**Project Reflection**

**Challenges and Solutions**

Building this task management app presented several significant challenges. The biggest hurdle was working with localStorage for data persistence - understanding how to serialize and deserialize JSON data properly took numerous attempts. Namely:

JSON serialization and deserialization were handled in these two functions:

javascript

*// Save tasks to local storage (serialization)*

function saveTasks(){

localStorage.setItem("tasks", JSON.stringify(tasks));

}

*// Load tasks from local storage (deserialization)*

function loadTasks(){

const storedTasks = localStorage.getItem("tasks");

if(storedTasks){

tasks = JSON.parse(storedTasks);

}

}

The serialization happens in saveTasks() with JSON.stringify(tasks), which converts the JavaScript tasks array to a JSON string before storing it in localStorage.

The deserialization happens in loadTasks() with JSON.parse(storedTasks), which converts the JSON string from localStorage back into a JavaScript array.

These functions are called at crucial points in the app --saving after adding/updating/deleting tasks, and loading when the app initializes.

Another major challenge was implementing the automatic detection of overdue tasks, which required learning how to compare JavaScript Date objects properly. Namely:

Happens in the checkOverdueTasks() function:

javascript

*//Check for overdue tasks//*

function checkOverdueTasks(){

const currentDate=new Date();

currentDate.setHours(0,0,0,0);*//Set to beginning of day for accurate comparsion//*

tasks.forEach(task => {

const deadlineDate = new Date(task.deadline);

deadlineDate.setHours(0, 0, 0, 0); *// Set to beginning of day*

*// Mark as overdue if deadline has passed and task is not completed*

if (deadlineDate < currentDate && task.status !== "Completed") {

task.status = "Overdue";

}

});

*//Save updated tasks to localStorage//*

saveTasks();

}

This function handles several aspects of JavaScript Date comparison:

\*It creates date objects from both the current time and the stored deadline strings.

\*It normalizes both dates by setting hours, minutes, seconds, and milliseconds to zero using setHours(0,0,0,0) to ensure only the dates themselves are compared, not the times.

\*It uses the < operator to compare dates (checking if the deadline is earlier than the current date).

\*It only marks tasks as overdue if they're not already completed.

This function is called during application initialization and after adding new tasks, ensuring overdue status is always up to date.

When faced with these obstacles, I adopted a systematic approach. For localStorage issues, I used console.log extensively to track data flow and identify where things went wrong. With the date comparison problems, I broke down the problem into smaller steps, first ensuring I could create valid Date objects before attempting comparisons.

**Future Improvements**

Given more time, I would enhance this project in several ways. First, I'd implement proper task editing functionality rather than just status updates. I'd also improve error handling with more user-friendly messages and add data validation. The UI could benefit from subtle animations to make interactions feel more polished. Finally, I'd reorganize the JavaScript code into modules for better maintainability as the application grows.

This project taught me that persistence and methodical debugging are just as important as coding knowledge when building real applications.