Main Reasons Why Turbolinks Are Not Worth The Effort

As a new developer, you advance by learning things every day. About a month ago, I developed a web app which is part of the Firehose core curriculum. Its design was to obtain a site which when visited, would array a random quote from the application database to the site user. The site was fluid and random, meaning that a user could contribute and track whatever they published. Things, however, took an interesting turn immediately after deploying and requested for feedback from users. Many users complained that the site is "sticky." But wait--- isn't it supposed to be random?

You should, therefore, see a quote several times (two or three times) in a row dependent on the number in the database. It's therefore not a problem, I thought…. I then noticed persistent loading of two quotes at a go. Well, now we have an issue that required urgent attention. I had to perform extensive research and schedule a session with a fellow developer to discuss at length the error popping up in the web app. We discovered the problem was as a result of an error in the turbolinks. What are Turbolinks? Turbolinks makes your web experience amazing by enabling faster navigation of web applications. It allows a single-page application without necessarily adding complex client-side JavaScript frameworks. It uses HTML to render views on the server side thus linking to the pages with ease. When a user follows a link, it inexorably fetches the web page, substitutes in its and merges its , all these without inducing the full page load costs. Fortunately, Ruby was updated sometimes back to incorporate Turbolinks.

However, there were many views and issues with it. Turbolinks is designed to aid with page load times; unfortunately, it is not often compatible with other libraries that you may be trying to apply. To solve the earlier mentioned issue, my fellow developer suggested that I do away with the Gemfile for Turbolinks. (For instance, on the above illustration line 24 was removed) After removing the Gemfile for Turbolinks, you are then required to update the apps/assets/javascripts/application.js file. You are also required to comment out the line that says//require =turbolinks. For the illustration below I had already removed it. Then run a bundle install in your terminal – and that it the page works well. Turbolinks in rails for applications A while back as I was playing with AJAX and jQuery as we had been learning on how to use asynchronous JavaScript request and event listeners. I designed a web app which can add, edit and also comment on posts using AJAX. In addition to that, I added in some event listeners using jQuery, for example, a slideToggle feature to show the full text in the post or even add a comment form. Things worked well until I tried to link to it and later on everything broke.

Turbolinks captures all the links that seem to go to the HTML pages and especially those links that use easier routes, creates an AJAX request for the content and later brings in the response’s body replacing the body. However, it can increase the speed of your app a little bit. Turbolinks can decrease the test time by about 11 seconds since the speed difference is measurable using JS files and Basecamp NEXT’s CSS. Problems associated with turbolinks Turbolinks does not call the document's ready event, for instance, if you wanted to apply JavaScript’s alert feature on a page. 1 2 3 4 alert("You're seeing me if this page was directly loaded"); // a popup window $("#thing-we-wanted-hidden").hide(); // won't work on a link. The code above only works when the page has been directly refreshed. In case a page is accessed through a link in Rails 4, the gemfile for turbolinks does not fetch it. Fortunately, there is still a

way to solve this issue using jQuery-turbolinks. However, it still sucks a little. In some instance, turbolinks speeds up things a little bit since it doesn't require reloading your assets. However, the whole global scope is saved rather than the assets. How to get rid of turbolinks on your app There are three steps to follow when removing turbolinks on a Rails 4 application

1. Comment out the line turbolinks in your Gemfile to get rid of the turbolinks gem.
2. Remove the line //=require turbolinks in your app/assets/javascripts/application.js file to stop the asset from needing turbolinks
3. Delete both the line with data-turbolinks-track=>true which are the critical pairs in your app/views/layouts/application.html.erb file to end incorporation of a now-invalid hash option.

Reasons why turbolinks is not worth the Effort

? Lack of clear global scope. You will have pay so much attention to global scoping since the JavaScript is not reloaded. Most of the existing JavaScript works under the assumption of a single DOMContentLoaded event, and a clean global scope. In a reasonable world, great JavaScript plugins should be architected to operate well with a solution like Turbolinks. However, the assumption of a clear scope per server-rendered HTML page is created into most of the jQuery and JavaScript libraries that people tend to use. Libraries that we often use won't consider their compatibility with turbolinks. In our world every library and plugin we use recognizes global scoping more carefully. However, it is not the role of the various developers to discuss compatibility with Turbolinks.

? Duplicating Bound Events. Since JavaScript is not reloaded events, remain bound even when the body is replaced. Therefore, if you are using generic selectors to bind events while you are binding those events on all the turbolinks page load, you might end up running into problems with the same event being bound multiple of times leading to undesirable behavior. You can eliminate this issue if you pay close attention here. However, this problem cannot be present if the JavaScript was reloaded.

? Complexity in Feature Specs. One of the main benefits of turbolinks is that it speeds up the applications. However, it makes the authoring feature test more difficult. Since turbolinks does not load a new page, capybara will assume that clicking a link is the end of the request and will then move on. This tells us that every time you click a link in a feature test, you will have to make sure that you are waiting for AJAX to finish loading to continue writing the next line in the feature spec.

Whereas turbolinks speeds up page loads, it also makes authoring feature test more complicated leading to inconsistent and unexpected results. Conclusion In this case, if you are trying to incorporate the right amount of jQuery and JavaScript, turbolinks look to be more trouble than what they seem to be worth. However I not always comment out turbolinks from my Rails project, but I mainly focus on optimizing my queries. In case I need to speed up things with javascript, I consider using an MVC framework to write a single page application.