Hotwire Cheatsheet

A comprehensive cheat sheet for Hotwire, covering Turbo and Stimulus, with examples and usage guidelines to quickly implement modern web features.

Turbo Basics

Turbo Drive

Turbo Drive automatically accelerates links and form submissions.

Key Features:

- Intercepts clicks on all same-origin links.
- Prevents the browser from following the link.
- Requests the page using `fetch`.
- Replaces the `<body>` of the current page with the `<body>` of the response.
- Merges the `<head>` section.

Enabling Turbo Drive:

Turbo Drive is enabled by default when you include `turbo.js` in your application.

Disabling Turbo Drive:

To disable Turbo Drive on a specific link or form, add `data-turbo="false"`.

```
<a href="/slow_page" data-turbo="false">Visit Slow Page</a>
```

Turbo Visit Options:

`data-turbo-action="replace"` - Replaces the current history entry.

`data-turbo-action="advance"` - Creates a new history entry.

`data-turbo-action="restore"` - Restores a previous history entry.

Turbo Frames

Turbo Frames allow you to isolate sections of a page into independent components that can be updated individually.

Key Features:

- Lazy loading of content.
- Parallel loading of content.
- Better perceived performance.

Defining a Turbo Frame:

Use the `<turbo-frame>` element with a unique `id`.

```
<turbo-frame id="messages">
  <!-- Content -->
</turbo-frame>
```

Updating a Turbo Frame:

When a link or form inside a Turbo Frame is activated, Turbo Drive will only update the contents of the frame.

```
# Example Rails controller action
def update
  @message = Message.find(params[:id])
  @message.update(message_params)
  render turbo_stream:
turbo_stream.replace(@message, partial:
'messages/message', locals: { message:
@message })
end
```

Turbo Streams

Turbo Streams deliver page changes as fragments of HTML to update parts of the page.

Key Features:

- Append, prepend, replace, update, remove actions.
- Broadcast updates via WebSockets.
- Server-Sent Events (SSE) support.

Example Turbo Stream Response:

```
<turbo-stream action="append"
target="messages">
  <template>
    <div id="message_123">New Message</div>
  </template>
</turbo-stream>
```

Rails Example:

```
# app/views/messages/create.turbo_stream.erb
<%= turbo_stream.append "messages", partial:
"messages/message", locals: { message:
@message } %>
```

Stimulus Basics

Stimulus Controllers

Stimulus organizes JavaScript in your application by connecting DOM elements to JavaScript objects called controllers.

Key Concepts:

- Controllers: JavaScript objects that manage a DOM element and its children.
- Actions: Methods on controllers that are triggered by DOM events.
- Targets: References to specific DOM elements within a controller's scope.

Defining a Controller:

```
//
app/javascript/controllers/example_controller.
js
import { Controller } from
"@hotwired/stimulus"

export default class extends Controller {
  connect() {
    console.log("Hello, Stimulus!")
  }
}
```

Connecting a Controller to HTML:

```
<div data-controller="example">
  <!-- Content -->
</div>
```

Actions

Actions connect DOM events to controller methods.

Syntax:

`data-action="event->controller#method"`

Example:

```
<button data-action="click-
>example#greet">Greet</button>
```

Controller Method:

```
//
app/javascript/controllers/example_controller.
js
import { Controller } from
"@hotwired/stimulus"

export default class extends Controller {
  greet() {
    alert("Hello!")
  }
}
```

Targets

Targets provide a way to reference specific DOM elements within a controller.

Syntax:

`data-controller-target="controller.targetName"`

Example:

```
<div data-controller="example">
  <input type="text" data-example-
target="name">
  <button data-action="click-
>example#greet">Greet</button>
</div>
```

Controller Access:

```
//
app/javascript/controllers/example_controller.
js
import { Controller } from
"@hotwired/stimulus"

export default class extends Controller {
  static targets = ["name"]

  greet() {
    alert(`Hello, ${this.nameTarget.value}!`);
  }
}
```

Advanced Turbo

Turbo Streams from Server

Sending Turbo Streams from the server enables real-time updates to specific parts of your page.

Use Cases:

- Real-time notifications.
- Live comment updates.
- Collaborative editing.

Rails Example:

```
# app/controllers/comments_controller.rb
def create
  @comment = Comment.new(comment_params)
  if @comment.save
    render turbo_stream: [
      turbo_stream.append("comments", partial: "comments/comment", locals:
{ comment: @comment }),
      turbo_stream.update("new_comment", "") # Clear the new comment form
    ]
  else
    render :new, status: :unprocessable_entity
  end
end
```

Broadcasting Turbo Streams:

```
# app/models/comment.rb
after_create_commit { broadcast_append_to :comments, partial:
"comments/comment", locals: { comment: self } }
```

Turbo Frames and Forms

Using Turbo Frames with forms allows you to update only the form or related sections of the page after submission.

Benefits:

- Improved perceived performance.
- Reduced data transfer.

Example:

```
<turbo-frame id="new_comment">
  <%= form_with model: @comment, url: comments_path do |form| %>
    <%= form.text_field :body %>
    <%= form.submit "Create Comment" %>
  <% end %>
</turbo-frame>
```

Controller Response:

```
# app/controllers/comments_controller.rb
def create
  @comment = Comment.new(comment_params)
  if @comment.save
    render turbo_stream: [
      turbo_stream.replace("new_comment", partial: "comments/form",
locals: { comment: Comment.new }),
      turbo_stream.append("comments", partial: "comments/comment", locals:
{ comment: @comment })
    ]
  else
    render :new, status: :unprocessable_entity
  end
end
```

Advanced Stimulus

Values API

The Values API in Stimulus allows you to define typed values that are automatically synchronized with data attributes.

Supported Types:

- String
- Number
- Boolean
- Array
- Object

Example:

```
// app/javascript/controllers/example_controller.js
import { Controller } from "@hotwired/stimulus"

export default class extends Controller {
  static values = {
    name: String,
    count: Number,
    isEnabled: Boolean
  }

  connect() {
    console.log(`Name: ${this.nameValue}, Count: ${this.countValue},
Enabled: ${this.isEnabledValue}`)
  }
}
```

HTML:

```
<div data-controller="example"
     data-example-name-value="John"
     data-example-count-value="10"
     data-example-is-enabled-value="true">
</div>
```

Classes API

The Classes API in Stimulus lets you automatically toggle CSS classes on elements based on controller state.

Benefits:

- Simplified class management.
- Reactive UI updates.

Example:

```
// app/javascript/controllers/example_controller.js
import { Controller } from "@hotwired/stimulus"

export default class extends Controller {
  static classes = ["active"]

  connect() {
    this.element.classList.add(this.activeClass)
  }
}
```

HTML:

```
<div data-controller="example"
     data-example-active-class="is-active">
  <!-- Content -->
</div>
```