# Delightful Multiplayer Editing with Phoenix

Tyler A. Young
Felt

# Hi, I’m Tyler

Recovering C++ developer

Lots of soft-real time work

Built an MMO server for the X-Plane flight sim

Now work at Felt on “the Figma for maps”

ElixirConf®


Felt

# Currently the Boat (Great Loop Trip)

Share

Maggie and Bruce Tate (@currently_the_boat) •...
instagram.com

Home | Currently The Boat
currentlytheboat.com

St Ours Lock, Day 151


Saugerties Marina, Days 140-142

Half Moon Bay Marina, Days 133-139

Calvert Marina, Days 109-110


Deltaville Town Dock, Days 107, 108

Tidewater Yacht Marina, Day 105

Dismal Swamp, Day 104

Aligator River Marina, Day 103

Bell Harbor Marina, Days 97-102

Midway Marina, Day 7


Columbus Marina, Day 8

Lower Cooks Bend Anchorage, Day 9

King Fisher Bay Marina, Days 10, 11

Bobby's Fish Camp, Day 12

Tensaw River Anchorage, Days 13, 14


Ranger Tug R-29

Jekyll Island Harbor, Day 78

Hilton Head Island Beach Harbor Marina, Days 77-79

Apalachicola Marina, Days 24-27


Titusville Marina, Day 63

Clearwater Beach Marina, Days 29-32


https://felt.com/map/Currently-N7bZNUKISn9BC9BviQktUrGA


Currently the Boat (Great Loop T...
Share


Currently the Boat (Great Loop T ...
Share
Maggie and Bruce Tate
(@currently the boat)
Home | Currently The Boat

# So you want a web app people will love to use.

- Low latency
- Low waiting
- High reliability
- Gets out of your way


# So you want a web app people will love to use.

- Low latency
- Low waiting
- High reliability
- Gets out of your way

There's some inherent tension between these!


# So you want a web app people will love to use.

1. Use Channels (LiveView?)
2. …
3. ???


# Channels are great!

- Persistent, stateful connection
- Low latency (one handshake rather than one per message)
- Bidirectional (no need for polling)

…but they don’t solve everything.

# Minimizing time to first (useful) render

Goal: Get to a useful state even before the WebSocket is open

Include on the page initial JSON that would have been a separate request


```
<div id="app-data" style="display: none">
  <%= Jason.encode!(%{
    user: %{name: @user_name, id: @user_id},
    title: @title,
    posts: render(
      AppServerWeb.PostView,
      "index.json",
      posts: @posts
    )
  }) %>
</div>
<script>
  const el = document.getElementById("app-data");
  window.appState = JSON.parse(el.textContent);
</script>
```

```
<div id="app-data" style="display: none">
  <%= Jason.encode!(%{
    user: %{name: @user_name, id: @user_id},
    title: @title,
    posts: render(
      AppServerWeb.PostView,
      "index.json",
      posts: @posts
    )
  }) %>
</div>
<script>
  const el = document.getElementById("app-data");
  window.appState = JSON.parse(el.textContent);
</script>
```

Use Phoenix's HTML escaping to prevent XSS vulnerabilities (OWASP Cheat Sheet)

# Minimizing time to first (useful) render

Cache this initial JSON for popular pages?

Getting Slashdotted could result in ~no database load!


# Reduce latency: Optimistic local edits

When a client makes a change, the frontend assumes it will be successful.

(That's true 99.?% of the time.)

Does have to handle the possibility that the server response will correct that assumption.


# Reduce latency: Optimistic broadcasts

Old workflow:

1. Client sends an edit to the server
2. Write it to the database
3. Broadcast the change to connected clients


# Reduce latency: Optimistic broadcasts

New

~~Old~~ workflow:

1. Client sends an edit to the server
2. Broadcast the change to connected clients
3. Write it to the database

# Reduce latency: Optimistic broadcasts

New

~~Old~~ workflow:

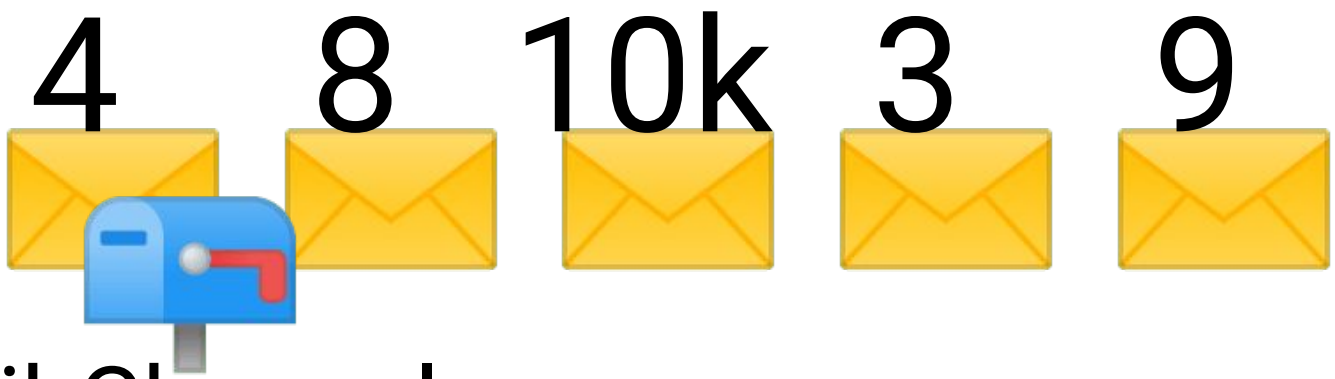
1. Client sends an edit to the server
2. Broadcast the change to connected clients
3. Write it to the database
4. (Maybe) broadcast any discrepancies

# Don’t block the Channel process

All messages on a given process get handled sequentially

Long-running tasks (say, longer than 500 ms) make the server unresponsive


# Don’t block the Channel process




FibChannel
mailbox

# Don’t block the Channel process


FibChannel
mailbox

# Don’t block the Channel process

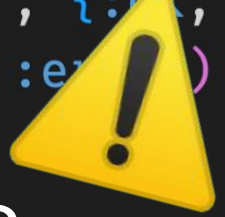

FibChannel
mailbox



ElixirConf®

```elixir
def handle_in("nth_fib", %{"n" => n}, socket) do
  ref = socket_ref(socket)

  Task.Supervisor.async_nolink(
    MyAppWeb.ChannelTaskSupervisor,
    fn ->
      case calculate_fibonacci_number(n) do
        {:ok, result} -> reply(ref, {:ok, %{fib: result}})
        {:error, _} -> reply(ref, :error)
      end
    end
  )

  {:noreply, socket}
end
```

```elixir
def handle_in("nth_fib", %{"n" => n}, socket) do
  ref = socket_ref(socket)

  Task.Supervisor.async_nolink(
    MyAppWeb.ChannelTaskSupervisor,
    fn ->
      case calculate_fibonacci_number(n) do
        {:ok, result} -> reply(ref, {:ok, %{fib: result}})
        {:error, _} -> reply(ref, :error)
      end
    end
  )

  {:noreply, socket}
end
```



Breaks sequential ordering guarantees!

# Don’t block the Channel process

Not always obvious!

Attach an event handler[†] to the `[:phoenix, :channel_handled_in]` event to log these in production




† https://gist.github.com/s3cur3/8a5fe8fc99eaa34dac985d98b3e60e78

```elixir
defmodule AppServerWeb.Telemetry do
  require Logger

  @doc """
  Attaches an event handler to the "handled in" event
  on your Phoenix Channels.
  Call from within your application.ex initialization.
  """
  def attach_telemetry_handlers() do
    channel_handled_in = [:phoenix, :channel_handled_in]

    :telemetry.attach(
      {__MODULE__, channel_handled_in},
      channel_handled_in,
      &__MODULE__.log_slow_handled_in/4,
      :ok
    )
  end
```

https://gist.github.com/s3cur3/8a5fe8fc99eaa34dac985d98b3e60e78

```elixir

  def log_slow_handled_in(_, %{duration: duration}, metadata, _) do
    duration_ms = System.convert_time_unit(duration, :native, :millisecond)
    max_duration_ms = 500

    if duration_ms > max_duration_ms do
      %{socket: socket, event: event, params: params} = metadata

      Logger.info("Slow channel event on #{socket.topic}: #{inspect(params)}")
    end
  end
end

```

https://gist.github.com/s3cur3/8a5fe8fc99eaa34dac985d98b3e60e78

# Detect when you've gone offline

Built-in detection of WebSocket disconnect isn't enough

- 30-60 seconds before the browser WebSocket API fires the disconnect event
- What happens if your *Channel* stops responding but the socket is still alive?


# Detect when you've gone offline

Easy step 1: watch `navigator.onLine`

→ Super fast for detecting when the client goes entirely offline


```
function ConnectionStatusAlertController() {
    const [isOnline, setIsOnline] = React.useState(navigator.onLine);

    React.useEffect(() => {
        window.addEventListener("online", () => setIsOnline(true));
        window.addEventListener("offline", () => setIsOnline(false));
    })

    if (isOnline) return null;
    return <OfflineAlert />;
}
```

# More heartbeats in more places

Channel-specific heartbeats let us very quickly detect when our messages aren't going through

- Time out if we don't hear back one way or another
- Easily detect when the initial connection negotiation has hung


```elixir
defmodule MyAppWeb.MyChannel do
  use MyAppWeb, :channel

  def handle_in("heartbeat", _, socket) do
    now =
      DateTime.to_unix(
        DateTime.utc_now(),
        :millisecond
      )

    payload = %{server_time_ms: now}
    {:reply, {:ok, payload}, socket}
  end
end
```

# Cache edits locally

If you go offline while edits are in flight, what do we do?

Is it safe to apply a particular edit later?


# Cache edits locally

`localStorage` for easily reconcilable changes (like adding a new element)

- Try to flush the next time we load the page
- Handle the case where the change went through, but the client hadn’t heard about it (idempotence)

CRDTs for other stuff?


# Be resilient to lost messages

Connection issues, browser refreshes, deploys, etc. could all cause a message to be lost.

How can we be more permissive in what we accept?

Do we really need the “create” message before an “update”? (Most things can be an upsert)


# Future work

Multi-node for seamless deploys

CRDTs?

Fully offline editing?

- What if you never come back online? 😱
- How do we handle missing tile data?

Global distribution via Fly.io?

Catchup mechanism?



# Summary


**Less waiting**

- Minimize time to first render
- Cache popular maps

**Less latency**

- Optimistic editing
- Optimistic broadcasts from the server
- Don't block the Channel process

**Less data loss**

- Detect problems with the WebSocket early
- Prevent or back up edits we can't save
- Make the protocol more resilient to dropped messages



ElixirConf®