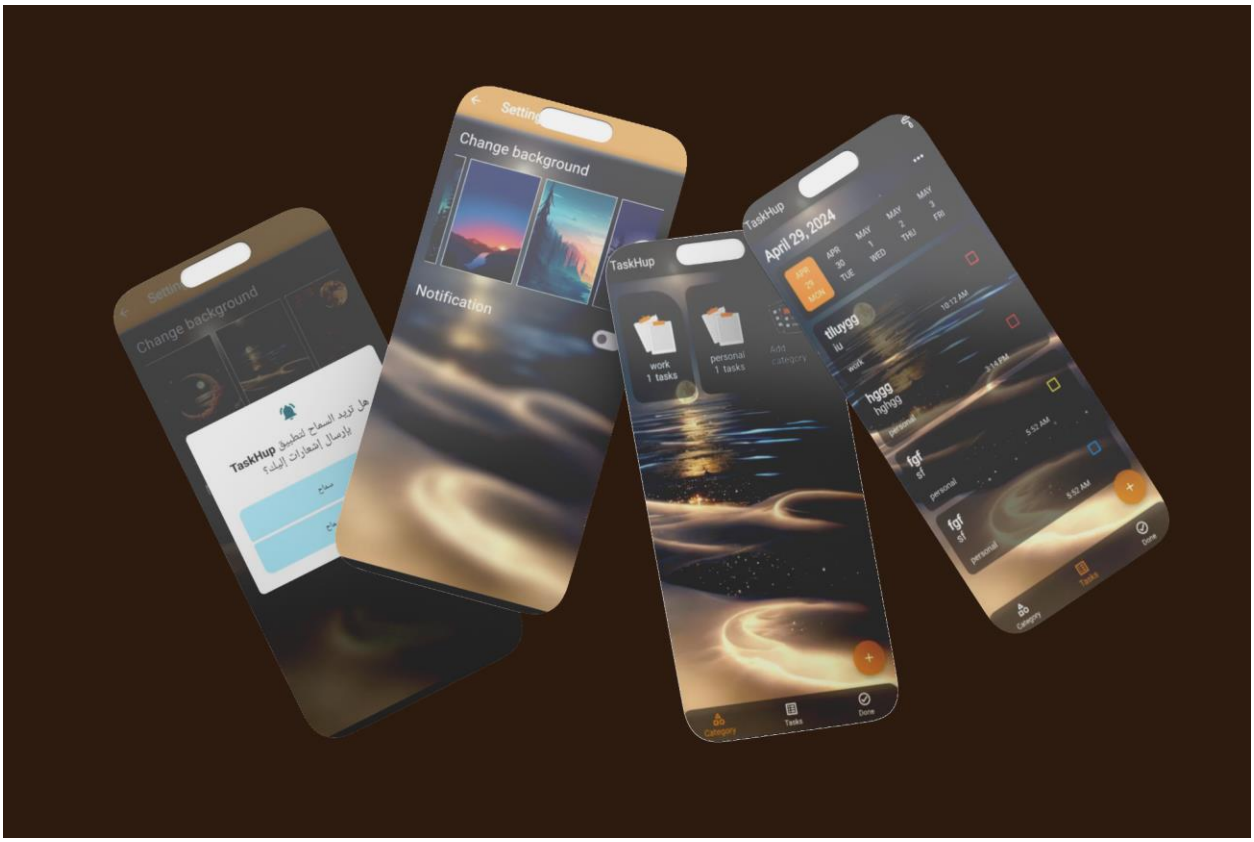
TaskHup App



Team members: -

نرمين محمد صادق

مريم محمد فتوح

نهي طعيمه إبراهيم

خلود رضا فايد

نادين محمود احمد

محمد ايمن حسن

احمد الوزير

Habit Flow

Introduction:

The Task app is your perfect companion for achieving daily productivity! It’s designed specifically to help you organize and manage your tasks with ease and speed. Whether you're looking to add new tasks, update lists, or prioritize them, this app provides all the tools you need to stay on top of your organization. It also gives you full control over the appearance of the app thanks to its background-changing feature, allowing you to customize the user experience to your liking. Built using Flutter, the app ensures great performance and a seamless experience across all devices. Get ready to explore its advanced features and make task completion more enjoyable than ever before!

Key Features:

1. Add Task

The "Add Task" feature allows users to create new tasks in the app. Users can input a task name, description, due date, priority, and assign a category. The task will be stored in the local SQLite database (sqflite).

Implementation:

- Use a form to collect task details.
- Store the task in the database using the sqflite package.
- After the task is added, display a success message using the awesome_dialog package.

Libraries Used:

- sqflite for local storage.
- flutter_bloc to manage the state of the task list after adding a new task.
- awesome_dialog for showing success messages.

2. Delete Task

The "Delete Task" feature allows users to remove a task from their task list. Once a task is no longer relevant, the user can delete it, which will remove the task from the local database and update the UI accordingly.

Implementation:

- Deleting a task triggers an action that removes it from the SQLite database.
- Once the task is deleted, a confirmation dialog is shown using awesome_dialog.

- The state of the task list is managed by flutter_bloc to ensure proper removal and update of the task list.

Libraries Used:

- sqflite for removing data from the database.
- flutter_bloc to manage state changes.
- awesome_dialog for confirmation dialogs.

3. Update Task

The "Update Task" feature allows users to modify an existing task. Users can edit the task details like the name, description, due date, etc., and save the changes.

Implementation:

- Retrieve the existing task details from the local database using sqflite.
- Display the task details in an editable form.
- Update the database with the modified task details.
- Use flutter_bloc to reflect the changes in the UI.

Libraries Used:

- sqflite for updating the database.
- flutter_bloc to update the task list.

4. Category Management

Users can categorize tasks to help organize their workflow better. Categories could include "Work," "Personal," "Shopping," etc. Each task can be assigned to one or more categories, and users can view tasks by category.

Implementation:

- Categories are predefined or user-generated and stored locally.
- Tasks can be assigned to categories.
- Use flutter_staggered_grid_view to display tasks grouped by categories in an intuitive grid layout.

Libraries Used:

- sqflite for storing categories.
- flutter_staggered_grid_view for presenting tasks by category in an organized manner.

5. Sort Tasks

Users can sort tasks based on various criteria such as due date, priority, or category. This feature helps users focus on high-priority tasks or upcoming deadlines.

Implementation:

- Sorting can be done directly in the database query using sqflite.
- The flutter_bloc library is used to update the UI with the sorted task list.

Libraries Used:

- sqflite for sorting queries.
- flutter_bloc for managing the sorted list.

6. Filter Tasks

Users can filter tasks based on categories, completion status, or due dates. Filtering allows users to see only the tasks that are relevant at a given moment.

Implementation:

- Use dropdown menus or toggle buttons to allow the user to apply filters.
- The filtering logic is applied to the task list, and only tasks that match the criteria are shown.
- Manage the state of the filtered list using flutter_bloc.

Libraries Used:

- flutter_bloc to manage filter states and update the UI accordingly.
- sqflite to retrieve filtered data from the database.

7. Change Background

The app allows users to personalize their experience by changing the app's background. Users can select from various background images or colors, and the selected option is saved in the user's preferences.

Implementation:

- Use shared_preferences to store the selected background choice.
- Apply the selected background when the app is loaded.
- Use cached_network_image if background images are downloaded from the internet.

Libraries Used:

- shared_preferences to store and retrieve the background settings.
- cached_network_image for loading images efficiently.

8. Notifications

TaskHup provides timely reminders for tasks by sending push notifications to the user. These notifications alert users about remind them to create new tasks, ensuring that nothing is missed.

Implementation:

- **flutter_local_notifications**: This package is used to display push notifications to users, allowing them to be notified even when the app is in the background or closed.
- **Scheduled Notifications**: Notifications are scheduled based on user input for specific tasks and times.
- **Reminders for Tasks**: Notifications can remind users to complete or review tasks at pre-set times, helping them stay on track.

Libraries Used:

- **flutter_local_notifications**: Handles notification scheduling and delivery.
- **timezone**: Ensures that notifications are accurate across different time zones.
- **flutter_timezone**: Manages timezone-related functionality to ensure proper notification timing.

9. Work Manager

The WorkManager in TaskHup handles background tasks such as delivering notifications, ensuring that task reminders are sent even if the app is closed. This allows users to receive timely reminders without needing to have the app open or running in the foreground.

Implementation:

- **workmanager**: The app leverages WorkManager to execute notifications and other periodic background tasks like syncing data.
- **Background Reminders**: The WorkManager is configured to trigger reminders for tasks based on their schedule, even when the app isn't actively running.
- **Efficient Background Task Execution**: WorkManager ensures efficient and reliable background operations without draining battery or affecting app performance.

Libraries Used:

- **workmanager**: Manages the execution of background tasks.
- **flutter_local_notifications**: Delivers notifications triggered by background tasks.

Libraries and Packages:

1. cupertino_icons: ^1.0.2

- **Purpose**: Provides Cupertino-style icons specifically designed for iOS devices.
- **Usage:** Use these icons throughout the app to maintain a consistent iOS design language.
- **Benefits:**
    - Ensures a native iOS look and feel.
    - Users instantly recognize familiar icons, improving usability.

2. bloc: ^8.1.2 & flutter_bloc: ^8.1.2

- **Purpose:** Enables predictable state management using the BLoC (Business Logic Component) pattern.
- **Usage:** Manage the app's state and handle events such as adding, updating, or deleting tasks.
- **Benefits:**
    - Simplifies state management by following a predictable pattern.
    - Enhances scalability for complex applications and improves code maintainability.

3. conditional_builder_null_safety: ^0.0.6

- **Purpose:** Simplifies the rendering of widgets based on conditions, particularly for handling null values.
- **Usage:** Ensure that tasks are displayed only when the list is not empty.
- **Benefits:**
    - Reduces boilerplate code for conditional rendering.
    - Enhances user experience by managing null-safe widget rendering.

4. shared_preferences: ^2.2.2

- **Purpose:** Stores small amounts of data locally, such as user preferences.
- **Usage:** Retain settings like app background and notification configurations.
- **Benefits:**
    - Creates a personalized app experience by retaining user settings.

    - Efficient for small data storage needs without requiring a database.

5. flutter_screenutil: ^5.9.0

- **Purpose:** Ensures that UI components are responsive across various screen sizes and resolutions.
- **Usage:** Use it to define sizes and layouts that adapt to different devices.
- **Benefits:**
    - Simplifies responsive design implementation.
    - Enhances user experience by making the app look great on all devices.

6. awesome_dialog: ^3.2.0

- **Purpose:** Displays beautiful, customizable dialog boxes.
- **Usage:** Provide feedback to users on actions like task addition or deletion.
- **Benefits:**
    - Enhances user interaction with intuitive feedback dialogs.
    - Supports animations and theming, improving user engagement.

7. sqflite: ^2.3.0

- **Purpose:** A SQLite plugin for Flutter that handles local data storage.
- **Usage:** Store and manage the app's task data locally on the device.
- **Benefits:**
    - Ensures offline access to task data, enhancing reliability.
    - Supports complex queries for effective task management.

8. path: ^1.8.3

- **Purpose:** Provides utilities for handling file and directory paths.
- **Usage:** Often used in combination with sqflite for database file management.
- **Benefits:**
    - Simplifies handling of file paths, reducing potential errors.
    - Cross-platform support for file management on Android and iOS.

9. flutter_launcher_icons: ^0.13.1

- **Purpose:** Generates app launcher icons for both Android and iOS platforms.

- **Usage:** Create high-quality launcher icons that conform to platform standards.
- **Benefits:**
    - Enhances the app's visual identity and branding.
    - Ensures high-quality icons at all required resolutions.

10. rename_app: ^1.3.1

- **Purpose:** Facilitates easy renaming of the app package and project.
- **Usage:** Simplify project setup and maintenance by renaming packages easily.
- **Benefits:**
    - Streamlines the app renaming process, saving time and effort.
    - Avoids potential deployment issues with proper package naming.

11. cached_network_image: ^3.3.1

- **Purpose:** Efficiently loads images from the internet and caches them for offline use.
- **Usage:** Use for displaying images in tasks or other features of the app.
- **Benefits:**
    - Reduces data usage by caching images.
    - Speeds up image loading, improving overall app performance.

12. intl: ^0.18.1

- **Purpose:** Supports internationalization and localization.
- **Usage:** Format dates, numbers, and other locale-specific data within the app.
- **Benefits:**
    - Enhances user experience for diverse audiences.
    - Provides accurate formatting based on the user's locale.

13. date_picker_timeline: ^1.2.5

- **Purpose:** Provides a customizable date picker for selecting dates from a timeline.
- **Usage:** Help users manage tasks with specific due dates.
- **Benefits:**
    - Simplifies date selection for task management.
    - Enhances user interaction with a visually appealing date picker.

14. flutter_staggered_grid_view: ^0.7.0

- **Purpose:** Displays tasks in a staggered grid layout.
- **Usage:** Use for visually appealing task categorization and representation.
- **Benefits:**
    - Provides a unique way to display tasks, improving visual organization.
    - Enhances user experience with a modern UI layout.

Packages Configuration

- **State Management & Logic**:
    - bloc
    - flutter_bloc
    - conditional_builder_null_safety
- **Storage & Persistence**:
    - sqflite
    - shared_preferences
    - path
- **UI Enhancements**:
    - flutter_screenutil: For responsive UI design.
    - awesome_dialog: Customizable dialogs for user interactions.
    - flutter_staggered_grid_view: For grid-based layouts in task views.
    - carousel_slider: Enables carousel-like displays for content.
- **Notifications & Background Tasks**:
    - flutter_local_notifications: Displays notifications.
    - workmanager: Manages background tasks.
    - timezone & flutter_timezone: Handle timezone-specific reminders for accurate notification timing.
- **Utility & Other Packages**:

    - intl: For internationalization and localization of date formats.
    - date_picker_timeline: Displays date timelines for choosing task dates.
    - permission_handler: Manages app permissions.
- **App Customization**:
    - flutter_launcher_icons: Customize the app icon.
    - rename_app: Modify app name and details easily.
    - cached_network_image: For efficient image loading.

Conclusion

The Task app is a powerful tool for enhancing your productivity and efficiently managing your tasks. With its diverse features, you can easily add, update, and delete tasks, as well as categorize and prioritize them. The app also allows you to customize your experience by changing backgrounds to suit your personal taste and push notification to remind user to write tasks.

Built with Flutter, the app ensures smooth performance and a comfortable user experience across various devices. With its unique interface and ease of use, Task is the ideal choice for anyone looking to achieve better organization and effective management of daily tasks. Get ready to explore all its capabilities and move forward toward achieving your goals more efficiently!