**Scenario: Design and Planning of a Simple Personal To-Do List Application**

**Scenario Description:**

Suppose you want to design and plan the development of a simple mobile application to help users manage their daily to-do lists. This application should have the following features:

1. **Create New Task:** The user should be able to add new tasks to the list with a title and an optional description.
2. **View Task List:** The user should be able to view all their tasks in a sorted list (e.g., based on creation time).
3. **Mark Task as Done:** The user should be able to mark a task as completed.
4. **View Completed Tasks:** The user should be able to view a separate list of completed tasks.
5. **Delete Task:** The user should be able to delete a task (whether completed or not) from the list.
6. **Edit Task:** The user should be able to edit the title and description of a task.
7. **Prioritize Tasks:** The user should be able to set a priority for each task (e.g., low, medium, high) and sort the task list by priority.
8. **Reminders for Tasks:** The user should be able to set a reminder time for each task and receive a notification at that time.

**Steps to Perform the Exercise:**

1. **Identify Requirements:** Based on the scenario, list all the requirements for the application. Try to include as many details as possible.
2. **Document as User Stories:** For each requirement, write one or more user stories. User stories should follow the structure:

As a <type of user>, I want <to do something> so that <reason/benefit>.

For example: "As a user, I want to add a new task with a title and description to the list so that I can keep track of my daily activities."

1. **Determine Epics:** Group related user stories into epics. Epics are larger collections of features. For example, the epic "Task Management" could include user stories related to creating, viewing, editing, deleting, and marking tasks as done. The epic "Prioritization and Reminders" could include user stories related to setting priority and reminders.
2. **Break Down User Stories into Tasks:** Break down each user story into smaller, actionable tasks that can be completed within a short period. Tasks are the technical activities needed to implement a user story.
3. **Assign Story Points to User Stories:** Estimate a story point value for each user story. Story points represent the complexity, effort, and risk involved in implementing that user story. You can use different scales like the Fibonacci sequence (1, 2, 3, 5, 8, ...).
4. **Plan Sprints:** Assume you will develop the project in two-week sprints. Define a number of sprints (e.g., 3 or 4) and assign user stories and their associated tasks to each sprint. In each sprint, try to ensure that the total story points assigned align with the capacity of your hypothetical team (e.g., based on the number of people and their available time).
5. **Present in Scrum Format:** Describe how these elements would be presented within a Scrum framework. For example, during the Sprint Planning meeting, the team decides which user stories will be implemented in the current sprint. Throughout the sprint, Daily Scrum meetings are held for coordination and progress tracking. At the end of the sprint, a Sprint Review meeting is held to showcase the deliverables, and a Sprint Retrospective meeting is held to improve the process.

By following these steps for the "To-Do List" application scenario, you can practice all the concepts related to requirement identification, documenting in the form of user stories, defining epics and tasks, assigning story points, and planning Scrum sprints.

Let's identify relevant business, functional, and non-functional requirements, determine the responsible or impacted stakeholders, and use the MoSCoW method to prioritize the required features for the "To-Do List" application scenario.

**1. Requirement Identification:**

* **Business Requirements:**
  + Provide a simple and user-friendly mobile application for managing personal tasks.
  + Increase user productivity by facilitating the organization and tracking of daily tasks.
  + Attract users due to the application's simplicity and efficiency.
  + Enable the expansion of application features in subsequent versions based on user feedback.
* **Functional Requirements:** (These were previously mentioned in the scenario)
  + Create a new task with a title and description.
  + View a list of all tasks.
  + Mark a task as completed.
  + View a list of completed tasks.
  + Delete a task.
  + Edit the title and description of a task.
  + Set a priority for tasks (low, medium, high).
  + Sort the task list by priority.
  + Set a reminder time for tasks.
  + Receive a reminder notification at the specified time.
* **Non-Functional Requirements:**
  + **Performance:** The application should respond quickly and without significant delays. The task list should load rapidly.
  + **Reliability:** The application should be stable and work without crashes or data loss.
  + **Usability:** The application's user interface should be simple, intuitive, and user-friendly. Users should be able to easily navigate and use all features.
  + **Security:** In this simple scenario, data security (even if stored locally) is important. (This becomes more critical for online versions.)
  + **Scalability:** Given the nature of this scenario, scalability is not a primary concern, but considering design principles for future scalability could be beneficial.
  + **Accessibility:** The application should be usable by individuals with disabilities (e.g., using larger fonts).

**2. Responsible or Impacted Stakeholders:**

* **End User:** The individual who uses the application to manage their personal tasks. This is the primary stakeholder, and all requirements are ultimately aimed at their satisfaction.
* **Development Team:** Includes software developers, UI/UX designers, and Quality Assurance (QA) testers. This team is responsible for implementing and delivering the application.
* **Product Manager:** The individual responsible for defining the product strategy, prioritizing requirements, and guiding the development team.
* **(Optional)** **Project Sponsor:** The individual or organization providing the project funding and defining the overall business goals. (In this educational scenario, this role might be hypothetical.)

**3. Prioritization using the MoSCoW Method:**

The MoSCoW method defines four levels of priority:

* **Must have:** Features that are critical for the application's basic functionality and without which the application would be unusable.
* **Should have:** Important features that add significant value to the application, but the initial version can be released without them.
* **Could have:** Useful features that enhance the user experience, but can be postponed to later versions if time or resources are limited.
* **Won't have:** Features that are not considered for this stage and might be added to the application in the future.

Using this method, we can prioritize the functional requirements of the "To-Do List" application:

|  |  |  |
| --- | --- | --- |
| **Functional Requirement** | **Priority (MoSCoW)** | **Explanation** |
| Create a new task with a title | **Must have** | Without this capability, the user cannot record any tasks, rendering the application unusable. |
| View a list of all tasks | **Must have** | The user needs to be able to see the tasks they have recorded. |
| Mark a task as completed | **Must have** | Tracking the status of tasks is a core objective of the application. |
| View a list of completed tasks | **Should have** | This feature helps the user see their progress and stay better organized. |
| Delete a task | **Must have** | The user needs to be able to remove tasks they no longer need. |
| Edit the title of a task | **Should have** | Allows the user to correct mistakes or change the task title, which is beneficial. |
| Set a priority for tasks (low, medium, high) | **Could have** | This feature helps users focus on more important tasks. |
| Sort the task list by priority | **Could have** | Displaying tasks based on priority makes managing them easier. |
| Set a reminder time for tasks | **Could have** | This feature helps users remember to perform tasks at the scheduled time. |
| Receive a reminder notification at the specified time | **Could have** | To make reminders effective, the application needs to display a notification at the set time. |
| Add a description for a task | **Should have** | Descriptions can provide more details about each task. |
| Edit the description of a task | **Could have** | Allows the user to modify the description if needed. |

Export to Sheets

**Further Explanations:**

* **Must have:** Core features without which the application would not achieve its primary purpose.
* **Should have:** Very important features that significantly improve the user experience, but their absence in the first version wouldn't make the application unusable.
* **Could have:** Nice-to-have features that can add value to the application, but can be considered for later versions if there are time or resource constraints.
* **Won't have:** No features are definitively categorized as "Won't have" at this stage, but some "Could have" features might be moved to this category after further discussions with stakeholders, especially if there are significant time constraints. For instance, the reminder functionality might be omitted in the initial release if time is very limited.

This analysis helps in focusing development efforts on the most crucial features first, allowing for the delivery of a usable initial version of the application to users. The initial sprints will likely focus on the "Must have" features, followed by "Should have" features in subsequent sprints, and then "Could have" features if time and resources permit.

here are the complete user stories for all the functional requirements of the To-Do List application, following the format you provided:

**Epic: Task Management**

**User Story 1:**

* **Title:** Create a new task with a title
* **Components:** As a user, I want to be able to quickly add new tasks to my list so I can remember what I need to do.
* **Acceptance Criteria:**
  + Given I am on the main task list screen, when I tap the "Add" button, then I should see a screen to enter a new task.
  + Given I am on the new task screen, when I enter a title for the task and tap "Save", then the new task should appear in my main task list.
  + Given I am on the new task screen, when I enter a title and no further details and tap "Save", then the task should be saved with only the title.
  + Given I am on the new task screen, when I do not enter a title and tap "Save", then I should see an error message prompting me to enter a title.
* **Priority:** Must have
* **Definition of Done:**
  + New task creation screen is implemented.
  + User can enter a task title.
  + Task title is saved and displayed in the main list.
  + Validation is implemented for mandatory title field.

**User Story 2:**

* **Title:** View a list of all tasks
* **Components:** As a user, I want to see all the tasks I have added so I can get an overview of my to-do list.
* **Acceptance Criteria:**
  + Given I have added several tasks, when I open the application or navigate to the main screen, then I should see all my active tasks listed.
  + Given I have no tasks added, when I open the application or navigate to the main screen, then I should see a message indicating that my list is empty.
  + The tasks in the list should be displayed in a clear and readable format.
* **Priority:** Must have
* **Definition of Done:**
  + Main task list screen is implemented.
  + All active tasks are retrieved and displayed.
  + Empty state is handled gracefully.
  + Task list is easy to read.

**User Story 3:**

* **Title:** Mark a task as completed
* **Components:** As a user, I want to be able to mark tasks as completed so I can track my progress and see what I have finished.
* **Acceptance Criteria:**
  + Given I am viewing my list of active tasks, when I tap on a task, then I should see an option to mark it as "Done".
  + Given I tap the "Mark as Done" option for a task, then the task should disappear from the active task list.
  + Given a task is marked as "Done", then it should appear in the "Completed Tasks" list.
* **Priority:** Must have
* **Definition of Done:**
  + Functionality to mark a task as done is implemented.
  + Marked tasks are removed from the active list.
  + Marked tasks are moved to the completed list.

**User Story 4:**

* **Title:** View a list of completed tasks
* **Components:** As a user, I want to see a separate list of all the tasks I have completed so I can review my accomplishments.
* **Acceptance Criteria:**
  + Given I have completed several tasks, when I navigate to the "Completed Tasks" section, then I should see all my completed tasks listed.
  + The completed tasks should be displayed clearly, possibly with a visual indication of their completed status (e.g., a strikethrough).
  + The "Completed Tasks" list should be accessible from the main task list screen.
* **Priority:** Should have
* **Definition of Done:**
  + "Completed Tasks" screen is implemented.
  + All completed tasks are retrieved and displayed.
  + Completed tasks have a clear visual indication.
  + Navigation to the completed tasks screen is implemented.

**User Story 5:**

* **Title:** Delete a task
* **Components:** As a user, I want to be able to remove tasks from my list that are no longer relevant or necessary.
* **Acceptance Criteria:**
  + Given I am viewing my active or completed task list, when I tap and hold (or swipe) on a task, then I should see an option to "Delete".
  + Given I confirm the deletion of a task, then the task should be permanently removed from both the active and completed task lists.
  + There should be a confirmation step to prevent accidental deletion.
* **Priority:** Must have
* **Definition of Done:**
  + Functionality to delete a task is implemented.
  + Task is removed from both active and completed lists.
  + Confirmation dialog is displayed before deletion.

**User Story 6:**

* **Title:** Edit the title of a task
* **Components:** As a user, I want to be able to edit the title of an existing task in case I made a mistake or the task description changes.
* **Acceptance Criteria:**
  + Given I am viewing my active task list, when I tap on a task, then I should see an option to "Edit".
  + Given I select "Edit", then I should be taken to a screen where I can modify the task title.
  + Given I update the title and tap "Save", then the task in my list should be updated with the new title.
* **Priority:** Should have
* **Definition of Done:**
  + Edit functionality for the task title is implemented.
  + User can modify the task title.
  + Updated title is saved and displayed in the task list.

**Epic: Prioritization and Reminders**

**User Story 7:**

* **Title:** Set a priority for tasks
* **Components:** As a user, I want to be able to assign a priority level to my tasks so I can easily identify and focus on the most important ones.
* **Acceptance Criteria:**
  + Given I am viewing or editing a task, then I should see an option to set a priority level (e.g., Low, Medium, High).
  + When I select a priority level and save the task, then the priority should be associated with the task.
  + The priority of a task should be visually indicated in the task list (e.g., with different colors or icons).
* **Priority:** Could have
* **Definition of Done:**
  + Priority setting option is available when creating or editing a task.
  + Priority levels (Low, Medium, High) are implemented.
  + Selected priority is saved with the task.
  + Visual indication of priority in the task list.

**User Story 8:**

* **Title:** Sort the task list by priority
* **Components:** As a user, I want to be able to sort my tasks by priority so I can easily see and work on the most important items first.
* **Acceptance Criteria:**
  + Given I have assigned priorities to some of my tasks, when I navigate to the task list, then I should see an option to "Sort by Priority".
  + When I select "Sort by Priority", then the tasks in the list should be reordered with higher priority tasks appearing at the top.
  + The sorting order should be High -> Medium -> Low.
* **Priority:** Could have
* **Definition of Done:**
  + "Sort by Priority" option is available in the task list.
  + Task list can be sorted based on the assigned priority.
  + Sorting order is correctly implemented.

**User Story 9:**

* **Title:** Set a reminder time for tasks
* **Components:** As a user, I want to be able to set a specific date and time as a reminder for a task so I don't forget important deadlines or activities.
* **Acceptance Criteria:**
  + Given I am viewing or editing a task, then I should see an option to set a "Reminder".
  + When I select "Reminder", I should be able to choose a date and time for the reminder.
  + The selected reminder time should be saved with the task.
  + There should be an option to disable or edit the reminder.
* **Priority:** Could have
* **Definition of Done:**
  + Reminder setting option is available when creating or editing a task.
  + User can select a date and time for the reminder.
  + Reminder time is saved with the task.
  + Option to disable or edit the reminder is available.

**User Story 10:**

* **Title:** Receive a reminder notification at the specified time
* **Components:** As a user, I want to receive a notification at the reminder time I set for a task so I am prompted to take action.
* **Acceptance Criteria:**
  + Given I have set a reminder for a task, when the current time reaches the reminder time, then I should receive a push notification on my device.
  + The notification should display the title of the task.
  + The notification should appear even if the application is not currently open.
* **Priority:** Could have
* **Definition of Done:**
  + Push notification functionality is implemented.
  + Notifications are triggered at the set reminder time.
  + Notification displays the task title.
  + Notifications work when the app is in the background.

**User Story 11:**

* **Title:** Add a description for a task
* **Components:** As a user, I want to be able to add a more detailed description to my tasks so I have all the necessary information readily available.
* **Acceptance Criteria:**
  + Given I am creating a new task or editing an existing one, then I should see an optional field to enter a description.
  + When I enter text in the description field and save the task, then the description should be saved with the task.
  + The description should be viewable when I view the details of a task.
* **Priority:** Should have
* **Definition of Done:**
  + Description field is available when creating or editing a task.
  + User can enter a task description.
  + Description is saved and displayed in the task details view.

**User Story 12:**

* **Title:** Edit the description of a task
* **Components:** As a user, I want to be able to modify the description of an existing task if I need to add more details or make changes.
* **Acceptance Criteria:**
  + Given I am viewing the details of a task, then I should see an option to "Edit" the task.
  + Given I select "Edit", then I should be able to modify the text in the description field.
  + When I update the description and tap "Save", then the updated description should be saved and displayed in the task details.
* **Priority:** Could have
* **Definition of Done:**
  + Edit functionality for the task description is implemented.
  + User can modify the task description.
  + Updated description is saved and displayed in the task details view.

These user stories provide a good starting point for planning your sprints. Remember that the "Definition of Done" can be further refined by the development team to include specific technical and quality standards.

let's break down the user stories into tasks, assign story points, plan sprints, and present this in a Scrum format.

**Assumptions:**

* We have a small development team with a capacity of roughly 8-10 story points per two-week sprint for the initial sprints. This is a rough estimate and can be adjusted based on the team's velocity.
* The Fibonacci sequence (1, 2, 3, 5, 8) will be used for assigning story points.

**Sprint 1 (Focus on Core Functionality - Must Haves)**

* **Total Estimated Story Points: 8**
  + **User Story 1: Create a new task with a title** (Story Points: 3)
    - Task 1.1: Design the "New Task" screen UI (0.5 days)
    - Task 1.2: Implement the input field for the task title (0.5 days)
    - Task 1.3: Implement the "Save" button and event handler (0.5 days)
    - Task 1.4: Write logic to save the task title to local storage (1 day)
    - Task 1.5: Display the newly created task in the main list (0.5 days)
    - Task 1.6: Implement validation for the mandatory title field (0.5 days)
    - Task 1.7: Write unit tests for task creation (0.5 days)
  + **User Story 2: View a list of all tasks** (Story Points: 2)
    - Task 2.1: Create the main task list UI (0.5 days)
    - Task 2.2: Write logic to retrieve all active tasks from local storage (0.5 days)
    - Task 2.3: Populate the task list with retrieved data (0.5 days)
    - Task 2.4: Implement handling for an empty task list (0.5 days)
    - Task 2.5: Write unit tests for task list retrieval (0.5 days)
  + **User Story 5: Delete a task** (Story Points: 3)
    - Task 5.1: Implement UI element for triggering the delete action (e.g., long press context menu) (0.5 days)
    - Task 5.2: Display a confirmation dialog before deletion (0.5 days)
    - Task 5.3: Write logic to remove the selected task from local storage (1 day)
    - Task 5.4: Update the task list UI after deletion (0.5 days)
    - Task 5.5: Write unit tests for task deletion (0.5 days)

**Sprint 2 (Adding Essential Features - Should Haves)**

* **Total Estimated Story Points: 10**
  + **User Story 3: Mark a task as completed** (Story Points: 3)
    - Task 3.1: Implement UI element to mark a task as done (e.g., a checkbox) (0.5 days)
    - Task 3.2: Write logic to update the task status to "completed" in local storage (1 day)
    - Task 3.3: Visually differentiate completed tasks in the active list (e.g., strikethrough) (0.5 days)
    - Task 3.4: Ensure marked tasks are removed from the active list (0.5 days)
    - Task 3.5: Update the UI to reflect the change in status (0.5 days)
    - Task 3.6: Write unit tests for marking tasks as done (0.5 days)
  + **User Story 4: View a list of completed tasks** (Story Points: 3)
    - Task 4.1: Create the "Completed Tasks" screen UI (0.5 days)
    - Task 4.2: Implement navigation from the main screen to the completed tasks screen (0.5 days)
    - Task 4.3: Write logic to retrieve all completed tasks from local storage (0.5 days)
    - Task 4.4: Populate the completed task list with retrieved data (0.5 days)
    - Task 4.5: Display completed tasks clearly (0.5 days)
    - Task 4.6: Write unit tests for completed task retrieval (0.5 days)
  + **User Story 6: Edit the title of a task** (Story Points: 4)
    - Task 6.1: Implement UI element to trigger the edit action (e.g., tap on a task in the active list) (0.5 days)
    - Task 6.2: Navigate to an "Edit Task" screen pre-filled with the task title (0.5 days)
    - Task 6.3: Implement input field for editing the task title (0.5 days)
    - Task 6.4: Implement "Save" button and event handler for updating the title (1 day)
    - Task 6.5: Write logic to update the task title in local storage (0.5 days)
    - Task 6.6: Update the task list UI with the new title (0.5 days)
    - Task 6.7: Write unit tests for editing task title (0.5 days)

**Sprint 3 (Enhancements - Could Haves & Remaining Should Have)**

* **Total Estimated Story Points: 10**
  + **User Story 11: Add a description for a task** (Story Points: 3)
    - Task 11.1: Add an optional input field for task description on the "New Task" screen (0.5 days)
    - Task 11.2: Update logic to save the task description to local storage (0.5 days)
    - Task 11.3: Display the task description when viewing task details (0.5 days)
    - Task 11.4: Design and implement a "Task Details" view (1 day)
    - Task 11.5: Implement navigation to the "Task Details" view (0.5 days)
    - Task 11.6: Write unit tests for adding task description (0.5 days)
  + **User Story 7: Set a priority for tasks** (Story Points: 3)
    - Task 7.1: Add a priority selection element (e.g., dropdown) on the "New Task" and "Edit Task" screens (0.5 days)
    - Task 7.2: Implement priority levels (Low, Medium, High) (0.5 days)
    - Task 7.3: Update logic to save the task priority to local storage (0.5 days)
    - Task 7.4: Visually indicate task priority in the main task list (e.g., using color-coding) (1 day)
    - Task 7.5: Write unit tests for setting task priority (0.5 days)
  + **User Story 8: Sort the task list by priority** (Story Points: 4)
    - Task 8.1: Add a "Sort by" option in the main task list (e.g., a dropdown or menu) (0.5 days)
    - Task 8.2: Implement the "Sort by Priority" option (1 day)
    - Task 8.3: Write logic to reorder the task list based on the selected priority (High -> Medium -> Low) (1.5 days)
    - Task 8.4: Ensure the sorting persists or can be easily changed (0.5 days)
    - Task 8.5: Write unit tests for sorting by priority (0.5 days)

**Sprint 4 (Optional Enhancements - Could Haves)**

* **Total Estimated Story Points: 7**
  + **User Story 9: Set a reminder time for tasks** (Story Points: 3)
    - Task 9.1: Add a reminder setting option (date and time picker) on the "New Task" and "Edit Task" screens (1 day)
    - Task 9.2: Implement logic to save the reminder time with the task in local storage (0.5 days)
    - Task 9.3: Allow disabling or editing the reminder (0.5 days)
    - Task 9.4: Write unit tests for setting reminders (1 day)
  + **User Story 10: Receive a reminder notification at the specified time** (Story Points: 4)
    - Task 10.1: Integrate a notification service for the mobile platform (1 day)
    - Task 10.2: Implement logic to trigger notifications at the saved reminder time (1.5 days)
    - Task 10.3: Ensure the notification displays the task title (0.5 days)
    - Task 10.4: Ensure notifications work when the app is in the background (1 day)
  + **User Story 12: Edit the description of a task** (Story Points: 2)
    - Task 12.1: Add the description field to the "Edit Task" screen (0.5 days)
    - Task 12.2: Implement logic to load the existing description for editing (0.5 days)
    - Task 12.3: Update logic to save the edited description to local storage (0.5 days)
    - Task 12.4: Write unit tests for editing task description (0.5 days)

**Presentation in Scrum Format:**

1. **Product Backlog:** All the identified user stories reside in the Product Backlog, prioritized based on the MoSCoW method (and potentially further refined by the Product Owner). Each user story has an estimated story point value.
2. **Sprint Planning Meeting:**
   * The Product Owner presents the prioritized user stories from the top of the Product Backlog to the Development Team.
   * The team discusses each user story to gain a shared understanding of its requirements and acceptance criteria.
   * The team breaks down the selected user stories into specific tasks and estimates the effort required for each task (often in hours or days).
   * Based on the team's velocity (the average number of story points completed in past sprints), the team collaboratively selects a set of user stories that they believe they can complete within the upcoming two-week sprint. This set of user stories forms the **Sprint Backlog**.
3. **Daily Scrum (Daily Stand-up):**
   * Every day during the sprint, the Development Team meets briefly (typically for 15 minutes).
   * Each team member answers three questions:
     + What did I do yesterday that helped the Sprint Goal?
     + What will I do today to help the Sprint Goal?
     + Are there any impediments that are blocking me or the team from achieving the Sprint Goal?
   * This meeting helps the team stay aligned, identify any roadblocks, and track progress towards the Sprint Goal (which is often implied by the set of user stories committed to the sprint).
4. **Sprint Review Meeting:**
   * At the end of the two-week sprint, the Development Team demonstrates the work they have completed to the Product Owner and other stakeholders.
   * The demonstration focuses on the completed user stories and whether they meet the agreed-upon acceptance criteria.
   * Stakeholders provide feedback, which can be used to refine the Product Backlog for future sprints.
5. **Sprint Retrospective Meeting:**
   * After the Sprint Review, the team (including the Product Owner and Scrum Master) holds a Sprint Retrospective.
   * The purpose of this meeting is to reflect on the past sprint and identify what went well, what could be improved, and what actions the team will take to make the next sprint more effective.

**In summary:**

* The **user stories** define what the application needs to do from the user's perspective.
* The **tasks** are the technical steps required to implement those user stories.
* The **story points** provide a relative measure of the effort and complexity involved in each user story, helping with sprint planning.
* The **sprint plan** organizes the work into manageable two-week iterations, allowing for regular delivery of value and adaptation based on feedback.
* The **Scrum ceremonies** provide a framework for the team to collaborate, track progress, and continuously improve their process.