# Hotwire or React ?

## ~Reactの録画機能をHotwireに置き換えて得られた知見~

Haruna Tsujita ( @haruna-tsujita )

2024/10/25

Kaigi on Rails 2024

# 自己紹介

@haruna-tsujita

-  → 株式会社キャタル（英語塾を運営）
- バックエンドエンジニア（Railsと、たまにReact）
- ライティングの添削サービスを開発

# 今日のテーマ

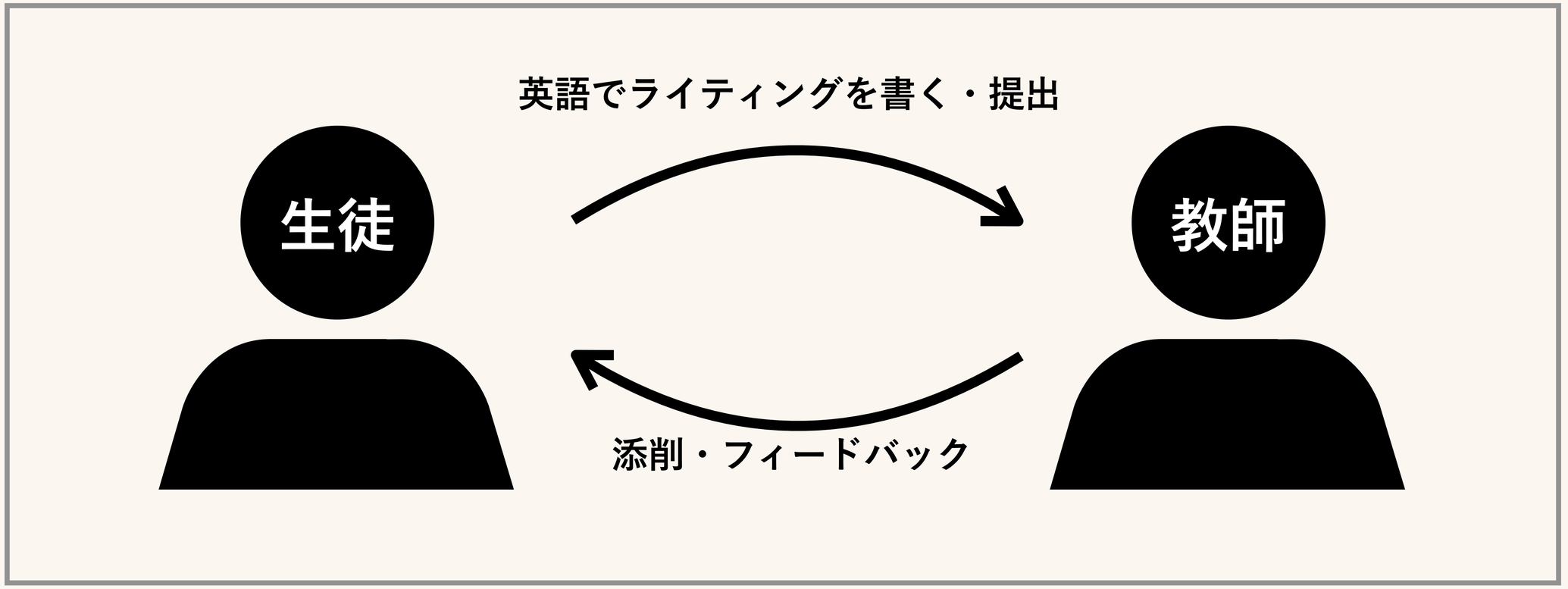
✦ メインはHotwire、特にStimulusの話です

✦ 技術選定に関する話もします

- 私はReactも好きです
- 本題の前にプロダクトやチームの状況を簡単に共有します

# ライティングの添削サービスとは？

- 英語塾キャタルに通う生徒に提供している小規模なアプリケーション
- Ruby on Rails / 部分的にReactでSPAを実現



このくらいの規模のサービスに

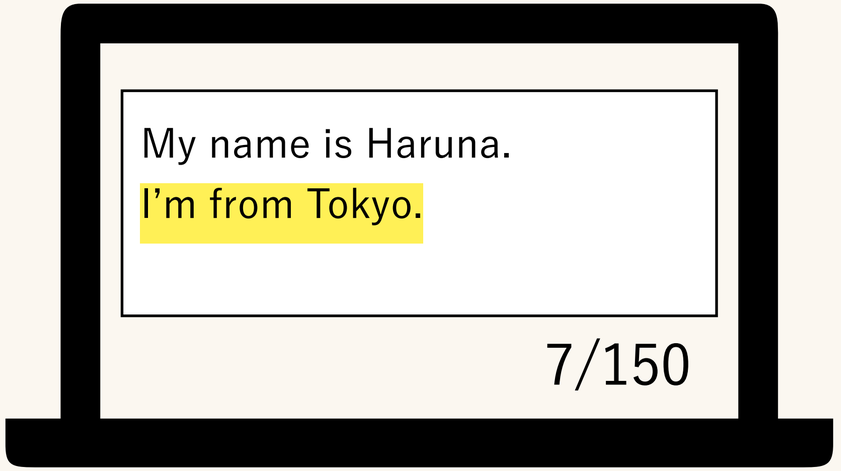
# React、本当に必要？

# React関連の課題

- Railsの方が得意なバックエンドエンジニア2名が開発
- 語数カウント・テキストエリアのハイライトなどちょっとした箇所をSPAにしたい
  - Reactで書かれた箇所が徐々に負債化
  - メインの機能開発に時間がかかってしまう

**大掛かりなことはしていない‥**

- **やりたいことはHotwireで十分実現できるのでは？**



# ReactをやめてHotwireにするのは怖い！！！

# 怖さの正体（と思われるもの）

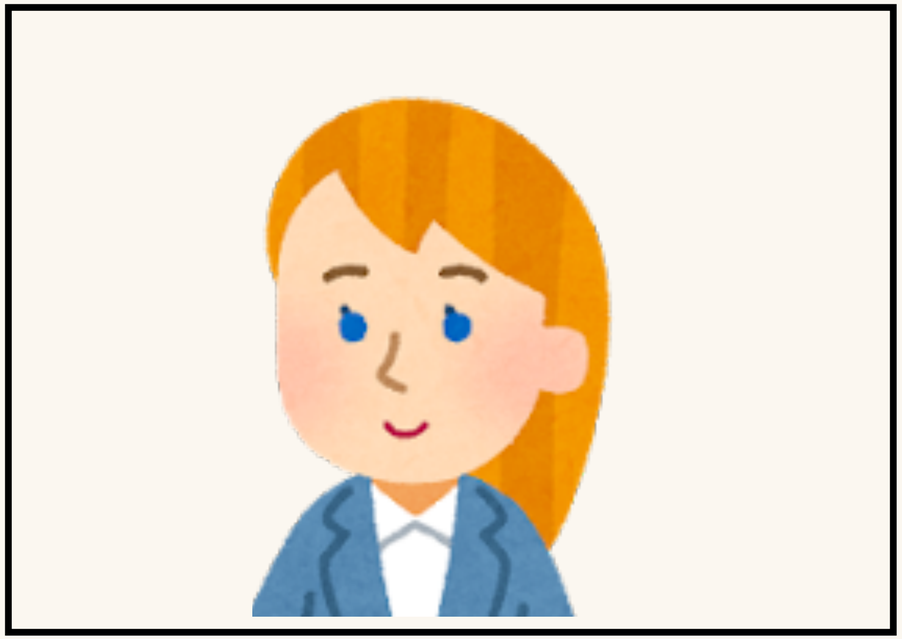
| | |
|---|---|
| **React** | エコシステムや知見が充実（とくにChakra UIを使いたい）<br>柔軟にUIを実装できる |
| **Hotwire** | TurboのイメージCRUD操作をNo JSでSPAライクにできる<br>管理機能くらいのプロダクトにフィット？<br>Stimulusの情報があまりなくて未知数 |

**Hotwireのみで十分開発できる？ Reactを手放して本当にいいの？**

# 技術検証する価値はありそう！

- 懸念の一つはStimulus
  - ▸ これがフィットしなければHotwire置き換えの選択肢が消える
- Reactを頑張ろう！と決意できるのはチームにとってプラスなはず

# 録画機能を置き換えてみる

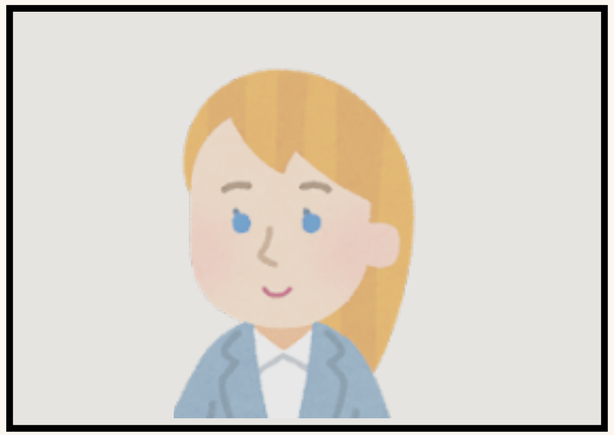


Record　Clear

- フィードバック動画の録画機能
  - 動画データの保存・再生・削除
  - PCのカメラ・マイクの操作が入るため確実にStimulusを使う
  - 機能改善が活発に行われない箇所

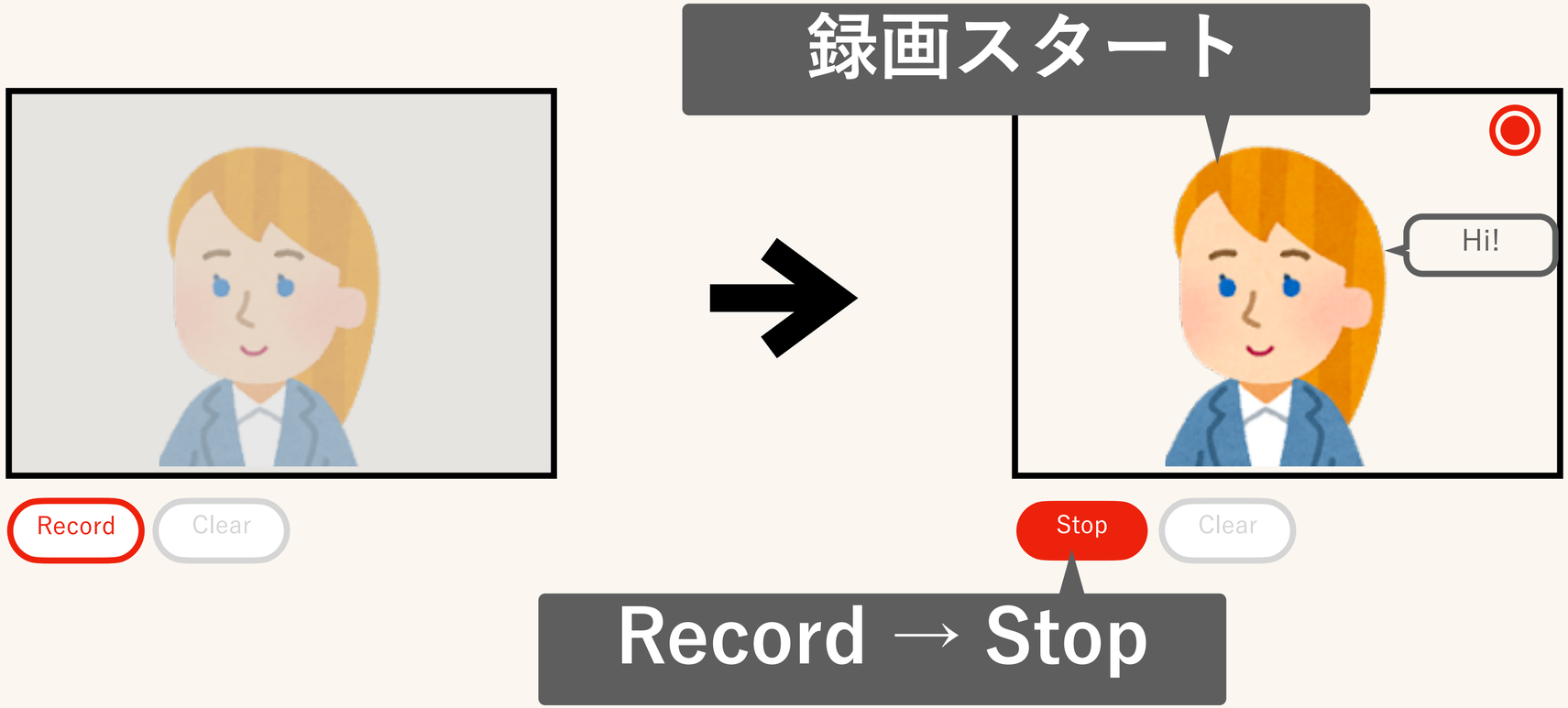
# 置き換えの前提条件

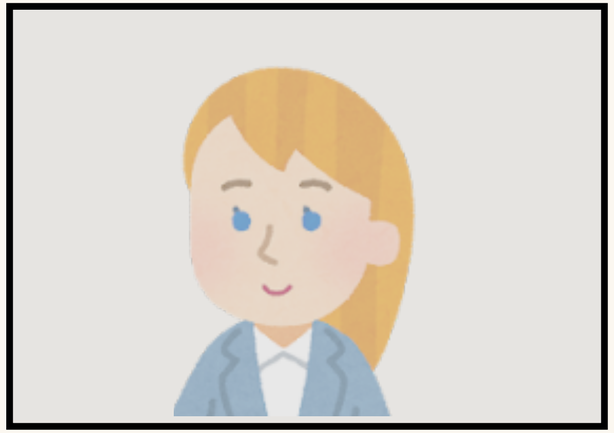
- ReactとHotwireの違いを検証することが目的
  - 既存のUI/UXを維持し、仕様は変更しない



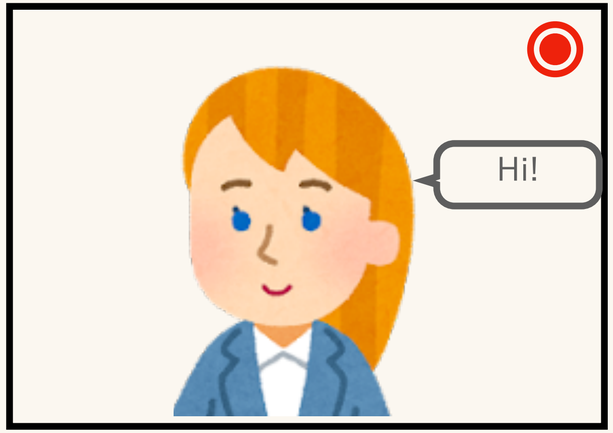
Record

Clear

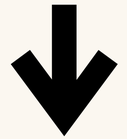

録画スタート
Hi!
Record
Clear
Stop
Clear
Record → Stop

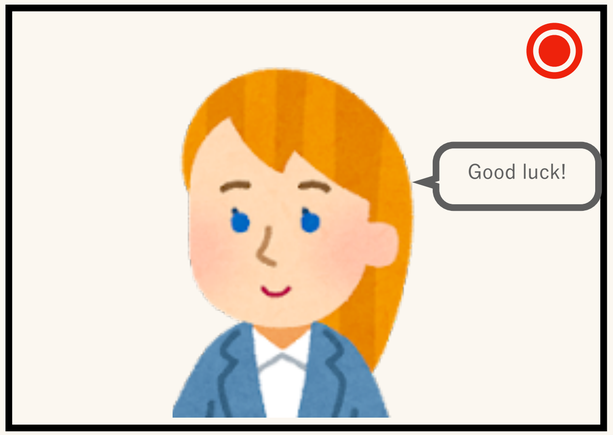


Record Clear

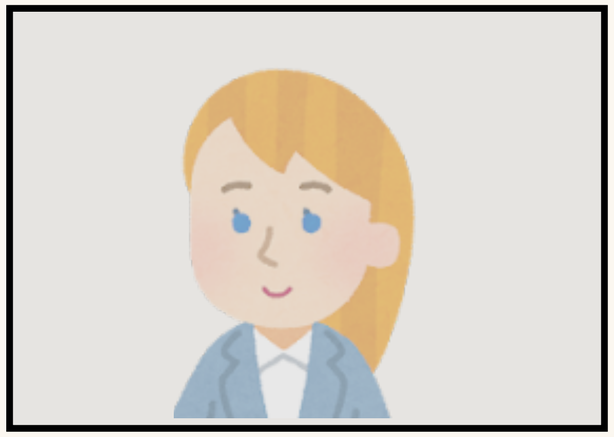



Stop Clear

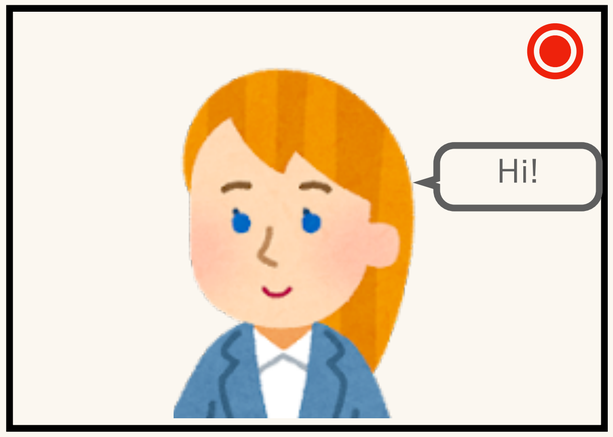




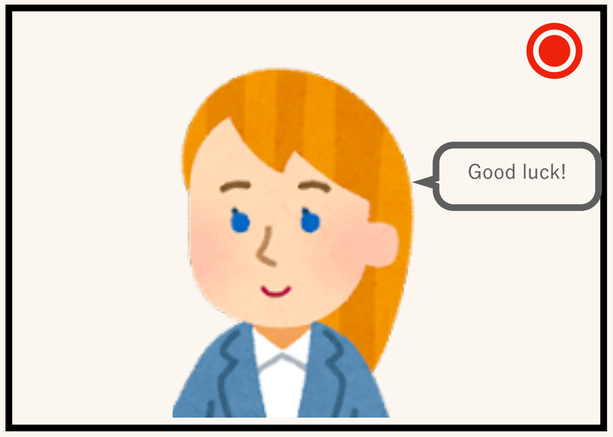

Stop Clear



Record Clear



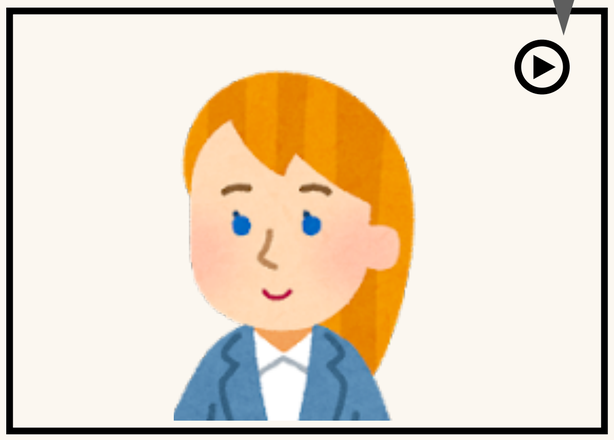

Stop Clear




Stop Clear

POST

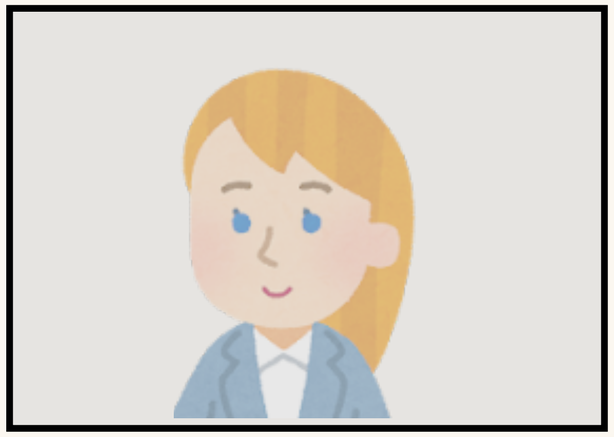
動画再生



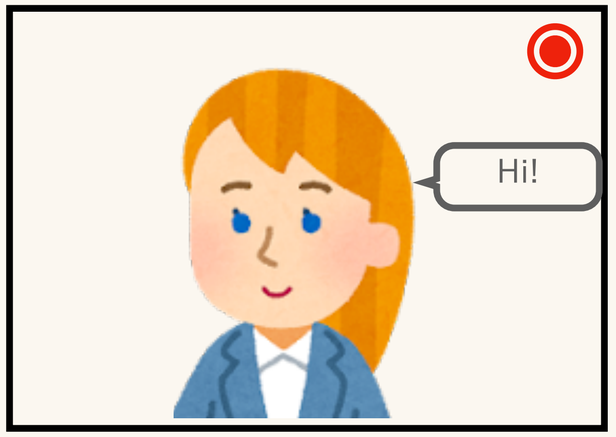
Record Clear

disabled: true

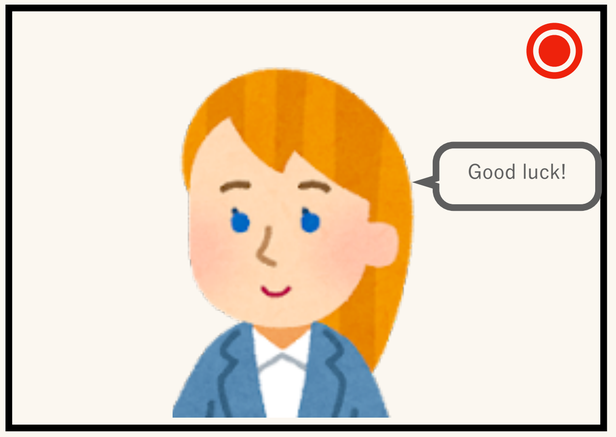
disabled: 解除



Record Clear

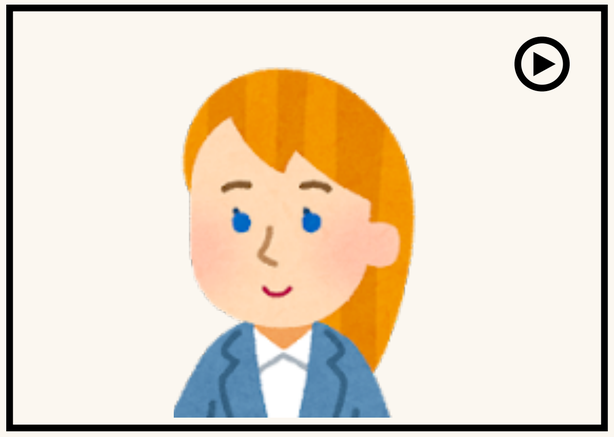



Stop Clear




Stop Clear

POST



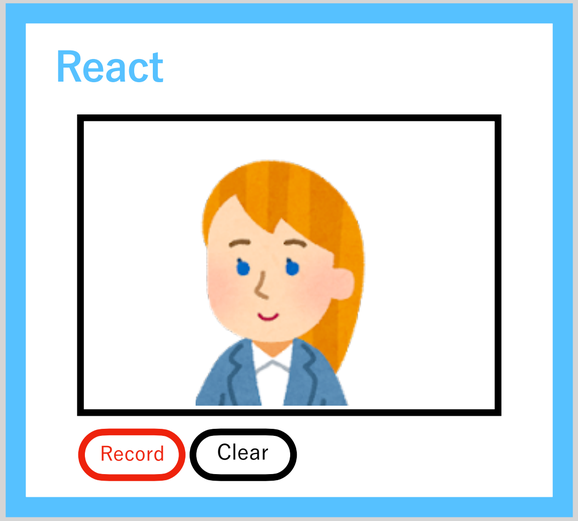
Record Clear

DELETE

# Reactでの実装を見ていきましょう

# 現状の実装

添削ページ

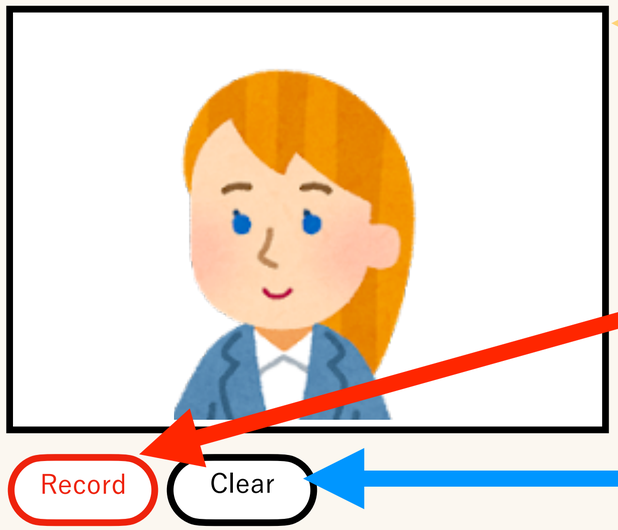



- 添削ページのviewからReactを呼び出す

```
// app/views/summaries/feedbacks/edit.html.haml

-# 省略

#summary-video-recorder{data: {'summary-id': @summary.id}}
= javascript_include_tag ‘SummaryVideoRecorder'

-# 省略
```

# 現状の実装

- 動画の表示・Recordボタン・Clearボタンがコンポーネント化
- ボタンのクリックイベントを起点に処理が実行される




```
return (
    <>
      <Video theme='large' videoUrl={videoUrl} mimeType='video/webm' />
      <p>
        <RecordButton
          startRecording={startRecording}
          stopRecording={stopRecording}
          id={summaryId}
          videoUrl={videoUrl}
        />
        <ClearButton
          onClick={handleClear}
          isDisabled={isPosting || !videoUrl}
        />
      </p>
    </>
  )
}
```

Turboと相性のよい
JavaScriptフレームワーク

# Hotwire/Stimulusとは？

引用：Hotwire Handbook

```html
<!--HTML from anywhere-->
<div data-controller="hello">
  <input data-hello-target="name"
type="text">

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

  <span data-hello-target="output">
  </span>
</div>
```

```js
// hello_controller.js
import { Controller } from "stimulus"

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

  greet() {
    this.outputTarget.textContent =
      `Hello, ${this.nameTarget.value}!`
  }
}
```

# Hotwire/Stimulusとは？

引用：Hotwire Handbook

```
<!--HTML from anywhere-->
<div data-controller="hello">
  <input data-hello-target="name"
type="text">

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

  <span data-hello-target="output">
  </span>
</div>
```

```
// hello_controller.js
import { Controller } from "stimulus"

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

  greet() {
    this.outputTarget.textContent =
      `Hello, ${this.nameTarget.value}!`
  }
}
```

# Hotwire/Stimulusとは？

引用：Hotwire Handbook

```
<!--HTML from anywhere-->
<div data-controller="hello">
  <button data-action="click->hello#greet">
    Greet
  </button>

  <span data-hello-target="output">
  </span>
</div>
```

イベントで発生するアクションを指定

```
// hello_controller.js
import { Controller } from "stimulus"

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

  greet() {
    this.outputTarget.textContent =
      `Hello, ${this.nameTarget.value}!`
  }
}
```

# Hotwire/Stimulusとは？

引用：Hotwire Handbook

```
<!--HTML from anywhere-->
<div data-controller="hello">
  <input data-hello-target="name"
type="text">

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

  <span data-hello-target="output">
  </span>
</div>
```

```
// hello_controller.js
import { Controller } from "stimulus"

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

  greet() {
    this.outputTarget.textContent =
      `Hello, ${this.nameTarget.value}!`
  }
}
```

Harunaと入力してGreetボタンを押すと”Hello, Haruna!”が出力

# Hotwireへ置き換え

```
// app/javascript/controllers/video_recorder_controller.js

export default class extends Controller {
  static targets = ["video"]
  static values = { summaryId: Number }

}
```

```
 -# app/views/summaries/feedbacks/edit.html.haml
 %div{data: {controller: "video-recorder", video_recorder_summary_id_value: @summary.id}}
   %video{data: {video_recorder_target: "video"}, controls: true, autoplay: true, muted: true}
   %button{data: {action: "click->video-recorder#startRecording"}} Record
   %button{data: {action: "click->video-recorder#stopRecording"} Stop
   %button{data: {action: "click->video-recorder#clearVideo"} Clear
```

# Hotwireへ置き換え

実装途中なのに、なんかJavaScript
いっぱい書いてるぞ・・？

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

// Connects to data-controller="feedback-video"
export default class extends Controller {
  static targets = ["video", "recordButton", "stopButton", "clearButton"]
  static values = { summaryId: Number }

  async connect() {
    console.log(this.summaryIdValue)
    this.chunks = []
    const stream = await navigator.mediaDevices.getUserMedia({ video: true, audio: true })
    this.videoTarget.srcObject = stream
    this.videoTarget.play()
    this.mediaRecorder = new MediaRecorder(stream)
    this.mediaRecorder.ondataavailable = (event) => this.chunks.push(event.data)
  }

  startRecording() {
    try {
      console.log('start recording')
      this.mediaRecorder.start()
    } catch(error) {
      console.error('Error accessing media devices.', error)
    }
  }

  stopRecording() {
    console.log('stop recording')
    this.mediaRecorder.stop()
    this.mediaRecorder.onstop = this.handleStop.bind(this)
  }

  handleStop() {
    const blob = new Blob(this.chunks, { type: 'video/webm' })
    const formData = new FormData()
    formData.append(
      'summary_video[video]',
      blob,
      `summary_video_id${this.summaryIdValue}.webm`
    )
    const response = await fetch(`/api/summaries/${this.summaryIdValue}/summary_feedback_videos`, {
      method: 'POST',
      body: formData,
      headers: {
      'X-CSRF-Token': document.querySelector('meta[name="csrf-token"]').content
    }
    })
    if (response.ok) {
      await response.json()
      console.log('Video uploaded')
    } else {
      console.error('Failed to upload video', await response.text())
    }
  }

  clearVideo() {
    try {
      fetch(`/api/summaries/${this.summaryIdValue}/summary_feedback_videos`, {
        method: 'DELETE',
        headers: {
          'X-CSRF-Token': document.querySelector('meta[name="csrf-token"]').content
        }
      })
    } catch(error) {
      console.error('Error deleting video', error)
    }
  }
}
```

※中身は読まなくて大丈夫なので、
混沌とした感じを汲み取っていただけると嬉しいです。

# Stimulusを触ってみた所感

- 動作は実現できそう。
- Stimulus自体に書きづらさはない。
- でも、しっくりこない。

と、思いつつチームメンバーに途中経過を共有したところ・・・

ゴリゴリにJavaScript書くことになっちゃうね

これならReactで良いのでは？

# プロポーザル提出時はこう結論を出していた

- **まとめ：弊チームはReactとHotwireをどう使い分けていくのか**
  - 弊チームにはReact派とHotwire派のメンバーで構成されており絶賛議論中ですが、現状では以下の結論を出しています。

本当にこれで、良いのか・・？？？

# （再掲）Stimulusを触ってみた所感

- 動作は実現できそう。
- **Stimulus自体に書きづらさはない。→ ???**
- でも、しっくりこない。

と、思いつつチームメンバーに途中経過を共有したところ・・・

# 書きづらさがない、の正体は？

- Reactに関する知識（useStateなどのHooks）が必要ない
  - Hooksがよしなに処理していた箇所を自分で実装することに
    - 楽にフロントエンドを書きたかったはずなのに

    ➡Reactを使用するよりもJavaScriptのコードが増加

# 改善の余地を探る

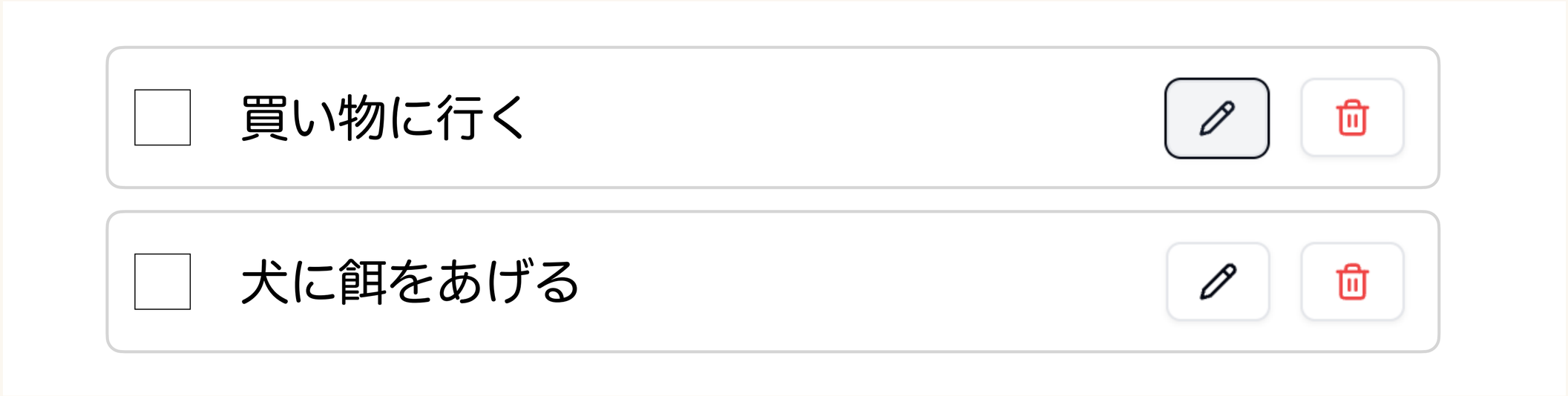
- Stimulusコントローラーに過度なJavaScriptのロジックが集中
  - それならばReactが良い
- しかし、別の箇所でTurboを使って実装したとき
  - Turboはチームで好評だった

# 実装方針を見直す

- 仮説１：Hotwireの旨みはTurboなのでは
  - Turboに寄せることで、Stimulusに書くJavaScriptが減るはず
- 仮説２：Stimulusのコントローラーを整理すると印象が違うのでは

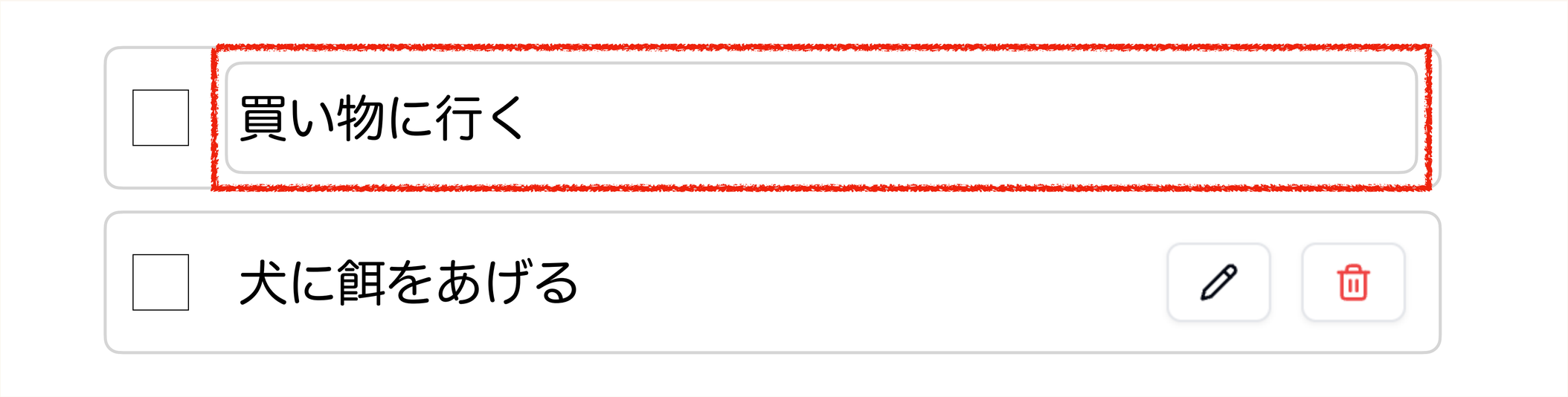
# Hotwire/Turboとは？

- CRUD操作を中心としたSPAライクな体験をJavaScriptを書かずに実現できるツール



# Hotwire/Turboとは？

- リクエストに対して、サーバーは指定した箇所のHTMLをレスポンス
  - 画面全体を再描画しないのでSPAライクな動作を実現できる



# 録画機能のどこでTurboを使える？

# Hotwire的な設計を追求して「Web紙芝居」に行き着いた話

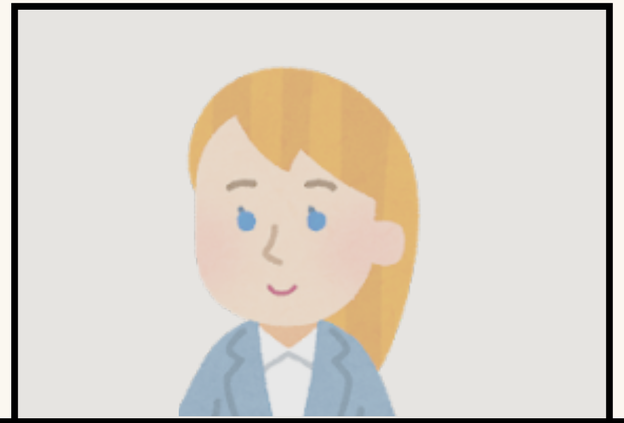
Yasuko Ohba (nay3)

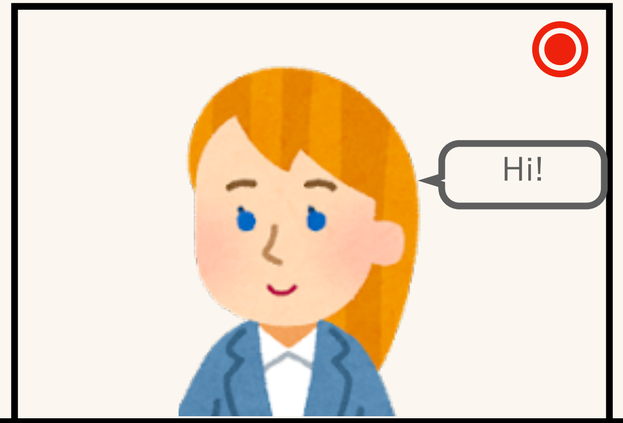
(※以下 Kaigi on Rails 2023 サイトから一部抜粋)

SPAとHotwireでは、アプリを構成するための物の考え方が大きく異なります。

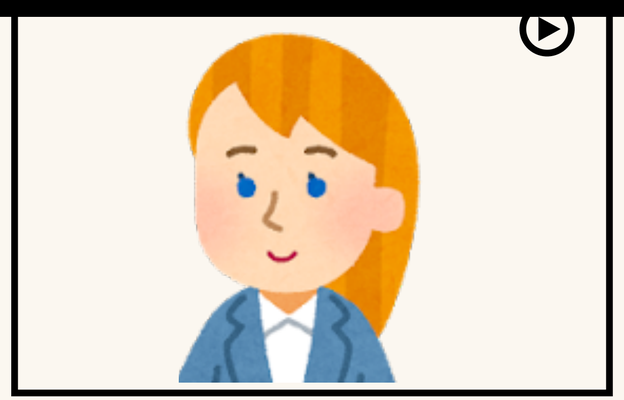
Hotwireを縦横無尽に使い倒すためには、Hotwire的な考え方で作る必要があるのです。

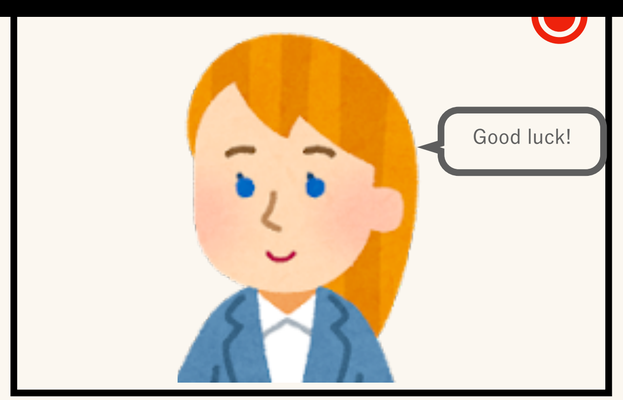
この、Hotwire的な考え方＝設計指針に私は「Web紙芝居」という名前をつけました。






# 画面の流れに沿って実装するのはダメだった・・





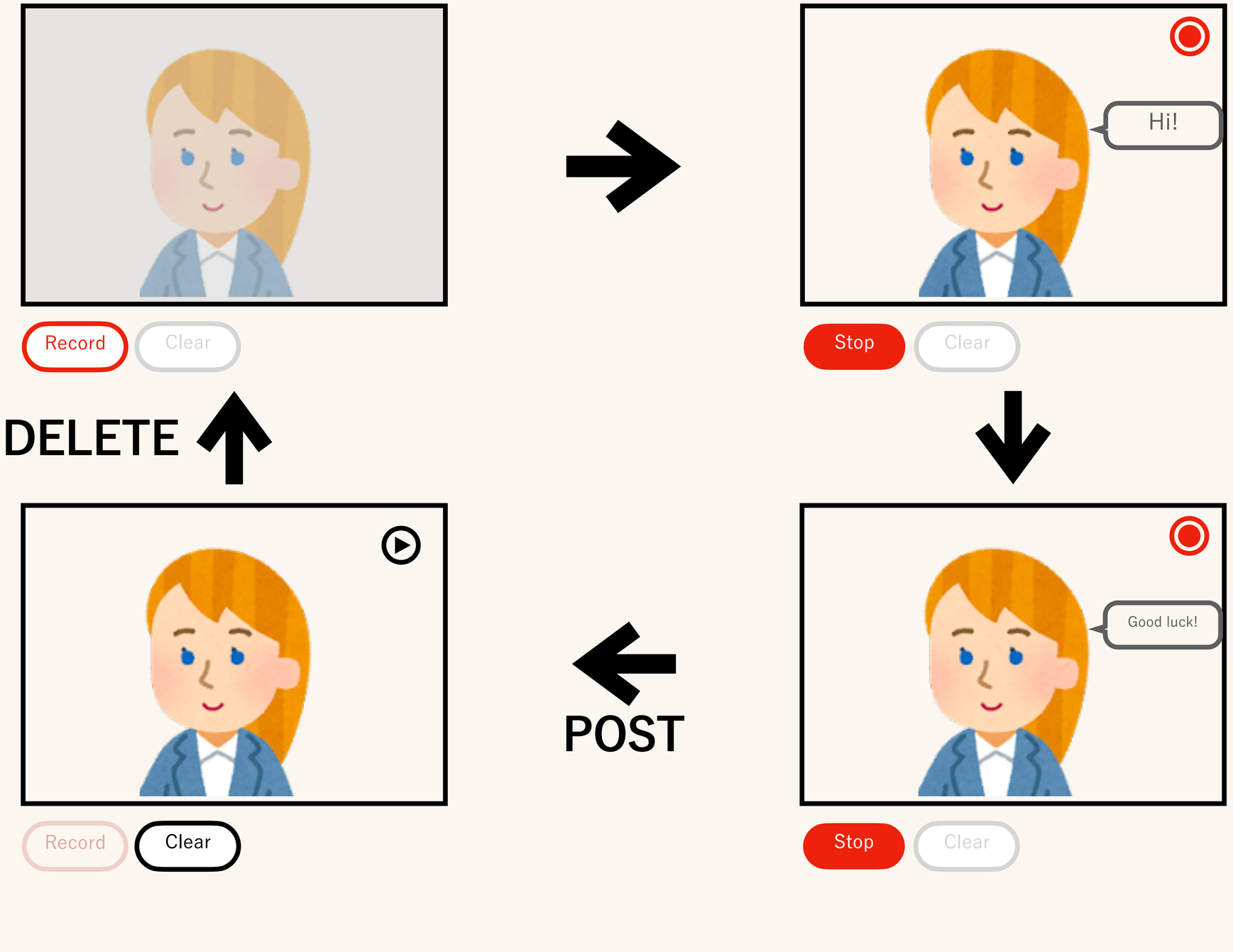

POST

Record Clear

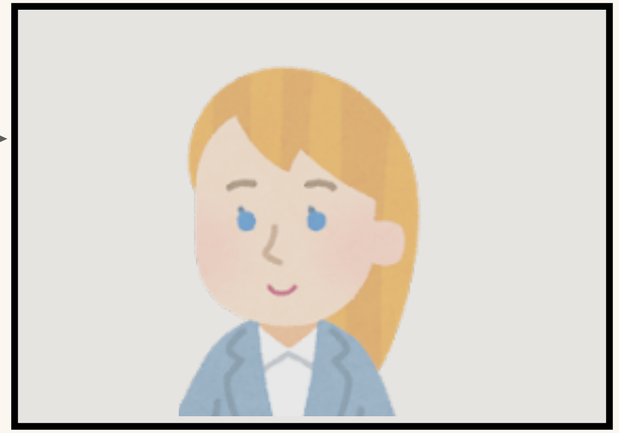
Stop Clear

# 考え方を変えてみる！

# （再掲）ボタンのクリックで状態が遷移する



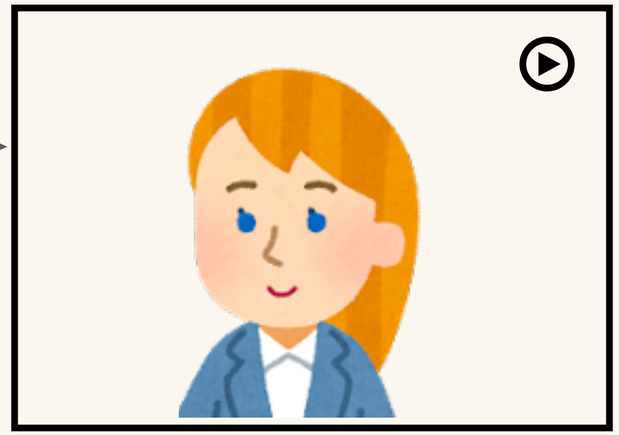
DBに動画
データなし



Record Clear

DELETE

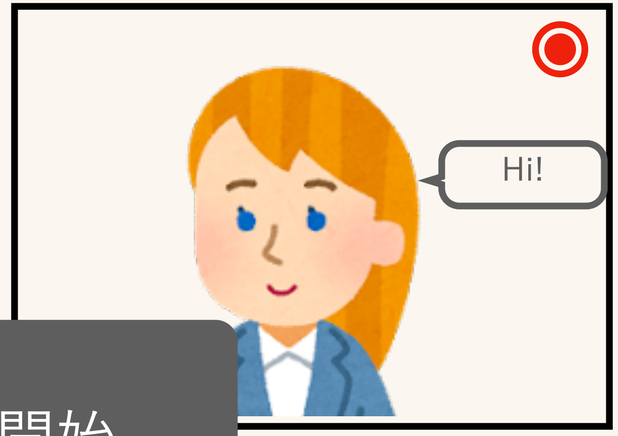
DBに動画
データあり



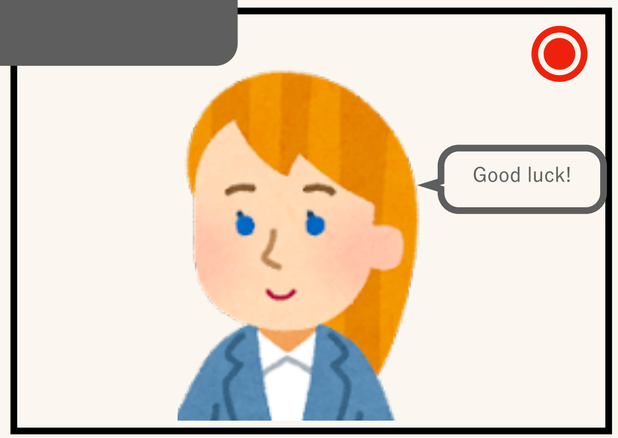
Record Clear

POST

録画開始
&
終了

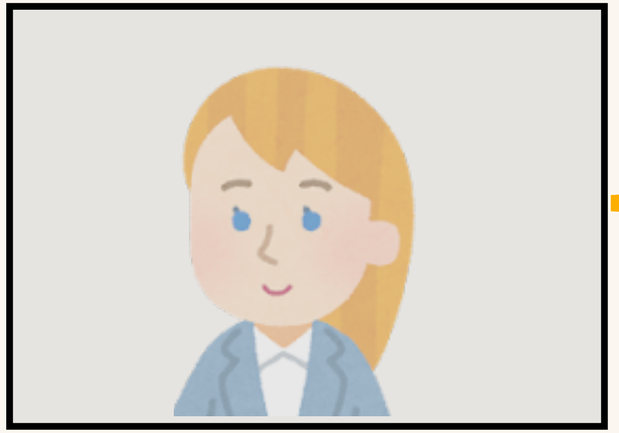



ear




Stop Clear

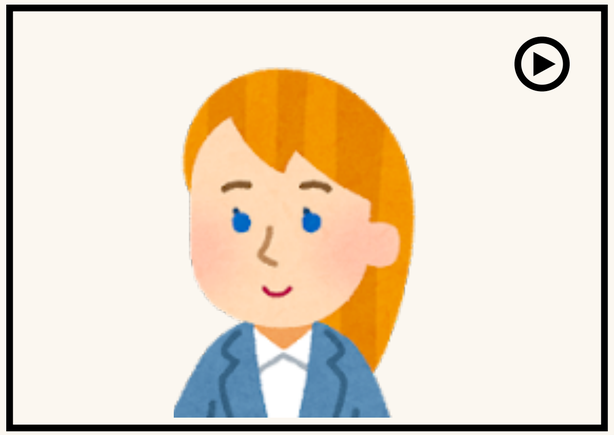
①動画データなし



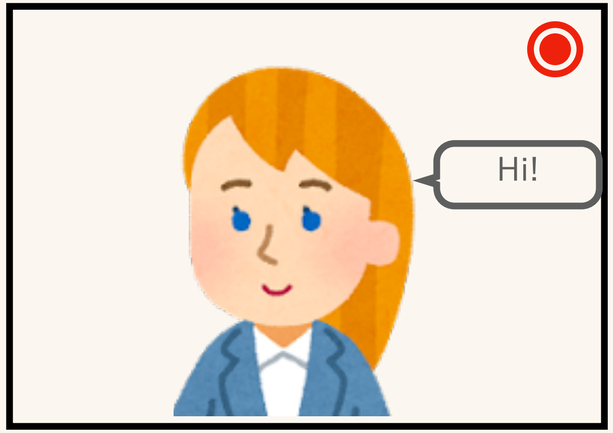
Record Clear

DELETE POST

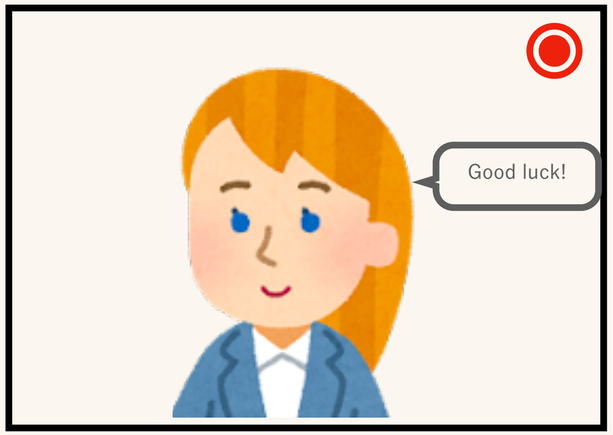
②動画データあり



Record Clear

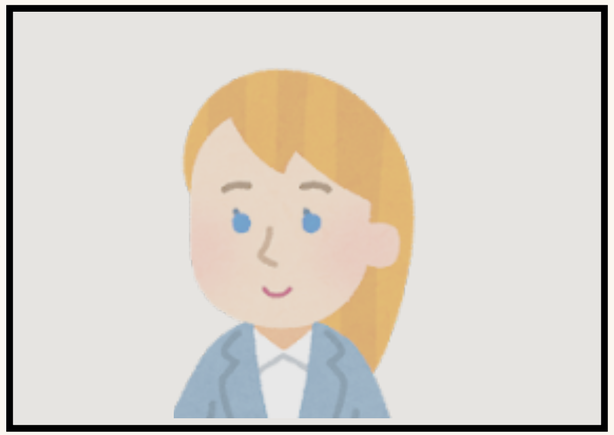



Stop Clear




Stop Clear

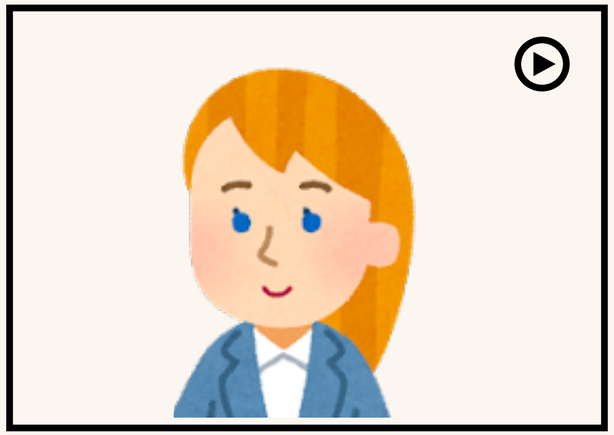
①動画データなし



Record Clear

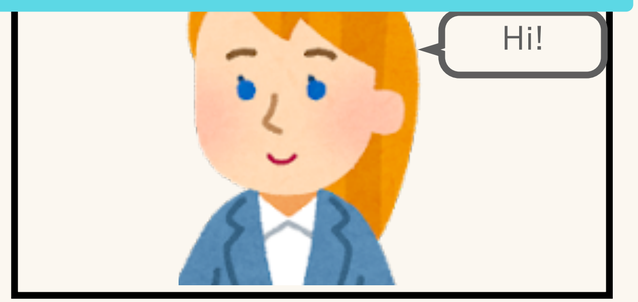
DELETE ↑↓ POST

②動画データあり

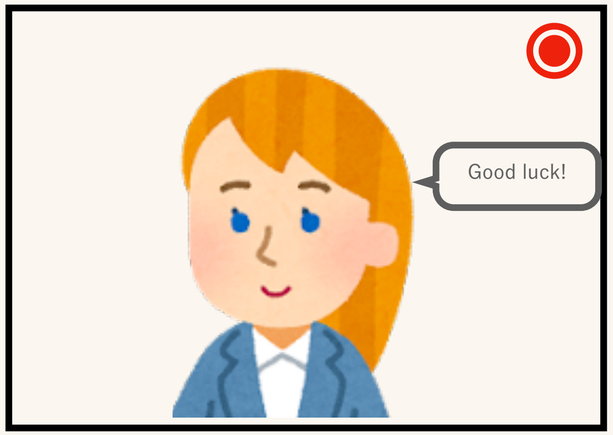


Record Clear

Turboを利用できそう



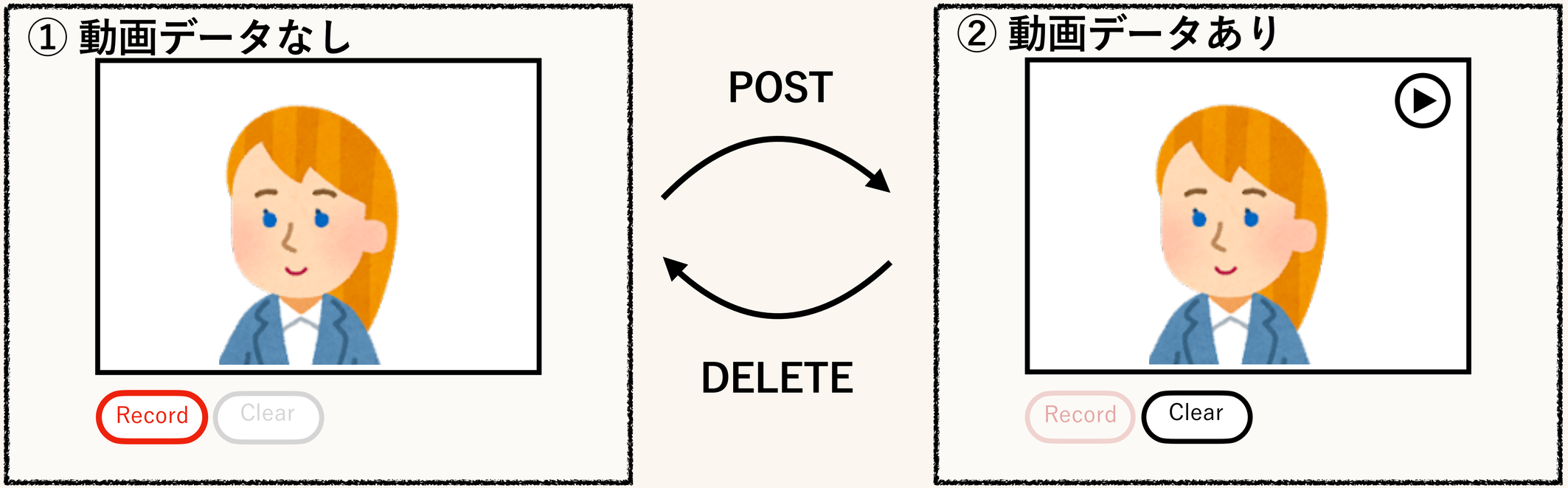

Stop Clear




Stop Clear

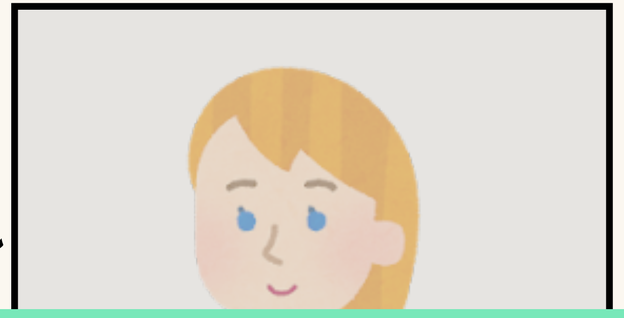
# 紙芝居として考えてみる

- CRUD操作がきっかけで発生する画面の切り替えはTurboに任せる




※『Web紙芝居』のトークは画面全体を1枚の紙芝居として捉えるお話なので、トークからヒントを得たうえで異なるアプローチをしています

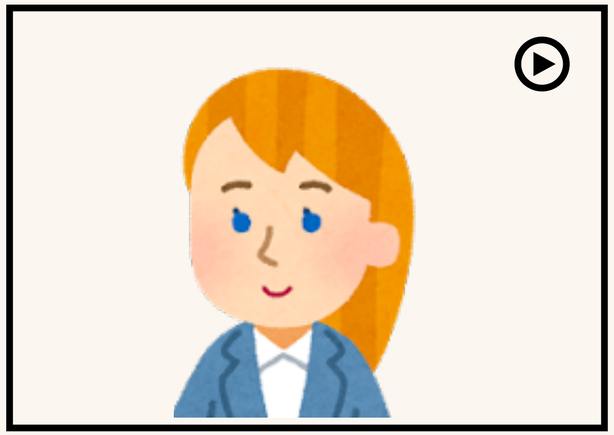
①動画データなし



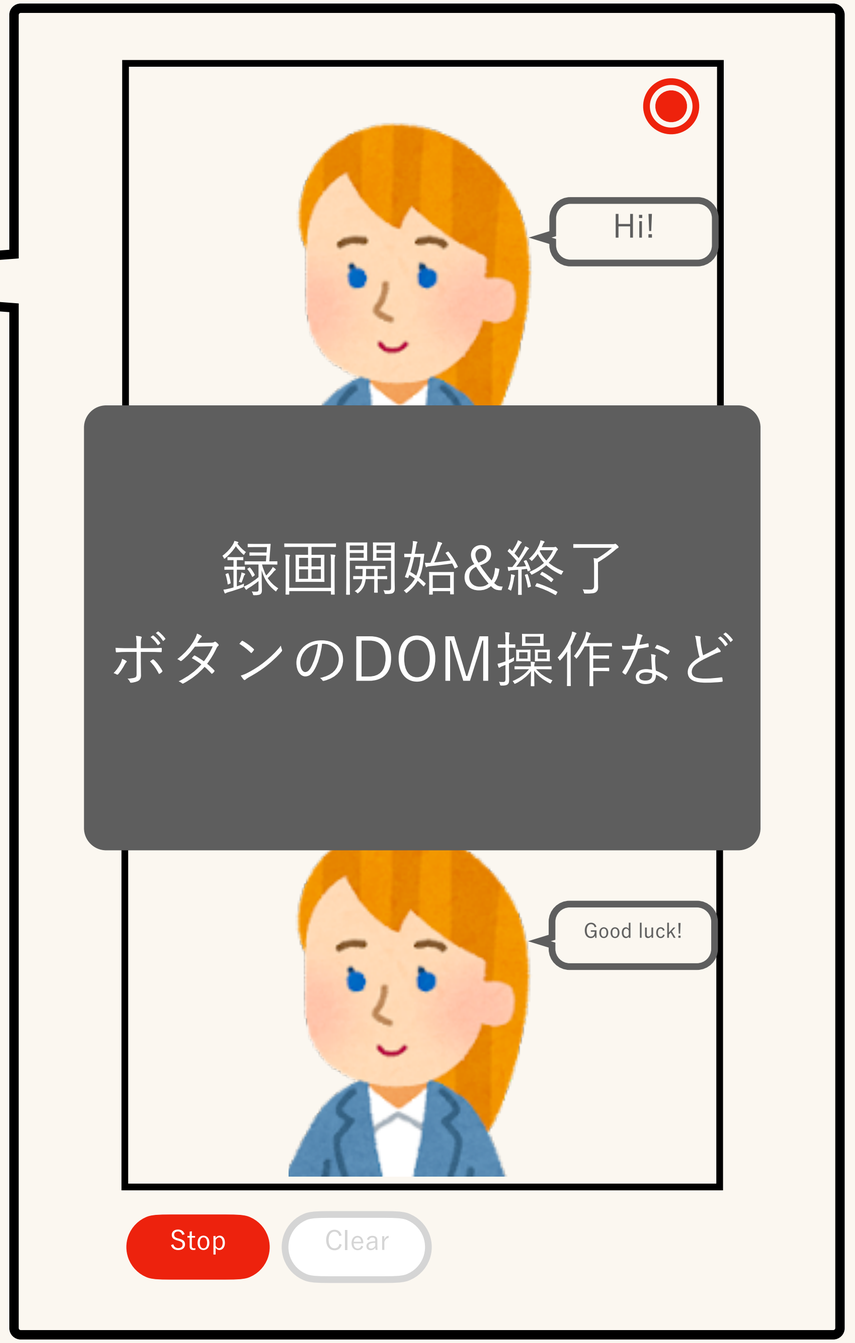
？？？

DELETE ↑↓ POST

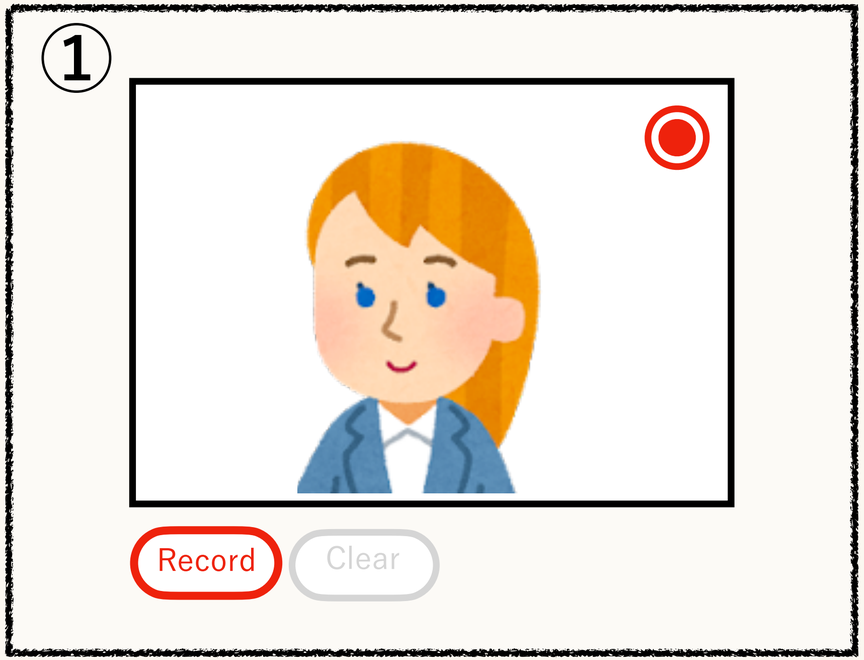
②動画データあり



Record Clear

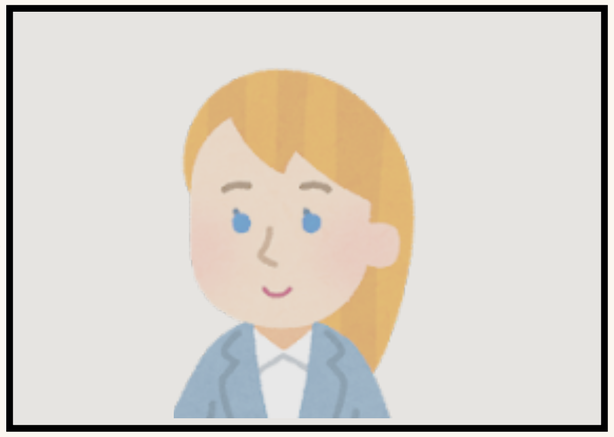


# 紙芝居として考えてみる②




- **“紙芝居の仕掛け”**と捉えてみる
  - Rails側で制御するのは難しい
  - Stimulusで実装する

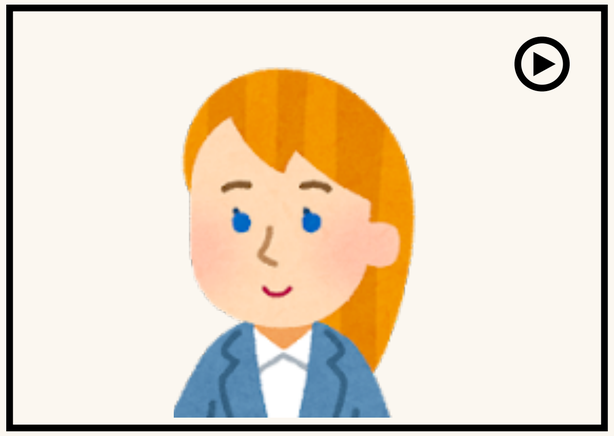
①動画データなし



Record Clear

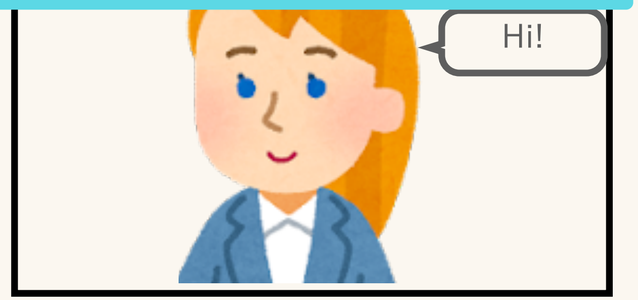
DELETE ↑↓ POST

②動画データあり

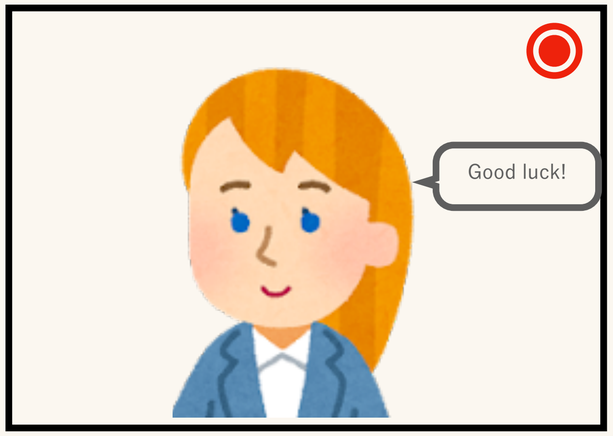


Record Clear

# Turboで実装




Stop Clear




Stop Clear

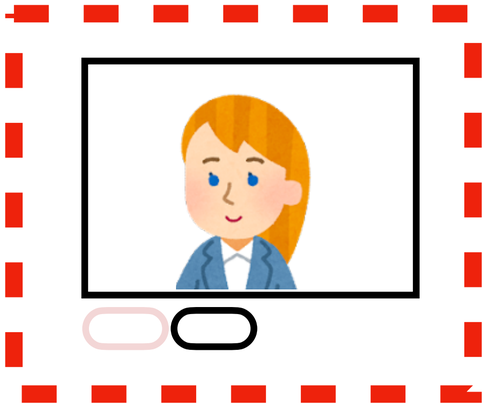
# Hotwireへ置き換え ~Turboを使う~

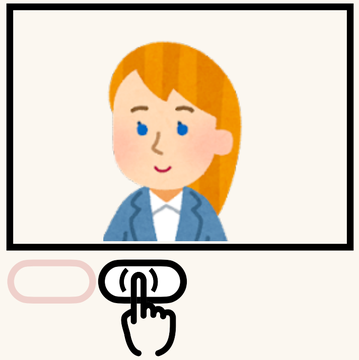
```
// 添削ページのview
= render partial: 'summaries/feedbacks/summary
```

```
// app/views/summaries/feedbacks/_summary_f

= turbo_frame_tag 'summary-feedback-video'
  %div{data: {controller: "video-recorder",
    .is-flex.is-flex-direction-column.is-al
      - if summary_feedback_video.present?
        // ビデオデータがある時の処理


      - else
        // ビデオデータがない時の処理
```

添削ページのview

↓ ‘summary-feedback-video’

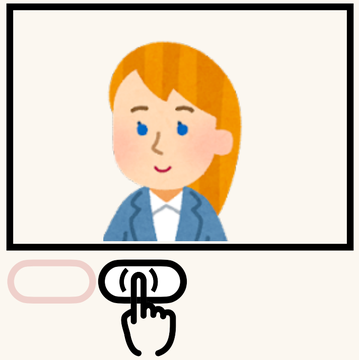




→ Turboリクエスト → Railsコントローラー

※スペースの関係で資料では一部省略している箇所があります

```
/       # app/controllers/summaries/summary_feedback_videos_controller.rb
        def destroy
=         @summary = Summary.find(params[:summary_id])
          @summary_feedback_video = @summary.summary_feedback_video
          @summary_feedback_video.destroy!
        end   ↑暗黙的にturbo_streamをrender

          %source{src: summary_feedback_video.video_url, type: "video/webm"}
         = link_to 'Clear', summary_summary_feedback_videos_path(summary),
data: { turbo_confirm: "本当に削除しますか？", turbo_method: :delete }

      - else
        // ビデオデータがない時の処理
```



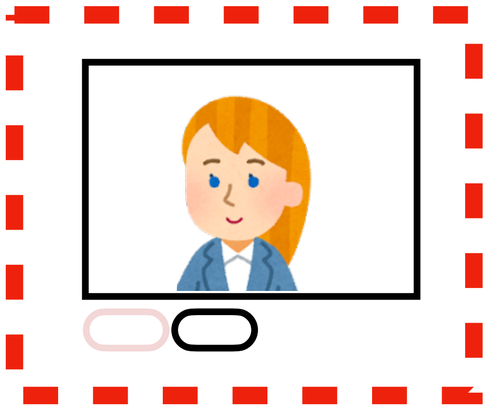
→ Turboリクエスト → Railsコントローラー → turbo_stream

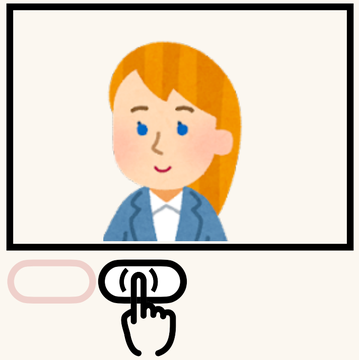
※スペースの関係で資料では改行を入れています

```
// app/views/summaries/summary_feedback_videos/destroy.turbo_stream.haml

= turbo_stream.replace 'summary-feedback-video',
  partial: 'summaries/feedbacks/summary_feedback_video',
  locals: { summary: @summary, summary_feedback_video: nil }
```

添削ページのview

↓ ‘summary-feedback-video’



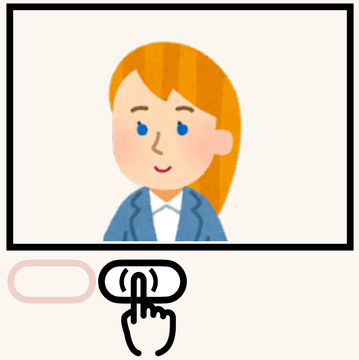


→ Turboリクエスト → Railsコントローラー → turbo_stream

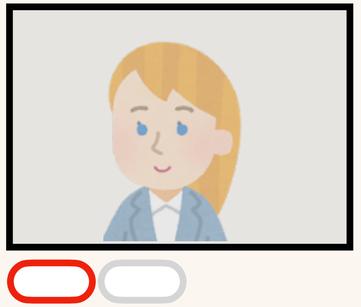
```
// app/views/summaries/summary_feedback_videos/destroy.turbo_stream.haml

= turbo_stream.replace 'summary-feedback-video',
  partial: 'summaries/feedbacks/summary_feedback_video',
  locals: { summary: @summary, summary_feedback_video: nil }
```

```
// app/views/summaries/feedbacks/_summary_feedback_video.html.haml

= turbo_frame_tag 'summary-feedback-video' do
  %div{data: {controller: "video-recorder", {// コントローラーへデータ受け渡し}}}

    .is-flex.is-flex-direction-column.is-align-items-start
      - if summary_feedback_video.present?
        // ビデオデータがある時の処理

      - else
        // ビデオデータがない時の処理
```

先ほどまではデータが
あったのでこちら

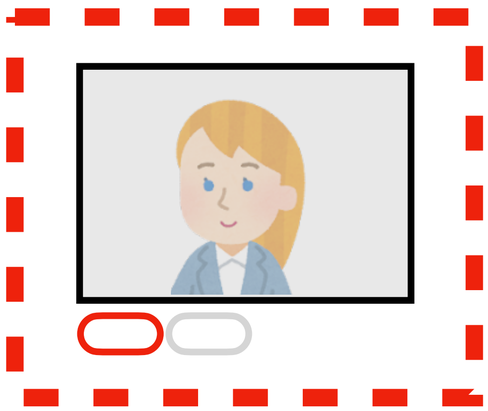


→ Turboリクエスト → Railsコントローラー → turbo_stream→



添削ページのview

↓ ‘summary-feedback-video’



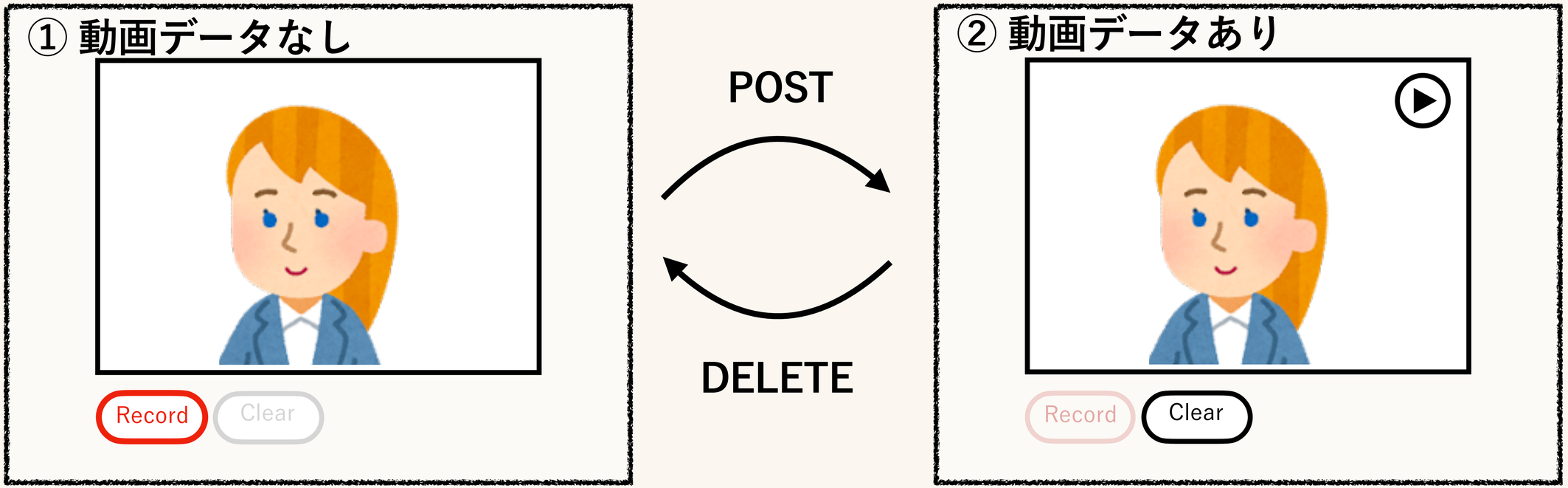
```
edback_videos/destroy.turbo_stream.haml

eedback-video',
ummary_feedback_video',
mmary_feedback_video: nil }
```

```
_summary_feedback_video.html.haml

ack-video' do
-recorder", {// コントローラーへデータ受け渡し}}}
```

```
.is-flex.is-flex-direction-column.is-align-items-start
  - if summary_feedback_video.present?
    // ビデオデータがある時の処理

  - else
    // ビデオデータがない時の処理
```

データが削除された
のでこちらが実行される

# （再掲）紙芝居の基礎が出来上がる



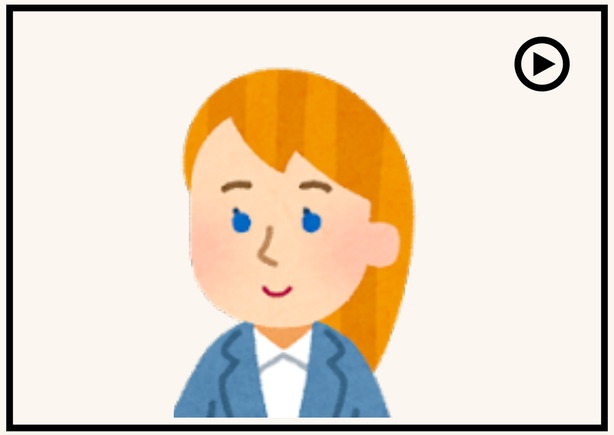
# Stimulusで実装

①動画データなし

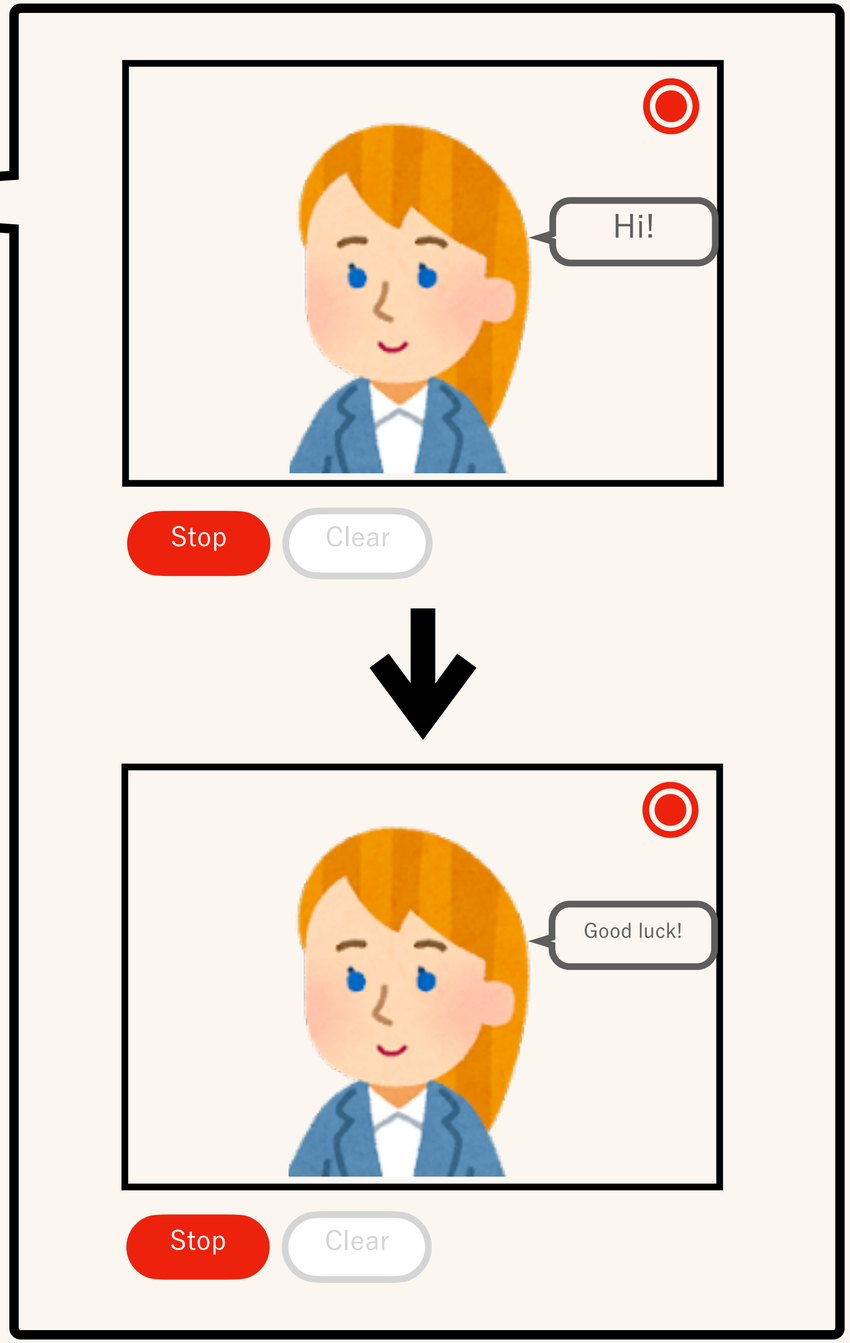
Record Clear

DELETE POST

②動画データあり



Record Clear



録画に関する処理はトークの本筋ではないので、

コードを整理した方法についてお話しします

# （再掲）実装方針を見直す

- 仮説１：Hotwireの旨みはTurboなのでは
  - Turboに寄せることで、Stimulusに書くJavaScriptが減るはず
- **仮説２：Stimulusのコントローラーを整理すると印象が違うのでは**
  - リファクタリング方法はまだ模索中ですが・・

```
// app/javascript/controllers/video_recorder_controller.js
import { Controller } from "@hotwired/stimulus"
import { // 利用する関数 } from “./video_recorder_utils"

export default class extends Controller {
  static targets = ["video", "recordButton"]
  static values = { writingId: Number, writingType: String, feedbackVideo: String,
uploadUrl: String }

  async connect() {
    // 接続時の処理
  }

  async startRecording() {
    // 録画開始の処理
  }

  stopRecording() {
    // 録画終了の処理
  }
}
```

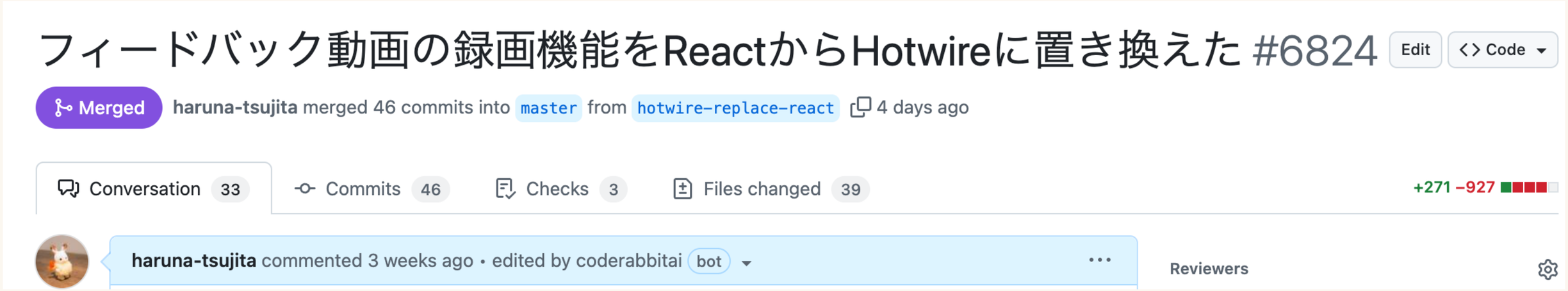
```
// app/javascript/controllers/video_recorder_controller.js
import { Controller } from "@hotwired/stimulus"
import { initializeMediaStream, setupMediaRecorder, setStopButton, uploadData } from "./video_recorder_utils"
// Connects to data-controller="video-recorder"
export default class extends Controller {
  static targets = ["video", "recordButton"]
  static values = { writingId: Number, writingType: String, feedbackVideo: String, uploadUrl: String }

  async connect() {
    if (!this.feedbackVideoValue) {
      this.mediaStream = await initializeMediaStream()
    }
  }

  async startRecording() {
    if (!this.mediaStream) {
      this.mediaStream = await initializeMediaStream()
      if (!this.mediaStream) return
    }
    this.mediaRecorder = setupMediaRecorder(this.mediaStream)
    this.mediaRecorder.start()
    setStopButton(this.recordButtonTarget)
    this.mediaRecorder.ondataavailable = (event) => uploadData(event, this.writingTypeValue,
this.writingIdValue, this.uploadUrlValue)
  }

  stopRecording() {
    this.mediaRecorder?.stop()
    this.mediaStream.getTracks().forEach(track => track.stop())
    this.recordButtonTarget.disabled = true
  }
}
```

**Total: 37行**

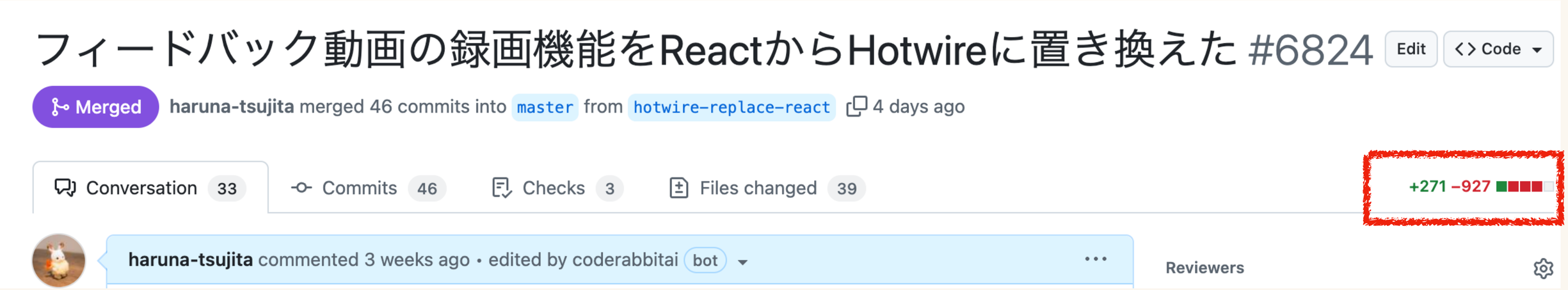
これで置き換えは完了です！

議論の結果…

# PRはマージされました🎉



# PRはマージされました🎉

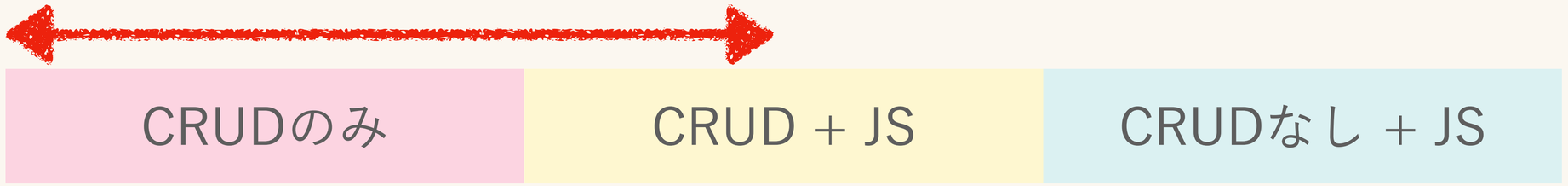



※差分 +217 -971 ですが、添削画面の録画機能**2箇所**の置き換えをおこない、
React側で共通化されていなかった箇所が今回共通化されての結果なので
1箇所の場合、減少量は1/2程度に収まります

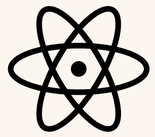
# Hotwire or React ？

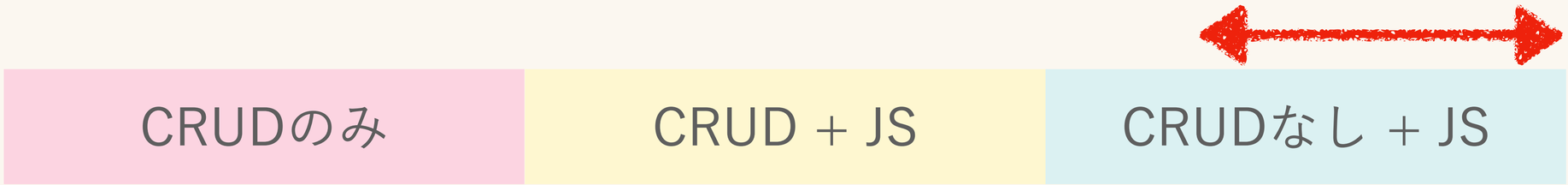
# Hotwire or React？

- CRUD操作が主体 + 補助的なJavaScriptが必要な画面
  - 紙芝居 + ちょっとした仕掛けがある場合
    - **Hotwireで十分対応できる、積極的に使っていきたい**



# Hotwire or React？

- CRUD操作を伴わずリッチなインタラクションが必要な画面
  - ex. 複雑なステート管理が必要
    - 私の担当プロダクトには該当しない
  - Reactの得意分野 



# Hotwire or React？

- CRUDを含む高度なSPA実装が必要な画面
- CRUDなしでシンプルなSPA実装が必要な画面
  - 担当プロダクトではテキストエディタが該当
  - このあたりは意見が分かれる

# Hotwire or React？

- CRUDなしでSPA実装が必要な画面
  - ＝ 多くはユーザーが長時間滞在するページ
  - 1枚の紙芝居の中に仕掛けがたくさんある場所

CRUDのみ | CRUD + JS | CRUDなし + JS

# Hotwire or React？

CRUDなしの
SPA実装をする場合

✦ React

- Hooksやエコシステムの真価を発揮し始める
  - ▸ しかしReactに対する深い理解が必要

✦ Hotwire

- サーバーサイドに実装を寄せられて扱いやすい
  - ▸ 実装の工夫・Hotwire的設計

    ➡実装をCRUDに落とし込む必要がある

CRUDのみ | CRUD + JS | CRUDなし + JS

# Hotwire or React？

- ReactとHotwire、どちらも一長一短あり、選択するのは難しい
- ただし、『Hotwireで実装できそうか？』という視点だと
  - **実装をCRUDに集約できるかどうか**、が大きなポイント

# まとめ

✦ Hotwire的な設計ができれば、適用可能な領域は想像より広い

✦ Hotwireを使ってみようかな、と思う方が増えたら嬉しいです！

# ご清聴ありがとうございました！

# 参考

- Hotwire Handbook https://hotwired.dev/
- Turbo Handbook（日本語訳）https://everyleaf.github.io/hotwire_ja/turbo/handbook/introduction/
- 猫でもわかるHotwire入門 Turbo編 https://zenn.dev/shita1112/books/cat-hotwire-turbo
- Hotwire的な設計を追求して「Web紙芝居」に行き着いた話 https://kaigionrails.org/2023/talks/nay3/