Documented by Samreen Chowlkar

Task Logger App

This document serves as a comprehensive guide to the **Task Logger** mobile application, developed using Kotlin and Jetpack Compose. The primary purpose of the application is to assist users in managing their daily tasks by providing a simple and intuitive interface to add, view, and track tasks.

The application's core functionality is centered around two main features: a real-time countdown timer that displays the time remaining until the next upcoming task, and a dynamic list that highlights this task for immediate visibility. This design focuses on a seamless user experience, ensuring the user can quickly identify and prepare for their most urgent task.

The project is built upon a modern and scalable architecture, adhering to best practices in Android development. It leverages the following key technologies:

- **Kotlin:** The primary programming language for all application logic.
- **Jetpack Compose:** The modern UI toolkit for building native Android user interfaces.
- **Room Persistence Library:** A robust database layer for local data storage, providing a reliable source of truth for all task data.
- **MVVM (Model-View-ViewModel) Architecture:** A clean and maintainable architectural pattern that separates the UI logic from the data handling.

This combination of technologies ensures the application is not only functional but also performant, scalable, and easy to maintain and extend in the future.

Architecture and Code Structure

The Task Logger app is structured using the **MVVM (Model-View-ViewModel)** architectural pattern. This separation of concerns allows for a modular codebase where the UI, business logic, and data layers are distinct and reusable.

Architectural Breakdown

- **View (UI):** Represented by the Composable functions in `MainActivity.kt`. The View's responsibility is to observe data from the ViewModel and display it to the user. It does not contain business logic.

- **ViewModel:** Defined in `TaskViewModel.kt`. The ViewModel holds and manages UI-related data in a lifecycle-conscious way. It acts as a bridge between the View and the Repository, handling user actions and updating the data as needed.
- **Repository:** The `TaskRepository.kt` class. This component is the single source of truth for all data. It abstracts the data source (in this case, the Room database) from the rest of the application. It handles data insertion and deletion.
- **Model:** The `Task.kt` data class and the Room database components. The `Task` class is the data entity, while the `TaskDao.kt` provides the methods to interact with the database.

Key Files and Their Roles

- **`build.gradle.kts`**: This file manages the project's dependencies, including those for Jetpack Compose, Room, and ViewModel. It ensures all necessary libraries are included and compatible.
- **`Task.kt`**: Defines the `Task` data entity using Room annotations. It specifies the table name and primary key, linking the Kotlin data class to a database table.
- **`TaskDao.kt`**: This is a Data Access Object (`@Dao`) that defines the methods for interacting with the `tasks` table. It contains queries to get all tasks, insert a new task, and delete a task.
- **`TaskDatabase.kt`**: An abstract class that represents the Room database. It defines the database version and links the `Task` entity and `TaskDao`. It also implements a singleton pattern to ensure only one instance of the database is created.
- **`TaskRepository.kt`**: The repository class that provides a clean API for the ViewModel to interact with the database. It exposes a `Flow` of all tasks, which the ViewModel observes for real-time updates.
- **`TaskViewModel.kt`**: The central component for managing UI data. It uses coroutines to execute database operations on a background thread, preventing the UI from freezing.
- **`MainActivity.kt`**: This is the main entry point for the UI. It contains the core Composable functions:
- **`TaskLoggerApp`**: The root composable that sets up the UI structure.
- **`HeaderSection`**: A composable that displays the current date and the next task's name and countdown timer.
- **`CountdownTimer`**: A separate, reusable composable that handles the live countdown logic.
- **`TaskList`**: A `LazyColumn` for efficiently displaying the list of tasks.

- `TaskItem`: Defines the visual representation of each task, including the highlighting effect for the upcoming task.
- `AddTaskDialog`: A dialog for adding new tasks with a name and time.

How to Use the App

The application features a straightforward user flow:

1. **Add a Task:** Tap the floating "Add" button at the bottom of the screen. A dialog will appear.
2. **Input Details:** Enter the task name and use the time pickers to set the due time.
3. **Save:** Tap the "Add" button in the dialog to save the task to the database.
4. **View Tasks:** The main screen automatically updates to show all tasks. The next upcoming task will be highlighted with a blue background.
5. **Track Time:** The `HeaderSection` will display the name of the next task and a live countdown to its due time.
6. **Delete a Task:** Tap the "Delete" icon next to any task to remove it from the list.

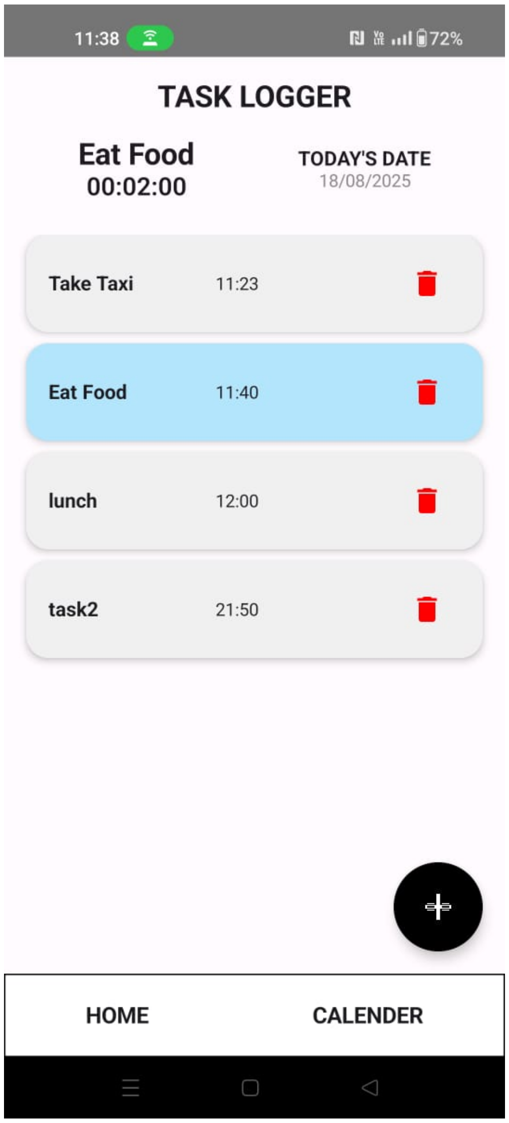
Recommended Future Enhancements

The current application provides a solid foundation, and several features could be added to enhance its functionality and user experience.

1. **Task Editing and Completion:** Implement a feature to allow users to edit existing tasks. A checkbox could be added to each `TaskItem` to allow users to mark tasks as completed, which would move them to a different list or visually distinguish them.
2. **User-Configurable Notifications:** Add a setting for users to receive push notifications a certain amount of time before a task is due (e.g., 15 minutes before). This would require a broadcast receiver and an alarm manager.
3. **Full Calendar View:** Integrate a full calendar view to allow users to see tasks scheduled for different days, not just the current day. This would require extending the `Task` entity to include a date field.
4. **Sorting and Filtering:** Add options to sort tasks by name or due time, and to filter the list to show only completed or incomplete tasks.
5. **Improved Time Picker:** While functional, the current time picker could be replaced with a more user-friendly and standard Material Design time picker for a better user experience.

6. **Data Synchronization:** Implement a backend service and use a library like Firebase to allow users to synchronize their tasks across multiple devices.

Screenshort



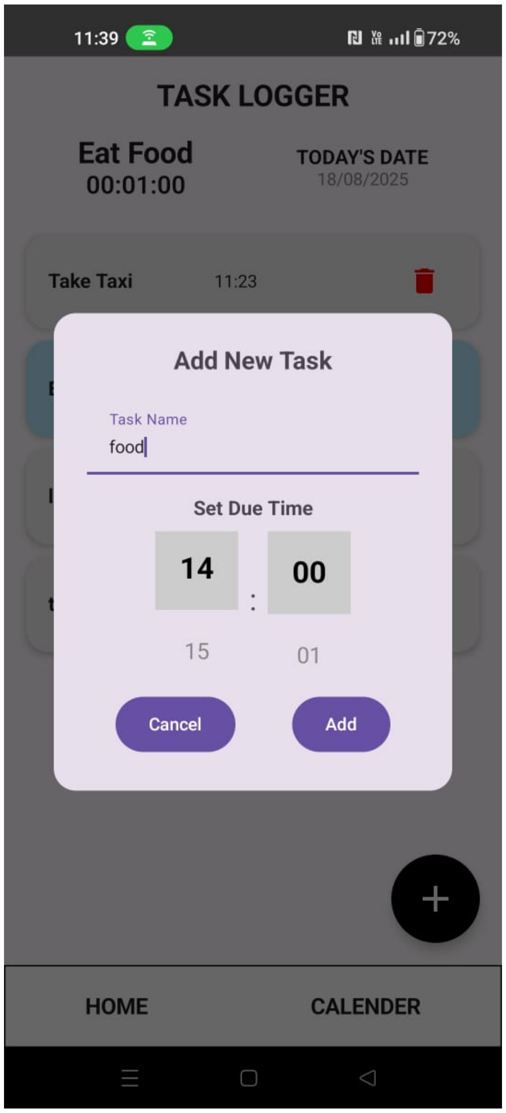
11:38 72%

TASK LOGGER

Eat Food
00:02:00

TODAY'S DATE
18/08/2025

Take Taxi	11:23
Eat Food	11:40
lunch	12:00
task2	21:50



11:39 72%

TASK LOGGER

Eat Food
00:01:00

TODAY'S DATE
18/08/2025

Take Taxi 11:23

Add New Task

Task Name
food

Set Due Time
14 : 00
15 01