Final Project Report: Trello Progress Dashboard

David Campbell III

Overview

The Trello Progress Dashboard is a web-based tool designed to assist small software development teams in tracking project progress with enhanced clarity across various interconnected departments that work on tasks and the software systems the tasks aim to build. The dashboard imports a JSON export of a Trello board and visually represents task data using dynamic JavaScript-powered UI elements.

Features & Functionality

Data Import and Parsing

- Users upload a JSON file exported from Trello.
- The dashboard parses this file to extract tasks, labels (departments), custom fields (systems, estimated hours), and list names (to infer task status).

Progress Visualization

- Progress bars show task completion (not started, in progress, completed) and estimated vs. completed hours.
- Labels (departments) and custom fields (systems, estimated hours) are visually mapped to the progress bars, representing the cumulative progress across various tasks in certain departments and systems.

Filtering & Sorting

- Two dropdown menus allow filtering by department and software system.
- Progress bars are automatically sorted from most to least backlogged systems/departments, where the system/department (determined by the current filter) with the most remaining tasks is shown first.

Tab System

- Instead of a page-based system as originally outlined in the project proposal, instead a tab-based UI is used, allowing users to switch between:
    - **Departments View**: Filtered or complete department progress. This also contains the progress for departments sharing the same tasks as the selected department, giving an overview of joint-department efforts and illustrating which departments are more tightly coupled than others.
    - **Systems View**: System-level progress regardless of department.
    - **Detailed Breakdown View**: Nested view showing each department's progress split by system.

Mobile Responsiveness

- Layout is fully responsive across screen sizes to allow practical use on mobile and tablet devices, down to the standard 320px width benchmark.

JavaScript Enhancements

JavaScript is central to the interactivity and data processing of the dashboard:

- **Dynamic DOM Manipulation**: Updates progress bars and stats dynamically based on user filter selections.
- **Event Listeners**: Listen to dropdown and tab interactions, dynamically updating the view.
- **Efficient JSON Parsing**: Reads deeply nested structures, mapping Trello labels and custom fields to their respective roles.
- **Weighted Progress Calculation**: For tasks missing estimated hours, an intelligent fallback mechanism assigns default weights, allowing balanced visual representation across all data, even if partial or incomplete (explained below).

Technical Challenges & Solutions

1. Parsing Large JSON Files

The Trello export used during development (taken from my team's actual Trello board for our current project) contained over 140,000 lines of JSON data. The sheer size made initial parsing and structure discovery difficult.

Solution:

A modular parser was developed that isolates relevant portions of the JSON (labels, cards, lists, and custom fields). Tasks were classified into "not started", "in progress", or "completed" based on matching their list names to known status indicators like "In Progress", "Review", or "Done".

2. Data Privacy

The test data used for this project contained real and sensitive information about the project and all of its tasks, which is essential to keep secure (especially when this project is later deployed on the open web).

Solution:

A small script was developed to "clean" the original Trello JSON export, removing any identifying information about the project and leaving only the data needed to sort each task by department and system, counted towards overall progress based on estimated time in the custom fields of a task.

3. Inconsistent Time Estimates

Some tasks had no estimated hours. This can be solved by simply measuring progress based on the number of tasks completed vs the number remaining, but this then negates the purpose of estimated time entirely. Simply ignoring them skewed progress percentages, as some systems/departments contained many tasks without estimated hours at all.

Solution:

A weighted task progress algorithm is used. Tasks with hours contribute their actual values; those without are assigned a dynamic default based on the average estimated hours across all tasks. This hybrid model results in a more realistic, proportional display of progress.

4. Counting, Grouping, and Processing Tasks

Counting each task and grouping it by department or system was fairly straightforward, but it was more challenging figuring out how to relate different combinations of data together (such as the Detailed Breakdown view, where all tasks grouped by system, but only for 1 department; or the Department view, where all tasks filtered by 1 department and 1 system, but

also sub-filtered by each department also working a task alongside the main filtered department and system).

Solution:

Each minimum subsection of how data could be sorted was developed to work independently from one another, allowing a chained set of filter functions to cascade one after the other until the more complex configuration of data was achieved. For example, to handle the Detailed Breakdown view, instead of having a single function that handles sorting the data by department, and then by system within that department, we reuse the exact same functions from the Department or Systems view. We simply filter all departments by all systems (to get the total progress of each department), and then pass that data back into the filter functions where each department data is individually sub-filtered by all systems.

By creating this modular system of filtering and grouping functions, we strip the problem down to its barest components, and rebuild them into whatever complex data configuration we might need. This also lends itself to greater extensibility, as adding more filters or complex data representations means just adding one more generalized function to solve that specific subgoal not yet achieved by the existing filter functions.

5. Mobile Responsiveness

Mobile responsiveness was particularly difficult for the progress bars, as information is densely packed on either end of most progress bars throughout the pro

Solution:

To maintain the same visual appearance while allowing the progress bars to remain responsive down to the target of 320px wide, a flex row layout is swapped for a flex col layout at the appropriate breakpoint. By extending the rows of the single column to full width, the text on the right side of the progress bar which was originally in a second column can then be pushed to the far right side of the second row, allowing each side of the progress bar to respond to screen sizes correctly. Before this switch to a row vs col layout at smaller screen sizes, it was common for the left side of the progress bar info to push the right side off the screen as it struggled to scale to the smaller width.

Plans for Future Features

To continue developing this into a fully integrated tool for our production pipeline, the following enhancements are planned:

- **Backend Integration**
    - Automate data retrieval via Trello API, removing the need for manual JSON exports. This also acts as a proxy server, allowing for sensitive data to be stripped out from the raw Trello responses before being forwarded to the client for rendering and sorting. This will be implemented via Firebase Cloud (serverless) functions, which are already commonly used across our other web-apps/tools across our pipeline.
- **User Authentication**
    - Secure dashboards for different team members, with role-based visibility (for example, only show Art-related tasks to Art team leads).
- **Long-Term Trends**
    - Visualize historical trends across departments and systems over a duration of time.

Conclusion

This dashboard successfully meets its initial goal of improving team insight into project progress using only a Trello JSON export. By leveraging JavaScript for real-time data parsing and visualization, and through iterative enhancements like the weighted progress model and responsive UI, the dashboard is now a functional prototype ready for production deployment and backend extension.