Redmine - Feature #43165

A Roadmap for the Future of Redmine View Development

2025-08-27 14:07 - Takashi Kato

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:	UI	**Estimated time:**	0.00 hour
Target version:			
Resolution:			

Description

A clear roadmap for the future of Redmine view development would be very helpful when considering new features or contributions.

- Currently, jQuery-UI and jQuery are still widely used, and some JavaScript is embedded directly in ERB templates.
- Some legacy jQuery syntax remains.
- Rails itself is moving forward with the introduction of Hotwire and the deprecation of rails-ujs.

As a starting point for discussion, I have created a branch that introduces Hotwire (the first few commits have already been submitted as patches: #42517, #42521):

https://github.com/tohosaku/redmine/tree/hotwire

My experience so far can be summarized as follows

- Around 120 commits were made, covering the following changes:
  - Removal of jQuery-UI and jQuery
  - Removal of JavaScript from ERB templates
  - Modularization of all JavaScript (including jstoolbar, attachment, contextmenu, etc.) using ES modules
  - Introduction of Turbo
  - All work was done without the use of AI coding tools

- Introducing Stimulus eliminates the need to rely on Rails' automatically generated DOM element IDs for JavaScript event handlers, improving code clarity.

- Moving all template JavaScript into Stimulus controllers resulted in 96 controller files in total.

- SJR (js.erb + rails-ujs) and Turbo Streams are nearly identical, which makes implementation relatively straightforward.

- When implementing Turbo Drive, the DOMContentLoaded event is not triggered during page transitions. This requires significant rewrites of event handler logic. (Hotwire's introductory articles often suggest simply replacing it with the turbo:load event, but this is not always sufficient.)

- In the importmap-rails environment, all registered libraries are preloaded by default. This means libraries used only in specific views, such as Chart.js and gantt.js, are downloaded by all users. By setting preload: false and using dynamic import in the Stimulus controller, libraries are only loaded when the relevant screen is opened.

Issues that have emerged during development

- **Plugin compatibility**
  A compatibility mode may be required for plugins that still depend on jQuery.

- **Stimulus does not provide namespacing**
  Conflicts can occur if multiple plugins define Stimulus controllers with the same name, similar to conflicts with non-namespaced CSS selectors or non-modular JavaScript functions.

- **Changes in JavaScript customization methods**
  ES module–based code makes direct function overrides more difficult. We need to consider how to extend them through plugins such as view_customize.
  It may also be worth confirming whether Stimulus controllers can be inherited by plugins and documenting this approach.

History

#1 - 2025-08-28 16:59 - Holger Just

Thank you for your contributions. I think it is very worthwhile to explore options to modernize these parts of Redmine to allow more a straight-forward development and better security.

With that being said, Redmine is quite mature software. It was able to keep up with development of internal changes and external requirements over the last 18 years. One reason this was possible with a rather small and changing number of contributors is that Redmine generally is and was quite conservative with adopting or requiring external software components.

Redmine deliberately did not follow many trends-of-the-day in both Rails and Javascript development which has helped the project to be able concentrate on Redmine itself rather than running behind fast-changing (and often also quickly abandoned) external dependencies which is quite extrem esp. with many Javascript frameworks and also a lot of past approaches of Rails/Javsscriot integration frameworks. I strongly believe that with Redmine relying on JQuery and JQueryUI, we have absolved us of A LOT of unnecessary busywork in the past.

As such, I believe we should continue to be **very** conservative when adopting external or additional frameworks, even if they may be considered the current state of the art. Specifically, I believe Redmine should keep its current approach of using regular page requests which return full-page server-rendered HTML as much as possible in order to control as much of the rendering as possible from the server.

I see the appeal of newer approaches for getting rid of the ERB-generated JS responses and to be able leverage more security features such as strict CSP rules. However, I'd rather prefer a more focused approach here which does not tie us into quickly changing external frameworks. If we are to tie us to this framework, I think its advantages should be clearly spelled out and must be shown to be overwhelming in favor of this approach.

From your proposal, I'm especially concerned about the adoption of Turbo and Hotwire. Right now, I do not believe that it would gain us significant advantages, compared with the hassles of having to deal with all the small gotchas and interaction particularities it brings and having to deal with required updates once the new hot trend comes around in a few years. In my eyes, the majority of the ecosystem of Hotwire and Turbo is overkill for Redmine and instead introduces a lot of moving parts and dependencies which would be very hard to get rid of later on.

Again: the stability of JQuery over the last > 15 years has been **very** helpful for Redmine and we should be very careful to throw this away...