Turbo.js Cheatsheet

A comprehensive guide to Turbo.js, covering its core concepts, components, and usage patterns for building modern web applications with enhanced speed and responsiveness.

Turbo Drive Fundamentals

Navigation & Page Updates

Turbo Drive: Automatically intercepts clicks on all `<a>` tags and form submissions, preventing full page loads.

Instead, Turbo Drive fetches the new page in the background and updates the current page's `<body>` using `morphdom`.

Turbolinks-classic Compatibility: Turbo is designed as a successor to Turbolinks. Many concepts remain similar, but Turbo offers significant improvements, including more robust handling of JavaScript and asset loading.

No Configuration Needed: To enable Turbo Drive, simply include the `turbo.js` file in your application. It automatically enhances existing links and forms.

Meta Tags: You can control Turbo Drive's behavior using meta tags in the `<head>` section of your pages.

Example: `<meta name="turbo-visit-control" content="reload">`

Page Visit Events

`turbo:before-visit`	Fired before Turbo Drive starts a visit.
`turbo:visit`	Fired when Turbo Drive is about to fetch a new page.
`turbo:before-cache`	Fired before Turbo Drive caches the current page.
`turbo:before-render`	Fired before Turbo Drive renders the new page.
`turbo:render`	Fired after Turbo Drive renders the new page.
`turbo:load`	Fired after Turbo Drive completes a visit and the new page is visible.

Disabling Turbo Drive

You can disable Turbo Drive on specific links or forms by adding the `data-turbo="false"` attribute.

Example: `<a href="/full_page_load" data-turbo="false">Full Page Load</a>`

To disable Turbo Drive completely, remove the `turbo.js` script from your application or set `Turbo.session.drive = false;`.

Turbo Frames

Encapsulating Page Sections

Turbo Frames: Allow you to update specific parts of a page without reloading the entire page. This is achieved by wrapping sections of your HTML in `<turbo-frame>` elements.

Lazy Loading: Turbo Frames can also be used for lazy loading content. Content within a frame is only loaded when the frame is scrolled into view (or when explicitly triggered).

Basic Frame Example

```
<turbo-frame id="user_profile">
  Loading user profile...
</turbo-frame>

<script>
  fetch('/users/123')
    .then(response => response.text())
    .then(html => {

document.getElementById('user_profile').innerHTML = html;
    });
</script>
```

In a Rails-like backend, a corresponding `users#show` action might render a partial that replaces the `user_profile` frame's contents.

Frame Events

`turbo:frame-load`	Fired after a Turbo Frame has loaded its content.
`turbo:frame-render`	Fired after a Turbo Frame has rendered the content

Frame Attributes

`id`	A unique identifier for the frame. Required for Turbo to target and update the frame.
`src`	The URL to load the frame's content from. The content fetched from this URL will replace the frame's current content.
`target`	Specifies the `id` of another Turbo Frame to update after a form submission or link click within the current frame. This allows you to chain updates across multiple frames.

Turbo Streams

Asynchronous DOM Updates

Turbo Streams: Deliver asynchronous DOM updates over WebSocket connections or server-sent events. Streams are particularly useful for real-time applications or scenarios where server-side events need to be reflected in the client-side UI immediately.

Stream Actions: Turbo Streams use actions like `append`, `prepend`, `replace`, `update`, and `remove` to modify the DOM.

Stream Message Format

Turbo Stream messages are typically sent as HTML fragments containing `<turbo-stream>` elements. These elements specify the action to perform and the target element to modify.

Example:

```
<turbo-stream action="append"
target="messages">
  <template>
    <div>New message!</div>
  </template>
</turbo-stream>
```

The `target` attribute specifies the `id` of the element to modify. The content within the `<template>` tag is used to perform the action.

Stream Actions

`append`	Appends the content to the end of the target element.
`prepend`	Prepends the content to the beginning of the target element.
`replace`	Replaces the entire target element with the content.
`update`	Replaces the content *within* the target element with the content.
`remove`	Removes the target element from the DOM.

Advanced Turbo Techniques

Using `data-turbo-stream`

You can trigger Turbo Stream updates directly from links and forms using the `data-turbo-stream` attribute. When a link or form with this attribute is clicked or submitted, Turbo will expect the server to return a Turbo Stream response.

Example:

```
<form action="/comments" method="post" data-
turbo-stream="true">
  ...
</form>
```

Redirects and Turbo

When handling form submissions with Turbo, you can return a redirect response. Turbo Drive will automatically follow the redirect and update the page.

If you need to perform additional actions after the redirect, you can use the `turbo:load` event.

JavaScript Considerations

Since Turbo Drive prevents full page loads, you need to ensure that your JavaScript code is compatible with Turbo. Use event delegation to attach event listeners to elements that may be replaced during Turbo Drive updates.

Example:

```
document.addEventListener('turbo:load', () =>
{
  document.addEventListener('click', '.my-
element', (event) => {
    // Handle click event
  });
});
```

Caching

Turbo Drive caches pages to improve performance. You can control caching behavior using meta tags and server-side headers. Use `turbo:before-cache` event to modify the page before caching.