LOCAL KANBAN DOCUMENTATION

This single HTML file implements an offline Kanban board application. It includes HTML for the layout, CSS for styling, and JavaScript for functionality. The application manages tasks using different columns and supports creating, editing, moving (via drag-and-drop), and exporting tasks. All data is stored in the browser's local storage.

---

1. GLOBAL VARIABLES

- **`tasks`**
  An array that stores all task objects. Each task is an object containing properties such as `id`, `name`, `description`, `deadline`, `creationDate`, `note`, `estimatedTime`, `workedHours`, `status`, and `labels`.

- **`labels`**
  An array used to store label objects. Each label object has an `id`, a `name`, and a `color`. Labels help to categorize tasks.

- **`currentStatus`**
  A string variable that holds the status (i.e., which column) the new task will belong to when using the Create Task modal.

- **`currentEditTaskId`**
  Stores the `id` of the task currently being edited. This helps the edit modal know which task to update.

---

2. LOCAL STORAGE FUNCTIONS

These functions handle saving to and retrieving data from the browser's local storage, ensuring that tasks and labels persist between sessions.

- **`saveToLocalStorage()`**
  Converts the `tasks` array into a JSON string and saves it in local storage under the key `"kanbanTasks"`.
  Usage: Called whenever a task is added, updated, or deleted.

- **`saveLabelsToLocalStorage()`**
  Converts the `labels` array into a JSON string and saves it under the key `"kanbanLabels"`.
  Usage: Called whenever labels are added or modified.

- **`loadFromLocalStorage()`**
  Checks if data exists in local storage for tasks. If found, it parses the JSON string back into the `tasks` array and then calls `renderAllTasks()` to display them on the board.

- **`loadLabelsFromLocalStorage()`**
  Similar to `loadFromLocalStorage()`, this function loads the saved labels (if any) from local storage.

---

3. Label Rendering & Management Functions

These functions manage the display and creation of labels within the task creation and editing modals.

- **`renderLabels(containerId, selectedLabels = [])`**
  Renders all available labels as checkboxes inside a specified container element.
  - **Parameters:**
    - `containerId`: The ID of the HTML element where labels will be shown.
    - `selectedLabels`: (Optional) An array of label IDs that should appear checked (e.g., for tasks that already have labels assigned).
      Usage: Called when opening the create or edit task modal to allow users to assign labels.

- **`toggleNewLabel(mode)`**
  Toggles the display of the section that allows users to create a new label.
  - **Parameter:**
    - `mode`: A string (`"create"` or `"edit"`) that determines which modal's label section to toggle.
      Usage: When the user clicks the "Create Label" button.

- **`addNewLabel(mode)`**
  Reads the new label name and color from the input fields, creates a label object, adds it to the `labels` array, and then updates local storage and the label displays.
  - **Parameter:**
    - `mode`: Determines if the label is being added in the create or edit context.
      Usage: Triggered when the user confirms adding a new label.

- **`getTaskLabels(taskId)`**
  Retrieves the list of label IDs assigned to a specific task.
  - **Parameter:**
    - `taskId`: The unique identifier of the task.
      Returns: An array of label IDs or an empty array if no labels exist.
      Usage: Used in the edit modal to pre-check the labels that the task already has.

---

4. Task Creation Functions

These functions manage the process of creating new tasks via a modal window.

- **`openCreateModal(status)`**
  Opens the "Create New Task" modal.

- **Parameter:**
  - `status`: The column status (e.g., `backlog`, `todo`, etc.) where the new task should be placed.
- **Actions:**
  - Sets `currentStatus` to the selected column.
  - Resets the form inputs.
  - Renders available labels for selection.
  - Displays the modal overlay.

- **`closeCreateModal()`**
  Hides the "Create New Task" modal by setting its overlay's display style to `none`.
- **Create Task Form Submission Listener**
  Attached to the form with ID `"createTaskForm"`, this listener handles the following:
  - Prevents the default form submission behavior.
  - Retrieves and trims user input for task name, description, deadline, note, estimated time, and worked hours.
  - Formats the deadline (replacing the "T" with a space if provided).
  - Collects selected label IDs from the rendered checkboxes.
  - Creates a new task object with a unique ID (using `Date.now()`), the current date/time for creation, and all provided details.
  - Pushes the new task into the `tasks` array.
  - Calls `renderAllTasks()` to update the board and `saveToLocalStorage()` to persist the changes.
  - Closes the modal afterward.

---

5. Task Editing Functions

These functions let users update the details of an existing task.

- **`openEditModal(taskId)`**
  Opens the "Edit Task" modal for a specific task.
  - **Parameter:**
    - `taskId`: The ID of the task to be edited.
  - **Actions:**
    - Sets `currentEditTaskId` with the given task ID.
    - Retrieves the task object from `tasks`.
    - Populates the modal input fields with the task's current data (name, description, note, deadline, estimated time, and worked hours).

        - Calls `renderLabels()` to display the labels, pre-checking those already assigned to the task.
        - Displays the modal overlay.

- **`closeEditModal()`**
  Hides the "Edit Task" modal.

- **Edit Task Form Submission Listener**
  Attached to the form with ID `"editTaskForm"`, this listener:
    - Prevents default form submission.
    - Retrieves the updated task details from the input fields.
    - Updates the corresponding task object's properties (name, description, note, deadline, estimated time, worked hours, and labels).
    - Calls `saveToLocalStorage()` and `renderAllTasks()` to save changes and update the board.
    - Closes the edit modal.

---

6. Export Work Report Functions

These functions allow the user to generate and download a report based on the tasks worked on between two dates.

- **`openExportModal()`**
  Opens the "Export Work Report" modal by resetting the export form and displaying the overlay.

- **`closeExportModal()`**
  Closes the export modal by hiding its overlay.

- **Export Form Submission Listener**
  Attached to the form with ID `"exportForm"`, this listener:
    - Prevents default form submission.
    - Retrieves the start and end dates input by the user.
    - Validates that both dates are selected.
    - Calls the `exportWorkReport(startDate, endDate)` function to generate the report.
    - Closes the modal afterward.

- **`exportWorkReport(startDate, endDate)`**
  Generates a work report based on tasks created within the specified date range.
    - **Parameters:**
        - `startDate`: Start date (inclusive).
        - `endDate`: End date (inclusive).
    - **Actions:**

        - Converts the dates to JavaScript Date objects, setting the time to cover the full days.
        - Iterates through each task, checking if its creation date falls within the range.
        - Groups worked hours by date and by each label attached to the task (or “No Label” if none).
        - Constructs a 2D array with headers (`["Date", "Label", "Total Worked Hours"]`) and subsequent rows for each day/label combination.
        - Depending on the selected export format (CSV or XLSX), the function converts the array into the appropriate file type:
            - For XLSX: Uses the SheetJS library to create a workbook, convert data to a sheet, and trigger a download.
            - For CSV: Converts the array to comma-separated values, creates a Blob, and triggers a download.
    - ***Usage:*** This function is called when the user submits the export form.

---

7. Helper Functions

These functions assist with various operations such as formatting, rendering, and task management.

- **`parseDeadline(deadline)`**
  Converts a deadline string (if provided) into a JavaScript `Date` object.
    - **Parameter:**
        - `deadline`: A string in the format `"YYYY-MM-DD HH:MM"`.
    - **Returns:**
        - A Date object or `null` if no deadline is provided.
    - ***Usage:*** Used in determining the background color of a task based on how close its deadline is.
- **`getTaskBackground(task)`**
  Determines the background color of a task card based on its deadline.
    - **Parameter:**
        - `task`: A task object.
    - **Logic:**
        - If no deadline exists, returns `"white"`.
        - If the deadline is more than 48 hours away, returns a light green color.
        - If between 24–48 hours away, returns a light yellow color.
        - If within 24 hours, returns a light red color.
    - ***Usage:*** Called when rendering each task to give a visual cue for urgency.

- **`renderTask(task)`**
  Creates the HTML for an individual task card and appends it to the correct column based on its status.
  - **Parameter:**
    - `task`: A task object.
  - **Actions:**
    - Creates a `div` element with the class `"task"`.
    - Sets the card as draggable and assigns an event listener for the drag event.
    - Sets the background color using `getTaskBackground()`.
    - Constructs inner HTML that displays the task details (name, description, deadline, note, estimated time, worked hours, and any labels).
    - Adds buttons for “Copy”, “Edit”, and “Delete” actions.
    - Appends the task element to the corresponding column’s task list.
  - ***Usage:*** Called by `renderAllTasks()` for each task.
- **`renderAllTasks()`**
  Clears all task lists on the board and re-renders each task.
  - **Actions:**
    - Clears the inner HTML of each element with class `"task-list"`.
    - For each column (Backlog, To Do, etc.), it filters tasks by the column’s status.
    - Sorts tasks (for example, by deadline).
    - Calls `renderTask(task)` for each task.
  - ***Usage:*** Called after any change (create, edit, drag-and-drop, deletion) to update the UI.
- **`drag(event)`**
  Event handler for the drag start event.
  - **Actions:**
    - Sets the data transfer object with the task’s ID so that it can be identified when dropped.
  - ***Usage:*** Attached to task elements to enable dragging.
- **`allowDrop(event)`**
  Prevents the default behavior to allow drop events.
  - ***Usage:*** Called on the task list areas (columns) to enable dropping.
- **`drop(event)`**
  Handles the drop event when a task is dragged and released over a column.
  - **Actions:**

        - Retrieves the task's ID from the data transfer object.
        - Finds the task in the `tasks` array.
        - Determines the new status from the target column's data attribute.
        - Updates the task's status, calls `renderAllTasks()`, and saves the updated tasks to local storage.
    - ***Usage:*** Enables the drag-and-drop functionality of moving tasks between columns.

- **`copyTask(taskId)`**
  Copies the details of a task to the clipboard.
    - **Parameter:**
        - `taskId`: The ID of the task to copy.
    - **Actions:**
        - Constructs a string with task details.
        - Uses the Clipboard API to copy the string.
        - Alerts the user on success or failure.
    - ***Usage:*** Triggered when the user clicks the "Copy" button on a task card.

- **`deleteTask(taskId)`**
  Removes a task from the `tasks` array.
    - **Parameter:**
        - `taskId`: The ID of the task to be removed.
    - **Actions:**
        - Filters out the task from the array.
        - Calls `renderAllTasks()` and updates local storage.
    - ***Usage:*** Triggered when the user clicks the "Delete" button on a task card.

- **`clearLocalStorage()`**
  Clears all saved tasks and labels from local storage.
    - **Actions:**
        - Confirms with the user before clearing.
        - Removes the keys for tasks and labels.
        - Resets the `tasks` and `labels` arrays.
        - Re-renders the board.
    - ***Usage:*** Allows the user to start fresh by deleting all data.

- **`backupData()`**
  Creates a JSON backup file containing both tasks and labels.
  - **Actions:**
    - Combines tasks and labels into one object.
    - Converts it into a formatted JSON string.
    - Creates a Blob and triggers a download of the JSON file.
  - ***Usage:*** Called when the user clicks "Save Data to JSON".
- **`restoreBackup(event)`**
  Restores tasks and labels from a JSON backup file.
  - **Actions:**
    - Reads the selected file.
    - Parses the JSON and checks if it contains both tasks and labels (or tasks only).
    - Updates the `tasks` and `labels` arrays accordingly.
    - Calls the save and render functions to update the board.
  - ***Usage:*** Triggered when the user selects a file via the "Restore Data from JSON" input.

---

8. Initialization

- **`window.onload`**
  This function is executed when the page loads. It calls:
  - `loadLabelsFromLocalStorage()` to load any previously saved labels.
  - `loadFromLocalStorage()` to load tasks and render them on the board. ***Usage:*** Ensures that the Kanban board displays existing tasks and labels when the user opens the page.

---

9. Modal Sections (HTML Structure)

The HTML includes several modal sections for different functionalities:

- **Create Task Modal:**
  Contains a form for entering task details and assigning labels.
- **Edit Task Modal:**
  Similar to the create modal but pre-filled with the task's current information for editing.
- **Export Report Modal:**
  Contains inputs for selecting a date range and export format (CSV or XLSX) for generating a work report.

Each modal is toggled by setting its overlay's `display` style between `"flex"` (visible) and `"none"` (hidden).