

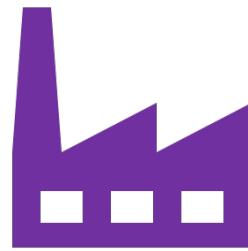


# Elixir Lang Now & beyond

**Start simple, go far**

March 2023, updated June 2024

# Product Lifecycle



Start

Idea

Continue

POC

MVP

Release

Stop

Decommissioning

...

# This is the situation - 1st iteration



Product Owner



Developers



The Codebase

Asks for an POC to be made.

Team structure:

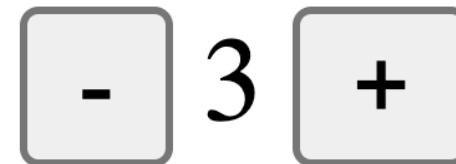
index.html (with some js code)

Title: Counter example web app

- 1 Ruby dev

Specs: Should INC/DEC

- 1 React dev
- 1 Scrum Master
- 1 QA



## This is the situation - 2nd iteration



Asks for a new increment of the application for the MVP.

the new specs are:

- It should save the number to DB
- It should be a SPA with a nice UI/UX touch



Team structure:

- 2 Ruby dev
- 2 React dev
- 1 Scrum Master
- 1 QA

Teams of two for redundancy and code reviews.



The initial index.html has been refactored 🤖 entirely.

A new Ruby project has been created.  
A new React project has been created.

- 3 +

# Here's the Problem

The MVP web application is a success

PO: Let's deliver it also as a  
native mobile real-time app... globally!



...

👨‍💻 Is the architecture scalable?

👩‍💻 Are new skills needed in the team?

👨‍💼 Am I over the budget?



Elixir Lang - Now & beyond

# Here's why it is hard to solve

Budgets, time, devs & computers are limited resources

- o Needed skills are hard to be find;
  - o Big teams are less productive than small ones;
  - o Human relations are hard to be established;
  - o Without adding extra complexity layers, the tech stack is not able to meet our new demands;
  - o Complex projects are hard to be maintained and prone to errors. That's a fact;
  - o Consistency (both visual and functional), it's another 🐘 in the room;
  - o Uncertainty is counterproductive;
- ...



# Productivity killers

1

Tech stack:



or



Big teams:

web

API

iOS

android

cross



3 or 4 teams that do not know the business logic of the app;  
frequent syncs needed.

Orchestration:



How do we organize?



What/how are we building?



What are we testing?



Elixir Lang - Now & beyond

# The Solution - Elixir

A mature, battle tested language with a large ecosystem

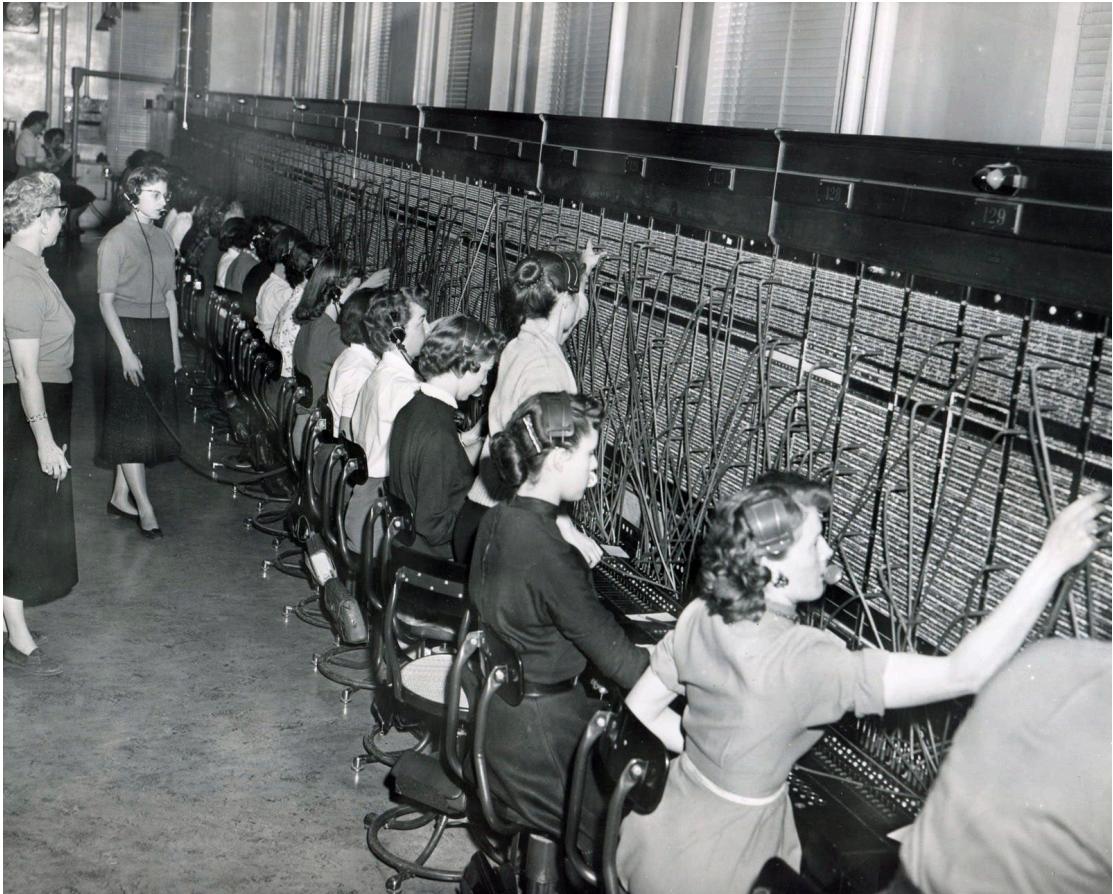
Various IT domains can be handled by this single language, which reduces the mixing of technologies as the application develops.

Domain	Project
 <b>Web Applications</b>	Phoenix, LiveView
 <b>Machine learning &amp; AI</b>	NX Project
 <b>Native Applications</b>	Phoenix, LiveView Native
 <b>Embedded/IoT</b>	Nerves, Firefly



Elixir Lang - Now & beyond

# Short History



Elixir Lang - Now & beyond

## BEAM Languages

Elixir  
Erlang  
Gleam  
Alpaca

...



**ERICSSON** ⚡

2000s

1986

# Why Erlang has been created?

Demanding requirements of TelCo systems back in the 1980s:

- o **Real-time** systems - Processes data with minimal delay.
- o **Concurrent** systems - Multiple tasks running simultaneously.
- o **Distributed** systems - Networked systems sharing resources.
- o **Fault-tolerance** - Ability to continue functioning after failures.



## Limitations of Existing Programming Languages

Inefficient handling of concurrency  
Poor support for distribution – client/server  
Difficulties with fault-tolerance

## Key Features of Erlang

Lightweight processes (BEAM process)  
Message-passing concurrency (Actor/Model)  
Fault-tolerance  
Distributed – run on multiple nodes  
Hot code swapping - Fix while running

Elixir Lang - Now & beyond

# Why Elixir has been created?

Elixir was created to provide developers with a modern, productive, and scalable language for building distributed and fault-tolerant systems.

Demanding requirements of systems  
back in the 1980s ...

- o **Real-time** systems - Processes data with minimal delay.
- o **Concurrent** systems - Multiple tasks running simultaneously.
- o **Distributed** systems - Networked systems sharing resources.
- o **Fault-tolerance** - Ability to continue functioning after failures.

... are the same NOW!

Elixir Lang - Now & beyond



# Companies using Elixir



**change.org**

**bet365**



**PEPSICO**



They migrated from Rails to Phoenix and reduced app servers from 150 to just eight. They also reduced their team size as they no longer required as many developers to support their ongoing needs.

Elixir Lang - Now & beyond

# The Solution - Elixir

Tech stack:



One team:

Cross Platform - truly native



A single codebase, singular business logic implementation.

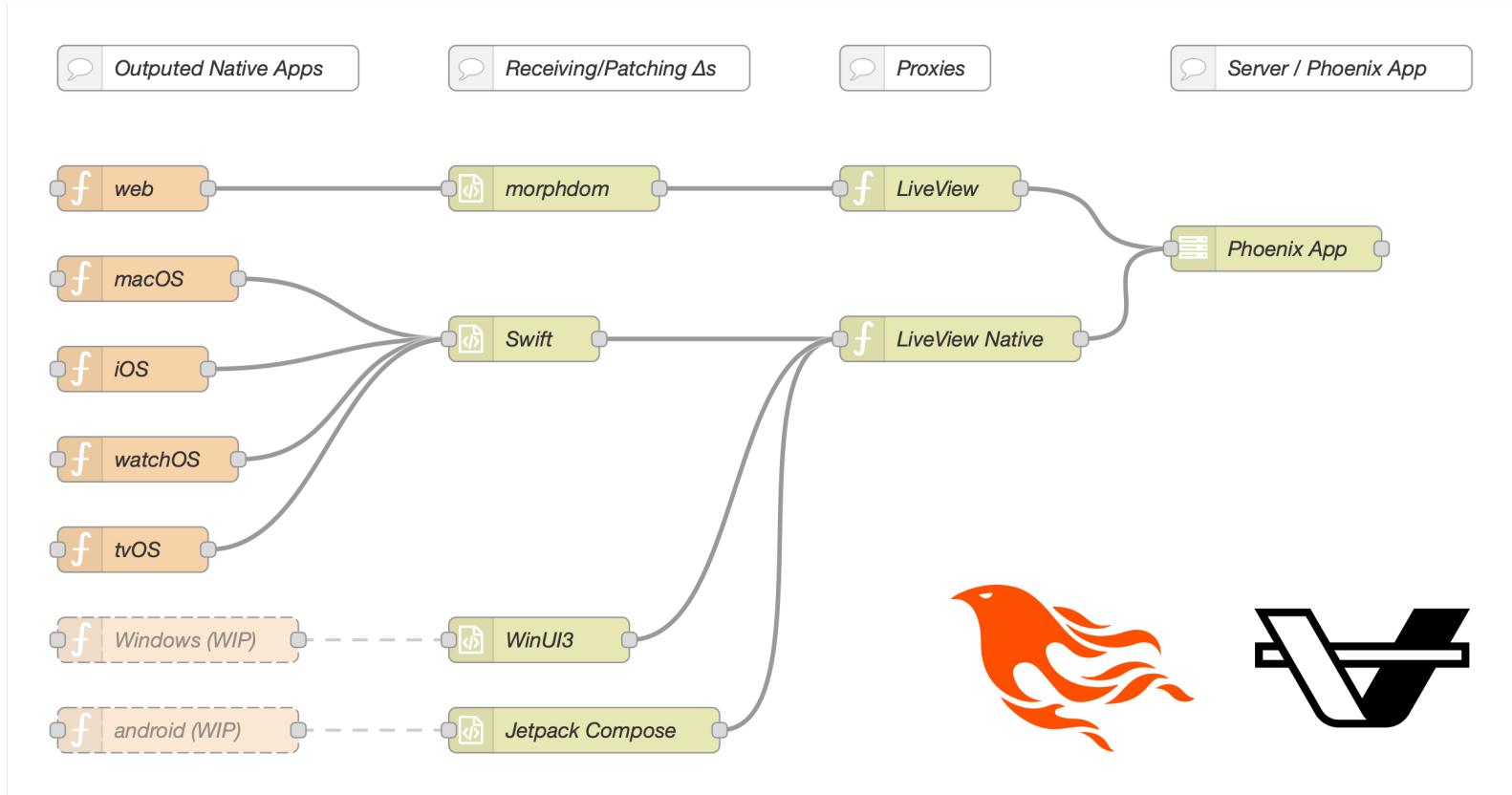
Orchestration:

-  We know what to deliver.
-  We know what to build.
-  We know what to test.



Elixir Lang - Now & beyond

# LiveView/LiveView Native

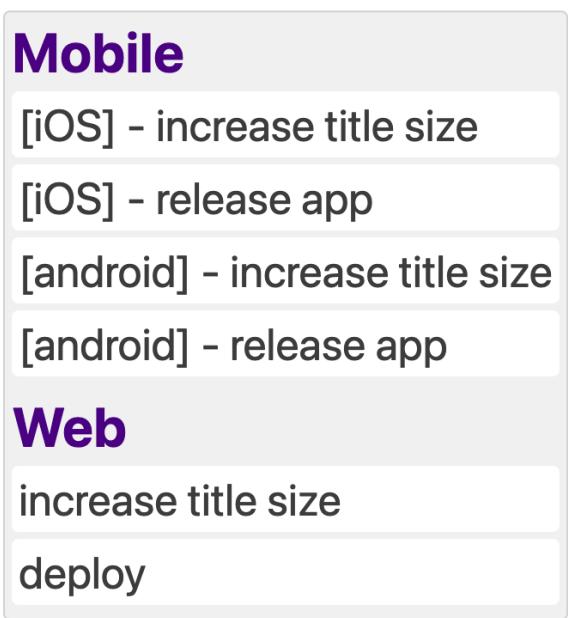


Elixir Lang - Now & beyond

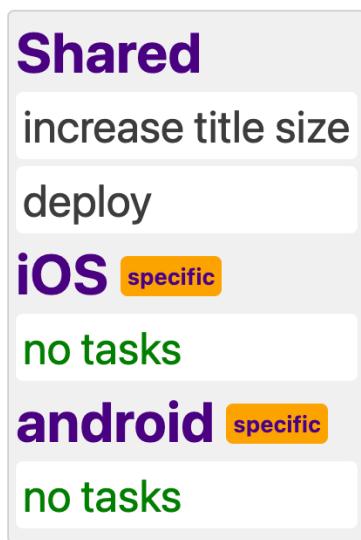
# Project management

Request: Increase title size

Acceptance criteria: Expect the title size to be bigger across the apps.



VS



specific Targets a specific platform matter.

# Costs & Consistency

CI/CD Costs

Less builds/less money



Consistent release

No need to wait for AppStore/  
Google Play



App Store



Google play



Elixir Lang - Now & beyond

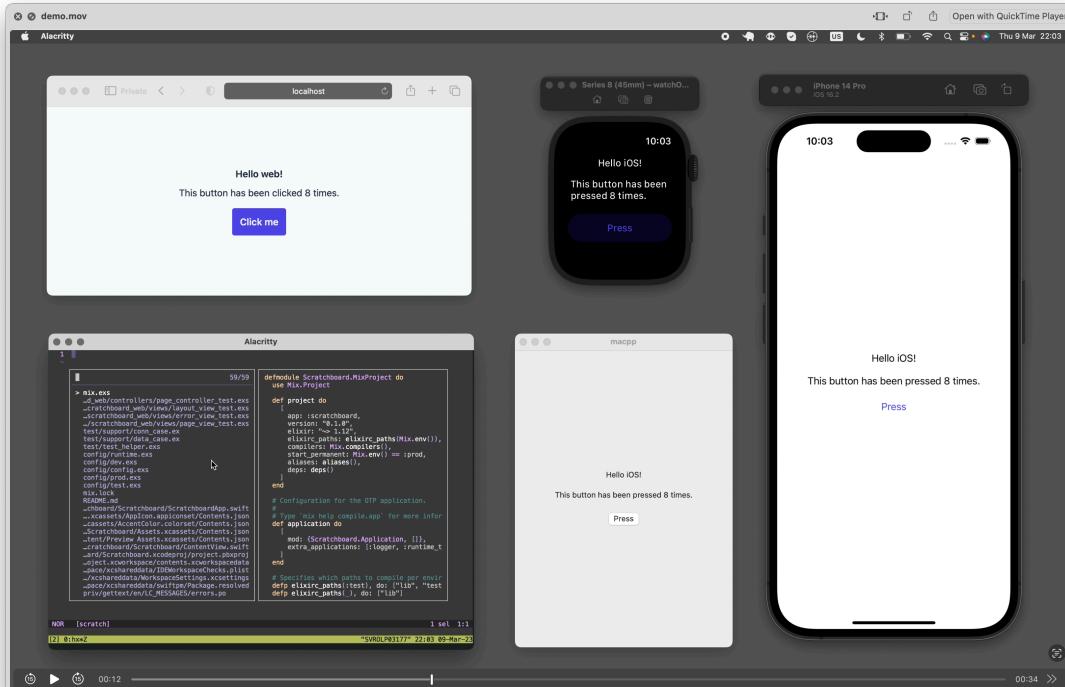
# Demo

Presentation:

<https://youtu.be/X4Hh8Oae9fk>

Demo

LiveView web app with LiveView  
Native MacOs, WatchOS and iOS  
apps. Rendered via a single  
Phoenix app



Elixir Lang - Now & beyond

# Thank you

Constantin Angheloiu

[constantin.angheloiu@gmail.com](mailto:constantin.angheloiu@gmail.com)  
[@cmnstmntmn](https://twitter.com/cmnstmntmn)