Hotwire & Turbo in Rails 8

Progetto di Sistemi WEB

Anno Accademico 2025/2026

Filippo Poltronieri

filippo.poltronieri@unife.it

Hotwire (HTML Over The Wire)

- **Hotwire** è un framework per costruire applicazioni web **reattive** e **veloci** usando principalmente **HTML**.
- In Rails 8, Hotwire è integrato di default.
- Utilizzando Hotwire, è possibile creare esperienze utente simili a quelle delle Single Page Applications (SPA) senza dover scrivere molto JavaScript.
- La documentazione ufficiale di Hotwire è disponibile su: hotwired.dev
- Vediamo come includere Hotwire per migliorare l'interattività delle nostre applicazioni Rails.
- Hotwire è composto da tre componenti principali:
    i. **Turbo**: Gestisce la navigazione e l'aggiornamento della pagina.
    ii. **Stimulus**: Un framework JavaScript leggero per l'interattività client-side.
    iii. **Strada**: (meno comune) per la gestione delle WebSocket.

Turbo

Lo scopo di **Turbo** è di trasformare ogni pagina HTML in un **Single Page Application (SPA) senza dover scrivere JavaScript**!

Funzionalità:

1. **Navigazione istantanea** (no ricaricamenti)
2. **Form dinamici** (submit senza refresh)
3. **Aggiornamenti parziali** (turbo_frame, turbo_stream)
4. **Stato del browser preservato** (back/forward)

Turbo Drive

- Drive permette di realizzare una navigazione simil Single Page Web Application, tramite l'inserimento di attributi specifici nei link e form HTML.
- Se il click a un link normale `<a href=/page>` richiede **normalmente** il caricamento di un'intera pagina HTML, con Turbo Drive il browser **intercetta** la richiesta e carica solo il contenuto necessario, modificando il `<body>` o parte di esso.
- Questo avviene tramite richieste AJAX che aggiornano dinamicamente il DOM della pagina corrente, senza ricaricare l'intera pagina.
- Sfrutta lo storico del browser `history.pushState` per mantenere la navigazione fluida.
- **Drive** potrebbe essere uno strumento per migliorare significativamente la velocità di navigazione nelle applicazioni Rails.

Turbo Drive

- Turbo Drive realizza questa navigazione istantanea iniettando del codice Javascript nella pagina.
- Il codice JavaScript generato da Turbo è trasparente durante la scrittura dell'applicazione Rails, in quanto è gestita dal framework stesso.
- Per abilitare la navigazione veloce nei link, possiamo utilizzare l'attributo `data-turbo="true"` (**è abilitato di default**).
- Per disabilitare Turbo Drive su un link specifico, possiamo utilizzare `data-turbo="false"`.

Turbo Frames

- Frames permette di creare degli *aggiornamenti parziali* della pagina, in un modo simile agli iframe (**vecchi dinosauri che non avrete mai visto**), ma molto più leggeri e integrati con Turbo.
- Un frame è una porzione della pagina che può essere aggiornata indipendentemente dal resto della pagina.
- Si definisce un frame con il tag `turbo_frame_tag`, per permettere l'aggiornamento dinamico di quella sezione.
- **Quando un link o un form viene inviato all'interno di un frame, solo quel frame viene aggiornato**.

Esempio: Contatore Dinamico

View (`index.html.erb`):

```
<%= turbo_frame_tag "counter" do %>
  <div id="counter">
    <%= @count %>
  </div>
  <%= link_to "++", increment_path %>
<% end %>
```

- Nell'esempio il tag `turbo_frame_tag` crea un frame con ID "counter".
- Quando si clicca sul link "++", viene inviata una richiesta al percorso `increment_path` .

Esempio: Contatore Dinamico

- Il server risponde con un aggiornamento solo per il frame "counter", senza ricaricare l'intera pagina.

Controller:

```
def increment
  @count = params[:count].to_i + 1
  render turbo_stream: turbo_stream.replace("counter", partial: "counter")
end
```

Turbo Streams

- **Turbo Streams** permette di inviare **aggiornamenti in tempo reale** al client, modificando il DOM senza ricaricare la pagina.
- Questo è particolarmente utile per funzionalità come chat, notifiche, o liste dinamiche.
- Si utilizzano i **turbo_stream actions** per specificare come modificare il DOM (**append, prepend, replace, update, remove**).
- I messaggi Turbo Stream possono essere inviati come risposta a richieste HTTP o tramite WebSocket.
- Le Websocket sono gestite tramite Action Cable in Rails e sono lo strumento più utilizzato per inviare Turbo Streams in tempo reale.

Turbo Streams: Azioni Principali

- Vediamo quali sono le azioni principali di Turbo Streams:

```
turbo_stream.append   "id", partial: "item"
turbo_stream.prepend  "id", partial: "item"
turbo_stream.replace  "id", partial: "new_item"
turbo_stream.update   "id", partial: "updated"
turbo_stream.remove   "id"
```

- Possiamo chiamare turbo_stream direttamente nel controller per inviare aggiornamenti al client.

```ruby
# Controller
def create
  @message = Message.create!(message_params)
  render turbo_stream: turbo_stream.append("messages", partial: @message)
end
```

Turbo Stream - Real Time Updates

- Per utilizzare Stream dobbiamo prestare attenzione al design delle nostre view.
- In particolare dobbiamo assegnare ai `div` che vogliamo aggiornare un `id` univoco, che permetta di identificare univocamente un elemento particolare.
- Nelle view generate da Rails, era presente la funziona `dom_id` che genera un id univoco per ogni risorsa

Turbo Streams - Real Time Updates

- All'interno della view possiamo utilizzare il tag `turbo_stream_from` per iscrivere un frame a un canale di Action Cable.
- Esempio di iscrizione a un canale `messages`

```
<%= turbo_stream_from "messages" %>
```

- In questo modo, ogni volta che un messaggio viene trasmesso sul canale "messages", il client riceverà un aggiornamento Turbo Stream.
- Tutto questo viene gestito tramite socket.

Turbo Streams - Real Time Updates

- Per inviare un messaggio Turbo Stream dal server, possiamo utilizzare il metodo `broadcast` di Action Cable.
- Esempio di trasmissione di un messaggio Turbo Stream:

```
        turbo_stream: turbo_stream.append("messages", partial: "message", locals: { message: @message })
```

Turbo Streams - Real Time Updates

- In questo esempio, stiamo trasmettendo un messaggio che aggiunge un nuovo messaggio alla lista dei messaggi.
- Il client che è iscritto al canale "messages" riceverà questo aggiornamento e aggiornerà il DOM di conseguenza.
- Possiamo anche utilizzare i metodi di trasmissione direttamente nei modelli, per esempio dopo la creazione di un nuovo record.

```
after_create_commit do
        broadcast_append_to "messages", partial: "message", locals: { message: self }
end
```

Turbo Streams - Real Time Updates

- Oppure si può utilizzare `broadcasts_to` nel modello per associare automaticamente le trasmissioni a un canale specifico.

```
class Message < ApplicationRecord
        broadcasts_to ->(message) { "messages" }
end
```

- In questo modo, ogni volta che un messaggio viene creato, aggiornato o eliminato, verrà automaticamente trasmesso al canale `messages`.
- Questo semplifica notevolmente la gestione delle trasmissioni Turbo Stream nelle applicazioni Rails.

Turbo Streams - Real Time Updates

- Nel caso di dipendenze tra modelli, possiamo utilizzare `broadcasts_to` con associazioni.

```
class Comment < ApplicationRecord
        belongs_to :post
        broadcasts_to :post
end
```

- Vediamo ora un esempio pratico di utilizzo di Turbo Streams in un'applicazione Rails.

Blog - Turbo Streams

- Nel caso dell'applicazione di Blog vista a lezione, possiamo utilizzare Turbo Streams per aggiornare la lista dei **commenti** in tempo reale quando un nuovo commento viene creato.
- L'aggiornamento automatico tramite Turbo Stream ci consente di creare una visualizzazione dinamica dei contenuti, senza dover ricaricare la pagina web.
- Quando un utente crea un nuovo commento, il server invia un messaggio Turbo Stream a tutti i client connessi, che aggiorna automaticamente la lista dei commenti visualizzati.

Blog - Turbo Streams

- Vediamo come modificare la nostra applicazione di blog per utilizzare Turbo Streams.
- Per esempio, nella vista `posts/show.html.erb`, aggiungiamo il tag `turbo_stream_from` per iscrivere il frame dei commenti al canale `comments`.

```
<%= turbo_stream_from "comments" %>
<div id="comments">
  <%= render @post.comments %>
</div>
```

Blog - Turbo Streams

- L'ultima cosa che ci rimane da modificare è il model di Comment per trasmettere automaticamente i nuovi commenti al canale `comments`.

```
class Comment < ApplicationRecord
  belongs_to :post
  broadcasts_to :post
end
```

- In questo modo, ogni volta che un nuovo commento viene creato, verrà automaticamente trasmesso al canale `comments`, e tutti i client connessi riceveranno l'aggiornamento in tempo reale.
- Vediamolo assieme!