Support for dynamic websites: Use Optimizely on single page applications

Skip Ahead

Enable
Activate a page for a SPA
Triggers
Conditions
Deactivate a page
How support for dynamic websites works

relevant products:

- Optimizely Web Experimentation
- Optimizely Performance Edge

THIS ARTICLE WILL HELP YOU:

- Use Optimizely with a **single page application** (SPA)
- Run experiments on a website built on a framework such as React, Vue, AngularJS, Ember, or Backbone
- **Enable enhancements to Optimizely pages** to support a single page application

Many sites today are built as single-page applications (SPAs) in order to optimize the speed of content delivery and improve overall site performance. SPAs offer a few key advantages over traditional static websites: they’re fast, responsive, and compact. If you’ve heard that your site is built on a framework such as React, Vue, AngularJS, Ember, or Backbone, it’s likely that you’re working with a single-page application.

Because SPAs load content differently from traditional static sites, Optimizely has a powerful kit of features that enable you to build experiments on SPAs out of the box.
These features support dynamic websites by allowing you to trigger experiments based on changes that happen on a page, even when the page doesn’t reload. Jump to the bottom for a more detailed explanation.

Enable support for dynamic websites to:

- **Activate experiments based on content that isn’t immediately displayed when the page loads**, or content that becomes visible based on your visitors’ navigation of your site. Single-page applications don’t reload the entire page when a visitor navigates the site. Instead, existing elements are replaced by new ones as the visitor explores.

- **Apply or reapply changes to elements on your page that appear after the initial page load**. Some single-page applications use infinite scroll to add new elements to the page, without fully reloading the page.

Read on to learn about using Optimizely with dynamic websites.

---

**Enable**

To gain access to features that allow you to experiment on a single page application, enable support for dynamic websites. Once you do, you’ll gain the following options for activating Optimizely pages, in addition to standard page activation triggers:

- When the URL changes

- When the DOM Changes

You’ll use these to define conditions under which the page will activate or deactivate. When the page activates, your experiment will run.

Here’s how to enable support for dynamic websites at the project level:

1. Navigate to *Project Settings > Implementation*.
2. Then, select *Enable Support for Dynamic Websites*.

Nice work! You’ve enabled a set of features that will help you experiment on dynamic content.

Now, instead of polling for elements on the site for two seconds after initial page load, Optimizely will use *MutationObserver* to “see” when something has changed on your site to apply Visual Editor changes. This will not have any negative impact on current experiments.

If your account uses *custom snippets*, you’ll need to enable support for dynamic websites in the settings for the custom
snippet (rather than the basic snippet).

Activate a page for a SPA

Once you’ve enabled support for dynamic websites in your project, you’ll gain access to a few additional options in your page settings.

1. Create an Optimizely page by navigating to Implementation > New Page.

2. Under Page Settings, note a set of triggers and conditions that you’ll use to fine-tune when your page activates.

Triggers

Triggers tell Optimizely when to start checking whether certain conditions are true:

- **When the URL changes**: Optimizely triggers activation of this page each time it sees the URL changing, even if the full page does not reload. This works well with applications that rely on client-side routing to handle user navigation. This will be triggered on initial page load AND subsequent URL changes.

- **When the DOM changes**: Optimizely triggers activation of this page each time an element on the page is inserted, removed or modified.

If you have a single-page application, you’re likely to use the two triggers mentioned above most of the time. However, you’ll still have access to the standard page activation triggers as well.

Once you decide when Optimizely should check for certain conditions on your page, you’ll tell Optimizely whether it should check whether any conditions are true or all conditions are true.

Conditions

Conditions tell Optimizely what criteria to check for, when deciding whether to activate the page:

- **URL Targeting**: Use URL Match Types to define where the experiment should run.

- **Element is present on page**: Add one or more CSS selectors that Optimizely should check for each time the trigger for this page fires. Optimizely uses `document.querySelectorAll` to determine whether the element exists at the time the page fires.

- **Custom JavaScript**: Add a custom JavaScript function that is evaluated when the trigger for this page fires. The function should return a boolean value.

Add and remove multiple conditions to define the exact set of conditions under which you want your page to activate. This will determine when an experiment will run.

Under Triggers, you can tell Optimizely to check whether any conditions are true. This joins separate conditions with an OR statement. When you tell Optimizely to check that all conditions are true, conditions are joined with an AND
Once the conditions return as true, the page activates and your experiment will run.

**Deactivate a page**

Once a page is activated and the experiment runs, the visual changes in your experiment will be applied to your website. Visitors who see the site under the conditions you defined and who are bucketed into the experiment will see those changes live in the world. Support for dynamic websites ensures that those changes will be **continually applied**, even as your visitors navigate around the site and conditions change -- the changes will still persist once the page is activated, regardless of what’s happening on the page.

Occasionally, you may not want the changes in your experiment to persist indefinitely as visitors navigate. Imagine for example that you have a page offering phones for sale, and on that page your experiment displays a banner advertising a promotion on a certain brand. When a user goes to a page displaying a different brand of phone, however, you don’t want the promotional banner to display. By setting conditions you can trigger deactivation of the banner by deactivating the page that contained it, when a user navigates away from the page.

To deactivate a page when the conditions are no longer met:

1. Under your Page Settings, navigate to Advanced.

2. Check **Automatically deactivate this page when the trigger fires and conditions are false**. Optimizely will deactivate the page and stop applying changes to that page when:
   
   - A trigger on the page fires
   - The conditions evaluate to false
3. If you also want Optimizely to remove **previously-applied** changes when the page deactivates, select *Undo changes when this page deactivates*:

- Changes made with the Visual Editor are removed
- Variation CSS is reverted
- An extension’s “Reset JS” is executed
- Custom JavaScript **cannot be reverted**

The next time the page trigger is fired, the page will deactivate.

If a change is made to the body tag using the visual editor, those changes can not be removed as outlined in our documentation. This is an open BUG issue CJS-2655

---

### How support for dynamic websites works

Optimizely works by applying changes to your site when a page is activated. If you have a traditional static site (and you're **not** using the feature kit that supports dynamic websites), Optimizely applies changes to elements on your page that load within 2 seconds of the initial page load. Ordinarily, this is plenty of time for elements to load so that visual changes seamlessly applied, without affecting your visitors’ experience.

However, unlike static sites, single page applications don’t fully reload the page in order to change the content that a visitor sees. Instead, new elements are changed or added to alter the site experience as a visitor explores. This means that two seconds is often not enough time to capture when an element loads on a single page application in order to apply the visual changes through an experiment.

Optimizely supports dynamic websites by monitoring when page elements load, even when the page itself doesn't fully reload. To do this, we use a Web API called **MutationObserver**.

**MutationObserver** provides hooks into DOM mutations and enable Optimizely’s snippet to know when a DOM node is inserted, destroyed, or modified so that we can apply (or reapply) changes at the right moment. In other words, instead of polling for elements on the site for two seconds after initial page load, Optimizely uses **MutationObserver** to “see” when something has changed on your site.

Once an experiment or campaign activates and a visitor is bucketed into the experiment, **MutationObserver** checks the DOM for as long as the page is active and applies (and reapplies) changes as appropriate. Once the page deactivates, Optimizely no longer attempts to change the elements on that page.

The observerSelector utility function acts as a wrapper for the **MutationObserver** Web API.

**This page** shows an example of a SPA built as a React application that is running Optimizely experiments. On the example **Cards** page, click **Additional Cards**. Note that the URL doesn’t change, but a new card is added to the page.

In this example, an experiment is running that changes the color of the **Apply Now** buttons to green, to measure how
that change affects engagement. Using Dynamic Support, Optimizely senses that elements are added to the page and applies the experiment parameters (makes these buttons green) as the elements appear, even though the page has not reloaded.

Next, click the Mortgages tab. Note that the last word in the URL changes from Cards to Mortgages. Now click to return to the Cards page again, and note that the URL changes back to Cards. Because of dynamic website support, Optimizely can detect which page is being displayed and can know what changes to apply even though the pages aren't reloading.

client-js has different levels of support for different kinds of changes:

- Insert HTML and Insert Image changes, which are only reapplied when new elements appear
- Rearrange changes, which are never reapplied
- Everything else, which is reapplied when new elements appear and when previously-modified elements are mutated

Optimizely does not currently support "rearrange" tests (tests that swap the positions of nodes in a document) on dynamic websites, although we may extend support in the future.

Technical details for not currently supporting "rearrange" tests are included below.

Potential issue; no clients have reported this issue so far.

Repro Steps:

1. Create a rearrange change with a selector that matches more than one element (like p)
2. In the shared code, append a <p> element to the body at DOM ready (long after the first, but before client times out)
3. Start the experiment (or preview) and navigate to the page

Expected: All <p> elements should be rearranged (both ones on the page initially and the one added via shared code)

Actual: Only the first matching <p> element is rearranged

Reason: Our current implementation of the rearrange change only attempts to execute a rearrange once for the given selector.

Technically speaking, we have 3 "visual" change types in the Editor UI:

1. "Element Change" - This UI element actually captures 2 types of changes and saves them in a single change object:
   --"Rearrange" - Applies ONCE for the first found element and does not reapply an already applied change.
   --Everything else, which we tend to call "attribute" changes. These apply and reapply indefinitely for all changes.
2. "Insert HTML" - Applies indefinitely for NEW applications but does not reapply an already applied change.
3. "Insert Image" - Applies indefinitely for NEW applications but does not reapply an already applied change.