Adding a Twitter Timeline to elixirmontreal.ca, the Elixir way

Montreal Elixir • 2017-10-11

Goals

- See Elixir and OTP in action
- Consume external API
- Deal with a Stream
- Some tips along the way

Assumptions

You are familiar with:

- Elixir
- GenServers
- Supervisors
- Streams
- Web sockets

Plan

- Part 1: OTP only
- Break
- Part 2: Web integration

Requirements

About Montreal Elixir

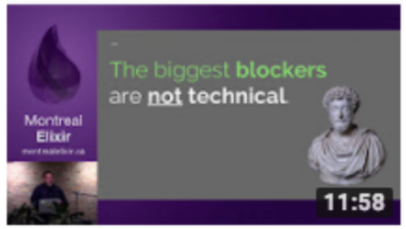
Montreal's community for Elixir, a dynamic, functional language designed for building scalable and maintainable applications.

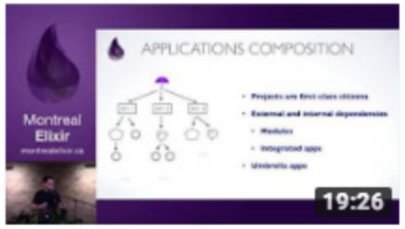
Recent updates (via Twitter)

- Would you like to share something you have recently learned about #elixirlang? Present at our next meetup, June 14
  May 24
- All the recordings from May's meetup have been posted. Enjoy!
  May 23
- The recording for "Learning Elixir by Contributing" by @joshnuss has been posted https://youtu.be/BoSkq43WLck
  May 19

View more on Twitter

New Videos

- 

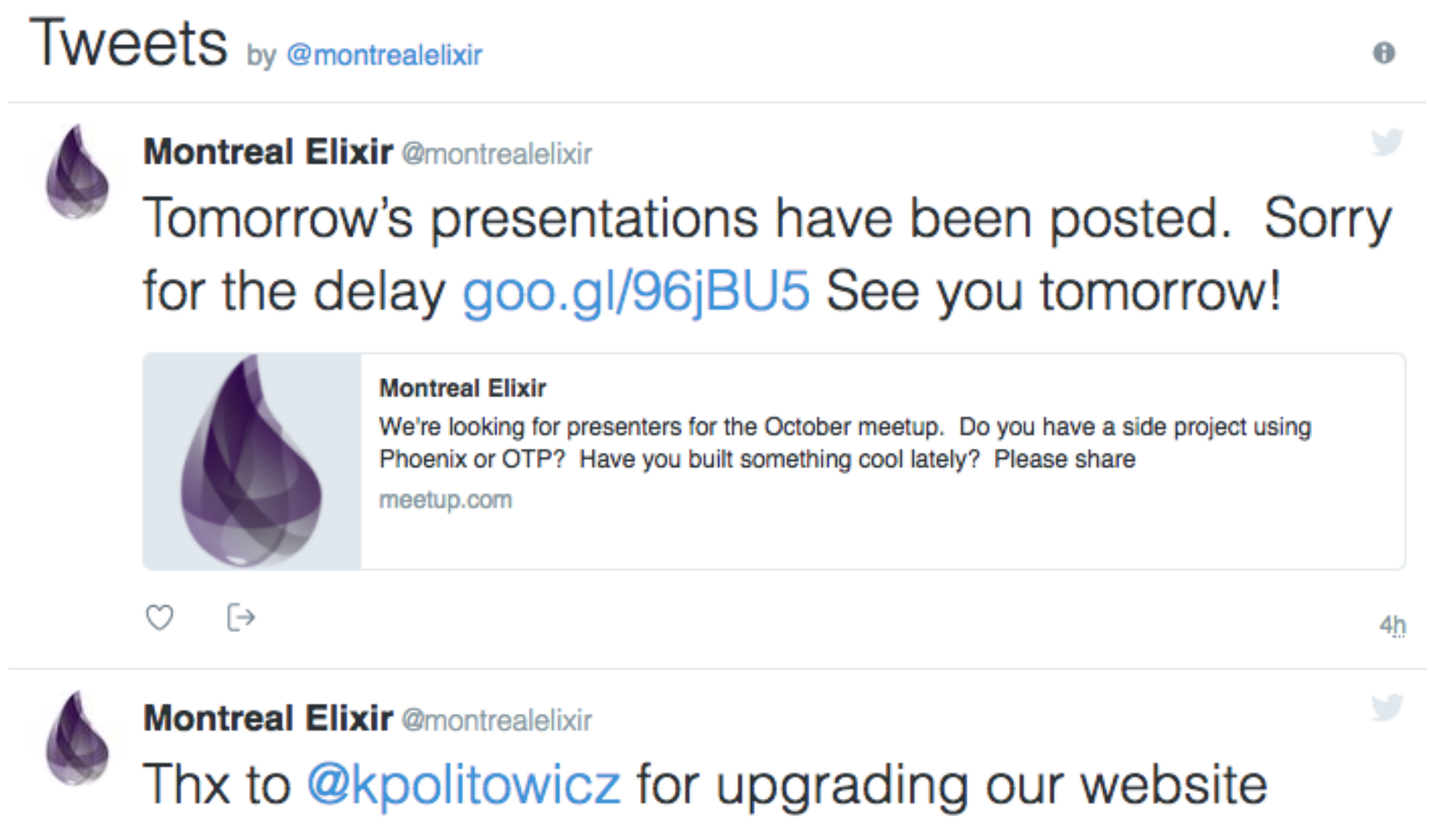
Learning Elixir by Contributing
44 views - 1 week ago
- 

Elixir and OTP for Node.js Developers
61 views - 2 week ago

Option 1: embedded timeline

- https://dev.twitter.com/web/embedded-timelines/user
- Very easy to integrate

```
<a class="twitter-timeline" data-height="500"
  href="https://twitter.com/montrealelixir?ref_src=twsrc%5Etfw">
  Tweets by montrealelixir
</a>
<script async src="//platform.twitter.com/widgets.js" charset="utf-8">
</script>
```



Option 2: Twitter API

- Search API
  - tweets

    `1.1/search/tweets.json`
  - user timeline

    `1.1/statuses/user_timeline.json`
- https://developer.twitter.com

Search all documentation...

Basics

Accounts and users

Tweets

Post, retrieve and engage with Tweets

Get Tweet timelines

Curate a collection of Tweets

Optimize Tweets with Cards

Search Tweets

Filter realtime Tweets

Sample realtime Tweets

Get batch historical Tweets

Tweet compliance

Tweet data dictionaries

Rules and filtering

Premium enrichments

Tweet updates

Direct Messages

Media

Search Tweets

Overview Guides API Reference

API Reference contents

30-Day Search API

Full-Archive Search API

GET search/tweets

GET saved_searches/list

GET saved_searches/show/:id

POST saved_searches/create

POST saved_searches/destroy/:id

GET search/tweets

Returns a collection of relevant Tweets matching a specified query.

Please note that Twitter's search service and, by extension, the Search API is not meant to be an exhaustive source of Tweets. Not all Tweets will be indexed or made available via the search interface.

In API v1.1, the response format of the Search API has been improved to return Tweet objects more similar to the objects you'll find across the REST API and platform. However, perspectival attributes (fields that pertain to the perspective of the authenticating user) are not currently supported on this endpoint.

To learn how to use Twitter Search effectively, consult our guide to Using the Twitter Search API. See Working with Timelines to learn best practices for navigating results by `since_id` and `max_id`.

Any existing package?

Packages

6 Results Found
(Sorted by recent downloads)

Sort By

extwitter 0.8.6
Twitter client library for elixir.
6 371 recent downloads

ueberauth_twitter 0.2.4
An Uberauth strategy for Twitter authentication.
2 793 recent downloads

bh 0.0.14
Twitter Bootstrap 4 and Bootstrap 3 helpers for Phoenix.
563 recent downloads

twittex 0.3.4
Twitter client library for Elixir
353 recent downloads

furlex 0.2.0
Furlex is a structured data extraction tool written in Elixir. It currently supports unfurling oEmbed, Twitter Card, Facebook Open Graph...
79 recent downloads

twittertex 0.2.0
Formats a tweet as HTML, using the entities from its JSON structure.
71 recent downloads

Showing 1–6 packages of 6 total

ExTwitter

Twitter client library for elixir. It uses oauther to call Twitter's REST API.

It only supports very limited set of functions yet. Refer to lib/extwitter.ex and test/extwitter_test.exs for available functions and examples.

Documentation

- http://hexdocs.pm/extwitter

Usage

1. Add `extwitter` to deps section in the `mix.exs`.
2. Use `ExTwitter.configure` to setup Twitter's OAuth authentication parameters. Refer to https://dev.twitter.com/docs for the detail.
3. Call functions in ExTwitter module (ex. `ExTwitter.search("test")`).

Configuration

The default behaviour is to configure using the application environment:

In `config/config.exs`, add:

```
config :extwitter, :oauth, [
   consumer_key: "",
   consumer_secret: "",
   access_token: "",
   access_token_secret: ""
]
```

Or manually at runtime:

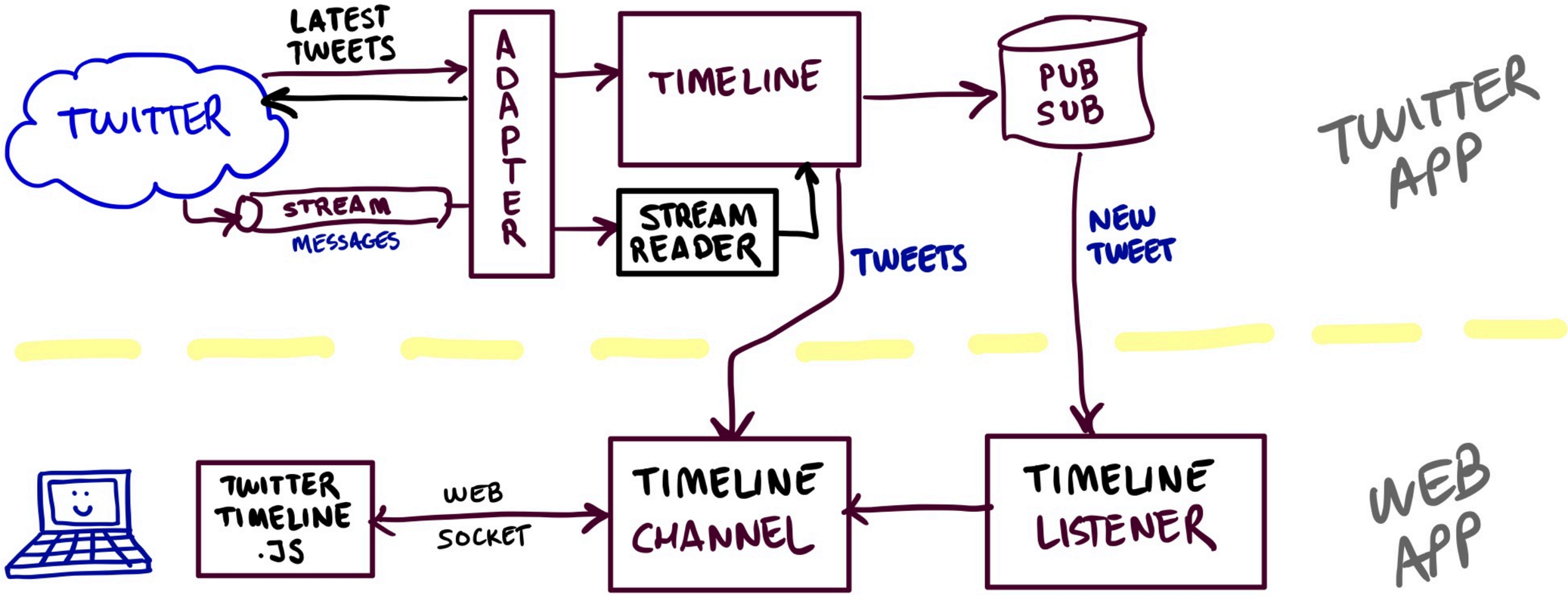
```
ExTwitter.configure([consumer_key: "", ...])
```

You can also configure the current process only:

```
ExTwitter.configure(:process, [consumer_key: "", ...])
```

Let's do it!

Architecture



Twitter umbrella app

- Standalone OTP application
- Publishes events to the Pub/Sub Application
- Needs configuration for using the Twitter API

Add new application in umbrella

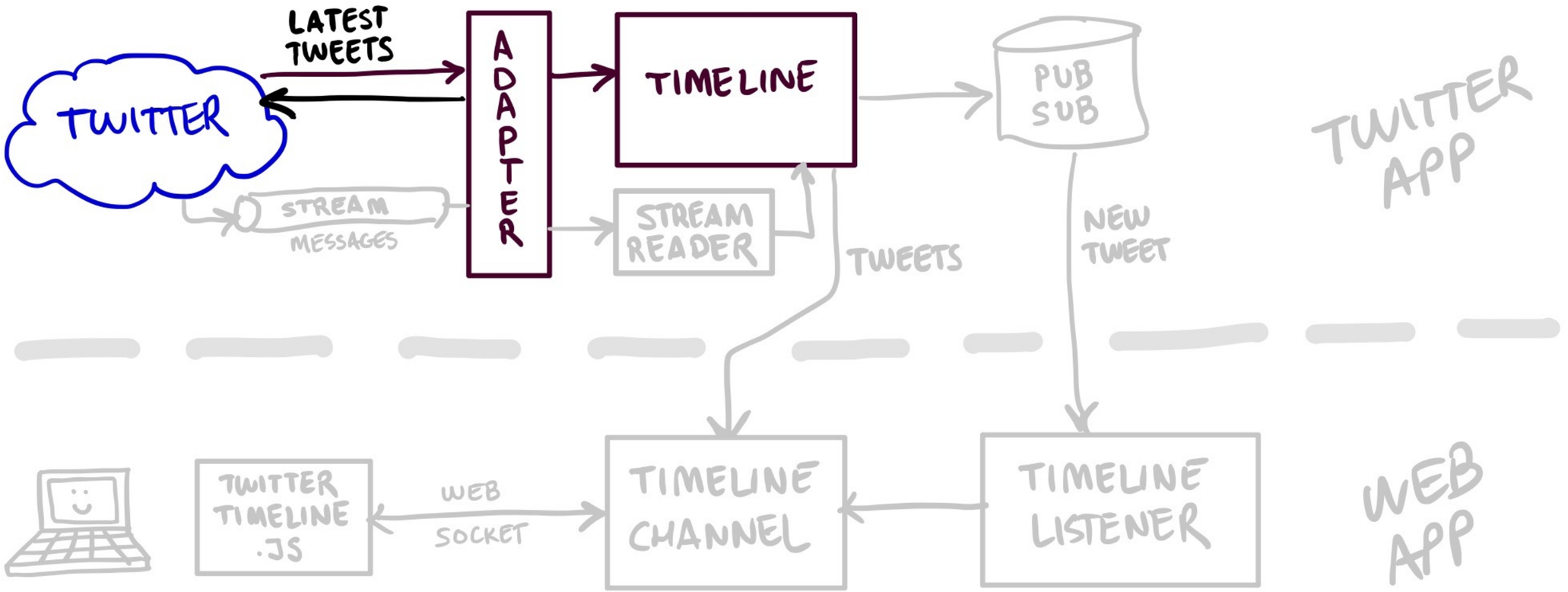
```
> cd apps
> mix new twitter --sup
* creating README.md
* creating .gitignore
* creating mix.exs
* creating config
* creating config/config.exs
* creating lib
* creating lib/twitter.ex
* creating lib/twitter/application.ex
* creating test
* creating test/test_helper.exs
* creating test/twitter_test.exs
>
```

Add new application in umbrella

```elixir
defmodule Twitter.Application do
  # See https://hexdocs.pm/elixir/Application.html
  # for more information on OTP Applications
  @moduledoc false

  use Application

  def start(_type, _args) do
    # List all child processes to be supervised
    children = [
      # Starts a worker by calling: Twitter.Worker.start_link(arg)
      # {Twitter.Worker, arg},
    ]

    # See https://hexdocs.pm/elixir/Supervisor.html
    # for other strategies and supported options
    opts = [strategy: :one_for_one, name: Twitter.Supervisor]
    Supervisor.start_link(children, opts)
  end
end
```

Timeline Features

- Get a list of most recent tweets (on-demand)
- Publish a new tweet received from Twitter
- Handle deleted tweets


LATEST
TWEETS
TWITTER
ADAPTER
TIMELINE
PUB
SUB
TWITTER
APP
STREAM
MESSAGES
STREAM
READER
TWEETS
NEW
TWEET
TWITTER
TIMELINE
.JS
WEB
SOCKET
TIMELINE
CHANNEL
TIMELINE
LISTENER
WEB
APP

Adapter pattern

- Avoid coupling with ExTwitter
  - Domain uses its own data structs (instead of ExTwitter ones)
  - Adapters implement a Behavior

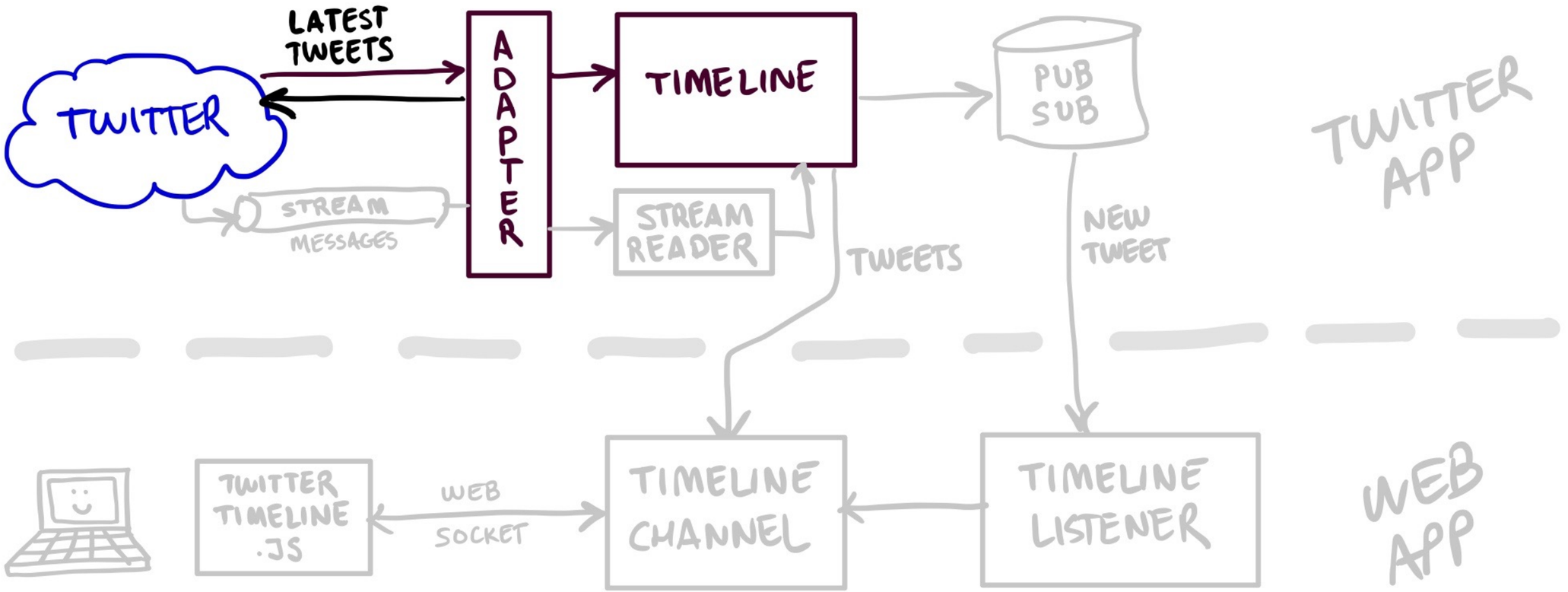
Structs

```elixir
defmodule Twitter.Tweet do
  defstruct [id: nil, text: nil, timestamp: nil]

  @type t :: %__MODULE__{}
end

defmodule Twitter.TweetDeletion do
  defstruct [tweet_id: nil]

  @type t :: %__MODULE__{}
end
```

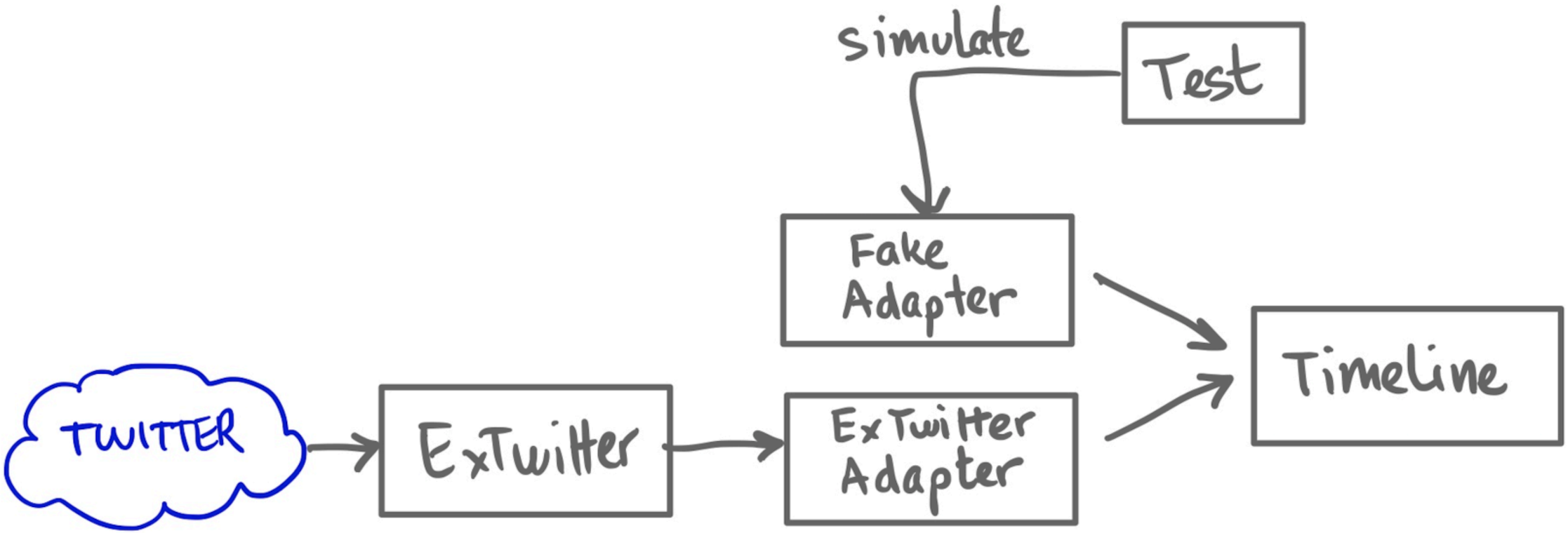
Adapter behavior

```elixir
defmodule Twitter.Adapter do
  alias Twitter.Tweet

  @doc """
  Returns a list of existing tweets in reverse order (most recent one first).
  """
  @callback fetch_user_timeline() :: [Tweet.t]

  @doc """
  Returns a stream of timeline events (new tweets, deleted tweets, ...).
  """
  @callback get_user_stream() :: Stream.t
end
```

Test-drive the Timeline


LATEST
TWEETS
TWITTER
ADAPTER
TIMELINE
PUB
SUB
TWITTER
APP
STREAM
MESSAGES
STREAM
READER
TWEETS
NEW
TWEET
TWITTER
TIMELINE
.JS
WEB
SOCKET
TIMELINE
CHANNEL
TIMELINE
LISTENER
WEB
APP

Adapters


simulate
Test
Fake Adapter
TWITTER
ExTwitter
ExTwitter Adapter
Timeline

```elixir
defmodule Twitter.TimelineTest do
  use ExUnit.Case, async: false

  alias Twitter.{Timeline, Tweet}

  setup do
    adapter = Twitter.Adapter.Fake
    {:ok, pid} = adapter.start_link()

    on_exit fn -> assert_down(pid) end

    [adapter: adapter]
  end

  describe "initializing the timeline" do
  end

  # As suggested here:
  # https://elixirforum.com/t/how-to-stop-otp-processes-started-in-exunit-setup-callback/3794/5
  defp assert_down(pid) do
    ref = Process.monitor(pid)
    assert_receive {:DOWN, ^ref, _, _, _}
  end
end
```

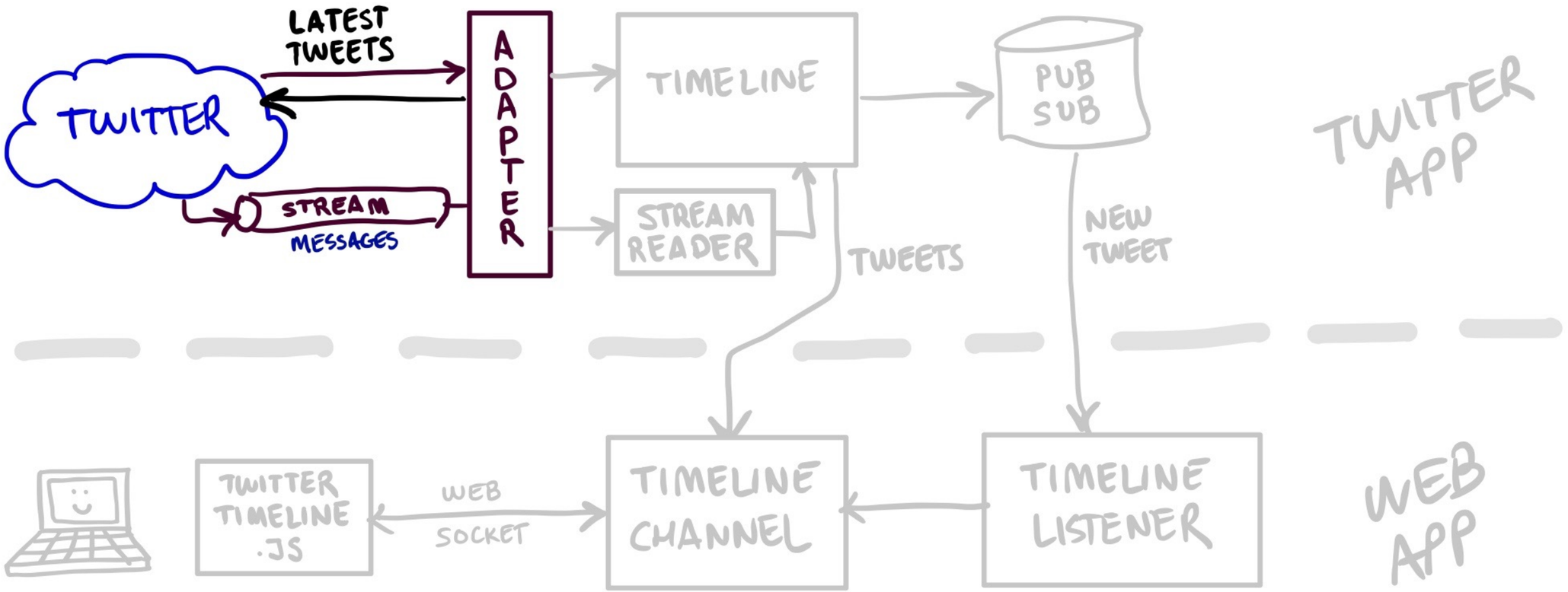
```elixir
describe "initializing the timeline" do
  test "initializes the state with existing user's tweets", %{adapter: adapter} do
    adapter.put_tweets([
      %Tweet{text: ":tweet-2:"},
      %Tweet{text: ":tweet-1:"}
    ])
    {:ok, timeline} = Timeline.start_link(adapter: adapter)

    tweets = Timeline.tweets

    assert length(tweets) == 2
    first_tweet = List.first(tweets)
    assert %Tweet{text: ":tweet-2:"} = first_tweet

    on_exit fn -> assert_down(timeline) end
  end
end
```

```elixir
defmodule Twitter.Timeline do
  use GenServer
  alias Twitter.Tweet

  # Public interface

  def start_link(opts \\ []) do
    adapter = Application.get_env(:twitter, :adapter, opts[:adapter])
    GenServer.start_link(__MODULE__, [adapter], name: __MODULE__)
  end

  def init([adapter]) do
    # Defer initialization to prevent timeout
    GenServer.cast(self(), :init)

    {:ok, %{adapter: adapter, tweets: []}}
  end

  # Server Callbacks

  def handle_cast(:init, %{adapter: adapter} = state) do
    latest_tweets = adapter.fetch_user_timeline()
    new_state = %{state | tweets: latest_tweets}

    {:noreply, new_state}
  end
```

```elixir
defmodule Twitter.Timeline do
  ...

  @doc """
  Returns the tweets, the first one being the most recent one.
  """
  @spec tweets :: [Tweet.t]
  def tweets do
    GenServer.call(__MODULE__, :list)
  end

  # Server Callbacks

  ...

  def handle_call(:list, _from, %{tweets: tweets} = state) do
    {:reply, tweets, state}
  end
end
```

```
> mix test
Compiling 1 file (.ex)
.

Finished in 0.04 seconds
1 test, 0 failures

Randomized with seed 79869
>
```

ExTwitter Adapter


LATEST
TWEETS
TWITTER
STREAM
MESSAGES
ADAPTER
TIMELINE
PUB
SUB
TWITTER
APP
STREAM
READER
TWEETS
NEW
TWEET
TWITTER
TIMELINE
.JS
WEB
SOCKET
TIMELINE
CHANNEL
TIMELINE
LISTENER
WEB
APP

Responsabilities

- Implement `Twitter.Adapter` behavior
  - Get user's timeline (latest tweets)
  - Provide a stream of incoming Twitter messages
- Convert `ExTwitter` structs to our own structs

```elixir
defmodule Twitter.Adapter.ExTwitter do
  @behaviour Twitter.Adapter

  @spec fetch_user_timeline() :: [Twitter.Tweet.t]
  def fetch_user_timeline do
  end

  @spec get_user_stream() :: Stream.t
  def get_user_stream do
  end
end
```

```elixir
defmodule Twitter.Adapter.ExTwitter.StateTest do
  use ExUnit.Case, async: false

  alias Twitter.{Adapter, Tweet}

  test "converts a ExTwitter.Model.Tweet to Tweet" do
    ex_twitter_tweet = %ExTwitter.Model.Tweet{id: ":id:", text: ":text:", created_at: ":time:"}

    tweet = Adapter.ExTwitter.State.convert(ex_twitter_tweet)

    assert tweet == %Tweet{id: ":id:", text: ":text:", timestamp: ":time:"}
  end
end
```

```elixir
defmodule Twitter.Mixfile do
  ...

  # Run "mix help deps" to learn about dependencies.
  defp deps do
    [
      # {:dep_from_hexpm, "~> 0.3.0"},
      # {:dep_from_git, git: "https://github.com/elixir-lang/my_dep.git", tag: "0.1.0"},
      # {:sibling_app_in_umbrella, in_umbrella: true},
      {:mix_test_watch, "~> 0.3", only: :dev, runtime: false},
      {:blocking_queue, "~> 1.0"},
      {:extwitter, "~> 0.8"}
    ]
  end
end
```

```elixir
defmodule Twitter.Adapter.ExTwitter do
  @behaviour Twitter.Adapter

  defmodule State do
    def convert(%ExTwitter.Model.Tweet{id: id, text: text, created_at: timestamp}) do
      %Twitter.Tweet{id: id, text: text, timestamp: timestamp}
    end
  end

  @spec fetch_user_timeline() :: [Twitter.Tweet.t]
  def fetch_user_timeline do
  end

  @spec get_user_stream() :: Stream.t
  def get_user_stream do
  end
end
```

```elixir
defmodule Twitter.Adapter.ExTwitter do
  @behaviour Twitter.Adapter

  defmodule State do
    def convert(%ExTwitter.Model.Tweet{id: id, text: text, created_at: timestamp}) do
      %Twitter.Tweet{id: id, text: text, timestamp: timestamp}
    end
  end

  @spec fetch_user_timeline() :: [Twitter.Tweet.t]
  def fetch_user_timeline do
    ExTwitter.user_timeline
    |> Enum.map(&State.convert/1)
  end

  @spec get_user_stream() :: Stream.t
  def get_user_stream do
  end
end
```

Let's try it!

But first...

Configuration

config/config.exs

```
use Mix.Config
import_config "../apps/*/config/config.exs"

# Configures Elixir's Logger
config :logger, :console,
  format: "$time $metadata[$level] $message\n",
  metadata: [:request_id]

config :twitter,
  adapter: Twitter.Adapter.ExTwitter

import_config "#{Mix.env}.exs"
```

Configuration

config/dev.exs

```
use Mix.Config

# Do not include metadata nor timestamps in development logs
config :logger, :console, format: "[$level] $message\n"

# Set a higher stacktrace during development. Avoid configuring such
# in production as building large stacktraces may be expensive.
config :phoenix, :stacktrace_depth, 20

import_config "dev.secret.exs"
```

Configuration

config/dev.secret.exs

```
use Mix.Config

config :extwitter, :oauth, [
    consumer_key: "something",
    consumer_secret: "here",
    access_token: "and-here",
    access_token_secret: "and-here-also"
]
```

Twitter Set-up

- Go to https://apps.twitter.com/
- Create New App
- Create access tokens

Create an application

Application Details

Name *

elixir-montreal-website

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *

List of latest tweets on Montreal Elixir website

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *

http://montrealelixir.ca

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL

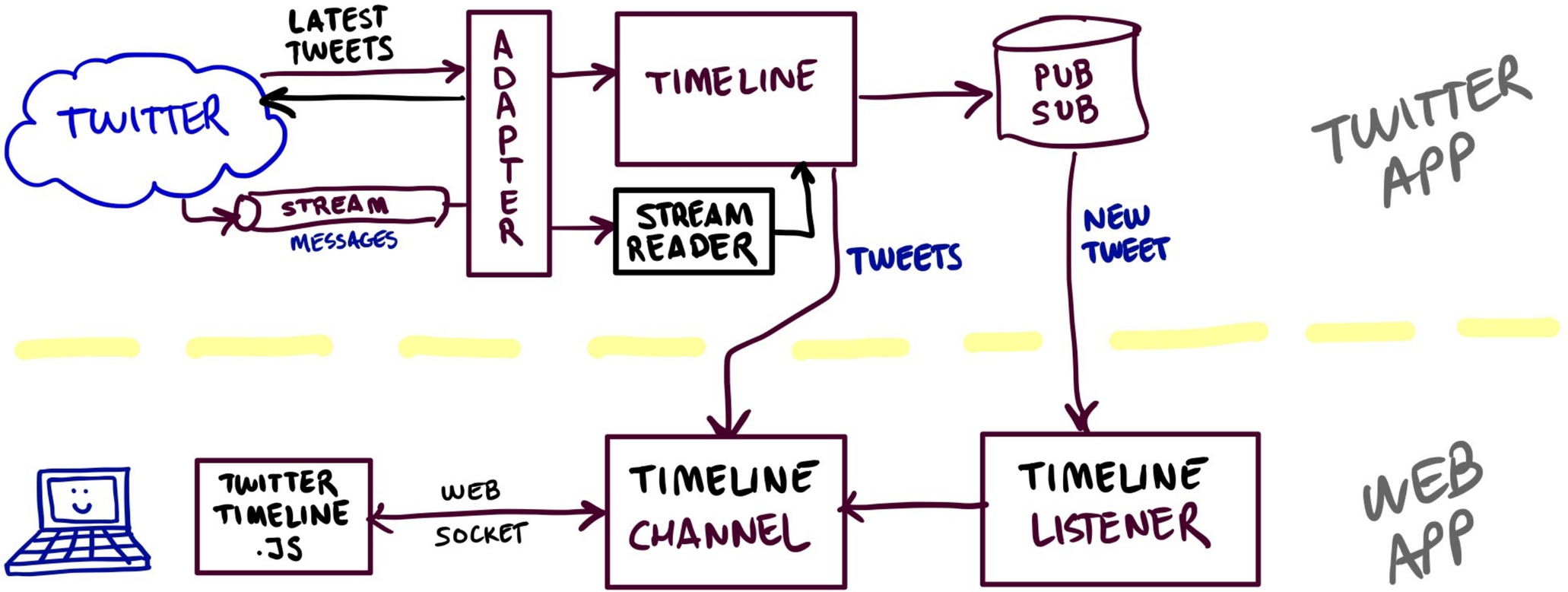
Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

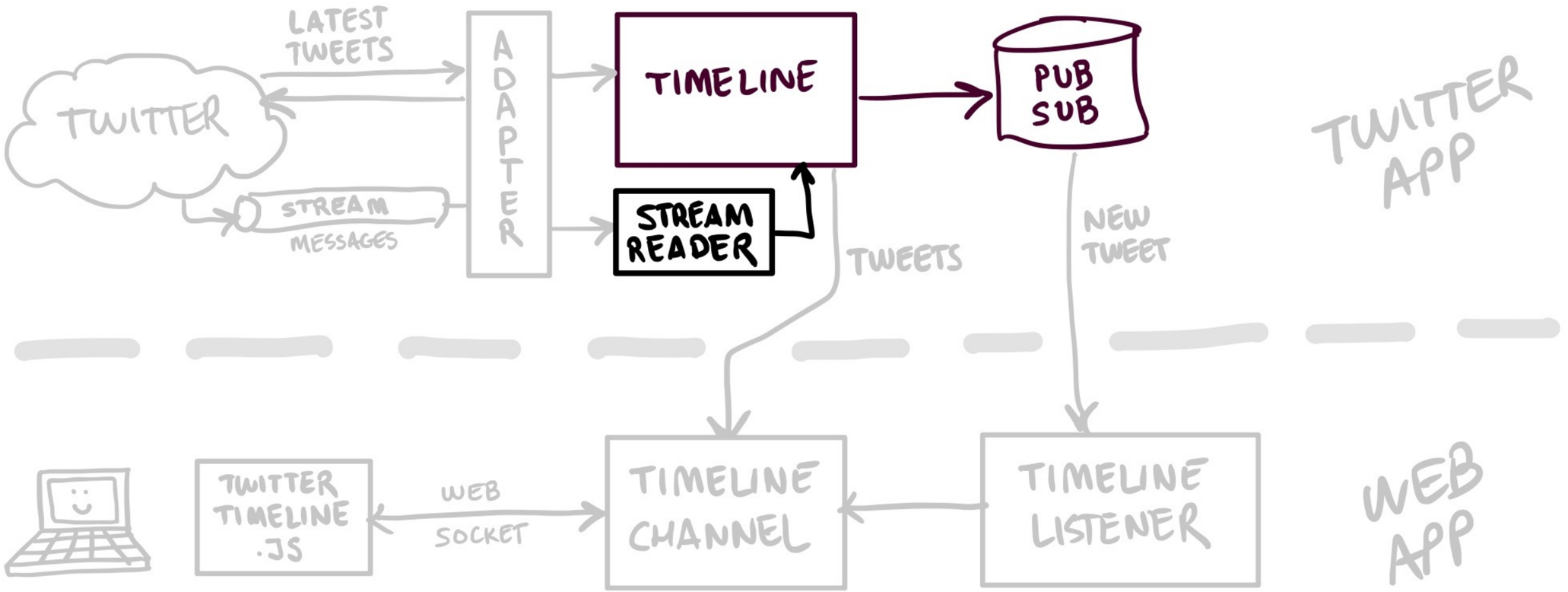
Developer Agreement

Demo 1

```
> cd ../..
> iex -S mix
> Twitter.Timeline.start_link
> Twitter.Timeline.tweets
>
```

Listen to user's stream


LATEST
TWEETS
TWITTER
STREAM
MESSAGES
ADAPTER
TIMELINE
STREAM
READER
PUB
SUB
TWITTER
APP
TWEETS
NEW
TWEET
TWITTER
TIMELINE
.JS
WEB
SOCKET
TIMELINE
CHANNEL
TIMELINE
LISTENER
WEB
APP


LATEST TWEETS
TWITTER
STREAM MESSAGES
ADAPTER
TIMELINE
STREAM READER
PUB SUB
TWITTER APP
TWEETS
NEW TWEET
TWITTER TIMELINE .JS
WEB SOCKET
TIMELINE CHANNEL
TIMELINE LISTENER
WEB APP

```elixir
defmodule Twitter.TimelineTest do
  ...

  describe "publishing tweets" do
    setup %{adapter: adapter} do
      topic = "test:timeline"
      PubSub.start_link()
      PubSub.subscribe(self(), topic)

      {:ok, timeline} = Timeline.start_link(adapter: adapter, topic: topic)
      on_exit fn -> assert_down(timeline) end

      :ok
    end

    test "publishes a new tweet", %{adapter: adapter} do
      tweet = %Tweet{text: ":text:"}
      adapter.stream_tweet(tweet)
      assert_receive {:new_tweet, ^tweet}
    end
  end
end
```

```elixir
defmodule Twitter.Mixfile do
  ...

  # Run "mix help deps" to learn about dependencies.
  defp deps do
    [
      # {:dep_from_hexpm, "~> 0.3.0"},
      # {:dep_from_git, git: "https://github.com/elixir-lang/my_dep.git", tag: "0.1.0"},
      # {:sibling_app_in_umbrella, in_umbrella: true},
      {:mix_test_watch, "~> 0.3", only: :dev, runtime: false},
      {:blocking_queue, "~> 1.0"},
      {:extwitter, "~> 0.8"},
      {:pubsub, "~> 1.0"}
    ]
  end
end
```

```elixir
defmodule Twitter.Timeline do
  ...

  # Public interface

  def start_link(opts \\ []) do
    adapter = Application.get_env(:twitter, :adapter, opts[:adapter])
    topic = Keyword.get(opts, :topic, "twitter:timeline")
    GenServer.start_link(__MODULE__, [adapter, topic], name: __MODULE__)
  end

  def init([adapter, topic]) do
    # Defer initialization to prevent timeout
    GenServer.cast(self(), :init)

    {:ok, %{adapter: adapter, tweets: [], topic: topic}}
  end

  ...
end
```

```elixir
defmodule Twitter.Timeline do
  ...

  def handle_cast(:init, %{adapter: adapter} = state) do
    latest_tweets = adapter.fetch_user_timeline()
    new_state = %{state | tweets: latest_tweets}
    listen_to_user_stream(adapter)

    {:noreply, new_state}
  end

  defp listen_to_user_stream(adapter) do
    stream = adapter.get_user_stream()
    timeline = self()

    Task.start_link(fn -> read_stream(stream, timeline) end)
  end
end
```

```elixir
defmodule Twitter.Timeline do
  ...

  defp read_stream(stream, timeline) do
    for message <- stream do
      case determine_call(message) do
        :noop -> :ok
        action -> GenServer.call(timeline, action)
      end
    end
  end

  defp determine_call(%Tweet{} = tweet), do: {:push, tweet}
  defp determine_call(_), do: :noop

  ...
end
```

```elixir
defmodule Twitter.Timeline do
  ...

  def handle_call({:push, tweet}, _from, %{tweets: tweets, topic: topic} = state) do
    PubSub.publish(topic, {:new_tweet, tweet})

    {:reply, :ok, %{state | tweets: [tweet | tweets]}}
  end

  ...
end
```

```elixir
defmodule Twitter.Adapter.ExTwitter do
  @behaviour Twitter.Adapter

  ...

  @spec get_user_stream() :: Stream.t
  def get_user_stream do
    ExTwitter.stream_user(with: :user, receive_messages: true, timeout: :infinity)
    |> State.filter_stream
  end
end
```

```elixir
defmodule Twitter.Adapter.ExTwitter do
  @behaviour Twitter.Adapter

  defmodule State do
    ...

    def filter_stream(stream) do
      stream
      |> Stream.filter(&valid_message/1)
      |> Stream.map(&convert/1)
    end

    defp valid_message(%ExTwitter.Model.Tweet{}), do: true
    defp valid_message(_), do: false
  end
end
```

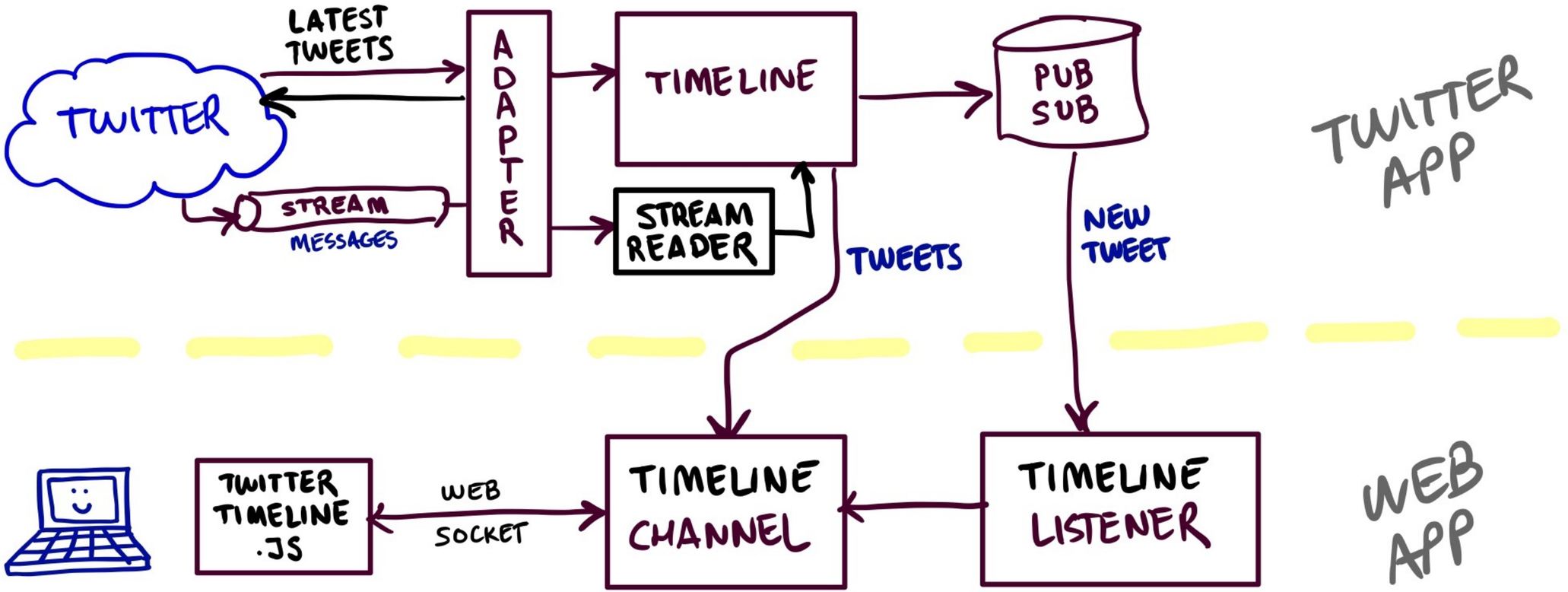
```elixir
defmodule Twitter.Application do
  # See https://hexdocs.pm/elixir/Application.html
  # for more information on OTP Applications
  @moduledoc false

  use Application

  def start(_type, _args) do
    PubSub.start_link()

    # List all child processes to be supervised
    children = workers(Mix.env)

    # See https://hexdocs.pm/elixir/Supervisor.html
    # for other strategies and supported options
    opts = [strategy: :one_for_one, name: Twitter.Supervisor]
    Supervisor.start_link(children, opts)
  end

  defp workers(:test), do: []
  defp workers(_) do
    [
      {Twitter.Timeline, []}
    ]
  end
end
```

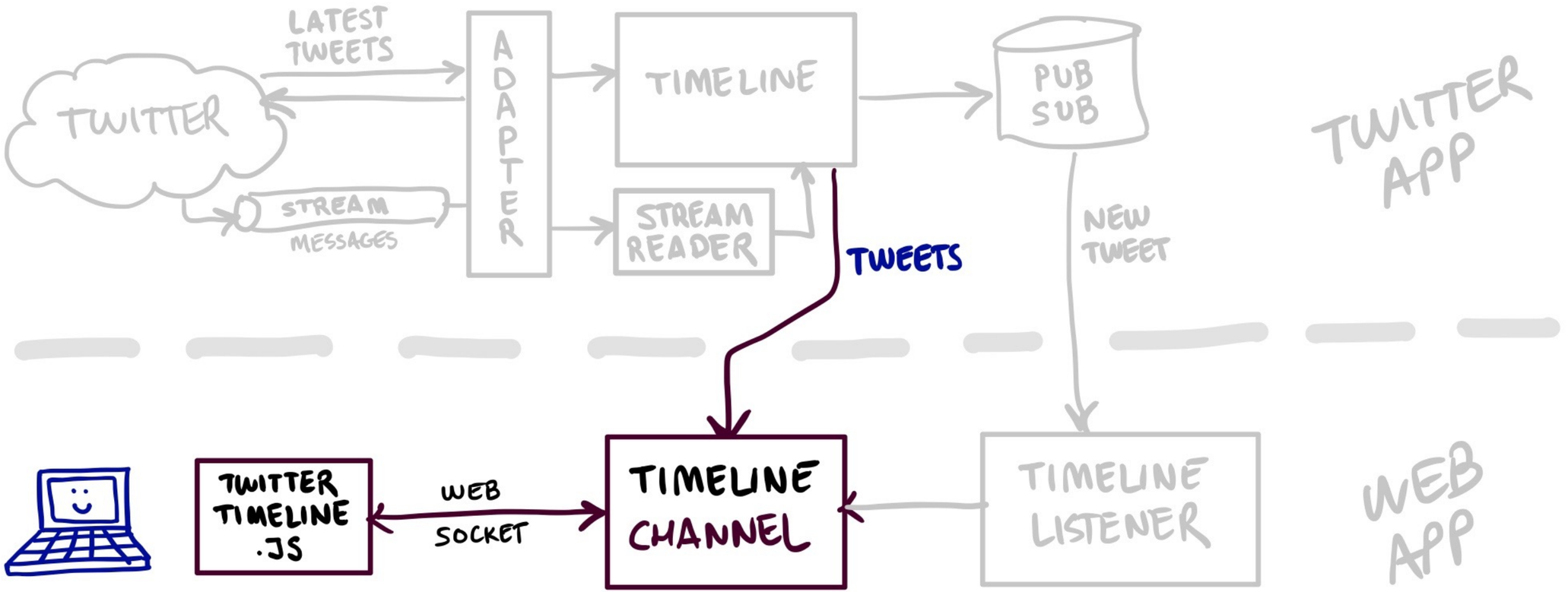
Demo 2

```
> iex -S mix
> PubSub.subscribe(self(), "twitter:timeline")
> flush
:ok
> flush
{:new_tweet, %Twitter.Tweet{id: 917487557969108992, text: "test", timestamp: ...}}
> :observer.start
>
```

Break

Let's hook it up with the web site


LATEST
TWEETS
TWITTER
STREAM
MESSAGES
ADAPTER
TIMELINE
STREAM
READER
PUB
SUB
TWITTER
APP
TWEETS
NEW
TWEET
TWITTER
TIMELINE
.JS
WEB
SOCKET
TIMELINE
CHANNEL
TIMELINE
LISTENER
WEB
APP


LATEST TWEETS
TWITTER
STREAM MESSAGES
ADAPTER
TIMELINE
STREAM READER
PUB SUB
TWEETS
NEW TWEET
TWITTER APP
TWITTER TIMELINE .JS
WEB SOCKET
TIMELINE CHANNEL
TIMELINE LISTENER
WEB APP

Web integration

- HTML container
- Client-side javascript code
- Timeline listener
- Channel logic

lib/templates/page/index.html.eex

```
...

  <div class="col-sm-6">
    <h4>Recent updates (via Twitter)</h4>
    <div id="tweets-container">
      <ul class="tweets"></ul>
    </div>
    <p>View more on <a href="https://twitter.com/montrealelixir">Twitter</a></p>
  </div>
```

assets/js/apps.js

```
import socket from "./socket"

import TwitterTimeline from "./twitter_timeline"

TwitterTimeline.init(socket, "#tweets-container")
```

```
let TwitterTimeline = {
  init(socket, tweets_container) {
    // Now that you are connected, you can join channels with a topic:
    this.channel = socket.channel("twitter_timeline", {})
    this.container = $(tweets_container)
    this.tweets = $(tweets_container).find(".tweets")

    this.channel.join()

    this.channel.on("updated_list", msg => {
      this._refreshAll(msg.tweets)
    })
  },

  _refreshAll(tweetsList) {
    this.tweets.empty()
    for(var tweet of tweetsList) {
      this.tweets.append(`<li>${tweet.text}<br><i>${tweet.timestamp}</i></li>`)
    }
    this.container.scrollTop(this.container[0].scrollHeight)
  }
}

export default TwitterTimeline
```

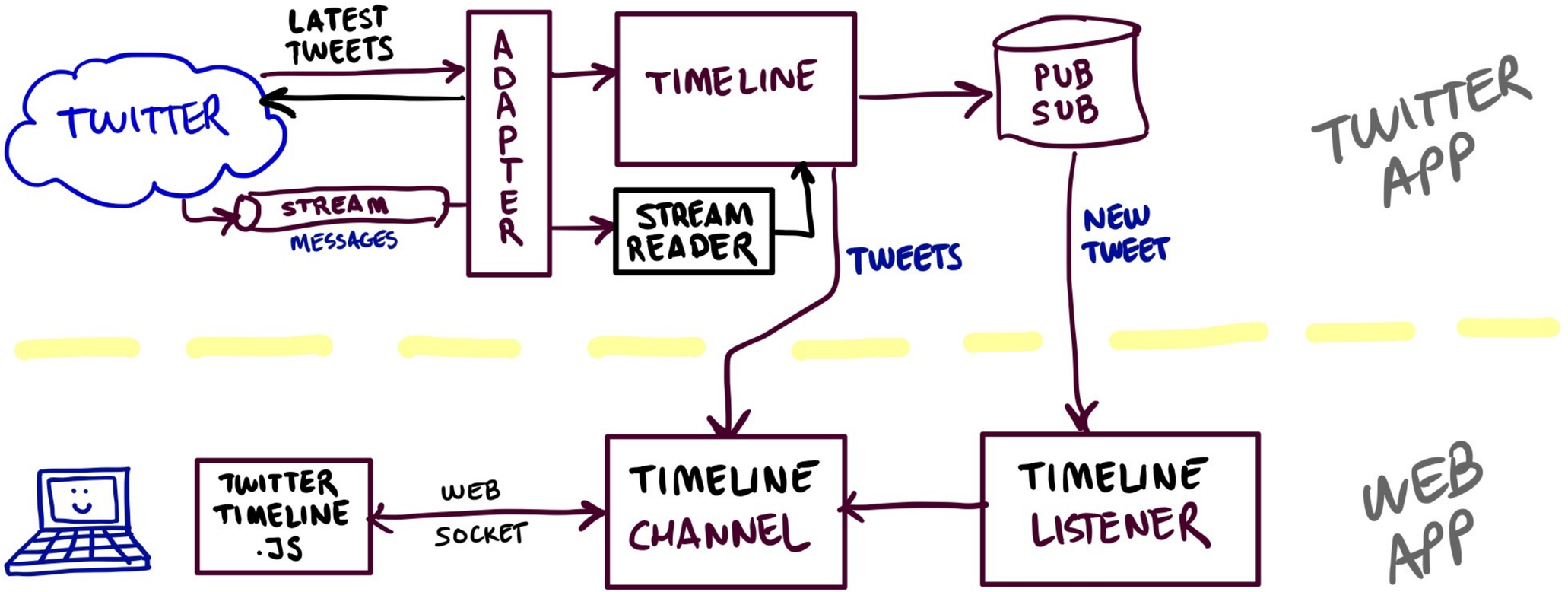
```elixir
defmodule MontrealElixirWeb.UserSocket do
  use Phoenix.Socket

  ## Channels
  channel "twitter_timeline", MontrealElixirWeb.TweetsChannel

  ## Transports
  transport :websocket, Phoenix.Transports.WebSocket, timeout: 45_000
  # transport :longpoll, Phoenix.Transports.LongPoll

  ...
```

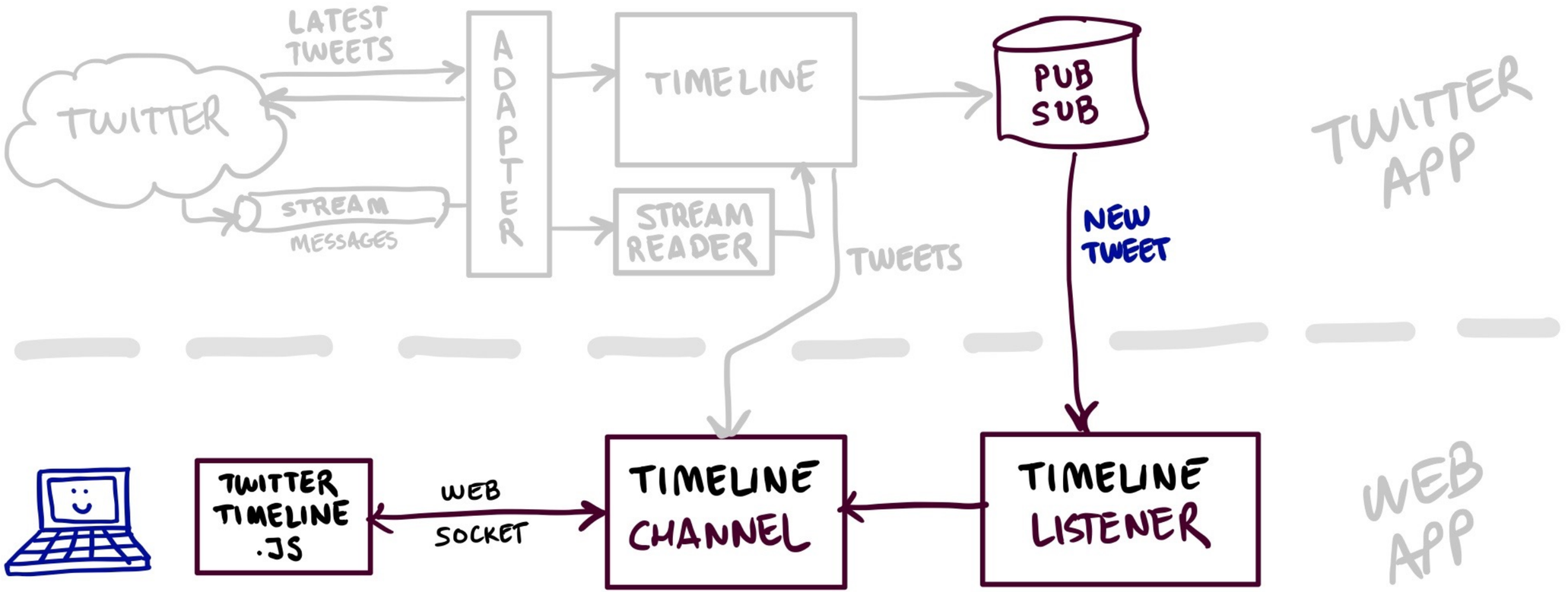
```elixir
defmodule MontrealElixirWeb.TwitterTimelineChannel do
  use Phoenix.Channel

  def join("twitter_timeline", message, socket) do
    send(self(), {:after_join, message})

    {:ok, socket}
  end

  def handle_info({:after_join, _msg}, socket) do
    latest_tweets = Twitter.Timeline.tweets
    push(socket, "updated_list", %{tweets: latest_tweets})

    {:noreply, socket}
  end

  def terminate(_reason, _socket) do
    :ok
  end
end
```

Demo 3

```
> iex -S mix phx.server
> open http://localhost:4000
>
```

Stream handling


LATEST
TWEETS
TWITTER
STREAM
MESSAGES
ADAPTER
TIMELINE
STREAM
READER
PUB
SUB
TWITTER
APP
TWEETS
NEW
TWEET
TWITTER
TIMELINE
.JS
WEB
SOCKET
TIMELINE
CHANNEL
TIMELINE
LISTENER
WEB
APP


LATEST TWEETS
TWITTER
STREAM MESSAGES
ADAPTER
TIMELINE
STREAM READER
PUB SUB
TWEETS
NEW TWEET
TWITTER APP
TWITTER TIMELINE .JS
WEB SOCKET
TIMELINE CHANNEL
TIMELINE LISTENER
WEB APP

Timeline Listener

- Subscribe to PubSub topic
- Broadcast events on channel

```elixir
defmodule MontrealElixirWeb.TwitterTimelineListener do
  use GenServer
  alias MontrealElixirWeb.Endpoint

  def start_link(_opts \\ []) do
    GenServer.start_link(__MODULE__, :ok, name: __MODULE__)
  end

  def init(:ok) do
    PubSub.subscribe(self(), "twitter:timeline")
    {:ok, []}
  end

  def handle_info({:new_tweet, tweet}, state) do
    Endpoint.broadcast!("twitter_timeline", "new_tweet", tweet)
    {:noreply, state}
  end
end
```

```elixir
defmodule MontrealElixirWeb.Application do
  @moduledoc false

  use Application

  def start(_type, _args) do
    import Supervisor.Spec

    # Define workers and child supervisors to be supervised
    children = [
      # Start the endpoint when the application starts
      supervisor(MontrealElixirWeb.Endpoint, []),
      worker(MontrealElixirWeb.TwitterTimelineListener, []),
    ]

    # See http://elixir-lang.org/docs/stable/elixir/Supervisor.html
    # for other strategies and supported options
    opts = [strategy: :one_for_one, name: MontrealElixirWeb.Supervisor]
    Supervisor.start_link(children, opts)
  end

  ...
end
```

```
let TwitterTimeline = {
  init(socket, tweets_container) {
    ...

    this.channel.on("new_tweet", tweet => {
      this._newTweet(tweet)
    })
  },

  _newTweet(tweet) {
    this.tweets.prepend(`<li>${tweet.text}<br><i>${tweet.timestamp}</i></li>`)
    this.container.scrollTop(this.container[0].scrollHeight)
  }

  ...
}

export default TwitterTimeline
```

One last thing

- Need to add dependency to `twitter` app so it gets started before the `web` app
  - `PubSub` process is started by the `twitter` app
  - Otherwise, the call to `PubSub.subscribe` will silently fail

```elixir
defmodule MontrealElixirWeb.Mixfile do
  ...

  defp deps do
    [
      {:phoenix, "~> 1.3.0"},
      {:phoenix_pubsub, "~> 1.0"},
      {:phoenix_ecto, "~> 3.2"},
      {:phoenix_html, "~> 2.10"},
      {:phoenix_live_reload, "~> 1.0", only: :dev},
      {:gettext, "~> 0.11"},
      {:cowboy, "~> 1.0"},
      {:montreal_elixir, in_umbrella: true},
      {:twitter, in_umbrella: true}
    ]
  end
end
```

Demo 4

```
> mix phx.server
> open http://localhost:4000
```

Nicholas to tweet something new.

What's Next

- Format timestamps
- Get full tweet text (currently truncated)
  - option in API request?
  - or ExTwitter needs some love?
- Use ExVCR to better test the ExTwitter adapter?

Thank You

Hugo Frappier
hugo@civilcode.io
@hugofrappier

Questions?