Below is the **Product Backlog** that further elaborates on each **Epic** or **User Story**. The goal is to provide enough detail for any Scrum team member—Product Owner, Scrum Master, or Developers—to understand the scope, requirements, and priorities.

**1. Basic Task Management (Epic)**

* **Priority**: 1
* **Title**: Basic Task Management
* **Type**: Epic
* **Estimate**: 5 SP
* **Description**:  
  As a user, I want to add, mark complete, and delete tasks so that I can manage my daily to-do list effectively.
* **Acceptance Criteria**:
  1. The UI provides a text input and an "Add Task" button.
  2. When I add a task, it appears in a list with a default “incomplete” status.
  3. I can mark a task as complete/incomplete by clicking or toggling a checkbox-like element.
  4. I can delete tasks and have them removed from the current list.
* **Additional Details**:
  1. **User Role**: General user (anyone who wants to keep track of tasks).
  2. **Business Value**: Fundamental to-do functionality is the core value proposition of this product.
  3. **Possible Implementation Steps**:
     1. Create an input field and button in index.html.
     2. Implement addTask() in script.js to capture the text input, validate it, and store it in a taskList array.
     3. Implement a renderTasks() function that populates the UI from the taskList.
     4. Add a toggle mechanism in toggleTask(index) to mark tasks complete or incomplete.
     5. Add a delete mechanism in deleteTask(index) to remove tasks.
  4. **Dependencies**: None (this is the first building block).

**2. Data Persistence (Epic)**

* **Priority**: 2
* **Title**: Data Persistence
* **Type**: Epic
* **Estimate**: 3 SP
* **Description**:  
  As a user, I want my tasks to remain saved after I refresh or close the browser, so I never lose track of my tasks.
* **Acceptance Criteria**:
  1. Tasks are stored in localStorage (JSON-structured).
  2. On page load, tasks are loaded from localStorage.
* **Additional Details**:
  1. **User Role**: Anyone accessing the web application from a single device/browser.
  2. **Business Value**: Without persistence, users would lose tasks every time the page is refreshed, undermining the app’s usefulness.
  3. **Possible Implementation Steps**:
     1. Define a saveTasksToLocalStorage() function that serializes the current taskList (and possibly other data) into JSON, storing in localStorage.
     2. Define a retrieveTasksFromLocalStorage() function that checks localStorage on page load, parses the JSON, and populates taskList.
     3. Integrate these functions into existing UI methods (addTask, toggleTask, deleteTask, etc.).
  4. **Dependencies**:
     1. **Basic Task Management** must be in place so that tasks exist to be persisted.

**3. Important Tasks (Epic)**

* **Priority**: 3
* **Title**: Important Tasks
* **Type**: Epic
* **Estimate**: 3 SP
* **Description**:  
  As a user, I want to differentiate important tasks from regular tasks so that I can focus on high-priority items.
* **Acceptance Criteria**:
  1. A separate list or section for “Important Tasks.”
  2. Ability to add, delete, and toggle these tasks similarly to regular tasks.
* **Additional Details**:
  1. **User Role**: Users who need to distinguish critical tasks from everyday items.
  2. **Business Value**: Enhances usability by allowing prioritization of tasks.
  3. **Possible Implementation Steps**:
     1. Create a new input field and "Add Important Task" button in the UI.
     2. Use a new array (e.g., importantTaskList) to store these tasks separately from taskList.
     3. Implement parallel functions (addImportantTask(), renderImportantTasks(), toggleImportantTask(index), deleteImportantTask(index)) in script.js.
     4. Integrate with localStorage so that important tasks also persist across sessions.
  4. **Dependencies**:
     1. **Basic Task Management** for the general task logic.
     2. **Data Persistence** for storing important tasks as well.

**4. Archive / History (Epic)**

* **Priority**: 4
* **Title**: Archive / History
* **Type**: Epic
* **Estimate**: 5 SP
* **Description**:  
  As a user, I want completed tasks to be moved to an archive so that I can keep a record without cluttering my main to-do list.
* **Acceptance Criteria**:
  1. There is a “History” or “Archive” section in the UI.
  2. Clicking an “Archive Completed Tasks” button moves all completed tasks to the archive list.
  3. Archived tasks remain visible in a read-only format but do not appear in the main list anymore.
* **Additional Details**:
  1. **User Role**: Users who like to keep a record of their completed tasks for reference.
  2. **Business Value**: Minimizes clutter in the main list while preserving historical data.
  3. **Possible Implementation Steps**:
     1. Add a “History” or “Archive” section in index.html.
     2. Implement archiveCompleted() in script.js to filter completed tasks out of taskList and add them to a historyList.
     3. Create a function renderArchive() to display archived tasks.
     4. Update saveTasksToLocalStorage() and retrieveTasksFromLocalStorage() to handle historyList.
  4. **Dependencies**:
     1. **Basic Task Management** to handle completed status.
     2. **Data Persistence** to store archived tasks.

**5. UI/UX Improvements (Story)**

* **Priority**: 5
* **Title**: UI/UX Improvements
* **Type**: User Story
* **Estimate**: 2 SP
* **Description**:  
  As a user, I want a modern, responsive interface so that I can easily use the app on any device.
* **Acceptance Criteria**:
  1. Polished design using CSS or a simple framework (e.g., Bootstrap or a custom styling).
  2. Buttons and text fields are easy to click on any screen size.
  3. Layout is responsive, adapting to different screen widths without breaking.
* **Additional Details**:
  1. **User Role**: Any user who values a pleasant visual and interactive experience.
  2. **Business Value**: Improves user retention and satisfaction.
  3. **Possible Implementation Steps**:
     1. Enhance existing CSS to ensure responsiveness (media queries, fluid layouts).
     2. Optimize font sizes and spacing for both desktop and mobile.
     3. Possibly introduce minimal animations or hover effects for better feedback.
  4. **Dependencies**:
     1. None strictly—can be performed in parallel, but benefits from having the main UI structure set by earlier tasks.

**6. Filtering / Sorting Tasks (Story)**

* **Priority**: 6
* **Title**: Filtering / Sorting Tasks
* **Type**: User Story
* **Estimate**: 5 SP
* **Description**:  
  As a user, I want to filter or sort tasks (e.g., by name, completed status) so that I can quickly locate tasks of interest.
* **Acceptance Criteria**:
  1. An option to switch views: all tasks vs. completed tasks vs. incomplete tasks.
  2. A quick sort by alphabetical order or creation date.
* **Additional Details**:
  1. **User Role**: Users who need advanced organizational features.
  2. **Business Value**: Helps users manage a large number of tasks more effectively.
  3. **Possible Implementation Steps**:
     1. Create UI controls (buttons or dropdowns) to filter tasks by status (all, completed, incomplete).
     2. Implement a sorting logic to reorder tasks by date or name.
     3. Decide whether to store creation timestamps in the taskList to enable date-based sorting.
     4. Modify renderTasks() (and renderImportantTasks() if needed) to apply the selected filters/sorting.
  4. **Dependencies**:
     1. **Basic Task Management** and **Data Persistence**.

**7. Share / Sync Feature (Epic)**

* **Priority**: 7
* **Title**: Share / Sync Feature
* **Type**: Epic
* **Estimate**: 8 SP
* **Description**:  
  As a user, I want to share or sync tasks across multiple devices or with other users so that I can collaborate on tasks easily.
* **Acceptance Criteria**:
  1. (Future) Some form of real-time backend or cloud sync capability (e.g., using an API or database).
  2. Possibly a link-based share or user-based share mechanism.
* **Additional Details**:
  1. **User Role**: Teams, families, or anyone who collaborates on to-do lists.
  2. **Business Value**: Greatly expands the product’s utility for group projects and cross-device usage.
  3. **Possible Implementation Steps** (conceptual):
     1. Introduce a server-side component or third-party service for storing tasks.
     2. Implement user authentication if necessary (for personalized lists).
     3. Provide a sync function that regularly polls or triggers updates in the web app.
     4. Potentially add real-time websockets or ephemeral ID-based sharing links.
  4. **Dependencies**:
     1. All existing local functionality.
     2. Additional infrastructure (server, real-time service).

**8. Mobile App Integration (Epic)**

* **Priority**: 8
* **Title**: Mobile App Integration
* **Type**: Epic
* **Estimate**: 8 SP
* **Description**:  
  As a user, I want a corresponding mobile application so that I can manage tasks on the go.
* **Acceptance Criteria**:
  1. A web or hybrid mobile approach that seamlessly integrates with the existing data or sync method.
  2. Uses the same data source or sync method as the web version.
* **Additional Details**:
  1. **User Role**: Users who prefer a native or near-native mobile experience.
  2. **Business Value**: Extends usability to smartphones and tablets with a polished, possibly offline-capable solution.
  3. **Possible Implementation Steps**:
     1. Evaluate frameworks (Ionic, React Native, Flutter, or standard PWA approach).
     2. Integrate the existing data logic (sync or localStorage).
     3. Optimize UI for smaller screens and touch input.
     4. Implement offline capabilities if required.
  4. **Dependencies**:
     1. **Share / Sync Feature** if multi-device usage is a priority.

**9. Accessibility (Story)**

* **Priority**: 9
* **Title**: Accessibility
* **Type**: User Story
* **Estimate**: 3 SP
* **Description**:  
  As a user with accessibility needs, I want the web app to be navigable by screen readers and keyboard so that I can manage tasks without barriers.
* **Acceptance Criteria**:
  1. Proper ARIA tags and semantic HTML elements to ensure screen-reader compatibility.
  2. Full keyboard navigation (tab order, focus states, etc.).
* **Additional Details**:
  1. **User Role**: Users with visual or motor impairments, or those who prefer keyboard navigation.
  2. **Business Value**: Increases the product’s user base and compliance with accessibility standards (e.g., WCAG).
  3. **Possible Implementation Steps**:
     1. Review each UI component for semantic markup (e.g., <ul> for lists, <button> elements, etc.).
     2. Add ARIA labels or roles where appropriate.
     3. Ensure focus states are visible and clearly indicated.
     4. Test with a screen reader (e.g., NVDA, VoiceOver).
  4. **Dependencies**:
     1. Basic UI in place.
     2. Potentially iterative improvements on top of existing features.

**Additional Backlog Clarifications**

1. **Story Point (SP) Estimates**:
   * These are **relative** measures of complexity or effort. They do not directly translate to hours but reflect how much effort each item will likely require compared to other items.
2. **Priorities**:
   * The Product Owner assigns these priorities based on business value, risk, and other organizational factors. High-priority items must be addressed earlier.
3. **Refinement & Grooming**:
   * The backlog items can be further **refined** over time. Acceptance criteria, estimates, or descriptions may evolve as the team gains more insight into the product and user needs.
4. **Sprint Planning**:
   * During sprint planning, the team selects the highest-priority items that fit within their capacity. They then break those stories down into smaller development tasks (the **Sprint Backlog**).
5. **Definition of Ready**:
   * A team may use a “Definition of Ready” to ensure each backlog item is adequately described, estimated, and feasible before committing it to a sprint.
   * For instance, “All user stories must have clear acceptance criteria, estimated story points, and no unresolved external dependencies.”