Habibi 3

Habits Tracker App

Project Overview

Name: Habibi 3

Platform: Android

Programming Language: Kotlin

UI Framework: Jetpack Compose

Local Data Storage: Room Database

Background Tasks: WorkManager

Asynchronous Operations: Kotlin Coroutines

***Creators**:*

Mariia Chinkova, Yana Hrynevich, Valeriia Kurinna

Main Features

The app offers a simple habit tracking system that allows users to monitor their daily and weekly habits efficiently. It provides functionality for adding, editing, and deleting habits, users can adapt their habit lists to their changing needs. Additionally, the app has a calendar view that allows users to visualize their habits over time and easily track their progress. For added convenience, habits can be reset daily or weekly, allowing users to maintain consistency in their habit-building journey.

Application Structure

Here I want to present a simple structure of the project.

There are some packages for all needs of the application, and all files are divided to tied parts by these packages.

All of them are in main package — **com.example.habibi_3**.

Then **db** package for all files connected to database.

And finally **interfaces** for UI and design here is interface package.

Setup Instructions

Environment Setup: Ensure you have the latest Android Studio installed with Kotlin support.

Opening Project: Open Android Studio, select "Open an Existing Project," and navigate to the project directory.

Dependencies: Ensure all dependencies are correctly specified in your build.gradle files.

Build and Run: Select an Android emulator or connect a device, then build and run the project.

Developer Instructions

Detailed description of all files inside and used functions one by one.

• MainActivity

MainActivity.kt in the com.example.habibi_3 package serves as the entry point for a habit tracking Android application built with Jetpack Compose. This class extends ComponentActivity and sets up the application's main user interface and periodic work requests for resetting daily and weekly habits.

Key Features

Periodic Work Requests: The activity schedules two periodic work tasks using Android's WorkManager for resetting daily and weekly habits at specific times.

User Interface Initialization: The setContent block is used to define the UI content for the activity using Composable functions, wrapped within a custom theme Habibi3Theme.

Usage

onCreate: This lifecycle method is overridden to perform initial setup, including scheduling work requests and setting the UI content.

WorkManager: Used to schedule and manage periodic work requests for habit resetting.

setContent: Defines the UI of the activity using Composable functions. The Habibi3Theme is applied to style the Composables according to the application's theme.

Work Requests Setup

Daily Work Request: Scheduled to run every 24 hours with an initial delay calculated to trigger the task at 23:59 today.

Weekly Work Request: Scheduled to run every 7 days with an initial delay calculated to trigger the task at 23:59 on the next Sunday.

Integration

MyApp Composable: The root Composable function called within setContent, serving as the container for the application's UI.

• DAOInterfaces

DaoInterfaces.kt within the com.example.habibi_3 package defines the Data Access Object (DAO) interfaces for accessing the application's database in a habit tracking application. These interfaces use annotations provided by the Room Persistence Library to map Kotlin functions to SQL queries, enabling straightforward database operations.

Key Components

HabitDao: Interface for managing CRUD (Create, Read, Update, Delete) operations related to Habit entities in the application's database. There're exactly the habits that we see on the main screen.

HabitLogDao: Interface for managing CRUD operations for HabitLog entities, which represent log entries for each habit tracking. There're exactly the habits that we see in the list of completed/not completed on the calendar screen.

Key Methods

HabitLogDao

addHabit(): Inserts a new habit into the database. If a conflict occurs, the operation is ignored.

updateHabit(): Updates an existing habit in the database.

deleteHabit(): Deletes a habit from the database.

getAllHabits(): Retrieves all habits stored in the database, ordered by their IDs in descending order.

getDailyHabits(): Fetches all habits marked as 'Daily'.

getWeeklyHabits(): Fetches all habits marked as 'Weekly'.

getHabit(): Retrieves a single habit by its ID.

deleteAll(): Deletes all habit entries from the database.

updateAllDailyHabitsIsDone(): Updates the isDone status for all daily habits.

updateAllWeeklyHabitsIsDone(): Updates the isDone status for all weekly habits.

Similar methods are implemented for HabitLogDao for logs and there is no need to duplicate them.

• Database

Database.kt within the com.example.habibi_3 package is a Kotlin file that defines the Room database for a habit this application. It includes entity definitions for Habit and HabitLog, type converters for handling LocalDate, and the abstract HabitDatabase class that extends RoomDatabase.

Key Features

Type Converters: Enables storing custom types, such as LocalDate, in the SQLite database by converting them to a compatible format. Automatically used by Room when reading from or writing to the database fields of type LocalDate.

Entity Definitions: Defines the schema for the habit_table and habit_log_table in the SQLite database, including the structure and types of data that will be stored. Serve as the schema definition for the database. When performing database operations (e.g., queries, inserts, updates), these entities are used to represent rows in the corresponding tables.

Room Database: Provides an abstraction layer over SQLite to allow for more strong database access and easier data handling within the application. To use the database, call HabitDatabase.getDatabase() with a valid Context. This method ensures that only one instance of the database is created and used throughout the application (singleton pattern).

- MyApp

MyApp.kt within the com.example.habibi_3 package defines the root composable function for the habit tracking Android application. It sets up the navigation architecture and enabling the transition between different screens within the app.

Navigation Routes

Navigation Setup: Utilizes rememberNavController and NavHost to configure the navigation graph for the application, specifying routes to different screens.

Main Screen: Serves as the application's home screen, displaying an overview of habits.

Edit Habit Screen: Allows users to edit the details of an existing habit, identified by habitId.

Add Habit Screen: Provides a UI for creating a new habit.

Calendar Screen: Displays a calendar view for tracking habit completion over time.

Settings Screen: Contains application settings and options for resetting habits.

- Repositories

Repositories.kt in the com.example.habibi_3 package defines repository classes for managing data operations related to habits and habit logs in a habit tracking Android application. These repositories abstract the data source layer, providing a clean API for the ViewModel to interact with the underlying data.

Components

HabitRepository: Manages data operations for Habit entities, including CRUD operations and specific queries like fetching all daily or weekly habits.

HabitLogRepository: Handles data operations for HabitLog entities, including CRUD operations and queries for fetching logs by date.

HabitRepository Methods

addHabit(): Inserts a new habit into the database.

updateHabit(): Updates an existing habit.

deleteHabit(): Deletes a habit from the database.

deleteAllHabits(): Clears all habit entries from the database.

getHabit(): Retrieves a habit by its ID.

updateAllDailyHabitsIsDone(): Marks all daily habits as done or not done.

updateAllWeeklyHabitsIsDone(): Marks all weekly habits as done or not done.

getAllDailyHabits(): Retrieves all habits marked as daily.

getAllWeeklyHabits(): Retrieves all habits marked as weekly.

HabitLogRepository Methods

addHabitLog(): Inserts a new habit log entry.

updateHabitLog(): Updates an existing habit log.

deleteHabitLog(): Deletes a habit log entry.

getDailyLogsByDate(): Fetches all daily habit logs for a given date.

getWeeklyLogsByDate(): Fetches all weekly habit logs for a given date.

Usage

Repositories are typically used within ViewModel classes to interact with the application's data layer, facilitating operations like fetching, inserting, updating, and deleting records.

LiveData and StateFlow properties are observed in the UI layer (Composable functions or Fragments) to update the UI reactively when data changes.

• CalendarUiModel

CalendarUiModel.kt is a Kotlin file within the com.example.habibi_3 package.

This file focuses on providing calendar-related features, such as displaying a calendar view, selecting dates and showing habit logs tied to specific dates.

Functions

CalendarHeader: Displays the calendar's header with navigational buttons (previous and next) and the currently selected date.

CalendarContentItem: Represents individual date items in the calendar. It uses a Card composable to display the day and date, indicating if it's the selected date.

CalendarContent: Arranges the CalendarContentItem composable in a LazyRow, allowing users to scroll through the visible dates in the calendar.

CalendarApp: Organizes the display of the calendar, including the header, the calendar content, and habit logs for the selected date.

CalendarScreen: The main composable function for the calendar feature, integrating all parts of the calendar UI and providing a scaffold for the layout.

HabitsListDisplay: A composable function to display a list of habit logs associated with the selected date in a vertically scrollable column.

• UIAddHabitScreen

UIAddHabitScreen.kt within the com.example.habibi_3 package is a Kotlin file that defines a composable function for the Add Habit screen in a habit tracking Android application. This screen allows users to create a new habit by entering details such as the habit's name and its frequency (daily or weekly).

Key Components

AddHabitScreen Composable: The main composable function that sets up the UI layout for adding a new habit, including text fields for the habit name and toggles for habit frequency.

Scaffold: Provides a basic Material Design layout structure, including app bars, floating action buttons, drawers etc.

UI Elements

Text Fields: Used for entering the name of the new habit.

Checkbox: Allows the user to select whether the habit is daily or weekly.

Button: Submits the new habit information to be saved.

- ## UIEditHabitScreen

UIEditHabitScreen.kt in the com.example.habibi_3 package defines a composable function for the Edit Habit screen in a habit tracking Android application. This screen allows users to modify details of an existing habit, including its name and frequency (daily or weekly), and provides options to save changes or delete the habit.

Key Components

EditHabitScreen Composable: The main composable function that sets up the UI layout for editing a habit, including text fields for the habit name, toggles for habit frequency, and buttons for saving changes or deleting the habit.

UI Elements

TextField: Used for editing the name of the habit.

Checkbox: Allows the user to toggle between daily and weekly frequency for the habit.

Buttons: Two buttons are provided, one for saving changes to the habit and another for deleting the habit.

- ## UIMainScreen

UIMainScreen.kt in the com.example.habibi_3 package defines the main screen of a habit tracking Android application using Jetpack Compose. It displays a list of habits categorized by their frequency (daily or weekly) and provides navigation options to other parts of the app.

Key Components

MainScreen: The primary composable function that sets up the layout for the main screen, displaying a list of habits and navigation buttons.

HabitList: A composable function that generates a scrollable list of habits using LazyColumn.

HabitRow: Represents a single row in the habit list, showing the habit's name and a checkbox to mark it as done.

MainText: Displays the main header text.

NavigationButtons: Provides buttons for navigating to the calendar, adding a new habit, and accessing settings.

UI Elements

LazyColumn: Used to display a list of habits in a scrollable column.

Card: Each habit is presented in a card format for clear delineation.

Checkbox: Allows users to mark a habit as completed directly from the main screen.

Buttons: Facilitate navigation to different parts of the application.

• UISettingsScreen

UISettingsScreen.kt in the com.example.habibi_3 package defines the Settings screen for a habit tracking Android application using Jetpack Compose. This screen provides options for users to reset daily or weekly habits and to navigate back to the main screen.

Key Components

SettingsScreen: The primary composable function setting up the UI layout for the settings screen, including buttons for resetting habits and a "Done" button for navigation.

Scaffold: Provides a basic Material Design layout structure, encapsulating the settings content.

UI Elements

Text: Displays the "Settings" title at the top of the screen.

Buttons: Provide options for resetting daily habits, resetting weekly habits, and navigating back to the main screen.

• ViewModels

ViewModels.kt in the com.example.habibi_3 package defines ViewModel classes for managing UI-related data and business logic in a habit tracking Android application. These classes extend AndroidViewModel and utilize repositories to interact with the data layer.

Components

HabitViewModel: Manages data and operations related to Habit entities, including fetching all habits, adding, updating, and deleting habits, as well as handling daily and weekly habit resets.

HabitLogViewModel: Manages data and operations related to HabitLog entities, such as adding, updating, and deleting habit logs, and fetching logs based on dates.

Key Features

LiveData and Flow: Both ViewModels use LiveData and Kotlin Flow to expose data to the UI layer, enabling reactive UI updates.

Coroutine Scopes: Operations that interact with the database are launched within the viewModelScope using Dispatchers.IO to ensure they are performed off the main thread.

HabitViewModel Methods

addHabit(): Adds a new habit to the database.

updateHabit(): Updates an existing habit.

deleteHabit(): Deletes a habit.

deleteAllHabits(): Deletes all habits from the database.

getHabit(): Retrieves a specific habit by ID.

setAllDailyHabitsDone(): Sets the completion status for all daily habits.

setAllWeeklyHabitsDone(): Sets the completion status for all weekly habits.

getAllDailyHabits(): Fetches all daily habits.

getAllWeeklyHabits(): Fetches all weekly habits.

HabitLogViewModel Methods

addHabitLog(): Adds a new habit log entry.

updateHabitLog(): Updates an existing habit log.

deleteHabitLog(): Deletes a habit log.

getDailyLogsByDate(): Fetches all daily habit logs for a specified date.

getWeeklyLogsByDate(): Fetches all weekly habit logs for a specified date.

- Workers

Workers.kt in the com.example.habibi_3 package defines worker classes for performing background tasks in a habit tracking Android application. These tasks include resetting the completion status of daily and weekly habits at specified intervals.

Components

ResetDailyHabitsWorker: Similar to ResetWeeklyHabitsWorker, but for daily habits. It resets the completion status of all daily habits to not done and logs each reset in the habit log.

ResetWeeklyHabitsWorker: A worker class extending CoroutineWorker for resetting all weekly habits. It marks all weekly habits as not done and logs this action in the habit log.

Key Features

Coroutine Support: Both workers use Kotlin Coroutines (suspend fun doWork()) for asynchronous operations, ensuring that the tasks do not block the main thread.

Database Interaction: Workers access the app's Room database directly to update habit completion statuses and log the resets.

Worker Tasks

ResetWeeklyHabitsWorker:

Fetches all weekly habits using getWeeklyHabits() from HabitDao.

Iterates over each habit, creating a HabitLog entry to mark the reset action.

Updates the isDone status of all weekly habits to false.

ResetDailyHabitsWorker:

Performs similar actions to ResetWeeklyHabitsWorker, but for daily habits.

Ensure that database operations within workers are efficient and do not lead to contention or locking issues, especially when scheduled tasks run frequently.

Usage Instructions

Main Screen: View your daily and weekly habits. Tap on a habit to mark it as «done» or «undone».

Add Habit: Use the "Add Habit" button to create a new habit. Specify the name and frequency (daily or weekly).

Edit Habit: Tap on a habit to edit its name or frequency.

Calendar View: Access the calendar view to see your habit logs. Select a date to view or add logs.

Settings: Reset all daily or weekly habits through the settings screen.