Adama Science and Technology University

School: School of Electrical Engineering and Computing

Department: Software Engineering

Course Title: Mobile Application Design and Development

Assignment on: Implementing Application Design and Development

Section-3

Members	Id
1. Yonas Zekarias	ugr/25332/14
2. Hanamariam Mesfin	ugr/25483/14
3. Yonas adane	ugr/25464/14
4. Abuzer Jemal	ugr/25351/14

Submition date June25 2024

Table of Contents

1. Project overview

The purpose of this mobile application project is to develop a task manager app using Flutter, aimed at enhancing user organization and productivity. The app allows users to create, manage, and monitor tasks across different categories, thereby facilitating efficient task management. The target audience includes individuals and professionals seeking a streamlined approach to managing their daily tasks and responsibilities.

2. User requirements

Overview

Understanding the specific requirements of the users is crucial for designing and developing a task management application that meets their needs effectively. This section outlines the identified user requirements based on analysis and feedback, highlighting their influence on the app's design and functionality.

Identified User Requirements

1. Category Creation and Management

Requirement: Users need the ability to create and manage categories to organize their tasks effectively.

Influence on Design: Implemented a straightforward category creation process with options for editing and deletion. Categories are prominently displayed on the home screen for easy access.

2. Task Creation and Editing

Requirement: Users should be able to create new tasks and edit existing ones with detailed information such as titles, descriptions, due dates, and priorities.

Influence on Design: Developed a user-friendly task creation form with validation to ensure accurate data entry. Tasks can be edited seamlessly to accommodate changes in priorities or details.

3. Task Listing and Organization

Requirement: Users want a clear and organized view of all tasks, both categorized and uncategorized, with essential details readily visible.

Influence on Design: Designed a task list interface that displays tasks grouped by categories for easy navigation and prioritization. Each task entry shows relevant details like title, description, and due date.

4. Notifications and Reminders

Requirement: Users require timely notifications whenever a new task or category is added

Influence on Design: Implemented local notifications to alert users about the tasks and category creation.

5. Task Completion Tracking

Requirement: Users need a way to track and mark tasks as completed to monitor progress and manage workload effectively.

Influence on Design: Incorporated a task completion checkbox feature where users can mark tasks as done. Completed tasks are visually distinguished, providing a sense of accomplishment and progress tracking using a pie chart.

6. Username Customization

Requirement: Users want the ability to personalize their experience by customizing their username within the app.

Influence on Design: Included a settings option for users to change their username easily. This feature enhances user engagement and personalization.

7. Data Persistence

Requirement: Users expect their tasks and settings to persist across app sessions for continuity and convenience.

Influence on Design: Utilized local data storage with Hive to ensure that user data, including tasks, categories, and settings, is saved securely on the device. This approach provides seamless access to data without requiring constant internet connectivity.

Justification of User Requirements

- **Enhanced Organization:** Category and task management features cater to users' needs for organizing their workload efficiently, improving productivity.
- **Improved User Engagement:** Notifications and reminders keep users informed and on track with their tasks, enhancing engagement and timely task completion.
- **Personalization:** Username customization and persistent settings contribute to a personalized user experience, increasing satisfaction and usability.
- **Data Security and Privacy:** Local data storage ensures that user data is secure and private, aligning with user expectations for confidentiality and trustworthiness.

Continuous User Feedback

Throughout the development process, user feedback was solicited and incorporated into iterative updates of the application. This agile approach ensured that the app aligned closely with user expectations and addressed any usability issues or feature requests promptly.

3. Design concepts

User Interface (UI) Design

Consistency:

- **Color Scheme:** A consistent color palette is applied throughout the app to ensure a cohesive visual experience. Categories and task priorities are color-coded to help users quickly distinguish between different types of tasks.
- **Typography:** Uniform fonts and text sizes are used to maintain readability and create a professional appearance. Different styles are applied to headings, task titles, and descriptions to establish a clear visual hierarchy.

Simplicity:

- **Minimalist Design:** The interface is designed to be clean and uncluttered, focusing on essential elements only. This reduces distractions and helps users concentrate on their tasks.
- **Intuitive Navigation:** The app features simple and intuitive navigation, including a bottom navigation bar for quick access to main sections such as Home, Categories, and Settings. Clear icons and labels make it easy for users to find what they need.

User Experience (UX) Considerations

Ease of Use:

- **User-Friendly Forms:** Task and category creation forms are designed to be straightforward and easy to use, with clear labels and input fields.

Accessibility:

- **Responsive Design:** The app is designed to work well on various screen sizes and orientations, providing a consistent experience across different devices.
- **Accessible Navigation:** The app includes features like high-contrast mode and adjustable text sizes to ensure it is accessible to users with varying abilities.

Navigation Flow

Logical Structure:

- **Home Screen:** Acts as the central hub, displaying an overview of all categories and the number of tasks in each. Users can quickly add new categories or navigate to existing ones.
- **Category Screen:** Shows tasks within a selected category, allowing users to add, edit, and delete tasks directly from this screen.
- **Task Details Screen:** Provides a detailed view of a task, including its title, description, due date, and priority. Users can edit or delete tasks and mark them as completed.
- **Settings Screen:** A dedicated area where users can change their username and access other app settings.

Seamless Transitions:

- **Smooth Animations:** Smooth transitions between screens enhance the user experience and provide visual feedback on user actions.
- **Back Navigation:** Consistent back navigation through the app's hierarchy ensures that users can easily return to previous screens without losing context.

Visual Elements

Icons and Graphics:

- **Custom Icons:** Icons are used to represent different actions
- **Task Indicators:** Visual indicators such as pie chart or icons are used to show task status.

Feedback Mechanisms:

- **Visual Feedback:** Users receive immediate visual feedback on their actions, such as button presses and form submissions, to confirm that their input has been recognized and processed.

4. Development Approach

The methodology we used was Agile Methodology: Iterative development with sprints for flexibility and continuous improvement.

Justification: Allows for regular feedback incorporation and adaptation to evolving requirements.

Challenges: The main 3 challenges we encountered while developing the app were

1. Integration of Notifications: Ensuring timely notifications without impacting app performance.
2. Data Persistence: Efficient local storage management for task data.
3. Integration of different packages: Adding and incorporating multiple packages without impacting the other elements of the app.

5. Technology stack

Overview

The development of the Task Manager app leverages a modern and efficient technological stack, ensuring a seamless user experience and robust performance. The chosen technologies facilitate cross-platform compatibility, local data storage, and real-time notifications.

Technologies

Flutter

Description: Flutter is an open-source UI software development kit created by Google. **Rationale:**

- **Cross-Platform Development:** Write once, run anywhere—single codebase for both iOS and Android.
- **High Performance:** Uses the Dart language and compiles to native ARM code for fast performance.
- **Rich UI:** Provides a wide range of customizable widgets to create a visually appealing user interface.
- **Community and Support:** Large and active community with extensive documentation and support.

Dart

Description: Dart is a client-optimized programming language for fast apps on multiple platforms. **Rationale:**

- **Performance:** Ahead-of-time (AOT) compilation for high performance on mobile.

- **Concurrency:** Supports asynchronous programming with the use of async/await, making it ideal for tasks like data fetching and I/O operations.
- **Productivity:** Easy to learn with a syntax similar to other C-style languages, boosting developer productivity.

Hive

Description: Hive is a lightweight and blazing fast key-value database written in Dart for Flutter applications. **Rationale:**

- **No Native Dependencies:** Pure Dart implementation, making it highly portable and easy to integrate.
- **High Performance:** Fast read and write operations, suitable for mobile applications.
- **Lightweight:** Minimal memory footprint, ideal for mobile devices.
- **Type Safety:** Provides strong typing with Dart's type system, reducing runtime errors.

Local Notifications

Description: Local notifications provide a way to alert users about events and updates without the need for an internet connection. **Rationale:**

- **Real-Time Alerts:** Ensures users receive timely reminders for their tasks.
- **Customizable:** Allows for flexible scheduling and customization of notifications.
- **User Engagement:** Helps in keeping users engaged and on top of their tasks.

Tools and Frameworks

IDE: Visual Studio Code

Description: A lightweight but powerful source code editor developed by Microsoft. **Rationale:**

- **Extensions:** Rich ecosystem of extensions, including Flutter and Dart plugins.
- **Debugging:** Integrated debugging capabilities for a seamless development experience.
- **Customization:** Highly customizable with themes and settings to suit individual preferences.

GitHub

Description: A web-based platform used for version control and collaboration. **Rationale:**

- **Version Control:** Git-based repository for managing code versions and collaboration.
- **Issue Tracking:** Tools for tracking bugs, feature requests, and project tasks.
- **CI/CD Integration:** Supports continuous integration and deployment pipelines.

Testing Framework: Flutter Test

Description: The built-in testing framework for Flutter applications. **Rationale:**

- **Comprehensive Testing:** Supports unit, widget, and integration testing.
- **Efficiency:** Provides a fast and efficient way to write and run tests, ensuring code quality and reliability.

Rationale Behind Technological Choices

- **Flutter and Dart:** The combination of Flutter and Dart provides a powerful toolkit for developing high-performance, cross-platform mobile applications. The extensive widget library and hot-reload feature significantly speed up the development process.
- **Hive:** Chosen for its simplicity, speed, and minimalistic nature, Hive is perfect for managing the local storage requirements of the Task Manager app without the overhead of more complex databases.
- **Local Notifications:** Essential for maintaining user engagement and ensuring that important task reminders are delivered promptly, even without an internet connection.
- **Visual Studio Code and GitHub:** Together, they provide a robust development environment and a seamless workflow for version control, collaboration, and continuous integration.

Integration and Workflow

- **Development Environment:** Set up in Visual Studio Code with necessary Flutter and Dart extensions.
- **Version Control:** Managed using Git and GitHub for collaborative development and version tracking.

- **Continuous Integration:** Automated builds and tests run through GitHub Actions to ensure code quality and streamline the deployment process.
- **Local Data Storage:** Implemented using Hive, with careful structuring of data models to optimize performance and maintain data integrity.
- **Notification Management:** Local notifications handled through Flutter's notification libraries to ensure timely alerts.

Potential Challenges and Considerations

- **Performance Optimization:** Ensuring smooth performance across various devices, particularly older models with limited resources.
- **Data Security:** Implementing secure data storage practices to protect user data, especially when introducing features like cloud sync.
- **Scalability:** Designing the app architecture to easily accommodate future enhancements and additional features.

Conclusion

The chosen technological stack and tools provide a solid foundation for developing a feature-rich, high-performance task management application. By leveraging the strengths of Flutter, Dart, Hive, and other tools, the development team can efficiently create and maintain an app that meets user needs and remains adaptable for future enhancements.

Technologies:

- **Flutter:** Cross-platform framework for mobile app development.
- **Dart:** Programming language for Flutter development.
- **Hive:** Local database for storing task and category data.
- **Local Notifications:** For push notifications.

Rationale:

- **Flutter:** Single codebase for Android and iOS platforms, reducing development time and maintenance.
- **Hive:** Lightweight and suitable for local data storage needs, with high performance and minimal latency.
- **Local Notifications:** Reliable solution for real-time notifications.

6. Implementation detail

Key Features and Functionalities

1. Category Management

Description: Allows users to create, edit, and delete categories to organize their tasks. **Implementation:**

- **Create Category:** Users can add a new category by entering a name. This is implemented using a form with validation to ensure the category name is not empty.
- **Edit Category:** Users can update the name of an existing category. This feature uses the same form as category creation with the current category name pre-filled.

2. Task Management

Description: Enables users to create, edit, and delete tasks, and to assign them to categories. **Implementation:**

- **Create Task:** Users can add tasks with a title, description, due date, priority level, and category. This is implemented using a form with fields for each attribute.
- **Edit Task:** Users can update any task attribute. The task edit form is similar to the task creation form but pre-filled with the current task details.
- **Delete Task:** Users can delete a task with a confirmation prompt to avoid accidental deletions.
- **Task List Display:** Tasks are displayed in a list format, grouped by category. Each task entry shows the title, description, and due date.

3. Notifications

Description: Sends reminders to users for tasks and categories to help them stay organized. **Implementation:**

- **Local Notifications:** Implemented using the Flutter local notifications package. Users receive reminders based on task deadlines.
- **Custom Scheduling:** Users can set custom notification times when creating or editing a task.

4. Task Completion

Description: Allows users to mark tasks as completed to track their progress. **Implementation:**

- **Completion Toggle:** Each task in the task list has a checkbox. When checked, the task is marked as completed and visually distinguished (e.g., struck-through text or different color).

5. Change Username

Description: Users can personalize their experience by changing their username. **Implementation:**

- **Username Form:** A form in the settings screen allows users to update their username. The new username is saved locally using Hive.

6. Data Persistence

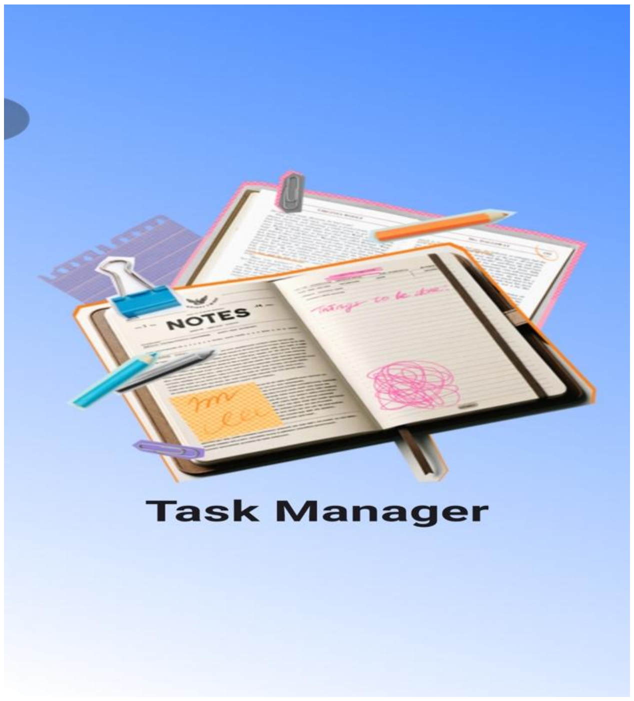
Description: Ensures that all user data (tasks, categories, username) is saved locally on the device. **Implementation:**

- **Local Storage:** Hive is used for data persistence. Data models are created for tasks, categories, and user settings.
- **Data Schema:** Structured to optimize read and write operations. Tasks and categories are stored in separate boxes in Hive

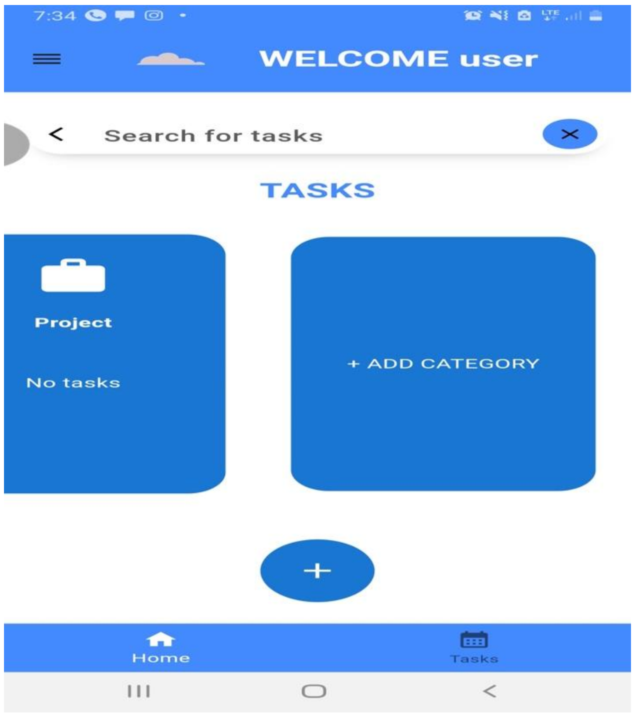
User Interface

Mockups

1. **Splash Screen**: Shows a splash screen when transitioning to the home screen

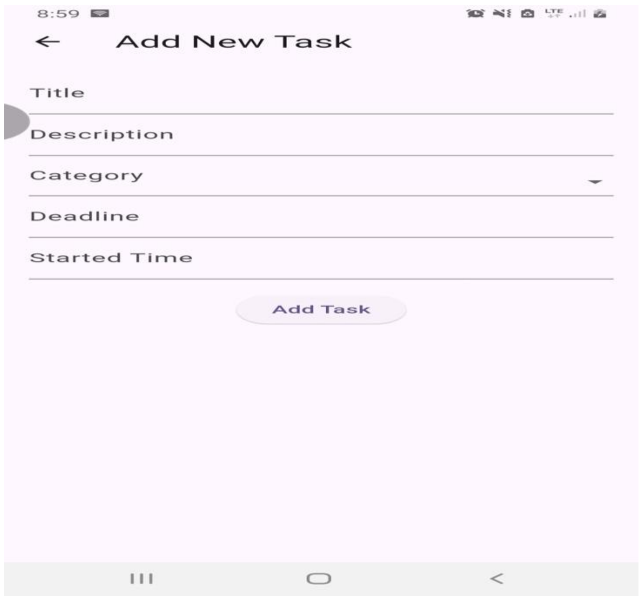


2. **Home Screen:** Shows the list of categories. Each category displays the number of tasks and a quick add button.



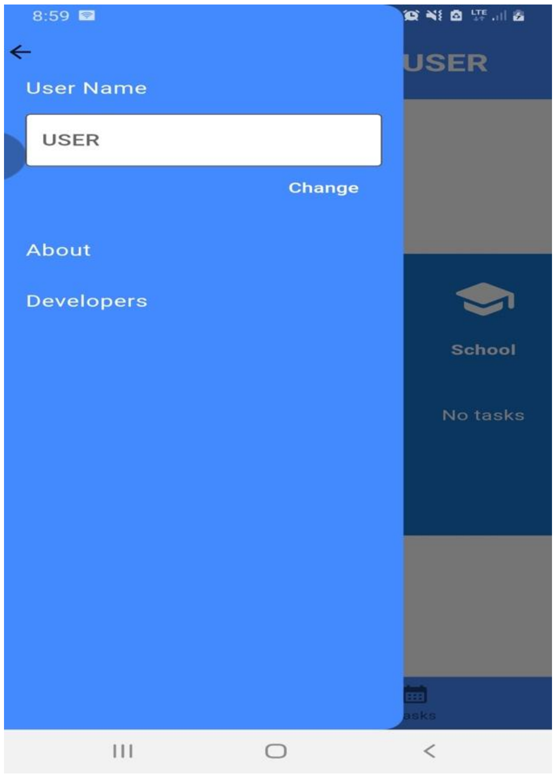
3.Category Screen: Displays tasks within a selected category. Users can add new tasks from this screen.

4.Task Form: Form for creating and editing tasks. Includes fields for title, description, due date,



category and a Calander for setting up deadline

5.Settings Screen: Allows users to change their username and access app settings.



Workflow

Task Creation: Users navigate to the category screen and tap the add button to open the task form. After entering task details, they save the task, which is then displayed in the task list.

Task Completion: Users mark a task as completed by checking the checkbox next to the task. The task's appearance changes to indicate its completion.

Notification Management: When creating or editing a task, users set a due date and notification time. The app schedules a local notification for the specified time.

Data Persistence: Upon any data change (task creation, update, deletion), the app saves the updated data to Hive. Data is automatically loaded from Hive when the app is launched.

7. Testing and quality assurance

Testing Strategies

To ensure the Task Manager app functions correctly, performs efficiently, and provides a positive user experience, we employed a comprehensive testing and quality assurance strategy. This includes the following approaches:

Unit Testing

- **Objective:** Verify the functionality of individual components and methods.
- **Approach:** Each function and method within the app is tested in isolation to ensure they work as expected.
- **Tools:** We used Flutter's built-in testing framework for unit tests.
- **Coverage:** Focused on key features such as task creation, editing, deletion, and notification handling.

Integration Testing

- **Objective:** Ensure that different parts of the application work together seamlessly.
- **Approach:** Test the interaction between modules and how they collaborate to fulfill user requirements.
- **Tools:** Flutter's integration testing capabilities were utilized to simulate real-world usage.
- **Scenarios:** Includes navigation between screens, data persistence, and real-time updates.

User Interface (UI) Testing

- **Objective:** Validate the user interface components for correctness and usability.
- **Approach:** Automated UI tests simulate user interactions to ensure the app responds correctly.
- **Tools:** Flutter Driver and widget testing for automated UI testing.
- **Focus:** Ensure that the layout is consistent across devices, user inputs are correctly handled, and visual feedback is appropriately given.

User Acceptance Testing (UAT)

- **Objective:** Confirm that the application meets user needs and requirements.
- **Approach:** Gather feedback from a selected group of target users through hands-on testing.
- **Process:** Users were asked to perform a set of tasks to evaluate the app's functionality and usability.
- **Feedback Loop:** Collected feedback was analyzed and used to make necessary adjustments and improvements.

Quality Assurance Measures

- **Performance Testing:** Evaluate the app's performance under various conditions to ensure it remains responsive and efficient.
  - **Tools:** Dart Observatory for performance profiling.
  - **Metrics:** Load times, memory usage, and CPU utilization.
- **Bug Tracking and Resolution:** Use a systematic approach to identify, track, and resolve bugs.
  - **Tools:** GitHub Issues and project boards for tracking bugs and managing the development process.
  - **Workflow:** Bugs were categorized by severity and prioritized for resolution.
- **Security Testing:** Ensure that the app securely handles user data.
  - **Focus:** Data encryption, secure storage, and protection against common vulnerabilities.
  - **Tools:** Code reviews and automated security scanners.

Effectiveness of Testing Approach

- **Comprehensive Coverage:** The multi-faceted approach ensured thorough testing of all aspects of the application.
- **Real-World Simulation:** User Acceptance Testing provided valuable insights into real-world usage and helped identify any remaining issues.
- **Continuous Improvement:** Agile methodology allowed for continuous feedback and iterative improvement, enhancing the app's quality over time.

Testing Artifacts

- **Test Cases:** Detailed documentation of test cases for unit, integration, and UI testing.
- **Test Reports:** Summary reports highlighting the results of each testing phase, including identified issues and their resolution status.
- **User Feedback:** Compiled feedback from UAT sessions, informing further refinements.

8. Future Enhancements

Proposed Features

1. Task Collaboration

Description: Enable users to share tasks with others, allowing for collaborative task management. **Benefits:** Facilitates teamwork and collective task management, especially useful for group projects and shared responsibilities. **Implementation:**

- **User Invitations:** Allow users to invite others to collaborate on tasks via email or in-app notifications.
- **Permissions:** Implement different levels of permissions (e.g., view-only, edit) for collaborators.
- **Real-Time Updates:** Use a real-time database to ensure all collaborators see the latest changes.

2. Cloud Sync

Description: Synchronize tasks across multiple devices using cloud storage. **Benefits:** Provides continuity and accessibility, ensuring users can manage their tasks from any device. **Implementation:**

- **User Authentication:** Integrate authentication mechanisms to securely identify users.
- **Data Storage:** Use cloud storage services such as Firebase Firestore or AWS Amplify to store and sync data.
- **Offline Access:** Implement offline capabilities, with automatic sync when the device is back online.

3. Advanced Notifications

Description: Provide customizable notification settings for different tasks and categories.

Benefits: Allows users to set personalized reminders, improving task management and time management. **Implementation:**

- **Custom Schedules:** Enable users to set specific times and frequencies for reminders.
- **Notification Types:** Provide options for different notification types, such as push notifications, emails, or SMS.
- **Snooze Functionality:** Allow users to snooze notifications and set follow-up reminders.

4. Data Analytics

Description: Offer insights into task completion rates and productivity metrics.

Benefits: Helps users understand their productivity patterns and improve time management. **Implementation:**

- **Dashboards:** Create interactive dashboards displaying key metrics such as tasks completed, average completion time, and overdue tasks.
- **Trend Analysis:** Provide visualizations for trends over time, helping users identify productivity peaks and troughs.
- **Export Options:** Allow users to export their data in various formats (e.g., CSV, PDF) for further analysis.

5. Enhanced User Interface and Experience

Description: Continuously improve the UI/UX based on user feedback and emerging design trends.

Benefits: Keeps the app engaging and user-friendly, enhancing overall satisfaction. **Implementation:**

- **User Feedback Loop:** Regularly collect and analyze user feedback to identify areas for improvement.

- **Design Updates:** Incorporate modern design elements and animations to keep the app visually appealing.
- **Accessibility Improvements:** Ensure the app is accessible to users with disabilities by adhering to accessibility guidelines.

Alignment with Project Objectives and User Needs

User Needs:

- **Collaboration:** Address the need for shared task management among teams and families.
- **Continuity:** Ensure users can access their tasks across multiple devices seamlessly.
- **Customization:** Allow users to personalize their notification settings to suit their schedules.
- **Productivity Insights:** Provide tools for users to track and improve their productivity.

Project Objectives:

- **Feature Expansion:** Enhance the app's functionality while maintaining simplicity and usability.
- **User Engagement:** Increase user engagement by adding features that address real user needs and preferences.
- **Market Competitiveness:** Stay competitive in the task management app market by continuously improving and adding new features.

Potential Challenges and Considerations

- **Data Security:** Ensuring that collaborative features and cloud sync are implemented securely to protect user data.
- **Performance Optimization:** Maintaining app performance and responsiveness, especially with real-time updates and advanced notifications.
- **User Adoption:** Encouraging users to adopt new features through intuitive design and clear communication of benefits.

Future Roadmap

1. **Short-Term (0-3 months):**
    - Implement basic task collaboration features.
    - Introduce advanced notification settings.

2. **Mid-Term (3-6 months):**
    - Roll out cloud sync capabilities.
    - Develop and integrate data analytics dashboards.
3. **Long-Term (6-12 months):**
    - Continuously refine the user interface based on feedback.
    - Expand collaboration features with real-time updates and advanced permissions.

Conclusion

Thank you for choosing To do as your task management solution. Let's get started on your journey to increased productivity and organization!

We are excited to have you on board with Todo and are committed to providing you with the best possible experience. This documentation is designed to be a valuable resource as you explore and utilize Task manager features. Should you have any questions or need further assistance,

please do not hesitate to reach out to our support team..

The technological stack for Tasks combines the simplicity and performance Flutter

and Async-Storage for local data persistence, provides a robust foundation for a reliable and user-friendly task management application .