historyFragment ->

findNavController(R.id.containerView).navigate(R.id.action_historyFragment_to_newMovementFrag
ment)


                R.id.budgetsFragment ->

findNavController(R.id.containerView).navigate(R.id.action_budgetsFragment_to_newMovementFrag
ment)


                R.id.statsFragment ->

findNavController(R.id.containerView).navigate(R.id.action_statsFragment_to_newMovementFragme
nt)
            }
        } catch (e: Exception) {
            Log.e("NAV_E", e.message.toString())
        }
    }


    private fun onMenuItemClick(menuItem: MenuItem): Boolean {
        ensureNavControllerInitialized()


        when (menuItem.itemId) {
            R.id.home -> {
                findNavController(R.id.containerView).navigate(R.id.homeFragment)
                lastFragmentId = R.id.homeFragment


                return true
            }


            R.id.history -> {
                findNavController(R.id.containerView).navigate(R.id.historyFragment)
                lastFragmentId = R.id.historyFragment
                return true
            }

            R.id.budget -> {
                findNavController(R.id.containerView).navigate(R.id.budgetsFragment)
                lastFragmentId = R.id.budgetsFragment

                return true
            }

            R.id.stats -> {
                findNavController(R.id.containerView).navigate(R.id.statsFragment)
                lastFragmentId = R.id.statsFragment

                return true
            }
        }

        return false
    }

    private fun restartApplication() {
        val intent = packageManager.getLaunchIntentForPackage(packageName) ?: return
        val startIntent = Intent.makeRestartActivityTask(intent.component)
        startIntent.`package` = packageName
        startActivity(startIntent)
        Runtime.getRuntime().exit(0)
    }
}
```

Explonation of code :-

This code is for a `Fragment` in an Android application, specifically for managing budgets. Let's break it down:

1. Imports: The fragment imports necessary classes from the Android framework and custom classes within the project.

2. Class Declaration: `BudgetsFragment` is declared as a subclass of `Fragment`.

3. Properties:

   - `viewModel`: An instance of `BudgetsViewModel`, which is used to manage the data and business logic related to budgets.

   - `_binding`: An instance of `FragmentBudgetsBinding`, which is auto-generated by Android's View Binding feature. It holds references to views within the fragment layout.

4. Lifecycle Methods:

   - `onCreateView()`: Inflates the fragment's layout, sets up data binding, and initializes UI components.

   - `onCreate()`: Initializes the view model.

   - `onDestroy()`: Cleans up resources, such as the binding instance.

5. Setup Methods:

   - `setupRecyclerViews()`: Configures and populates the RecyclerViews for displaying active and past budgets. It sets adapters, layout managers, observes changes in the budgets, and updates the UI accordingly.

   - `setupRecyclerGestures()`: Configures swipe gestures (left and right) on RecyclerView items for editing and deleting budgets.

   - `setupRecyclerDecorator()`: Adds custom decoration (spacing) to RecyclerView items.

   - `setupButtons()`: Sets click listeners for UI buttons.

6. Event Handling:

   - `onRemoveMovementInvoked()`: Handles the action when a budget is swiped to delete. It deletes the budget from the database and shows a Snackbar with an undo option.

Overall, this fragment manages the display and interaction of budgets, including adding, editing, and deleting budgets, using RecyclerViews and ViewModel for data management.

II History :-

```
// Copyright (c) 2024 Filippo Ferrario

// Licensed under the MIT License. See the LICENSE.


package com.ferrariofilippo.saveapp.view


import android.content.Context

import android.os.Bundle

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import android.view.inputmethod.InputMethodManager

import android.widget.AutoCompleteTextView

import androidx.fragment.app.Fragment

import androidx.lifecycle.Observer

import androidx.lifecycle.ViewModelProvider

import androidx.lifecycle.lifecycleScope

import androidx.recyclerview.widget.ItemTouchHelper

import androidx.recyclerview.widget.LinearLayoutManager

import androidx.recyclerview.widget.RecyclerView

import com.ferrariofilippo.saveapp.MainActivity

import com.ferrariofilippo.saveapp.R

import com.ferrariofilippo.saveapp.SaveAppApplication

import com.ferrariofilippo.saveapp.databinding.FragmentHistoryBinding

import com.ferrariofilippo.saveapp.model.entities.Movement

import com.ferrariofilippo.saveapp.model.entities.Tag

import com.ferrariofilippo.saveapp.model.enums.Currencies

import com.ferrariofilippo.saveapp.model.taggeditems.TaggedMovement

import com.ferrariofilippo.saveapp.util.BudgetUtil

import com.ferrariofilippo.saveapp.util.RecyclerEditAndDeleteGestures

import com.ferrariofilippo.saveapp.util.SettingsUtil

import com.ferrariofilippo.saveapp.util.SpacingUtil

import com.ferrariofilippo.saveapp.util.StatsUtil
import com.ferrariofilippo.saveapp.view.adapters.HistoryAdapter
import com.ferrariofilippo.saveapp.view.adapters.TagsDropdownAdapter
import com.ferrariofilippo.saveapp.view.viewmodels.HistoryViewModel
import com.google.android.material.bottomsheet.BottomSheetBehavior
import com.google.android.material.snackbar.Snackbar
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.launch
import kotlinx.coroutines.runBlocking
import java.time.LocalDate

class HistoryFragment : Fragment() {
  private lateinit var viewModel: HistoryViewModel

  private var _binding: FragmentHistoryBinding? = null
  private val binding get() = _binding!!

  // Overrides
  override fun onCreateView(
    inflater: LayoutInflater,
    container: ViewGroup?,
    savedInstanceState: Bundle?
  ): View {
    _binding = FragmentHistoryBinding
      .inflate(inflater, container, false)
      .apply {
        lifecycleOwner = viewLifecycleOwner
        vm = viewModel
      }

    setupRecyclerView()
    setupBottomSheet()
    return binding.root
}

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    viewModel = ViewModelProvider(this)[HistoryViewModel::class.java]
}

override fun onDestroyView() {
    super.onDestroyView()
    _binding = null
}

// Methods
private fun setupRecyclerView() {
    val adapter = HistoryAdapter(
        Currencies.from(runBlocking { SettingsUtil.getCurrency().first() }),
        SpacingUtil.padding
    )

    binding.movementsRecyclerView.adapter = adapter
    binding.movementsRecyclerView.layoutManager = LinearLayoutManager(context)

    viewModel.movements.observe(viewLifecycleOwner, Observer { movements ->
        movements?.let {
            setFilteredItems(adapter, it)
        }
    })

    binding.sortButton.setOnClickListener {
        binding.searchBar.editText?.clearFocus()
        viewModel.sortAscending.value = !viewModel.sortAscending.value!!
        adapter.submitList(adapter.currentList.reversed())
    }

    binding.searchBar.editText?.setOnFocusChangeListener { _, b ->
        viewModel.searchBarHint.value =
            if (b) "" else requireContext().resources.getString(R.string.searchbar_hint)
    }

    viewModel.searchQuery.observe(viewLifecycleOwner, Observer { query ->
        query?.let {
            if (viewModel.movements.value != null) {
                setFilteredItems(adapter, viewModel.movements.value!!)
            }
        }
    })

    binding.movementsRecyclerView.setOnTouchListener { _, _ -> onRecyclerClick() }

    setupRecyclerGestures()
}

private fun setupRecyclerGestures() {
    val gestureCallback = object : RecyclerEditAndDeleteGestures(requireContext()) {
        override fun onSwiped(viewHolder: RecyclerView.ViewHolder, direction: Int) {
            val position = viewHolder.adapterPosition
            val adapter = binding.movementsRecyclerView.adapter as HistoryAdapter
            val movement = adapter.getItemAt(position)

            if (direction == ItemTouchHelper.RIGHT) {
                (activity as MainActivity).goToEditMovementOrSubscription(movement.id, true)
            } else if (direction == ItemTouchHelper.LEFT) {
                onRemoveMovementInvoked(movement)
            }
        }
    }

    val itemTouchHelper = ItemTouchHelper(gestureCallback)
    itemTouchHelper.attachToRecyclerView(binding.movementsRecyclerView)
}

private fun setupBottomSheet() {
    val bottomSheet = BottomSheetBehavior.from(binding.filtersBottomSheet)
    val bottomSheetCallback = object : BottomSheetBehavior.BottomSheetCallback() {
        override fun onSlide(bottomSheet: View, slideOffset: Float) {
        }

        override fun onStateChanged(bottomSheet: View, newState: Int) {
            viewModel.isSearchHidden.value = newState == BottomSheetBehavior.STATE_HIDDEN
        }
    }
    bottomSheet.addBottomSheetCallback(bottomSheetCallback)

    binding.searchButton.setOnClickListener {
        binding.searchBar.editText?.clearFocus()
        bottomSheet.state = BottomSheetBehavior.STATE_HALF_EXPANDED
    }

    setupTagPicker()

    binding.decreaseYearButton.setOnClickListener {
        viewModel.year.value = (viewModel.year.value ?: LocalDate.now().year) - 1
    }
    binding.increaseYearButton.setOnClickListener {
        viewModel.year.value = (viewModel.year.value ?: LocalDate.now().year) + 1
    }
}

private fun setupTagPicker() {
    val tagAutoComplete = binding.tagFilterInput.editText as AutoCompleteTextView
    viewModel.tags.observe(viewLifecycleOwner, Observer {
        it?.let {
            val adapter = TagsDropdownAdapter(
                binding.tagFilterInput.context,
                R.layout.tag_dropdown_item,
                it
            )

            tagAutoComplete.setAdapter(adapter)
            tagAutoComplete.setOnItemClickListener { parent, _, position, _ ->
                val selection = parent.adapter.getItem(position) as Tag
                viewModel.tagId.value = selection.id
                setFilteredItems(
                    binding.movementsRecyclerView.adapter as HistoryAdapter,
                    viewModel.movements.value!!
                )
            }
        }
    })

    binding.clearTagFilterButton.setOnClickListener {
        tagAutoComplete.text = null
        tagAutoComplete.clearFocus()
        viewModel.tagId.value = 0
        setFilteredItems(
            binding.movementsRecyclerView.adapter as HistoryAdapter,
            viewModel.movements.value!!
            )
        }
    }

    private fun setFilteredItems(adapter: HistoryAdapter, movements: List<TaggedMovement>) {
        val isFilteringByTag = viewModel.tagId.value != 0

        val values = if (viewModel.searchQuery.value!!.isNotBlank() && isFilteringByTag) {
            val query = viewModel.searchQuery.value!!.lowercase()
            movements.filter {
                it.description.lowercase().contains(query) && it.tagId == viewModel.tagId.value
            }
        } else if (isFilteringByTag) {
            movements.filter {
                it.tagId == viewModel.tagId.value
            }
        } else if (viewModel.searchQuery.value!!.isNotBlank()) {
            val query = viewModel.searchQuery.value!!.lowercase()
            movements.filter {
                it.description.lowercase().contains(query)
            }
        } else {
            movements
        }

        adapter.submitList(if (viewModel.sortAscending.value!!) values.reversed() else values)
    }

    private fun onRecyclerClick(): Boolean {
        binding.searchBar.editText?.clearFocus()
        val imm = context?.getSystemService(Context.INPUT_METHOD_SERVICE) as?
InputMethodManager
        imm?.hideSoftInputFromWindow(view?.windowToken, 0)
        return false
    }

    private fun onRemoveMovementInvoked(taggedMovement: TaggedMovement) {
        val app = requireActivity().application as SaveAppApplication
        val movement = Movement(
            taggedMovement.id,
            taggedMovement.amount,
            taggedMovement.description,
            taggedMovement.date,
            taggedMovement.tagId,
            taggedMovement.budgetId
        )
        lifecycleScope.launch {
            app.movementRepository.delete(movement)
            BudgetUtil.removeMovementFromBudget(movement)
            viewModel.updateMovements()
            movement.amount *= -1
            StatsUtil.addMovementToStat(app, movement)
        }

        Snackbar.make(binding.coordinatorLayout, R.string.movement_deleted,
Snackbar.LENGTH_SHORT)
            .setAction(R.string.undo) {
                lifecycleScope.launch {
                    movement.amount *= -1
                    BudgetUtil.tryAddMovementToBudget(movement)
                    app.movementRepository.insert(movement)
                    viewModel.updateMovements()
                    StatsUtil.addMovementToStat(app, movement)
                }
            }.show()
    }
}
```

Explonation :-

This is another fragment in Android application, responsible for displaying and managing the history of movements. Let's go through the key parts of the code:

1. Imports: Similar to the previous fragment, necessary classes from the Android framework and custom classes within the project are imported.

2. Class Declaration: `HistoryFragment` is declared as a subclass of `Fragment`.

3. Properties:

  - `viewModel`: An instance of `HistoryViewModel`, responsible for managing the data and business logic related to movement history.

  - `_binding`: An instance of `FragmentHistoryBinding`, generated by View Binding, which holds references to views within the fragment layout.

4. Lifecycle Methods:

  - `onCreateView()`: Inflates the fragment's layout, sets up data binding, and initializes UI components.

  - `onCreate()`: Initializes the view model.

  - `onDestroyView()`: Cleans up resources, such as the binding instance.

5. Setup Methods:

  - `setupRecyclerView()`: Configures and populates the RecyclerView for displaying movement history. It sets adapters, layout managers, observes changes in movements, and updates the UI accordingly.

  - `setupRecyclerGestures()`: Configures swipe gestures (left and right) on RecyclerView items for editing and deleting movements.

  - `setupBottomSheet()`: Sets up the bottom sheet for applying filters to the movement history. It handles the state changes of the bottom sheet, sets click listeners for buttons, and configures the tag picker.

  - `setupTagPicker()`: Sets up the tag picker for selecting tags to filter movements.

6. Event Handling:

- `setFilteredItems()`: Filters the movements based on search query and tag filtering. It updates the RecyclerView with the filtered items.

- `onRecyclerClick()`: Handles click events on the RecyclerView by clearing focus and hiding the keyboard.

- `onRemoveMovementInvoked()`: Handles the action when a movement is swiped to delete. It deletes the movement from the database, updates budgets and stats, and shows a Snackbar with an undo option.

This fragment provides functionality for managing and viewing movement history, including filtering, sorting, and deleting movements, as well as applying filters based on tags.

III Home page :-

```
package com.ferrariofilippo.saveapp.view


import androidx.lifecycle.ViewModelProvider

import android.os.Bundle

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import androidx.fragment.app.Fragment

import com.ferrariofilippo.saveapp.MainActivity

import com.ferrariofilippo.saveapp.databinding.FragmentHomeBinding

import com.ferrariofilippo.saveapp.view.viewmodels.HomeViewModel


class HomeFragment : Fragment() {

    private lateinit var viewModel: HomeViewModel


    private var _binding: FragmentHomeBinding? = null

    private val binding get() = _binding!!


    // Overrides

    override fun onCreateView(

        inflater: LayoutInflater,

        container: ViewGroup?,

        savedInstanceState: Bundle?

    ): View {
```

```
    _binding = FragmentHomeBinding
        .inflate(inflater, container, false)
        .apply {
            lifecycleOwner = viewLifecycleOwner
            vm = viewModel
        }

    setupUI()

    return binding.root
}

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    viewModel = ViewModelProvider(this)[HomeViewModel::class.java]
}

override fun onDestroyView() {
    super.onDestroyView()
    _binding = null
}

// Methods
private fun setupUI() {
    binding.settingsButton.setOnClickListener {
        (activity as MainActivity).goToSettings()
    }

    binding.subscriptionsButton.setOnClickListener {
        (activity as MainActivity).goToSubscriptions()
    }
}
```

}

Explonation:-

This fragment appears to represent the home screen of your application. Here's what it does:

1. Imports: Import necessary classes, including `ViewModelProvider`, `Bundle`, `LayoutInflater`, `View`, `ViewGroup`, and `Fragment`.

2. Class Declaration: `HomeFragment` is declared as a subclass of `Fragment`.

3. Properties:

- `viewModel`: An instance of `HomeViewModel`, which likely manages data and business logic related to the home screen.

- `_binding`: An instance of `FragmentHomeBinding`, generated by View Binding, which holds references to views within the fragment layout.

4. Lifecycle Methods:

- `onCreateView()`: Inflates the fragment's layout, sets up data binding, initializes UI components, and sets up click listeners for UI elements.

- `onCreate()`: Initializes the view model.

- `onDestroyView()`: Cleans up resources, such as the binding instance.

5. Setup Methods:

- `setupUI()`: Sets up the user interface by assigning click listeners to buttons (`settingsButton` and `subscriptionsButton`). When clicked, these buttons navigate the user to settings or subscriptions, respectively.

Overall, this fragment seems to be simple, responsible for displaying the home screen layout and providing navigation options to other parts of the application.

IV About applicatioin :-

This `SaveAppApplication` class extends the `Application` class in Android and serves as the entry point for your application. Here's a breakdown of its key components:

1. Companion Object:

- `settingsFileName`: A constant representing the filename for storing application settings.

2. Properties:

- `applicationScope`: An instance of `CoroutineScope` used for managing coroutines within the application.

- `database`: An instance of `AppDatabase`, representing the application's database. It's lazily initialized using the `getInstance()` method.

- Repositories: Instances of various repositories (`movementRepository`, `budgetRepository`, `subscriptionRepository`, `tagRepository`, `utilRepository`), which provide access to data and handle data operations.

- `ratesStore`: An instance of `DataStore` for storing currency exchange rates.

- `settingsStore`: An instance of `DataStore` for storing application settings.

- `currentActivity`: A reference to the currently active `Activity`.

3. Lifecycle Methods:

- `onCreate()`: Initializes various components of the application, such as the database and repositories.

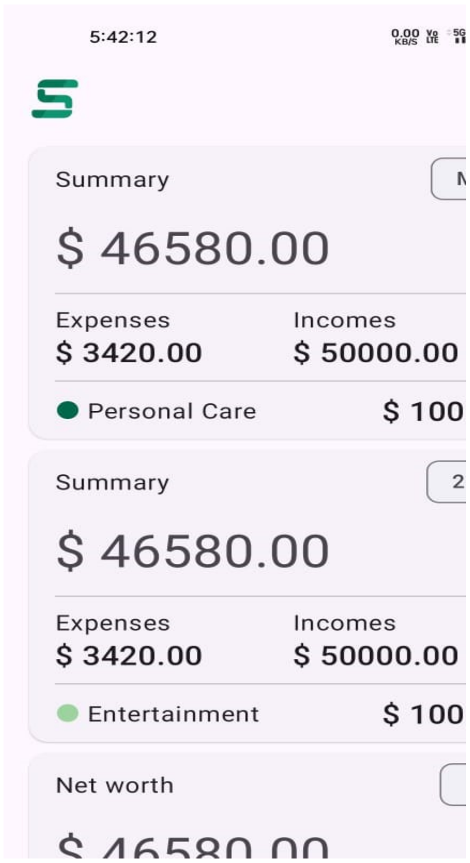
4. Methods:

- `getCurrentActivity()`: Retrieves the currently active `Activity`.

- `setCurrentActivity()`: Sets the currently active `Activity`.

Overall, this class initializes essential components of your application, such as the database and repositories, and provides access to them throughout the application's lifecycle. Additionally, it manages the storage of application settings and currency exchange rates using `DataStore`.

Screenshots of emulator and activities:-



5:42:12

Summary

$ 46580.00

Expenses $ 3420.00

Incomes $ 50000.00

● Personal Care $ 100

Summary

$ 46580.00

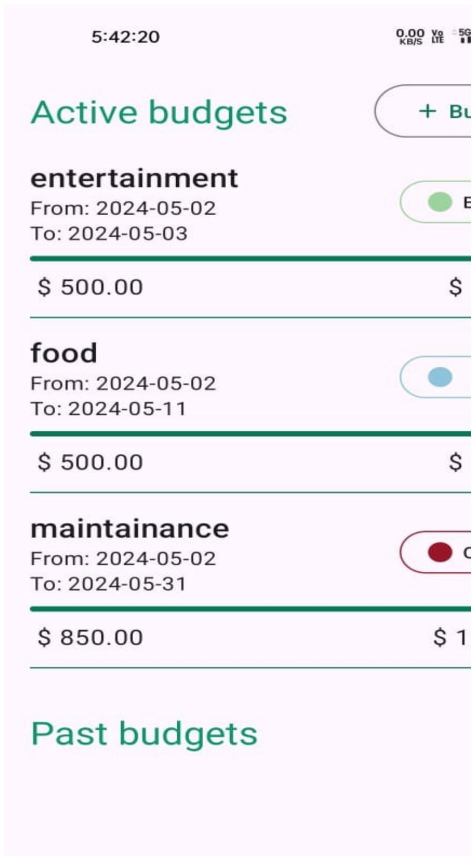
Expenses $ 3420.00

Incomes $ 50000.00

● Entertainment $ 100

Net worth

$ 46580.00



5:42:20

Active budgets

+ Bu

entertainment

From: 2024-05-02

To: 2024-05-03

$ 500.00 $

food

From: 2024-05-02

To: 2024-05-11

$ 500.00 $

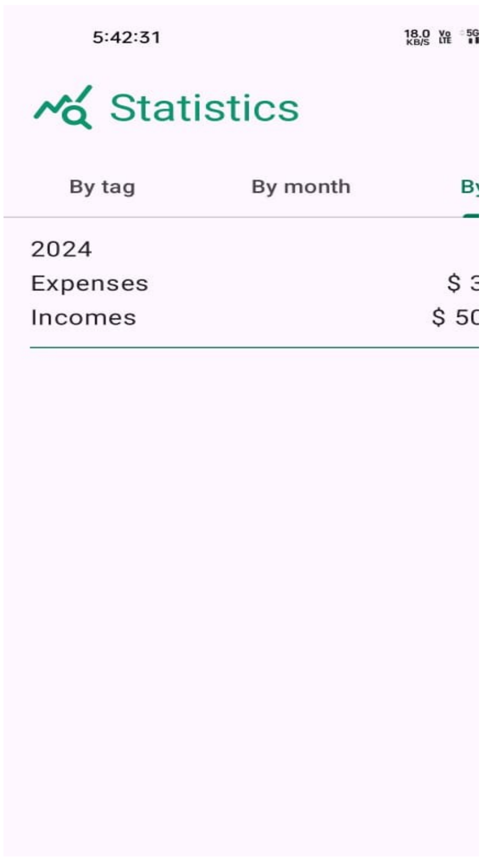
maintainance

From: 2024-05-02

To: 2024-05-31

$ 850.00 $ 1

Past budgets



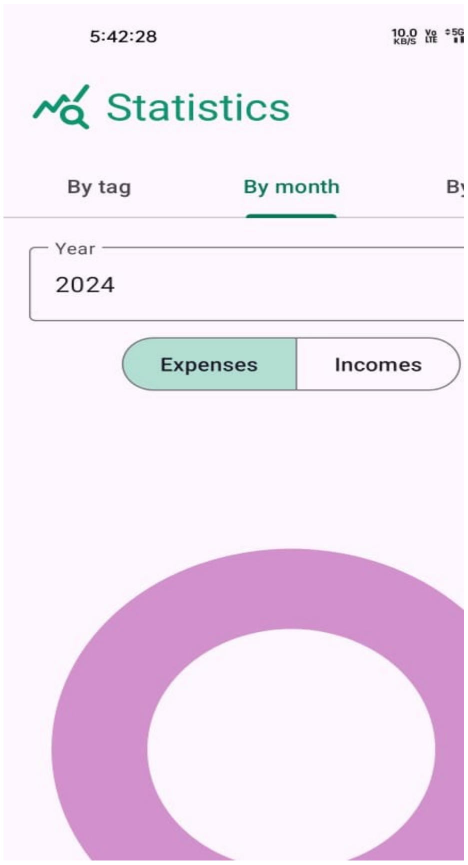
5:42:31

Statistics

By tag By month By

2024

Expenses	$ 3
Incomes	$ 50

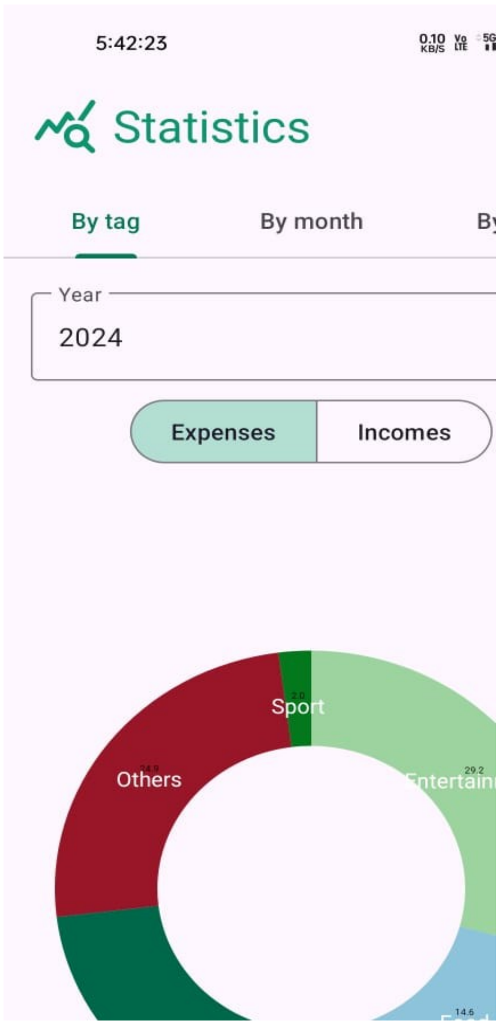


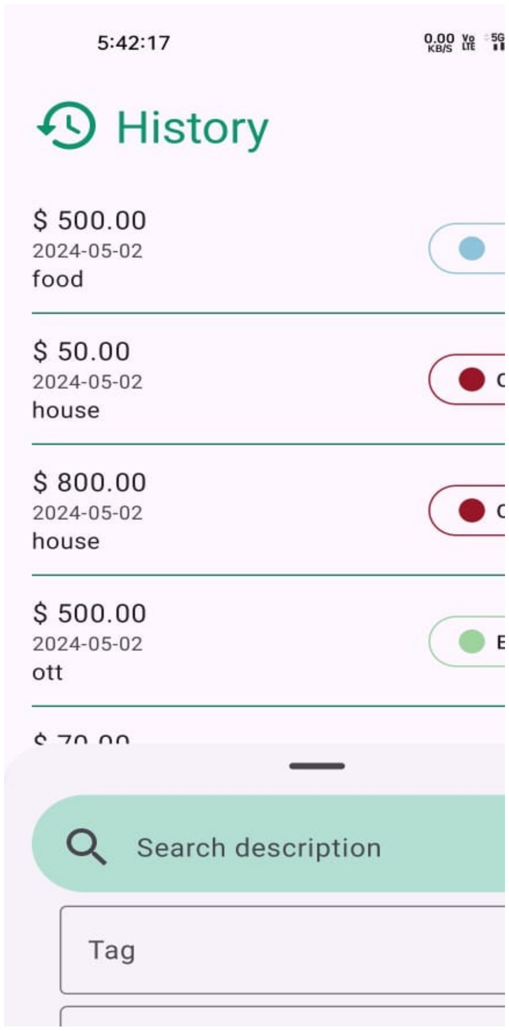
5:42:28

Statistics

By tag By month By

Year
2024



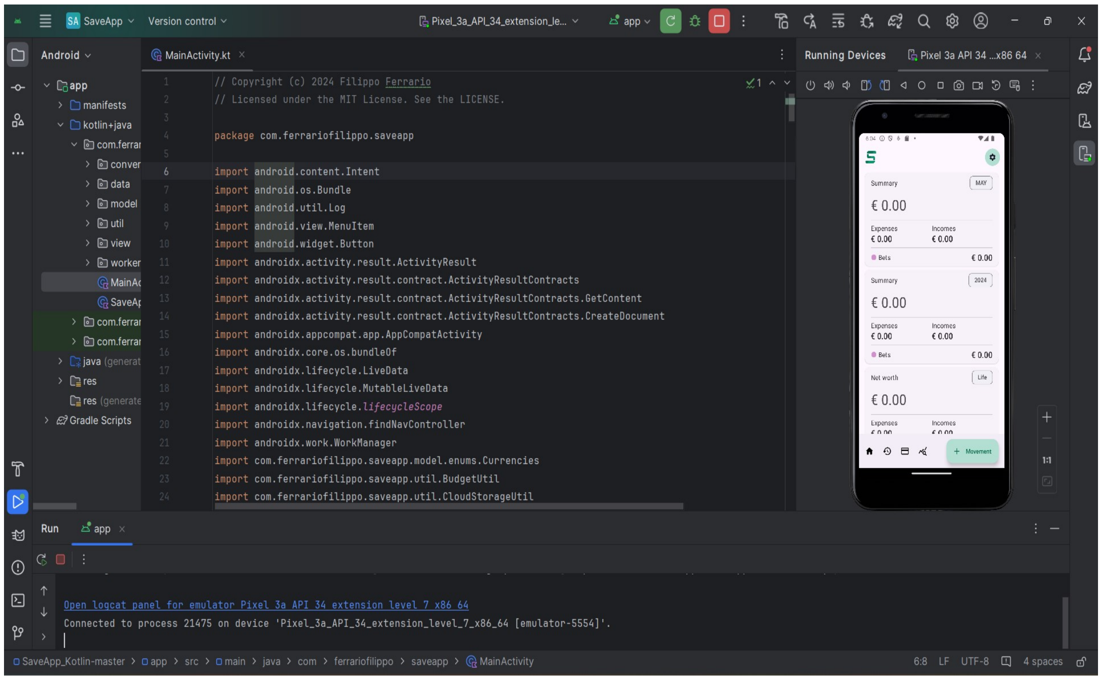


$ 500.00
2024-05-02
food

$ 50.00
2024-05-02
house

$ 800.00
2024-05-02
house

$ 500.00
2024-05-02
ott



```
// Copyright (c) 2024 Filippo Ferrario
// Licensed under the MIT License. See the LICENSE.

package com.ferrariofilippo.saveapp

import android.content.Intent
import android.os.Bundle
import android.util.Log
import android.view.MenuItem
import android.widget.Button
import androidx.activity.result.ActivityResult
import androidx.activity.result.contract.ActivityResultContracts
import androidx.activity.result.contract.ActivityResultContracts.GetContent
import androidx.activity.result.contract.ActivityResultContracts.CreateDocument
import androidx.appcompat.app.AppCompatActivity
import androidx.core.os.bundleOf
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.lifecycleScope
import androidx.navigation.findNavController
import androidx.work.WorkManager
import com.ferrariofilippo.saveapp.model.enums.Currencies
import com.ferrariofilippo.saveapp.util.BudgetUtil
import com.ferrariofilippo.saveapp.util.CloudStorageUtil
```

```
Open logcat panel for emulator Pixel 3a API 34 extension level 7 x86 64
Connected to process 21475 on device 'Pixel_3a_API_34_extension_level_7_x86_64 [emulator-5554]'.
```

SaveApp_Kotlin-master > app > src > main > java > com > ferrariofilippo > saveapp > MainActivity

Conclusion & Future Scope: -

Conclusion:-

In short, this app is here to make managing your money a breeze. It's designed to be user-friendly, so you can easily keep track of your finances without any hassle. this app prioritize your security, ensuring that your sensitive information stays safe and protected. Whether you're saving up for a big purchase or just want to keep tabs on your daily expenses, our app has everything you need to stay on top of your finances. With this simple interface and powerful features, taking control of your financial future has never been easier.

Future Scope:-

Looking ahead, excited to enhance this app with even more useful features. One area that focusing on is integrating with banks and financial institutions. This will allow to sync your accounts seamlessly, giving you real-time updates on your transactions and balances. Also exploring ways to automate expense tracking, using advanced technology to categorize your expenses automatically. This will save you time and effort, making it even easier to stay on budget.

Another area of focus for this app is providing personalized insights and recommendations. By analyzing your spending habits, this can offer tailored advice on how to save more and spend wisely. Additionally, considering adding features for expense sharing, making it simple to split bills with friends or roommates. And to help you make the most of your money, and planning to provide educational resources and tips on financial management. Whether you're a beginner or an expert, there's always more to learn about managing your finances.

In summary, the future of this app is committed to continually improving and expanding features to better serve daily needs. From seamless bank integration to personalized financial advice, dedicated to helping you achieve financial goals.

Github link :- https://github.com/sklatheef/Financial-Tracker-Assistance