Elixir

Learning

- https://elixir-language-tour.swmansion.com/
  - https://hexdocs.pm/elixir/main/introduction.html
  - https://hexdocs.pm/elixir/main/introduction-to-mix.html
  - https://hexdocs.pm/elixir/debugging.html
  - https://elixirschool.com/en
  - https://runelixir.com/welcome.html
  - https://flaviuspopan.com/elixir-year-technical-sabbatical/
  - https://github.com/thmsmlr/livescript
  - https://github.com/oban-bg/oban_training
  - https://www.openmymind.net/Elixirs-With-Statement/
  - https://fly.io/phoenix-files/single-file-elixir-scripts/
  - https://hexdocs.pm/phoenix/up_and_running.html
    - https://hexdocs.pm/phoenix_live_view/form-bindings.html#nested-inputs
  - Study https://github.com/fly-apps/live_beats
  - Elixir in Action
  - Functional Programming with Elixir
  - From Ruby to Elixir
  - Designing Elixir Systems with OTP
  - https://github.com/JKWA/funpark_notebooks
  - The BEAM Book: Understanding the Erlang Runtime System
  - Designing for Scalability with Erlang/OTP
  - https://www.youtube.com/@srcrip/videos
  - Code Generation by using Macros by Filipe Cabaço
  - Keynote: Gang of None? Design Patterns in Elixir - José Valim
  - The Architecture of Oban
  - https://elixirpatterns.dev/

Books

- Elixir in Action
  - Designing Elixir Systems with OTP
  - From Ruby to Elixir
  - Programming Elixir
  - Testing Elixir
  - Programming Ecto

- Programming Phoenix
- Programming Phoenix LiveView
- https://github.com/JKWA/funpark_notebooks
- Real-Time Phoenix
- Concurrent Data Processing in Elixir
- Metaprogramming Elixir
- Engineering Elixir Applications
- Machine Learning in Elixir
- Elixir Patterns
- The BEAM Book: Understanding the Erlang Runtime System
- Designing for Scalability with Erlang/OTP

Templates

- https://github.com/gmcabrita/petal_pro_web
  - https://github.com/gmcabrita/petal_framework
  - Mise

```
[tasks.watch]
# Can't get this one to work correctly 😭
# --no-process-group is apparently not supported by
mise watch
# run = "mise watch format --timings --exts
'ex,exs,leex,heex,eex,sface,escript'"
run = "watchexec --no-process-group --timings --exts
'ex,exs,leex,heex,eex,sface,escript' -- 'mix
format'"
alias = ["default", "w"]

[tasks.format]
run = "mix format"
alias = "f"

[tasks.test]
run = "mix test"
depends = ["compile"]
alias = "t"

[tasks.test-stale]
run = "mix test --stale"
depends = ["compile"]
alias = "ts"
```

```
[tasks.compile]
run = "mix compile"
depends = ["install"]
alias = "c"

[tasks.install]
run = "mix deps.get --all"
alias = "i"
```

CI

- https://github.com/felt/ultimate-elixir-ci
    - https://felt.com/blog/hashrocket-ultimate-elixir-to-the-next-level

Packages

- https://github.com/ash-project/usage_rules
    - https://elixirforum.com/t/usage-rules-a-tool-for-synchronizing-llm-rules-files-with-your-dependencies/70991
    - https://hexdocs.pm/usage_rules/readme.html
    - https://github.com/ash-project/ash/blob/main/usage-rules.md
  - elixir-saas: Elixir tools for building SaaS applications
    - https://elixir-saas.com/
  - https://github.com/dominicletz/certmagex
  - akoutmos/hog: track memory hungry processes
  - thmsmlr/livebook_tools: Powertools for livebook.dev — AI Code Editing, MCP Servers, and Running Livebooks from the CLI
  - solnic/text_parser: TextParser is an Elixir library for extracting and validating structured tokens from text, such as URLs, hashtags, @-mentions etc
  - Gazler/termite: A dependency-free NIF-free terminal library
  - software-mansion/live-debugger: Tool for debugging LiveView applications
  - mirego/telemetry_ui: Telemetry based metrics UI. Take your telemetry metrics and display them in a web page

- dashbitco/broadway: Concurrent and multi-stage data ingestion and data processing with Elixir
- mimic: A mocking library for Elixir
- mox: Mocks and explicit contracts in Elixir
  - carsdotcom/credo_mox: Credo check for ensuring that Mox expect calls have been verified
- prom_ex: Prometheus metrics collection library built on top of Telemetry with accompanying Grafana dashboards
- schrockwell/bodyguard: Simple authorization/permissions conventions for Phoenix apps
- woylie/let_me: Authorization library for Elixir
- lalabuy948/PhoenixAnalytics: Plug and play analytics for Phoenix applications
- mjml_eex: Create emails that WOW your customers using MJML and EEx
- akoutmos/sql_fmt: Format and pretty print SQL queries
- mhanberg/schematic: A library for data specification, validation, and transformation
- functional-rewire/dune: A sandbox for Elixir to safely evaluate untrusted code from user input
- ityonemo/protoss: Evil, Powerful Protocols for Elixir
  - https://www.youtube.com/watch?v=dCRGgFkCkmA
- acalejos/flint: Practical Ecto embedded schemas for data validation, coercion, and manipulation
- hauleth/mix_unused: Find unused functions in your project
- seanmor5/hop: A tiny web crawling framework for Elixir
- s3cur3/parameterized_test: A utility for defining eminently readable parameterized (or example-based) tests in Elixir
- bencheeorg/benchee: Easy and extensible benchmarking in Elixir providing you with lots of statistics!
- batteries-included/heyya: Heyya the snapshot testing utility for Phoenix framework components
- devinus/poison: An incredibly fast, pure Elixir JSON library
- thmsmlr/livescript: 1 part Phoenix Livereload, 1 part Livebook. All for .exs scripts
- mirego/mix_audit: MixAudit provides a mix deps.audit task to scan a project Mix dependencies for known Elixir security vulnerabilities
- thmsmlr/vendor_sync: Vendor ETL tool for Elixir, continously snyc Stripe (et. al) into your own database

- elixir-wallaby/wallaby: Concurrent browser tests for your Elixir web apps
- ftes/phoenix_test_playwright: Execute PhoenixTest cases in an actual browser via Playwright
- ajvondrak/remote_ip: A plug to rewrite the Plug.Conn's remote_ip based on forwarding headers
- ExHammer/hammer: An Elixir rate-limiter with pluggable backends
- woutdp/live_svelte: Svelte inside Phoenix LiveView with seamless end-to-end reactivity
- ash-project/igniter: A code generation and project patching framework
- ash-project/spark: Tooling for building DSLs in Elixir
- srcrip/live_toast: A beautiful drop-in replacement for the Phoenix Flash system
  - https://toast.src.rip/
- wojtekmach/req: Req is a batteries-included HTTP client for Elixir
  - derekkraan/curl_req: Req is awesome, but the world speaks curl
  - zachallaun/req_telemetry: Req plugin to instrument requests with Telemetry events
- burrito-elixir/burrito: Wrap your application in a BEAM Burrito!
- hauleth/defconst: Helper macros for defining constant values in modules
- sabiwara/iter: A blazing fast compile-time optimized alternative to the Enum and Stream modules
- sabiwara/cmp: Semantic comparison and sorting for Elixir
- sabiwara/aja: Extension of the Elixir standard library focused on data stuctures, data manipulation and performance
- 100phlecs/tailwind_formatter: Sorts tailwind classes within elixir projects
- zachdaniel/tails: Utilities for working with tailwind classes, like semantic merge, and conditional class lists
- lpil/mix-test.watch: 🏇 Because TDD is awesome
  - randycoulman/mix_test_interactive: Interactive watch mode for Elixir's mix test
  - mix test.watch /path/to/file.exs
  - fswatch -l 0.1 lib test | xargs -I {} mix test /path/to/file.exs
  - https://mise.jdx.dev/tasks/running-tasks.html#watching-

    files
      - watchexec/watchexec: Executes commands in response to file modifications
- Artur-Sulej/excellent_migrations: An Elixir tool for checking safety of database migrations
  - fly-apps/safe-ecto-migrations: Guide to Safe Ecto Migrations
- surface-ui/surface: A server-side rendering component library for Phoenix
- testcontainers/testcontainers-elixir: Testcontainers is an Elixir library that supports ExUnit tests, providing lightweight, throwaway instances of common databases, Selenium web browsers, or anything else that can run in a Docker container
- germsvel/phoenix_test: PhoenixTest provides a unified way of writing feature tests – regardless of whether you're testing LiveView pages or static (non-LiveView) pages
- sasa1977/boundary: Manage and restrain cross-module dependencies in Elixir projects
- solnic/drops: 🛠️ Tools for working with data effectively - data contracts using types, schemas, domain validation rules, type-safe casting, and more.
  - https://solnic.dev/introducing-elixir-drops
- cabol/nebulex: In-memory and distributed caching toolkit for Elixir
  - https://hexdocs.pm/nebulex/cache-usage-patterns.html
  - https://whitfin.io/blog/cachex-v4-0-optimization-consolidation-and-routing/
- Oban - Robust Job Processing
  - https://github.com/linqueta/oban-console
  - https://github.com/oban-bg/oban_training
- monarch: data migrations, powered by Oban
- Short Unique IDs in Elixir · Sqids
- sloanelybutsurely/typeid-elixir: Elixir implementation of TypeIDs: type-safe, K-sortable, and globally unique identifiers inspired by Stripe IDs
- aj-foster/open-api-generator: Open API code generator for Elixir
- open-api-spex/open_api_spex: Open API Specifications for Elixir Plug applications
- https://kaffy.fly.dev/admin/

- wyeworks/boom: Exception notification for plug based applications
- remoteoss/phx_gen_solid: A SOLID generator for Phoenix applications
- underjord/entrace: Tracing experiments
- underjord/entrace_live_dashboard: Tracing experiments with live dashboard
- naymspace/backpex: Backpex is a highly customizable administration panel for Phoenix LiveView applications
- Erlang - tprof
- Stratus3D/eflambe: A tool for rapid profiling of Erlang and Elixir applications
- DockYard/flame_on: Flame Graph LiveView Component and LiveDashboard plugin
- LivewareProblems/Orion: Distributed Dynamic Profiling for the BEAM
- zhongwencool/observer_cli: Visualize Erlang/Elixir Nodes On The Command Line
- Voronchuk/ecto_gss: Elixir library to persist Ecto records and changesets in Google Spreadsheets
- pnezis/tucan: An Elixir plotting library on top of VegaLite
- mimiquate/blend: 🥣 Test your package against different versions of its dependencies
- sportradar/elixir-workspace: Set of tools for working with elixir monorepos
- utahplt/chorex: Choreographic programming in Elixir
  - Chorex: Guaranteeing Deadlock Freedom in Elixir
- elixir-dbvisor/sql: Brings an extensible SQL parser and sigil to Elixir, confidently write SQL with automatic parameterized queries
- ityonemo/vsr: Viewstamped Replication
  - maelstrom-adapter

Useful blog posts / tricks

- https://andrewian.dev/blog/phoenix-email-defaults
  - https://johnelmlabs.com/posts/magic-link-auth
    - https://github.com/srcrip/phoenix-magic-links
  - https://peterullrich.com/simulate-latency-jitter-and-package-loss-in-phoenix-live-view

- https://community.fly.io/t/custom-health-check-on-phoenix-fails-and-creates-zombie-machines/18932
  - https://gist.github.com/gmcabrita/6a65a894b18cfb86952aed3b46de2a81
- Kill your Phoenix Context
  - Pivot: https://peterullrich.com/resurrect-your-phoenix-context
- https://saltycrackers.dev/posts/bye-bye-async-false/
  - https://andrealeopardi.com/posts/async-tests-in-elixir/
- https://bitcrowd.dev/idempotent-seeds-in-elixir/
- https://andrealeopardi.com/posts/get-rid-of-your-old-database-migrations/
- https://dashbit.co/blog/how-to-debug-elixir-erlang-compiler-performance
- https://dashbit.co/blog/speeding-up-re-compilation-of-elixir-projects
- https://gonglexin.com/posts/managing-elixir-main-branch-with-asdf
- One SQLite DB per tenant
- Protecting LiveView
- Dashbit: Writing assertive code with Elixir
- Dashbit: Soft deletes with Ecto and PostgreSQL
- Dashbit: Req API Client Testing
- Dashbit: SDKs with Req: Stripe
- Dashbit: SDKs with Req: S3
- Dashbit: How we verify webhooks
- https://samrat.me/til-file-uploads-using-the-req-elixir-library/
- zoi: schema validation library

```
def transcribe_audio(file_path, token) do
  model = "whisper-1"
  filename = Path.basename(file_path)
  {:ok, file_contents} = File.read(file_path)

  multipart =
    Multipart.new()
    |>
Multipart.add_part(Multipart.Part.text_field(model,
"model"))
    |> Multipart.add_part(
      Multipart.Part.file_content_field(
        filename,
```

```
          file_contents,
          :file,
          filename: filename
        )
      )

    content_length =
  Multipart.content_length(multipart)
    content_type = Multipart.content_type(multipart,
  "multipart/form-data")

    headers = [
      {"authorization", "Bearer #{token}"},
      {"Content-Type", content_type},
      {"Content-Length", to_string(content_length)}
    ]

    Req.post(
      "<https://api.openai.com/v1/audio/
  transcriptions>",
      headers: headers,
      body: Multipart.body_stream(multipart)
    )
  end
```

- https://benreinhart.com/blog/verifying-slack-requests-elixir-phoenix/
  - One Weird Trick™ for making timing or async-related bugs way easier to reproduce in Elixir:
    - ExUnit.configure(max_cases: System.schedulers_online() * 8)
    - Make sure to also set your :pool_size to the same value in test.
    - The default number of parallel tests, of course, is 2 times the number of hardware threads on your CPU. By bumping it to 8, it's way more likely tests will get descheduled in the middle of their execution, so timing bugs that were previously unlikely become common.
    - https://x.com/tylerayoung/status/1749818156510535874?s=46
  - Phoenix LiveReload
    - web_console_logger: true

    - Jumping to HEEx function definitions
    - Hot reload while preserving socket state:

```
  config :my_app, MyAppWeb.Endpoint,
    live_reload: [
      notify: [
        live_view: [
          ~r"lib/my_app_web/core_components.ex$",
          ~r"lib/my_app_web/(live|components)/.*(ex|
heex)$"
        ]
      ],
      patterns: [
        ~r"priv/static/(?!uploads/).*(js|css|png|
jpeg|jpg|gif|svg)$",
        ~r"lib/my_app_web/controllers/.*(ex|heex)$"
      ]
    ]
```

- Add backtraces to Ecto telemetry events:
    - config :my_app, Repo, stacktrace: true
    - https://hexdocs.pm/ecto/Ecto.Repo.html#:~:text=%3Astacktrace- when true%2C publishes the stacktrace in telemetry events and allows more advanced logging.
  - Persistent connections with gen_statem
  - How Phoenix LiveView Form Auto-Recovery works
  - https://diff.hex.pm/

See what changed during a LiveView update

```
// Mutation observer to highlight changed elements
new MutationObserver((mutations) => {
  mutations.forEach((mutation) => {
    if (mutation.type === 'childList') {
      mutation.addedNodes.forEach((node) => {
        if (node.nodeType === Node.ELEMENT_NODE) {
          node.style.transition = 'outline 0.3s
ease-in-out';
          node.style.outline = '2px solid red';
          setTimeout(() => {
            node.style.outline = 'none';
            node.style.transition = '';
          }, 1000);
```

```
        }
      });
    }
  });
}).observe(document.body, {
  childList: true,
  subtree: true,
});
```

Multi-tenant query scoping

https://x.com/atomkirk/status/1806303250208927844

```
def ownership_query(query, _user_id) do
  from(
    u in query,
    join: mtg in assoc(u, :meetings),
    join: ans in assoc(mtg, : answers)
  )
end

defp ownership_query_for_resource(user_id,
resource_id) do
  module = TypeKeyMap. module_from_id(resource_id)

  from(
    [u, ..., resource] in
module.ownership_query(User, user_id),
    where: u.id == ^user_id and resource.id ==
^resource_id
  )
end
```

See also: Scopes in Phoenix 1.8

UUID casting

https://x.com/tylerayoung/status/1818084381447033049

```
@doc """
Casts a UUID string to an Ecto.UUID.t().
```

```
  This is a safer wrapper around `Ecto.UUID.cast/1`,
  which accepts either a UUID string or
  a raw 16 byte UUID (like `<<0x60, 0x1D, 0x74, 0x4,
  0xA8, 0X3, 0x4B, 0x6, 0x83, 0x65, 0xED,
  0xDB, 0x4C, 0x89, 0x33, 0x27>>`). The latter means
  that *any* 16-character string
  (`user@example.com`, `1234567890123456`, `warehouse
  worker`, etc.) will be successfully
  cast, which is almost certainly not what you want.

  ## Examples

      iex> cast_uuid_string("a03de5ed-a9f7-436d-8644-
  b85e6a413e86")
      {:ok, "a03de5ed-a9f7-436d-8644-b856a413e86"}

      iex> cast_uuid_string("user@example.com")
      :error

      iex> cast_uuid_string(42)
      :error
  """
  @spec cast_uuid_string(term) :: {:ok, Ecto.UUID.t()}
  | :error
  def cast_uuid_string(value) when byte_size(value) >
  16 do
    Ecto.UUID.cast(value)
  end

  def cast_uuid_string(_), do: :error
```

Elixir Concurrent Testing Architecture

https://sensaisean.medium.com/elixir-concurrent-testing-architecture-13c5e37374dc
https://github.com/SophisticaSean/elixir-async-testing

- Every test file should be configured as async: true
  - Use Mox to mock external or extraneous services
  - Use start_supervised to start unique GenServers or other required async processes in the setup block of that test

- Simple approach:

```
@doc """
Looks up the bucket pid for `name` stored in
`server`.

Returns `{:ok, pid}` if the bucket exists, `:error`
otherwise.
"""
def lookup(server, name) do
  GenServer.call(server, {:lookup, name})
end

# In the test:
setup do
  registry = start_supervised!({KV.Registry, name:
__MODULE__, test_pid: self()})
  %{registry: registry}
end

test "can put into bucket 1", %{registry: registry}
do
  bucket_id = Ecto.UUID.generate()
  :ok = KV.Registry.create(registry, bucket_id)
  {:ok, bucket} = KV.Registry.lookup(registry,
bucket_id)

  assert {:ok, _bucket} = KV.Bucket.put(bucket,
"hello", 12)
  out = KV.Bucket.get(registry, bucket, "hello")
  assert "12" = out.value
end
```

- Manager layer approach

```
@registry_manager
Application.compile_env(:mox, :registry_manager,
KV.Registry.Manager)
def lookup(name) do
  GenServer.call(@registry_manager.get_server(),
{:lookup, name})
end

setup do
  registry = start_supervised!({KV.Registry, name:
```

```
__MODULE__, test_pid: self()})
  Hammox.stub(KV.Registry.ManagerMock, :get_server,
fn -> registry end)

  :ok
end

test "can put into bucket 1" do
  bucket_id = Ecto.UUID.generate()
  :ok = KV.Registry.create(bucket_id)
  {:ok, bucket} = KV.Registry.lookup(bucket_id)

  assert {:ok, _bucket} = KV.Bucket.put(bucket,
"hello", 12)
  out = KV.Bucket.get(bucket, "hello")
  assert "12" = out.value
end
```

- GenServers/Supervisors need to be made configurable on startup

```
alias Ecto.Adapters.SQL.Sandbox

# ...

def start_link(opts) do
  GenServer.start_link(__MODULE__, {:ok,
Keyword.get(opts, :test_pid, nil)}, opts)
end
@impl true
def init({:ok, parent_pid}) do
  if parent_pid != nil do
    :ok = Sandbox.allow(AsyncTesting.Repo,
parent_pid, self())
  end
  names = %{}
  refs = %{}
  {:ok, {names, refs}}
end
```

- All primary/unique keys used in tests should be made unique via randomization

Clustering

- https://docs.render.com/deploy-elixir-cluster
  - https://fly.io/docs/elixir/the-basics/clustering/

Elixir scripts

- https://github.com/wojtekmach/mix_install_examples
  - Phoenix Playground: Single file Phoenix applications
  - https://fly.io/phoenix-files/single-file-elixir-scripts/
  - generate_deps_changelogs.exs

hex publish via GitHub Actions

https://x.com/emjii/status/1790853377137463522
https://github.com/elixir-mint/castore/blob/22d0a4efd41b97f55aab48e170f2520c338341b9/.github/workflows/publish.sh

Credo

- Custom credo check performance tip
  - carsdotcom/credo_mox: Credo check for ensuring that Mox expect calls have been verified
  - adobe/elixir-styler: An Elixir code-style enforcer that will just FIFY instead of complaining
  - Artur-Sulej/excellent_migrations: An Elixir tool for checking safety of database migrations
  - karolsluszniak/ex_check: One task to efficiently run all code analysis & testing tools in an Elixir project
  - Interesting credo rules:
    - https://hexdocs.pm/credo/Credo.Check.Readability.AliasOrder.html
    - https://hexdocs.pm/credo/Credo.Check.Readability.StrictModuleLayout.html

Ecto generated always as identity

In config.exs add:

```
config :my_app, MyApp.Repo, migration_primary_key:
[type: :"bigint generated always as identity"]
```

Using a VERSION file

```
defp project_version do
  __ENV__.file
  |> Path.dirname()
  |> Path.join("VERSION")
  |> File.read!()
  |> String.trim()
end
```

Mix aliases

```
defp aliases do
  [
    "ecto.setup": [
      "ecto.create",
      "ecto.load --skip-if-loaded --quiet",
      "ecto.migrate",
      "run priv/repo/seeds.exs"
    ],
    "ecto.migrate": ["ecto.migrate", "ecto.dump"],
    "ecto.rollback": ["ecto.rollback", "ecto.dump"],
    test: [
      "ecto.create --quiet",
      "ecto.load --quiet --skip-if-loaded",
      "ecto.migrate --quiet",
      "test"
    ],
    dump_migrations: ["ecto.dump",
&delete_migration_files/1],
    # Deploy from: <https://x.com/wojtekmach/status/
1783072561850401021>
    deploy: ["deploy.check", &deploy_confirm/1,
"deploy.run"],
    "deploy.check": [
      "cmd git status --porcelain | grep . && echo \
\"'error: Working directory is dirty'\\" && exit 1
|| exit 0",
      "cmd mix test --warnings-as-errors"
    ],
    "deploy.run": ["cmd flyctl deploy --build-arg
```

```
APP_SHA=`git rev-parse --short HEAD`"],
    "deploy.console" ["cmd flyctl ssh console --pty
--command=\\"'/app/bin/app_name remote'\\""]
  ]
end

defp delete_migration_files(_args) do
  # Match all files in the 21st century (year is
20xx).
  Enum.each(Path.wildcard("priv/repo/migrations/
20*.exs"), fn migration_file ->
    File.rm!(migration_file)
    Mix.shell().info([:bright, "Deleted:
", :reset, :red, migration_file])
  end)
end

defp deploy_confirm(_) do
  unless Mix.shell().yes?("Ready to deploy?") do
    IO.puts("error: User cancelled")
    System.halt(1)
  end
end
```

Repo.fetch

From: https://tomkonidas.com/repo-fetch/

```
defmodule MyApp.Repo do
  use Ecto.Repo,
    otp_app: :my_app,
    adapter: Ecto.Adapters.Postgres

  @spec fetch(Ecto.Queryable.t(), binary(),
keyword()) ::
          {:ok, Ecto.Schema.t()} |
{:error, :not_found}
  def fetch(queryable, id, opts \\\\ []) do
    case get(queryable, id, opts) do
      nil -> {:error, :not_found}
      record -> {:ok, record}
    end
```

```
  end

  @spec fetch_by(Ecto.Queryable.t(), keyword() |
map(), keyword()) ::
          {:ok, Ecto.Schema.t()} |
{:error, :not_found}
  def fetch_by(queryable, clauses, opts \\\\ []) do
    case get_by(queryable, clauses, opts) do
      nil -> {:error, :not_found}
      record -> {:ok, record}
    end
  end
end
```

Unnest

From: https://kobrakai.de/kolumne/unnest-for-runtime-sorted-results

```
defmodule MyApp.QueryHelpers do
  @doc """
  Unnest into a table format.

  ## Example

      import MyApp.QueryHelpers

      status = [:waiting, :running, :done]
      order = [1, 2, 3]

      from jobs in "jobs",
        join: ordering in unnest(^status, ^order),
        on: jobs.status == ordering.a,
        order_by: ordering.b

  """
  defmacro unnest(list_a, list_b) do
    quote do
      fragment("select * from unnest(?, ?) AS t(a,
b)", unquote(list_a), unquote(list_b))
    end
  end
end
```

Unnest multi-update

From: https://geekmonkey.org/updating-multiple-records-with-different-values-in-ecto-repo-update_all

```
Tree
|> join(
  :inner
  [t],
  tn in fragment(
    "SELECT * FROM unnest(?::integer[], ?::text[])
    AS id_to_new_name(id, new_name)",
    type(^ids, {:array, :integer}),
    type(^new_names, {:array, :string})
  ),
  on: t.id == tn.id
)
|> update([_, tn], set: [name: tn.new_name])
|> DB.Repo.update_all([])
```

Scroll into view on reload

A little snippet I put into my LiveView apps to make development easier.

Add an id to an element and included it in the URL hash to have the element scroll into view on every reload

From: https://x.com/atimberlake/status/1823319021992763403

```
window.addEventListener('phx:page-loading-stop',
(_info) => {
    if (document.location.hash) {
        const cid =
document.location.hash.substring(1);
        const el = document.getElementById(cid);
        if (el) {
            el.scrollIntoView({ behavior:
'smooth' });
        }
    }
```

```
});
```

Optimize compile times

- https://dashbit.co/blog/how-to-debug-elixir-erlang-compiler-performance
  - https://dashbit.co/blog/speeding-up-re-compilation-of-elixir-projects
  - MIX_OS_DEPS_COMPILE_PARTITION_COUNT=$(sysctl -n hw.perflevel0.logicalcpu 2>/dev/null || sysctl -n hw.ncpu 2>/dev/null || nproc --all 2>/dev/null || getconf _NPROCESSORS_ONLN 2>/dev/null || echo 1)

Identify runtime dependencies causing large compilation cycles

Before this commit, LivebookWeb had runtime dependencies into the project, causing large compilation cycles.

Using the following command in Elixir v1.17.3+

```
$ mix xref graph --format stats --label compile-connected
```

Would reveal:
Top 10 files with most incoming dependencies:
- lib/livebook_web.ex (97)
  - lib/livebook/config.ex (3)
  - proto/lib/livebook_proto/deployment_group.pb.ex (2)

After this patch:

Top 10 files with most incoming dependencies:
- lib/livebook/config.ex (3)
  - proto/lib/livebook_proto/deployment_group.pb.ex (2)
  - lib/livebook_web/plugs/memory_provider.ex (2)

From: https://github.com/livebook-dev/livebook/commit/edefa6649ab783c1c2f6a2c067b44fa6dc9de642

Validating function keyword list input

If your function accepts a keyword list of options, you can use Keyword.validate!/2 to:

1. Validate that you only received the expected keys, and
2. Set default values for any or all of your expected keys

For instance, this will raise a really clear error:

```
iex> Keyword.validate!([falback: 123],
[:fallback, :timeout, :max_size])
** (ArgumentError) unknown keys [:falback] in
[falback: 123], the allowed keys are:
[:fallback, :timeout, :max_size]
```

Here's what it looks like to get back an updated keyword list with your default values interpolated into it:

```
iex> Keyword.validate!([fallback: 123], [fallback:
nil, timeout: 5_000, max_size: 100])
[fallback: 123, timeout: 5_000, max_size: 100]
```

Alias module within its definition

```
# Before
defmodule MyApp.Businesses.Business do
  def open(%__MODULE__{} = business) do
    Map.put(business, :open, true)
  end

  def close(%__MODULE__{} = business) do
    Map.put(business, :open, false)
  end
end

# After
defmodule MyApp.Businesses.Business do
  alias __MODULE__

  def open(%Business{} = business) do
    Map.put(business, :open, true)
  end

  def close(%Business{} = business) do
    Map.put(business, :open, false)
```

```
  end
end
```

Update LiveView node, but ignore specific attributes

https://x.com/thmsmlr/status/1901081275844194343

```
let liveSocket = new LiveSocket("/live", Socket, {
  longPollFallbackMs: 2500,
  params: { _csrf_token: csrfToken },
  hooks: Hooks,
  dom: {
    onBeforeElUpdated(fromEl, toEl) {
      if (fromEl.hasAttribute("phx-update-ignore"))
{
        const ignoreAttributes =
fromEl.getAttribute("phx-update-ignore").split(",");
        ignoreAttributes.forEach(attr => {
          if (fromEl.hasAttribute(attr)) {
            toEl.setAttribute(attr,
fromEl.getAttribute(attr));
          }
        });
      }
    }
  }
});
```

In LV v1.1 you will be able to do:
`phx-mounted={JS.ignore_attribute("style")}`
via https://x.com/josevalim/status/1931996863550439536

Disable os_mon in tests

In config/test.exs:

```
# Disable os_mon for tests
config :os_mon,
  start_cpu_sup: false,
  start_disksup: false,
  start_memsup: false
```

Jump into editor from iex

- Set VISUAL environment variable
  - Esc + O

Close iex

Ctrl + \

Make private functions available in dev/test

```
defmodule MyModule do
  if Mix.env() in [:dev, :test] do
    @compile :export_all
  end
end
```

Task.await_one

https://gist.github.com/bcardarella/4046fb0e644129c77b443bb4229993af

```
defmodule Task do
  defp await_one(tasks, timeout \\ 5_000) when
is_list(tasks) do
    awaiting =
      Map.new(tasks, fn %Task{ref: ref, owner:
owner} = task ->
        if owner != self() do
          raise ArgumentError,
invalid_owner_error(task)
        end

        {ref, true}
      end)

    timeout_ref = make_ref()

    timer_ref =
      if timeout != :infinity do
        Process.send_after(self(), timeout_ref,
```

```
timeout)
      end

    try do
      await_one(tasks, timeout, awaiting,
timeout_ref)
    after
      timer_ref && Process.cancel_timer(timer_ref)
      receive do: (^timeout_ref -> :ok), after: (0
-> :ok)
    end
  end

  defp await_one(_tasks, _timeout, awaiting,
_timeout_ref) when map_size(awaiting) == 0 do
    nil
  end

  defp await_one(tasks, timeout, awaiting,
timeout_ref) do
    receive do
      ^timeout_ref ->
        demonitor_pending_tasks(awaiting)
        exit({:timeout, {__MODULE__, :await_one,
[tasks, timeout]}})

      {:DOWN, ref, _, proc, reason} when
is_map_key(awaiting, ref) ->
        demonitor_pending_tasks(awaiting)
        exit({reason(reason, proc),
{__MODULE__, :await_many, [tasks, timeout]}})

      {ref, nil} when is_map_key(awaiting, ref) ->
        demonitor(ref)
        await_one(tasks, timeout,
Map.delete(awaiting, ref), timeout_ref)

      {ref, reply} when is_map_key(awaiting, ref) ->
        awaiting = Map.delete(awaiting, ref)
        demonitor_pending_tasks(awaiting)

        reply
```

```
    end
  end

  defp demonitor_pending_tasks(awaiting) do
    Enum.each(awaiting, fn {ref, _} ->
      demonitor(ref)
    end)
  end

  defp reason(:noconnection, proc), do: {:nodedown,
monitor_node(proc)}
  defp reason(reason, _), do: reason

  defp monitor_node(pid) when is_pid(pid), do:
node(pid)
  defp monitor_node({_, node}), do: node

  defp demonitor(ref) when is_reference(ref) do
    Process.demonitor(ref, [:flush])
    :ok
  end

  defp invalid_owner_error(task) do
    "task #{inspect(task)} must be queried from the
owner but was queried from #{inspect(self())}"
  end
end
```

Pipe vs With

Pipe when you know each step will run successfully and you don't have error cases to worry about (typically because you've already checked). All steps of a pipe (or all but the very last, in some cases — Ecto query composition feeding into an actual query, for example) should return some value, rather than an ok/error tuple.

Use with when you have a multi-step operation with possible failures to account for. Typically many steps of a with-chain will return ok/error tuples.

Pattern match on any struct

```
iex(1)> %{__struct__: _} = %Order{id: 10}
%Order{id: 10}
# Shorthand
iex(2)> %_{} = %Order{id: 10}
%Order{id: 10}
```

For even more fun, you can capture the struct module:

```
def add_one_hour(%date_or_time{} = value) when
date_or_time in [DateTime, NaiveDateTime, Time] do
  ...
end
```

https://katafrakt.me/til/short-notation-for-pattern-matching-on-any-struct-in-elixir/
https://bsky.app/profile/tylerayoung.com/post/3mdgp5k4o7c2i