Gleam on Beam: Elixir’s safe escape hatch

Briefly about me:

- Diploma degree in Sociology, Economics & Psychology.
- Full Stack Dev since 1997/2001.
- Contributor to Gleam’s stdlib: https://github.com/gleam-lang/stdlib 😄 and a bit to Gleam Playground.
- When looking for new tech, I have tried a bunch of technologies, which included picking Phoenix and Flutter for creating mobile apps for iOS/Android.

Story/1

PICKING NEW TECHNOLOGIES

- For the mobile app I picked Dart/Flutter as the frontend technology.
  - Nowadays one could try https://github.com/elixir-desktop/desktop (Elixir-Desktop), which lets you run Elixir code on iOS/Android as well as Mac, Linux and Windows.
- For the backend I picked Elixir/Phoenix.
- I had a great time learning both, but I really missed the typing in Elixir/Phoenix while onboarding.

Story/2

FIRST EXPERIENCES

- The first iterations of the backend/API (Phoenix) some runtime match errors sneaking into staging.
- Errors were due to using pattern matching and not enough ``` `with` ``` or change-sets for validation.
- I simply happen to *"let it crash"* - unreasonably though.

Story/3

DART/FLUTTER

- The Dart/Flutter experience was different.
- Bugs existed as well, but the overall onboarding was awesome.
- Reasons:
  - Flutter was mostly declarative and DOM-like.
  - Flutter/Dart tooling made it rewarding to ***not*** use `` `dynamic` ``, but almost always use static typing.
  - Why? →

Story/4

DX BENEFITS OF STATIC TYPING

- The IDE (vscode) showed type hints right away.
- Without any delay (like in Elixir with ElixirLS, type specs and Dialiyzer).
- The IDE offered options to pick and function documentation alongside when moving through the vast amounts of Flutter build-in types/components to compose the application of.

➔ Dart has its culprits but I clearly missed that kind of type safety and DX in Elixir/Phoenix.

Story/5

"LET IT CRASH!"

1. `Bad crashes`: Application errors happen at runtime and we just do not handle them.
   - When taking over larger Phoenix projects I have for instance seen a lot of `(MatchError) no match of right hand side value` in Sentry.io logs.
2. `Good crashes`: A failing process does not need to kill the system. This also allows for self healing where certain networked resources are not available briefly.
   - On the BEAM without global state and objects we can more or less safely just crash and restart processes.

Disclaimer: Obviously crashes are *NEVER* good! 💩

Story/6

"LESSONS LEARNED!"

Erlang and Elixir help a lot on following happy paths:

- When resources, an application requires, are set to be available under usual circumstances, it runs.
- When such resources are temporarily unavailable, some processes crash and restart.

➔ The `` `good crashes` `` are handled by well written Beam/OTP apps.

What about the `` `bad crashes` ``?

- A lot of runtime errors can occur that have nothing to do at all with unexpected errors.
- Change-sets or other forms of input validation help.
- Type specs, and schemata such as Json-Schema, XML-Schema/XSD, GraphQL help.
- Unit tests help.

➔ Static typing can help to avoid many of these bad crashes

Story/7

TYPES TO THE RESCUE

- Erlang community has tried to fix this within and outside of Erlang, see https://github.com/stars/michallepicki/lists/erlang-and-static-types.
- Reason for tools like Dialzyer and to some degree Credo.
- Reason for numerous attempts to bring typing to Erlang, the last one being `` `erlt` `` by the WhatsApp-team, released into the public in spring 2021: https://github.com/WhatsApp/erlt.
- Reason for numerous to bring typing to Erlang or write typed languages against the Beam VM such as: Lisp-Flavoured-Erlang, Alpaca, Purerl, Elchemy, Hamler, Caramel and Gleam.

Gleam/1

WHAT GLEAM AVOIDS

Great care around simplicity and DX:

- Many of the attempts to bring typing to the Beam make it hard to pick up for developers not familiar with say Lisp, Haskell, PureScript, Elchemy or OCaml.
- And let’s be honest: Compared to say JavaScript these are ultra-niche.
- Why? Next to immutability, recursion and strict typing developers are also forced to know these and/or understand these niche technologies and be willing to read their syntax and live with their abstractions.

Gleam/2

WHAT SETS GLEAM APART

1. Average developer happiness matters:
   - The Gleam language is developed with great care limiting the languages' *strangeness-budget*.
   - Syntax appears to be familiar to JavaScript.
   - Not every feature one could think of or desire is implemented.
   - The language interface/surface is being kept small for this reason (Chess vs Go).
   - It should be easy to pick up.
   - Expressive and clear error messages.
   - Powerful type inference to get started or for the lazy (appears if types are optional or dynamic).
2. Compiles to targets Erlang and JavaScript.
   - Once there is more financial support the main developer would like to add C/native as a compile target.
3. The compiler is written in Rust:
   - Compiling Gleam is ultra snappy and at the same time yields compile time guarantuees.
   - Can be compiled via WASM within the browser - Gleam' playground uses this.
4. Interacts with Erlang/OTP, Browser-Javascript, NodeJS and possibly Deno.

Gleam/3

Gleam/4

With all that being said, let’s dive in a bit!

Gleam/5

DEMO TIME

Demo and this talk in available here: https://github.com/inoas/gleam-elixir-phoenix-liveview-counter.

Run at home? Make you have got Erlang, Elixir, Gleam and Rebar installed. See below for some instructions.

Run demo app via:

```
asdf install erlang 25.0
asdf install elixir 1.13.4-otp-25
asdf install nodejs 18.2.0
asdf install gleam 0.21.0
# Make sure to run following outside of your gleam project dir and with the elixir version your gleam up requires by `.tool-
mix archive.install hex mix_gleam
bin/dev/run && open http://localhost:4000
```

Run the slides via:

```
asdf install nodejs 18.2.0
bin/dev/slides && open http://localhost:3030
```

DEMO LINK

Gleam/6

HOW TO INSTALL GLEAM

1. Install ASDF https://asdf-vm.com/guide/getting-started.html

```
brew install asdf # Mac OS only, other instructions above
```

2. Install Erlang, NodeJS, Gleam

```
asdf install erlang latest
asdf install elixir latest
asdf install nodejs latest
asdf install gleam latest
```

3. Create a Gleam dummy app, run tests on both targets:

```
gleam new my_app
cd my_app
gleam test --target erlang; gleam test --target javascript
```

If you want to use Gleam inside an Elixir project, see: https://github.com/gleam-lang/mix_gleam.

Gleam/7

PLAYTIME

- Toying around together on https://johndoneth.github.io/gleam-playground.
- Caveat: Only one module, no dependencies except gleam's included prelude and `gleam_stdlib`.
- Going through some examples found here: https://gleam.run/book/tour.

Gleam/8

PLAYGROUND EXAMPLES

- Strings attached
  - Strings and StringBuilder, there is also BitStrings
- Very expressive
  - Keywords: Expressions everywhere: Functions, anonymous functions, case statements, expression blocks
- Lovely patterns
  - Keywords: Discards ands spreads
- Pipe and capture
  - Keywords: Piping, capture symbol, complete functions assumed to return functions to push the pipe value into
- Bonus: Racing without safety belts
  - Keywords: assert, pipelines

Closing words/1

IS GLEAM PRODUCTION READY?

To my knowledge the main author certainly thinks so. People are using it in production. In the end Gleam generated rather readable Erlang (or JavaScript).

ANYTHING BAD?

- Complete exhaustiveness checks on `` `case` `` statements.
- The compiler, to be able to deal with broken/partial ASTs and still offer good help
  - Thus that we can have autosuggestions/completion and better IDE tooling.

💬 To my knowledge, these features are on the core developer’s lists.

Closing words/2

THE STATE OF MATTER

- The language is pretty stable:
  - The main author has stated that they intend to not break any language syntax or core language interfaces.
  - Syntax high-lightning and the first LSP is available.
  - Github officially supports the language.
- The eco-system needs help, libraries, more developers, and more users:
  - A lot of things already exist: HTTP1/2 servers, HTTP clients, JSON decoders, Protocol-Buffers, PostgreSQL client, Mustache and Matcha templating and many more.
  - For more see https://github.com/gleam-lang/awesome-gleam.

Closing words/3

THANKS & SHOUT-OUTS

- Joe & Team for creating Erlang!
- José & Chris for creating Elixir, Phoenix & LiveView!
- Louis for creating Gleam!
- The Gleam community and especially *michallepicki*, *rattard* and *HarryET* for helping out!

Closing words/4

BECOME SHINY

The Gleam community is very friendly, not snobbish at all, and welcomes developers of all trades and levels!

😄 **Gleam needs you - Gleam wants you! You want Gleam - you need Gleam!** 😄

... and no, it is not going to replace Elixir anytime soon!

- Discord: https://discord.gg/twY7ZhKTM3
- GitHub Discussions: https://github.com/gleam-lang/gleam/discussions
- Gleam on the web: https://gleam.run