FULL-STACK SCALA

JONAS CHAPUIS, PH.D. TERASOL

What? *front-end* in Scala?

What? *front-end* in Scala?

Yep, *front-end* and even *ops* in Scala 🚀

Cool, but what for? 🤔

Who *wouldn't* want... 🤩

→ 👬 *Type-based* team alignment

→ 🔷 Smaller & leaner codebase

→ 👣 Reduced overall dependency surface

→ 💄 Cutting-edge UX

→ 🚀 WASM performance with flawless interop

Past 10 months: my full-stack journey with Scala

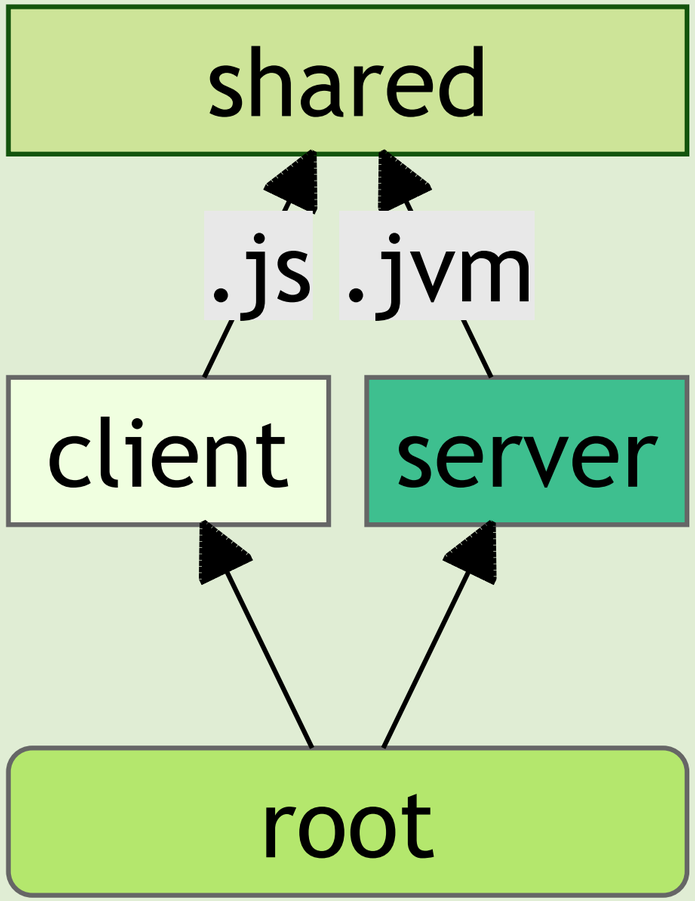
→ After 7 years of backend & microservices about 🚐 and 🚗

→ Asphalt is too 🥵! Let's plant some 🌳

→ `http://exomap.com`


Exomap

`build.sbt` structure



`shared` **project namespaces**

→ 📁 **`endpoints`**: endpoint definitions (using **`tapir`**)

→ 📁 **`model`**: data classes

→ 📁 **`i18n`**: translation data classes

tapir endpoint definitions

```scala
import sttp.tapir.*
import sttp.tapir.json.upickle.*
import com.terasol.exomap.shared.i18n.Translations
import com.terasol.exomap.shared.model.Language
import sttp.tapir.codec.iron.given

trait I18nEndpoints:
  val i18n = endpoint.get
    .in("i18n" / path[Language]("lang"))
    .out(jsonBody[Translations])
    .description("Get translations for a certain language")
    .name("i18n")
```

Iron types


```scala
import io.github.iltotore.iron.*
import io.github.iltotore.iron.constraint.string.*

type Language = Language.T
object Language extends RefinedType[String, Match["^(en|fr|de|es|it)$"]]
```

→ 💅 Compile-time & runtime type "refinements"

→ ✅ Capture domain value constraints

→ ⏬ Thanks to modules ecosystem, all the way into the DB 🛢️ !

Codec derivation

```scala
import upickle.default.*

case class Translations(popup: PopupLabels, map: MapLabels, settings: SettingsLabels) derives ReadWriter
case class PopupLabels(computing: String, parcel: String) derives ReadWriter
// ...
```

→ ⚙️ autogeneration of **`ReadWriter`** JSON codec

→ 1️⃣ plain case classes, single definition

client project

```
lazy val client = project
  .in(file("client"))
  .dependsOn(shared.js)
  .enablePlugins(ScalaJSPlugin)
  .enablePlugins(ScalablyTypedConverterExternalNpmPlugin)
  .settings(
    scalaJSUseMainModuleInitializer := true,
    scalaJSLinkerConfig ~= {
      _.withModuleKind(ModuleKind.ESModule)
        .withModuleSplitStyle(ModuleSplitStyle.SmallModulesFor(List("exomap")))
    },
    externalNpm := baseDirectory.value,
    libraryDependencies ++= Client.dependencies.value,
  )
```

Just a standard Vite project ⚡

 vite.config.js

```
import scalaJSPlugin from "@scala-js/vite-plugin-scalajs";
```

Just a standard Vite project: `package.json`

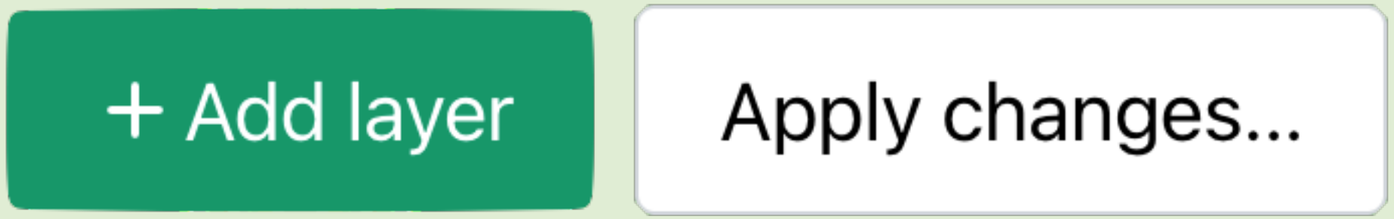
```json
{
  "name": "exomap",
  "private": true,
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview",
    "clean": "rm -rf dist"
  },
  "devDependencies": {
    "@scala-js/vite-plugin-scalajs": "^1.0.0",
    "autoprefixer": "^10.4.20",
    "tailwindcss": "^4.1.5",
    "typescript": "^5.8.2",
    "vite": "^6.2.0"
  },
  "dependencies": {
    "@tailwindcss/vite": "^4.1.5",
    "flowbite": "^3.1.2",
    "maplibre-gl": "^5.2.0"
  }
}
```

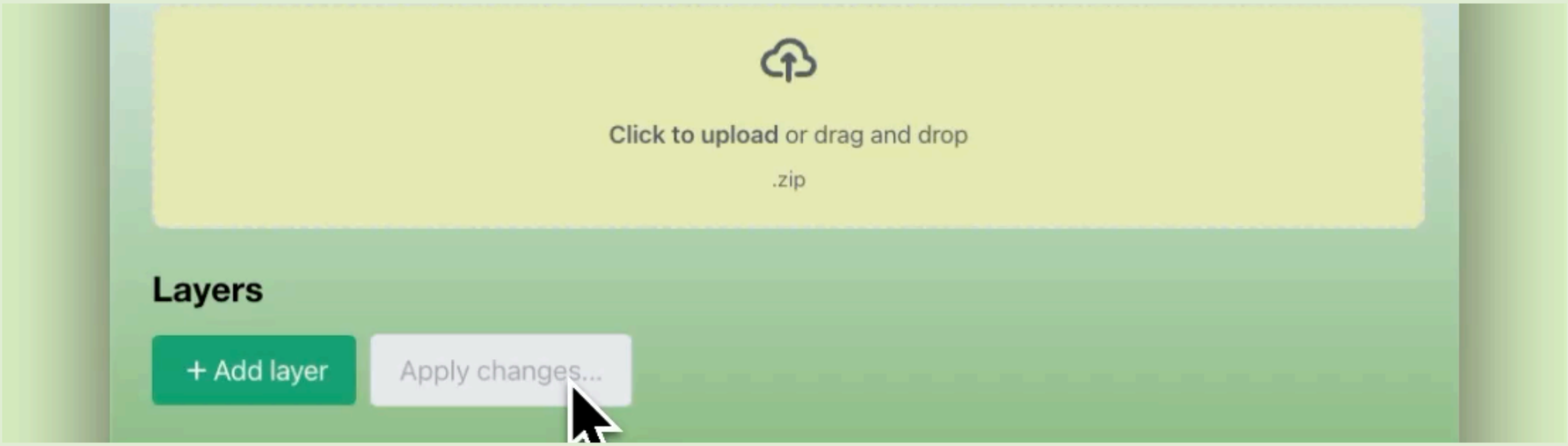
Web app entry point 🚪

```scala
@main
def Exomap(): Unit =
  renderOnDomContentLoaded(
    dom.document.getElementById("app"),
    rootElement()
  )

def rootElement() = div( ... )
```

```html
<!DOCTYPE html>
<html>
  <body>
    <div id="app"></div>
    <script type="module">import 'scalajs:main.js';</script>
  </body>
</html>
```

Let's create a button with Laminar!



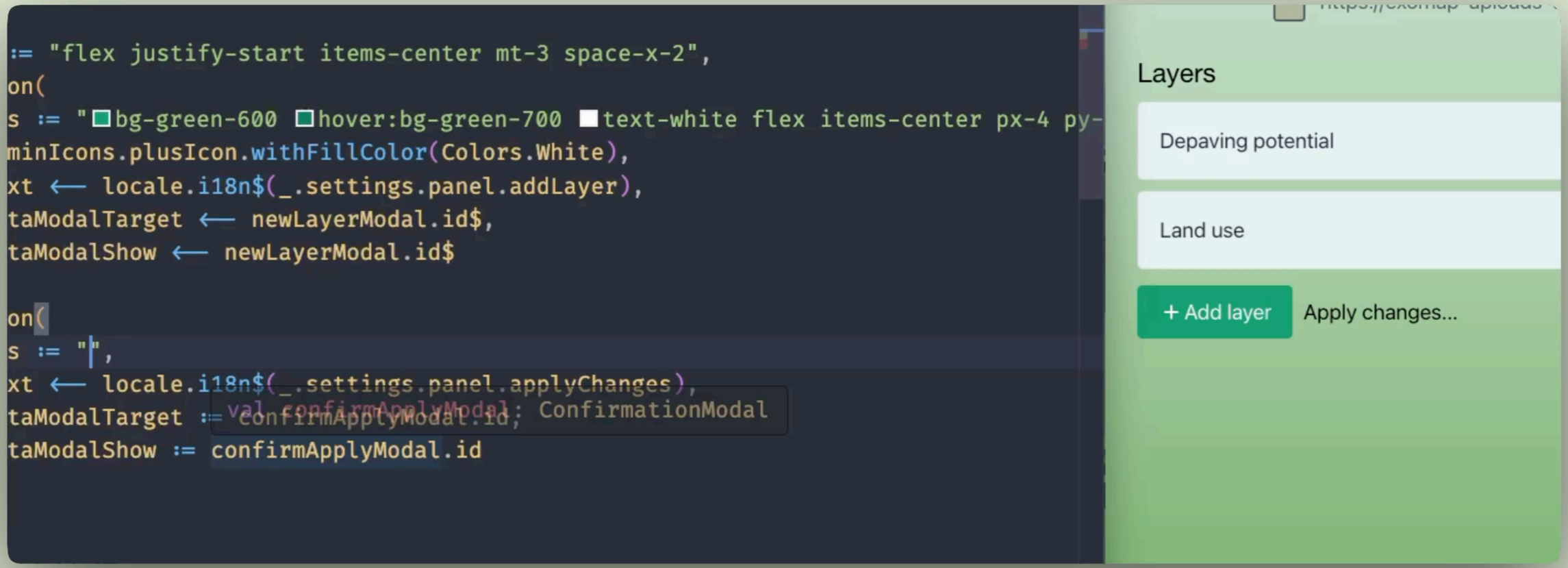


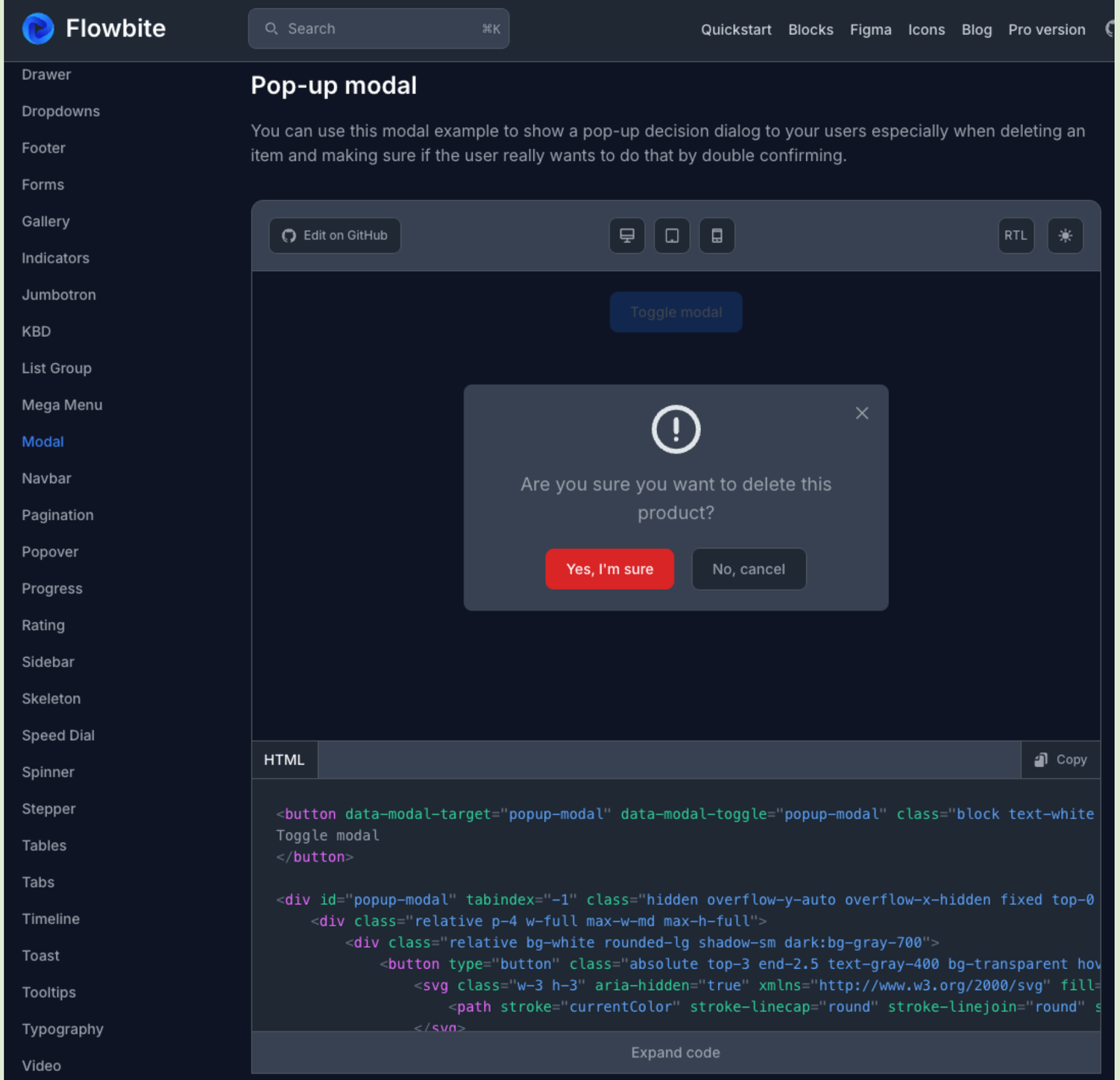
Laminar elements

```
import com.raquo.laminar.api.L.*

def applyChangesButton = button("Apply changes")
```

Tailwind CSS styling

```
import com.raquo.laminar.api.L.*

def applyChangesButton = button("Apply changes",
  cls := """bg-white hover:bg-gray-200
  border-1 border-gray-300
  disabled:text-gray-400
  px-4 py-2
  rounded""")
```


:= "flex justify-start items-center mt-3 space-x-2",
on(
s := "bg-green-600 hover:bg-green-700 text-white flex items-center px-4 py-
minIcons.plusIcon.withFillColor(Colors.White),
xt ⟵ locale.i18n$(_.settings.panel.addLayer),
taModalTarget ⟵ newLayerModal.id$,
taModalShow ⟵ newLayerModal.id$
on(
s := "",
xt ⟵ locale.i18n$(_.settings.panel.applyChanges),
taModalTarget := confirmApplyModal.id,
taModalShow := confirmApplyModal.id
val confirmApplyModal: ConfirmationModal
Layers
Depaving potential
Land use
+ Add layer
Apply changes...


Flowbite
Search ⌘K
Quickstart Blocks Figma Icons Blog Pro version
Drawer
Dropdowns
Footer
Forms
Gallery
Indicators
Jumbotron
KBD
List Group
Mega Menu
Modal
Navbar
Pagination
Popover
Progress
Rating
Sidebar
Skeleton
Speed Dial
Spinner
Stepper
Tables
Tabs
Timeline
Toast
Tooltips
Typography
Video
Pop-up modal
You can use this modal example to show a pop-up decision dialog to your users especially when deleting an item and making sure if the user really wants to do that by double confirming.
Edit on GitHub
RTL
Toggle modal
Are you sure you want to delete this product?
Yes, I'm sure
No, cancel
HTML
Copy
<button data-modal-target="popup-modal" data-modal-toggle="popup-modal" class="block text-white
Toggle modal
</button>
<div id="popup-modal" tabindex="-1" class="hidden overflow-y-auto overflow-x-hidden fixed top-0

<button type="button" class="absolute top-3 end-2.5 text-gray-400 bg-transparent hov
<svg class="w-3 h-3" aria-hidden="true" xmlns="http://www.w3.org/2000/svg" fill=
<path stroke="currentColor" stroke-linecap="round" stroke-linejoin="round"
Expand code

Flowbite how-to

Copy HTML to 📋

```
<div id="popup-modal" tabindex="-1" class="...">
  <div class="relative p-4 w-full max-w-md max-h-full">
    ...
```

👇 convert elements to

```
div(id:="popup-modal", tabIndex := -1, cls:="...",
  div(cls:="relative p-4 w-full max-w-md max-h-full"),
    ...
)
```

Optional: for built-in JS, include dependency

```
<body>
  <div id="app"></div>
  <script type="module" src="./node_modules/flowbite/dist/flowbite.min.js"></script>
  <script>import 'scalajs:main.js';</script>
</body>
```

and define custom attributes

```
val dataModalTarget: HtmlAttr[String] = htmlAttr("data-modal-target", StringAsIsCodec)
val dataModalHide: HtmlAttr[String] = htmlAttr("data-modal-hide", StringAsIsCodec)
val dataModalShow: HtmlAttr[String] = htmlAttr("data-modal-show", StringAsIsCodec)
val dataModalToggle: HtmlAttr[String] = htmlAttr("data-modal-toggle", StringAsIsCodec)
```

Ok, but let me press this button! 👊

```
import com.raquo.laminar.api.L.*

def applyChangesButton = button("Apply changes",
  cls := """bg-white hover:bg-gray-200
  border-1 border-gray-300
  disabled:text-gray-400
  px-4 py-2
  rounded""",
  onClick ⟶ (event ⇒ dom.console.log(s"Button clicked: ${event.button}")))
```

Internationalization 🌐

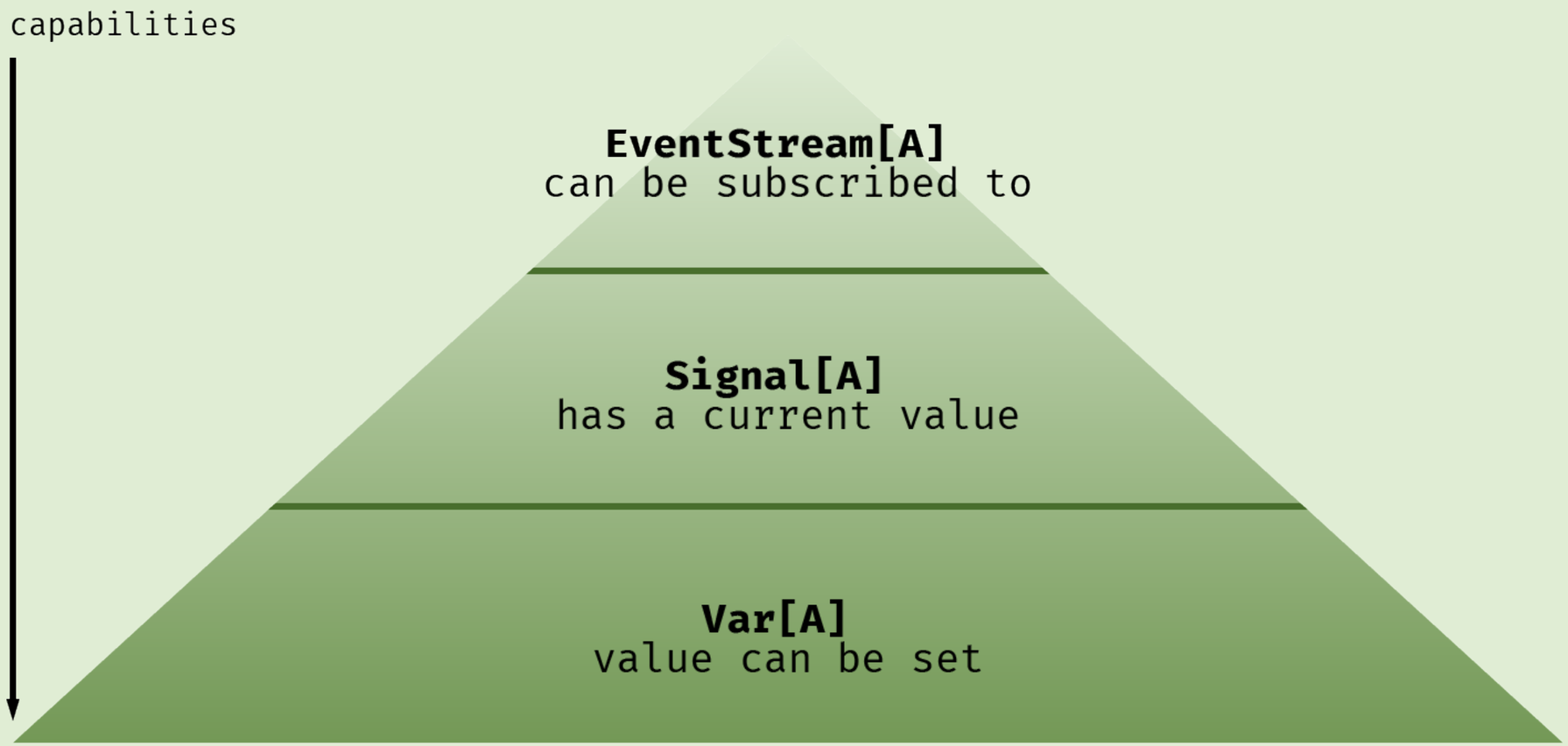
```scala
import com.raquo.laminar.api.L.*

def applyChangesButton(using locale: LocaleState) =
  button(
    cls := """bg-white hover:bg-gray-200
    border-1 border-gray-300
    disabled:text-gray-400
    px-4 py-2
    rounded""",
    onClick --> (event => dom.console.log(s"Button clicked: ${event.button}")),
    text <-- locale.i18n$(_.settings.panel.applyChangesButton))
```

Vars and signals 🔌

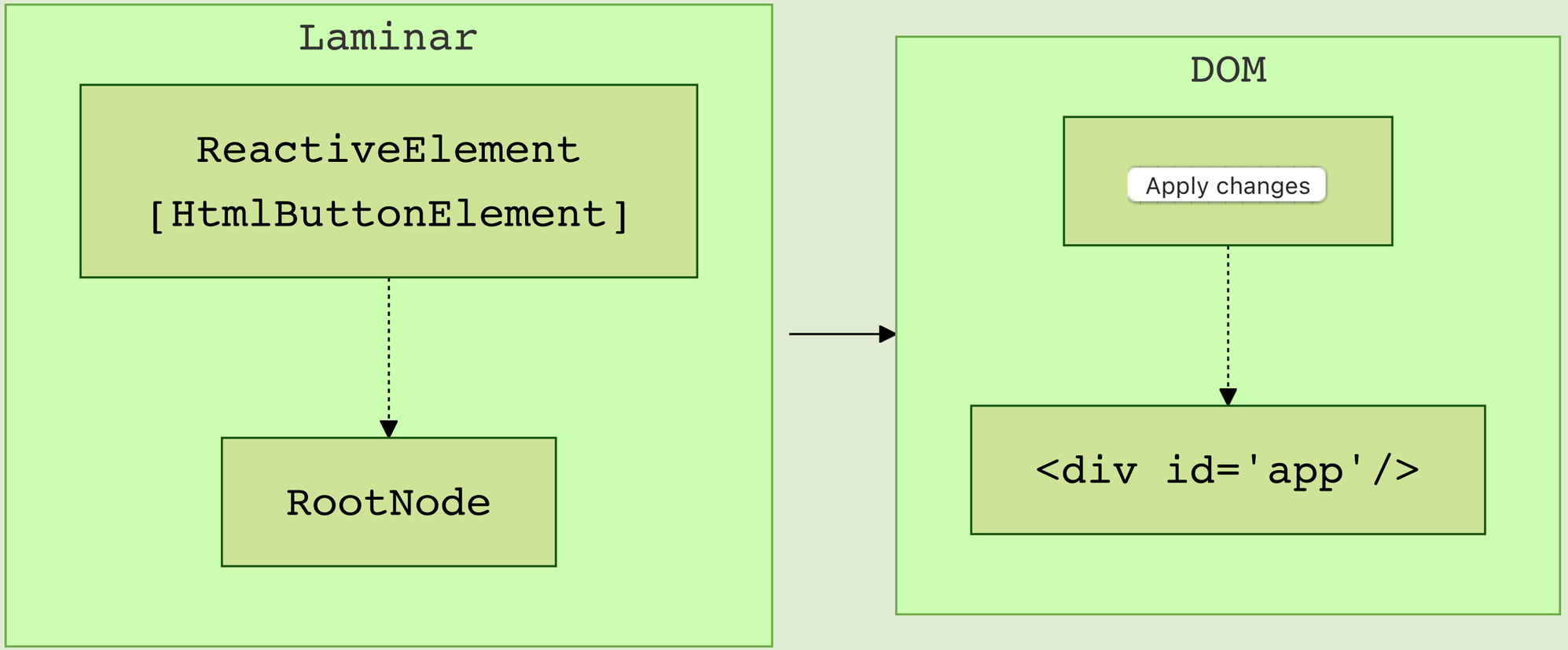
```scala
import com.raquo.airstream.state.Var
import com.raquo.airstream.core.Signal
import com.terasol.exomap.shared.i18n.Translations

trait LocaleState:
  val currentTranslations: Var[Translations]

  def i18n$(selector: Translations ⇒ String): Signal[String] = currentTranslations.signal.map(selector(_))
```

→ subscription: `text ⟵ locale.i18n$ (_.settings.panel.applyChangesButton)`

Fundamental abstractions in



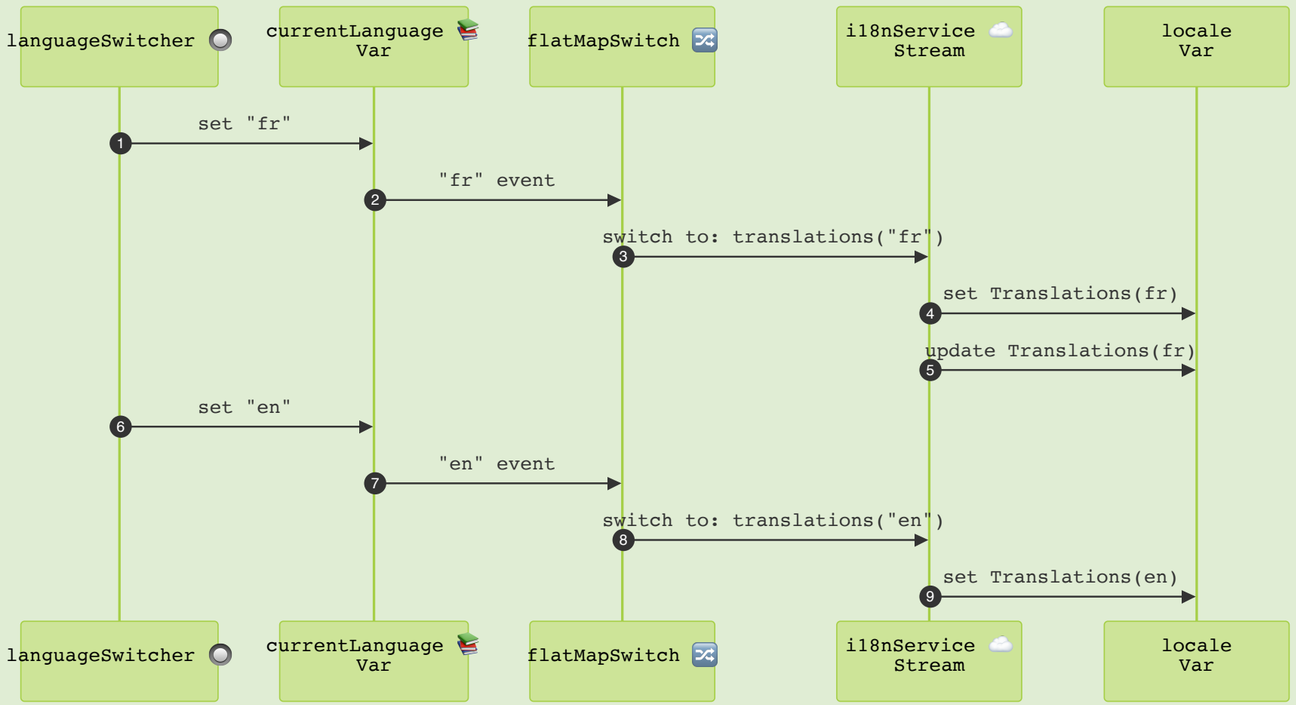
Laminar's parallel 🌳



Behaviors: functional reactive logic 🛝

```
def i18nBehavior()(using locale: LocaleState, i18nService: I18nService): Modifier.Base =
  locale.currentLanguage.signal.flatMapSwitch(i18nService.translations) ⟶ locale.currentTranslations
```

Essential FRP operator: flatMapSwitch 🛝

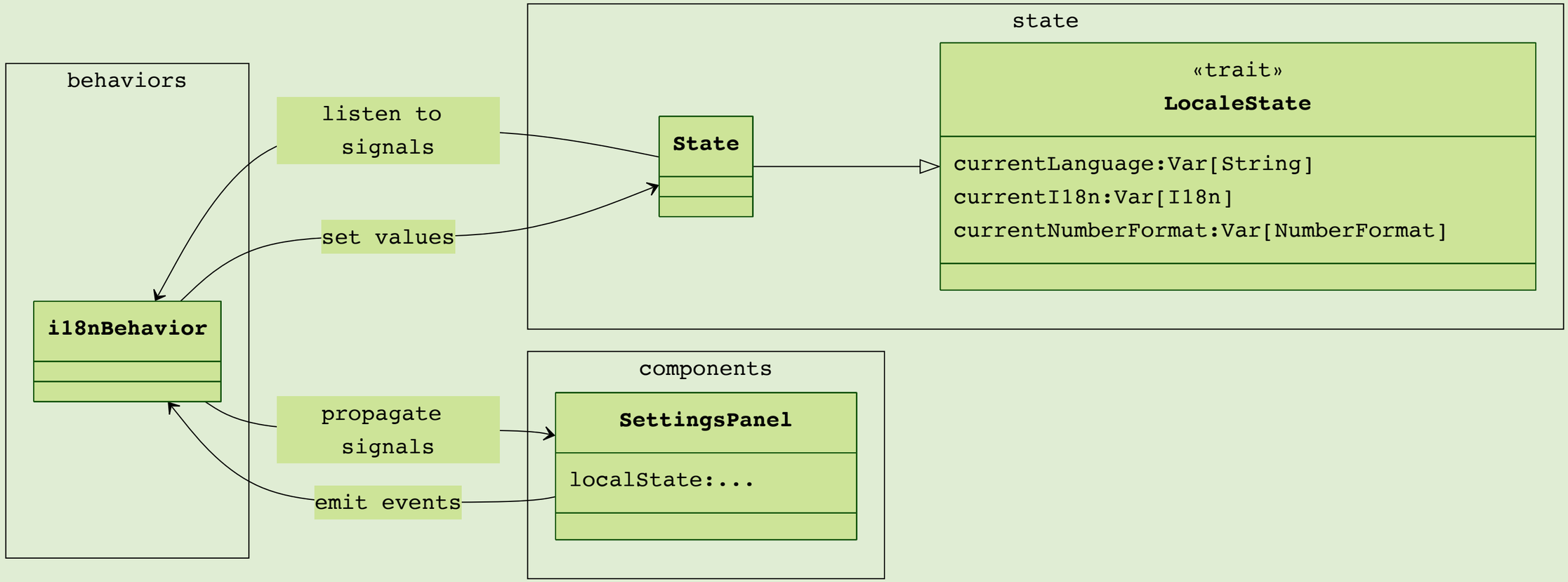


`currentLanguage$` ***flatMapSwitch*** `i18nService.translations ⟶ locale.currentTranslations`

Behavior ⚙️ unit coverage ✅

```
property("translations match the current language"):
  forAll: (additionalTranslations: Map[Language, Translations]) ⇒
    val translationsMap = additionalTranslations + (Language("en") → Translations.default)
    given locale: LocaleState = mockedLocale
    given I18nService = new I18nService:
      def translations(language: Language) = EventStream.fromValue(translationsMap(language))

    withActivatedSubscriptions(i18nBehavior()):
      translationsMap.keys.foreach(locale.currentLanguage.set)
      assertEquals(
        locale.currentTranslations.now(),
        translationsMap(locale.currentLanguage.now())
      )
```

Application state management 🗄️



Redux in Laminar

```scala
import com.raquo.laminar.api.L.*

class ReduxBehavior[Action, State, Effect](
    actionsBus: EventBus[Action],
    maybeState: Var[Option[State]],
    reducer: (Action, Option[State]) ⇒ (Option[State], List[Effect]),
    effector: EventStream[Effect] ⇒ EventStream[Action]
):
  def modifier: Modifier.Base =
    val reducedActionsStream = actionsBus.toObservable.withCurrentValueOf(maybeState).map(reducer.tupled)
    val reducedStateStream = reducedActionsStream.map:
      case (newState, _) ⇒ newState
    val reducedEffectsStream = reducedActionsStream
      .map:
        case (_, effects) ⇒ effects
      .flatMapMerge(EventStream.fromSeq(_))
    modSeq(
      reducedStateStream ⟶ maybeState,
      effector(reducedEffectsStream) ⟶ actionsBus.writer
    )
```

Integration with JS libraries

→ A number of wrappers exist for mainstream libs or you can roll your own

→ `ScalablyTyped` can convert TypeScript bindings to scala.js 🪄

Integration with maplibre

```
class MapLibreAdapter(using Locale, ...):
  val embedder: Modifier[ReactiveHtmlElement[HTMLElement]] =
    onMountUnmountCallbackWithState(
      mount = context ⇒ maplibre.Map(MapOptions(context.thisNode.ref)),
      unmount = (_, maybeMap) ⇒ maybeMap.foreach(_.remove())
    )
```

Integration with maplibre

```
class MapLibreAdapter(using Locale, ...):
  val embedder: Modifier[ReactiveHtmlElement[HTMLElement]] =
    onMountUnmountCallbackWithState(
      mount = context ⇒ maplibre.Map(MapOptions(context.thisNode.ref)),
      unmount = (_, maybeMap) ⇒ maybeMap.foreach(_.remove())
    )
```

🪝 hook it in the laminar tree:

```
div(
  cls := "h-screen w-screen antialiased absolute inset-0",
  mapLibre.embedder
)
```

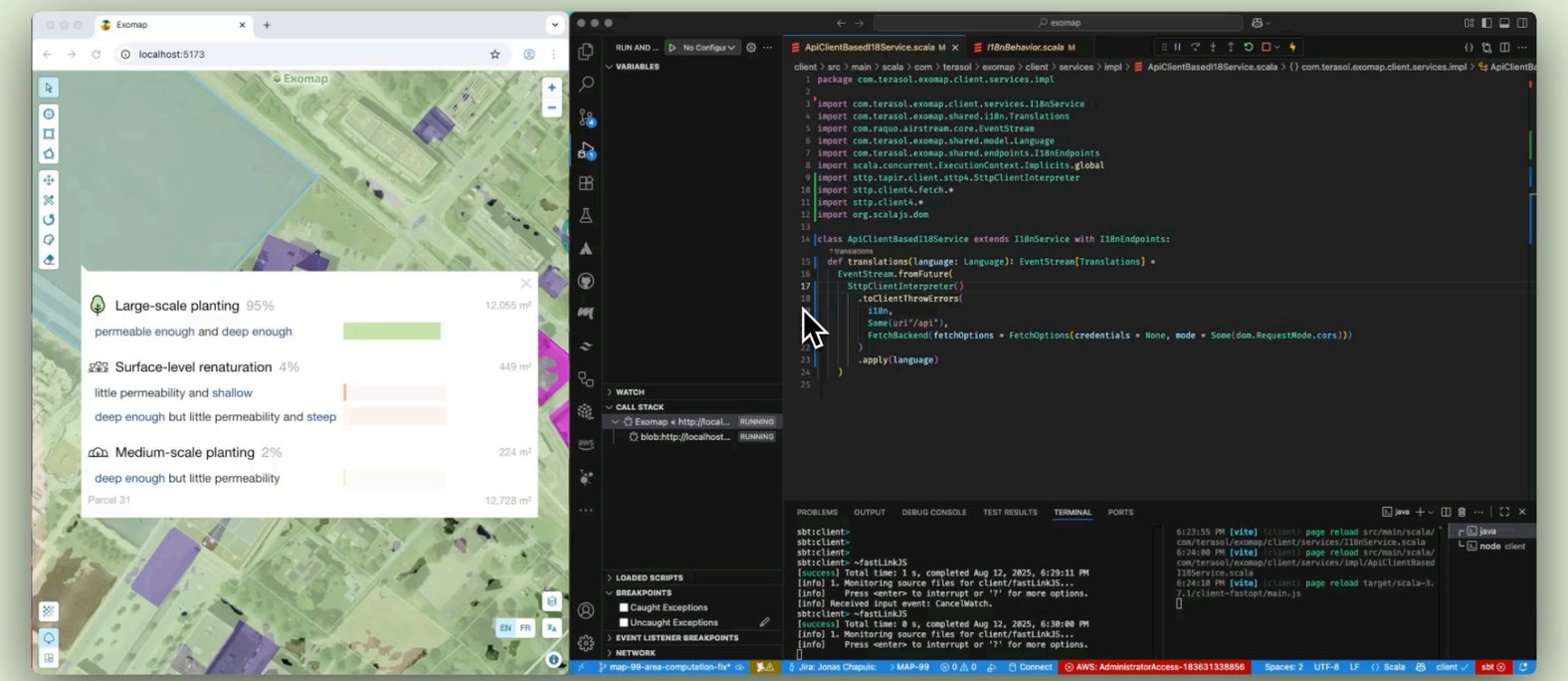
Service abstraction

```
trait I18nService:
  def translations(language: Language): EventStream[Translations]
```

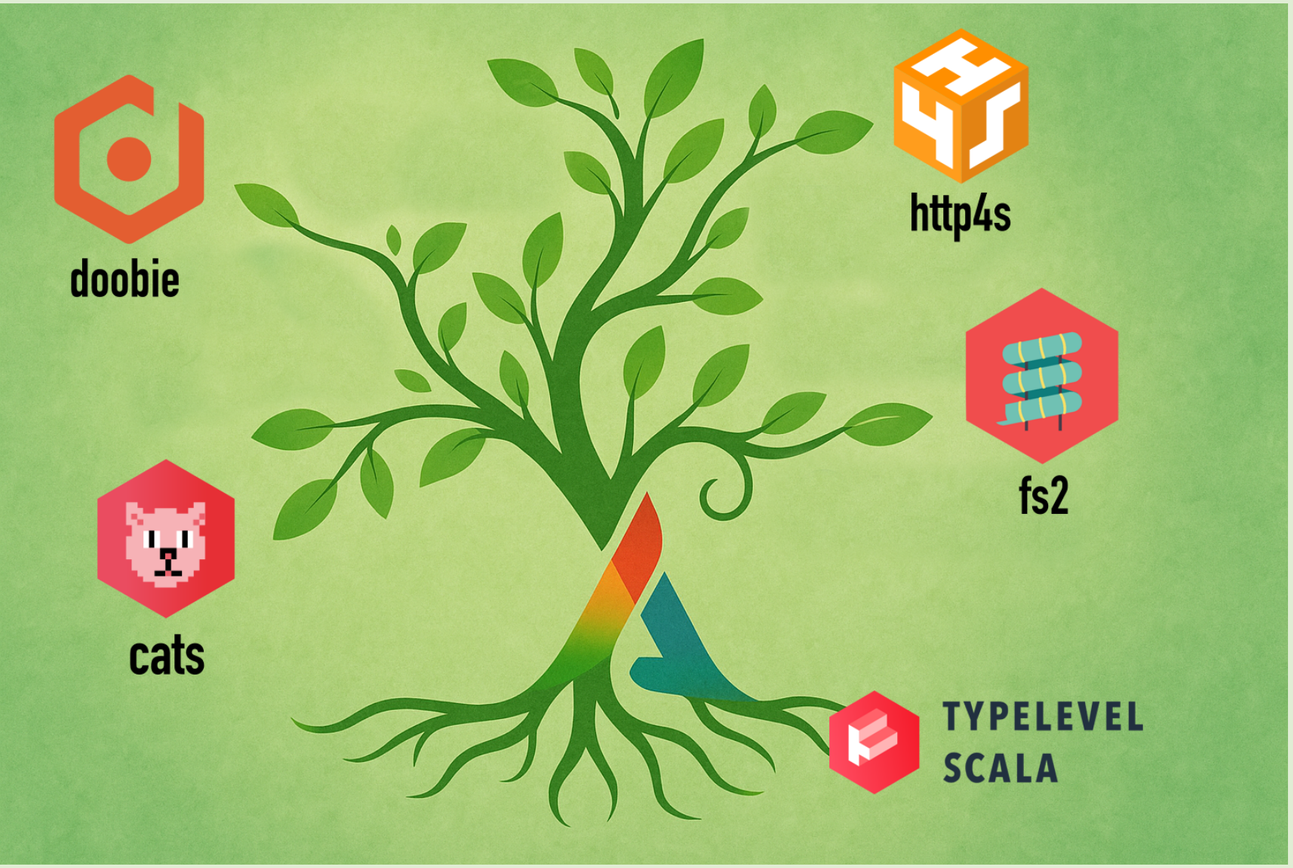
Sttp fetch-based implementation

```
class ApiClientBasedI18Service extends I18nService with I18nEndpoints:
  def translations(language: Language): EventStream[Translations] =
    EventStream.fromFuture(
      SttpClientInterpreter()
        .toClientThrowErrors(
          i18n,
          Some(uri"/api"),
          FetchBackend(fetchOptions = FetchOptions(credentials = None, mode = Some(dom.RequestMode.cors)))
        )
        .apply(language)
    )
```

Let's see it in action! 🐛



Backend: typelevel stack 🐱



→ proven, performant

→ `IO`: a control-freak's dream 🤓

→ `Http4s` with `http4-netty` backend for HTTP2 support

→ `fs2`-driven tile server combined with `doobie` DB streaming

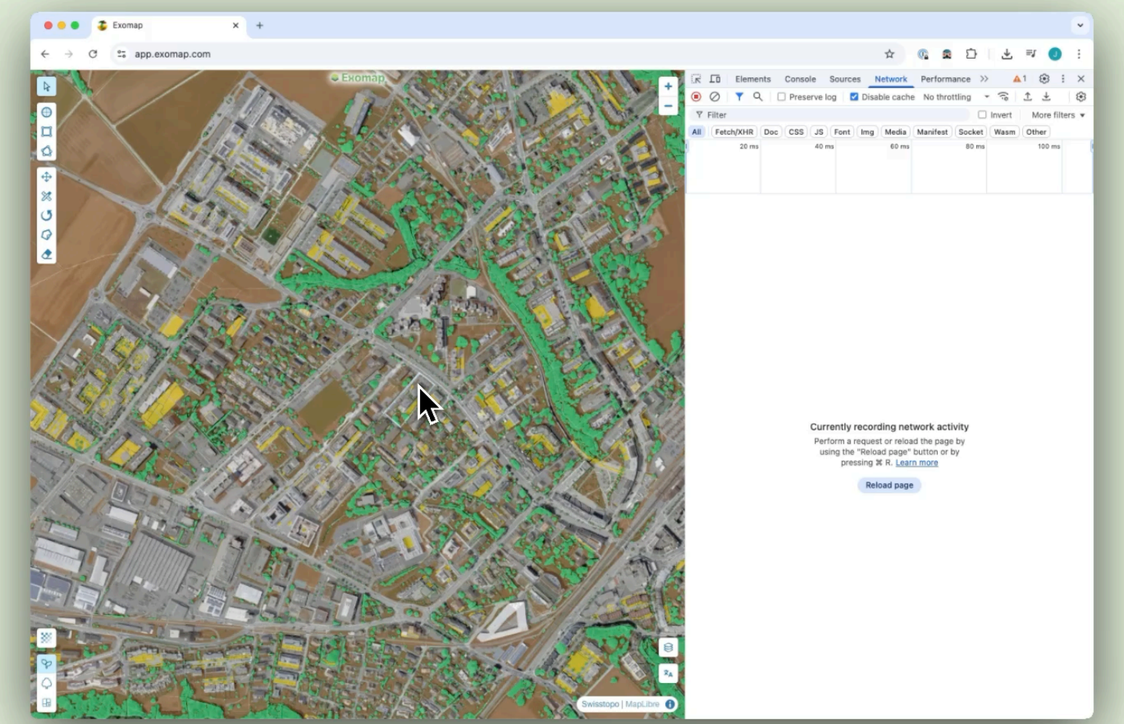
http4s server using tapir interpreter 🌐

```scala
class I18nHttpApp extends IOApp.Simple with I18nEndpoints:
  def run: IO[Unit] =
    val appResource =
      for
        given I18nService ← BabelBasedI18nService().toResource
        _ ← server
      yield ()
    appResource.useForever

  private def server(using i18nService: I18nService): Resource[IO, Server] =
    val interpreter = Http4sServerInterpreter[IO]()
    val serverLogic = i18n.serverLogicSuccess: language ⇒
      i18nService.translationsForLanguage(language)
    val routes = interpreter.toRoutes(serverLogic)
    val equippedRoutes = middlewareStack(routes)
    NettyServerBuilder[IO].bindHttp(8080, "0.0.0.0").withHttpApp(equippedRoutes).resource

  private def middlewareStack(routes: HttpRoutes[IO]): HttpApp[IO] = ???
```

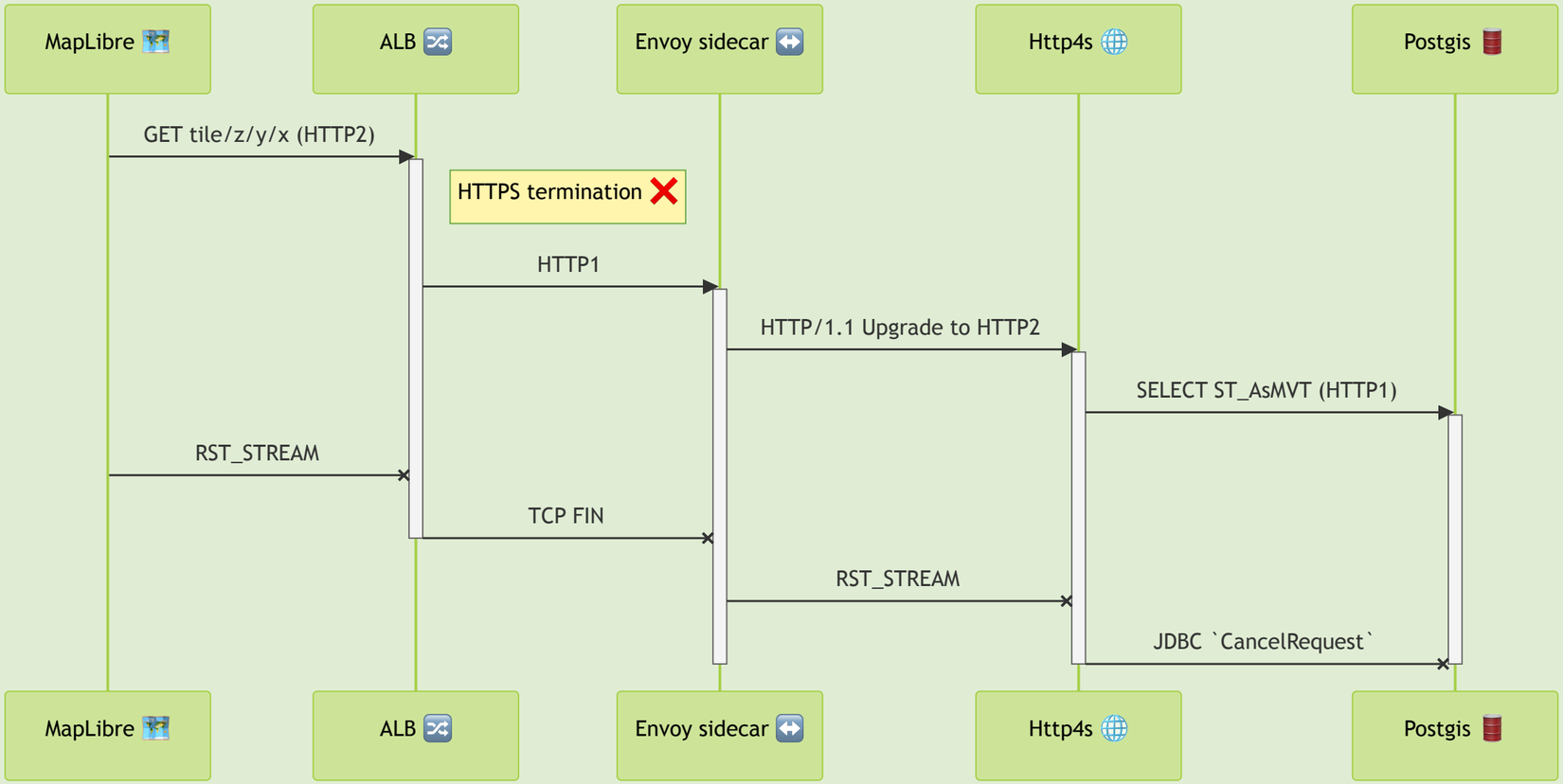
Rich middleware 🧰

```scala
private def middlewareStack(routes: HttpRoutes[IO]): HttpApp[IO] =
  CORS.policy.withAllowOriginAll.withAllowMethodsAll
    .withAllowHeadersAll(
      ResponseTiming(
        Date.httpApp(
          GZip(
            Logger
              .httpRoutes(
                logHeaders = true,
                logBody = false,
                logAction = Some(Console[IO].print)
              )(KamonSupport(routes, "localhost", 8080))
              .orNotFound
          )
        )
      )
    )
```

Tile server 


Exomap
app.exomap.com
Exomap
Elements
Console
Sources
Network
Performance
Preserve log
Disable cache
No throttling
Filter
Invert
More filters
All
Fetch/XHR
Doc
CSS
JS
Font
Img
Media
Manifest
Socket
Wasm
Other
20 ms
40 ms
60 ms
80 ms
100 ms
Currently recording network activity
Perform a request or reload the page by using the "Reload page" button or by pressing ⌘ R. Learn more
Reload page
Swisstopo | MapLibre

Deep cancellation: from `RST_STREAM` to `Statement#cancel()`



fs2 and doobie to stream from DB 🌊

```scala
def layerTileQuery(layer: LayerRef, zoom: Int, x: Int, y: Int): fs2.Stream[ConnectionIO, Array[Byte]] =
  val query = fr"""
    WITH ... AS (
      ...
    )
    SELECT ST_AsMVT(mvt_geom.*, ${layer.id})
    FROM mvt_geom
  """
  query.query[Array[Byte]].stream

  def layerTile(layer: LayerRef, zoom: Int, x: Int, y: Int)(using xa: Transactor[IO]): fs2.Stream[IO, Byte] =
    layerTileQuery(layer, zoom, x, y).transact(xa).flatMap(fs2.Stream.emits)
```

Route definition with http4s 


```
case GET → Root / LocationSlugVar(slug) / "metrics" / MetricLayerIDVar(metricLayerID) /
    LongVar(v) / IntVar(z) / IntVar(x) / IntVar(y) ⇒
  Ok(
    service.layerTile(LayerRef(slug, metricLayerID), z, x, y)
      .onFinalizeCase:
        case ExitCase.Canceled ⇒
          logger.info(s"Cancelled streaming tile for $metricLayerID v$v $z/$x/$y")
        case _ ⇒ () // other cases elided
      ,
    contentTypeHeader,
    cacheControlHeader
  )
```

Container packaging: `sbt-jib` FTW 📦

```
.settings(
    jibBaseImage := "eclipse-temurin:21",
    jibName := "exomap",
    jibExtraMappings := {
      val base = baseDirectory.value
      val viteDistDir = base / ".." / "client" / "dist"
      val files = (viteDistDir ** "*").get
      files pair Path.rebase(viteDistDir, "/public")
    },
    jibVersion := sys.env.getOrElse("APP_VERSION", s"v${version.value}"),
    jibUser := Some("65534:65534"), // Set the user and group ID for the container (noboby:nogroup, minimal privileges)
    jibRegistry := ecrSplit.headOption.getOrElse("localhost"),
    jibCustomRepositoryPath := (if (ecrSplit.drop(1).nonEmpty) Some(ecrSplit.drop(1).mkString("/")) else None),
    jibPlatforms := Set(JibPlatforms.amd64),
    jibJvmFlags += "-XX:+EnableDynamicAgentLoading",
    javaOptions += "-XX:+EnableDynamicAgentLoading" // Enable dynamic agent loading for Kamon
  )
```

Scala 4 Ops with  Pulumi &  BESOM

```scala
import besom.*
import ...

@main def main: Unit = Pulumi.run:
  given Input[EcsConfig] = config.require("ecsConfig")
  given Input[Networking] = config.require("networking")
  given Input[DatadogConfig] = config.require("datadogConfig")
  given Input[DatabaseConfig] = DatabaseConfig(config.require[RawDatabaseConfig]("dbConfig"))
  given Input[ContainerImage] =
    config.require[NonEmptyString]("ecrUrl").zip(config.require[NonEmptyString]("appVersion")).map(ContainerImage.apply)

  val service: Output[FargateService] = ExoService("exomap")

  Stack(service).exports(fargateUrn = service.fargate.urn)
```

"I suddenly felt an infra superman" 🦸

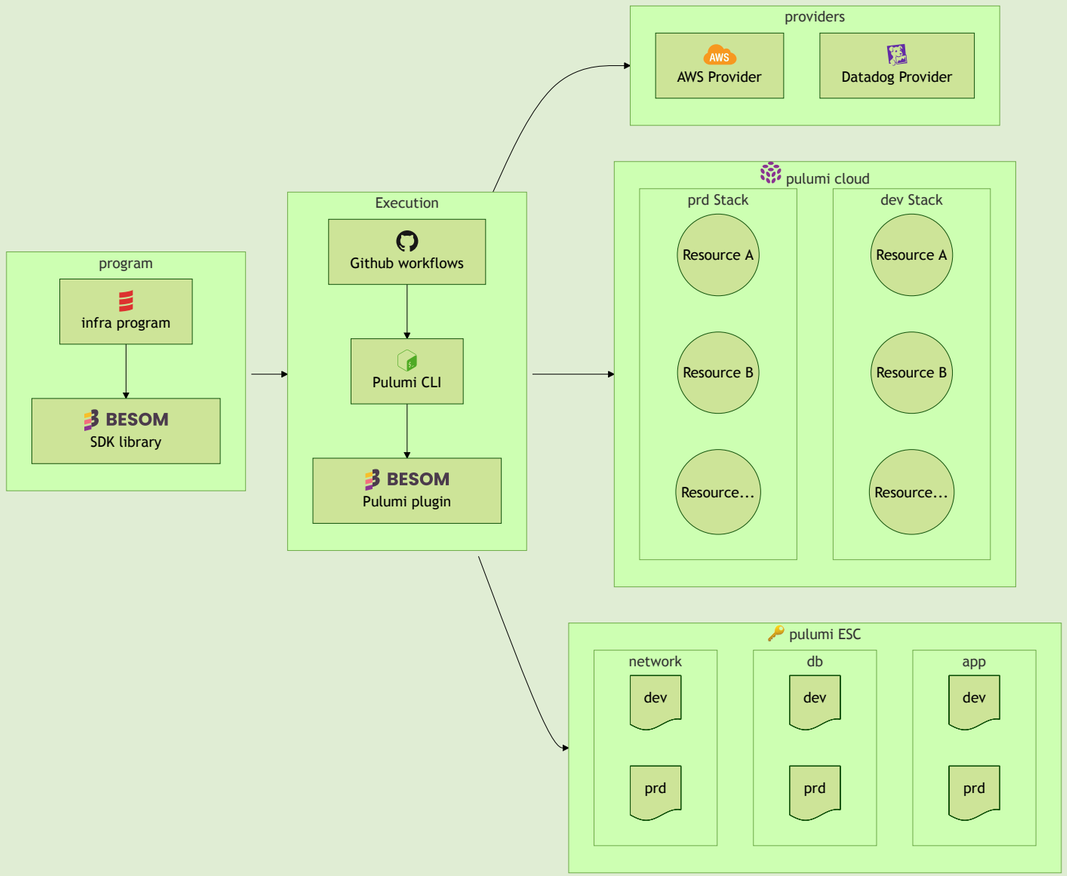
— long-time coder and YAML alergic

```scala
import besom.*
import besom.api.awsx.ecs.*
import besom.api.awsx.ecs.inputs.*

object ExoService:
  def apply(name: NonEmptyString)(using image: Input[ContainerImage], ...): Output[FargateService] =
    val serviceContainer = TaskDefinitionContainerDefinitionArgs(s"$name-container",
        image = image.imageRef,
        // ...
    )

    val serviceTask = FargateTaskDefinition(s"$name-service-definition",
      FargateTaskDefinitionArgs(container = Some(serviceContainer)), // ...)

    // ...

    FargateService(taskDefinition = serviceTask.taskDefinition.arn, /// ...)
```

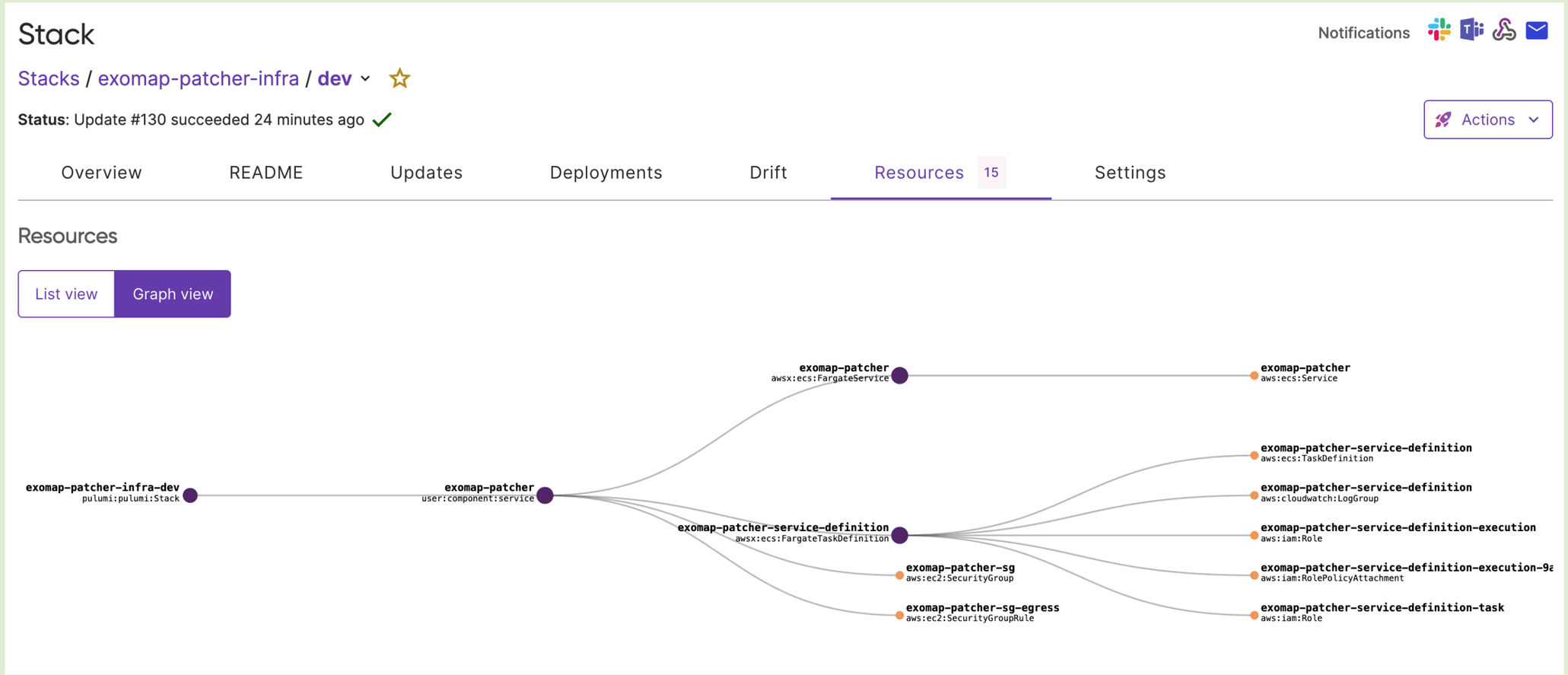
→ **Pulumi**: *OSS* infrastructure-as-code platform.

→ **Besom**: Pulumi SDK for Scala.

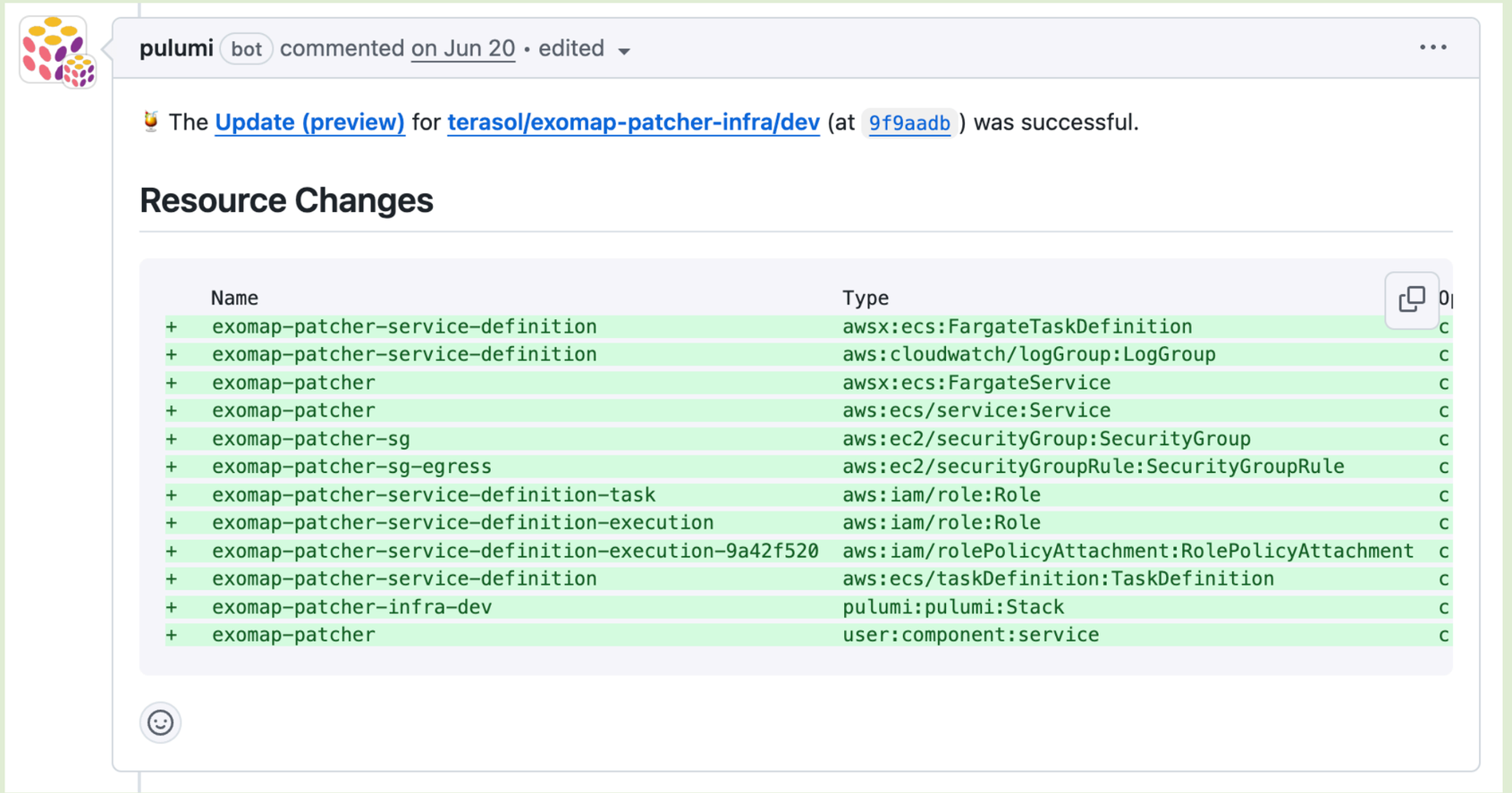
Ok, but how does it work?



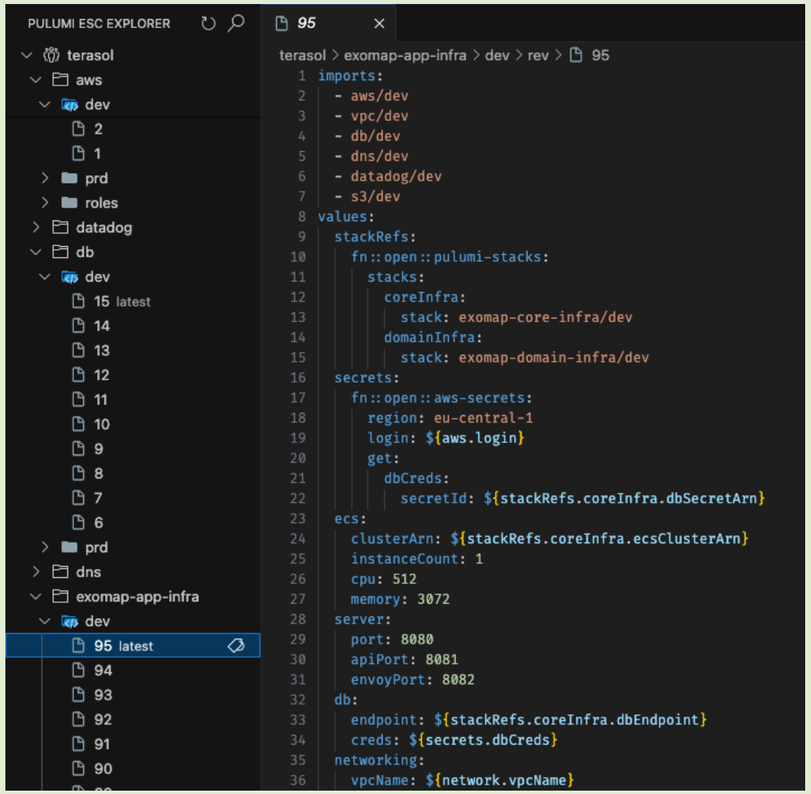
Pulumi cloud dashboard



Pulumi PR previews



Pulumi ESC 🔒



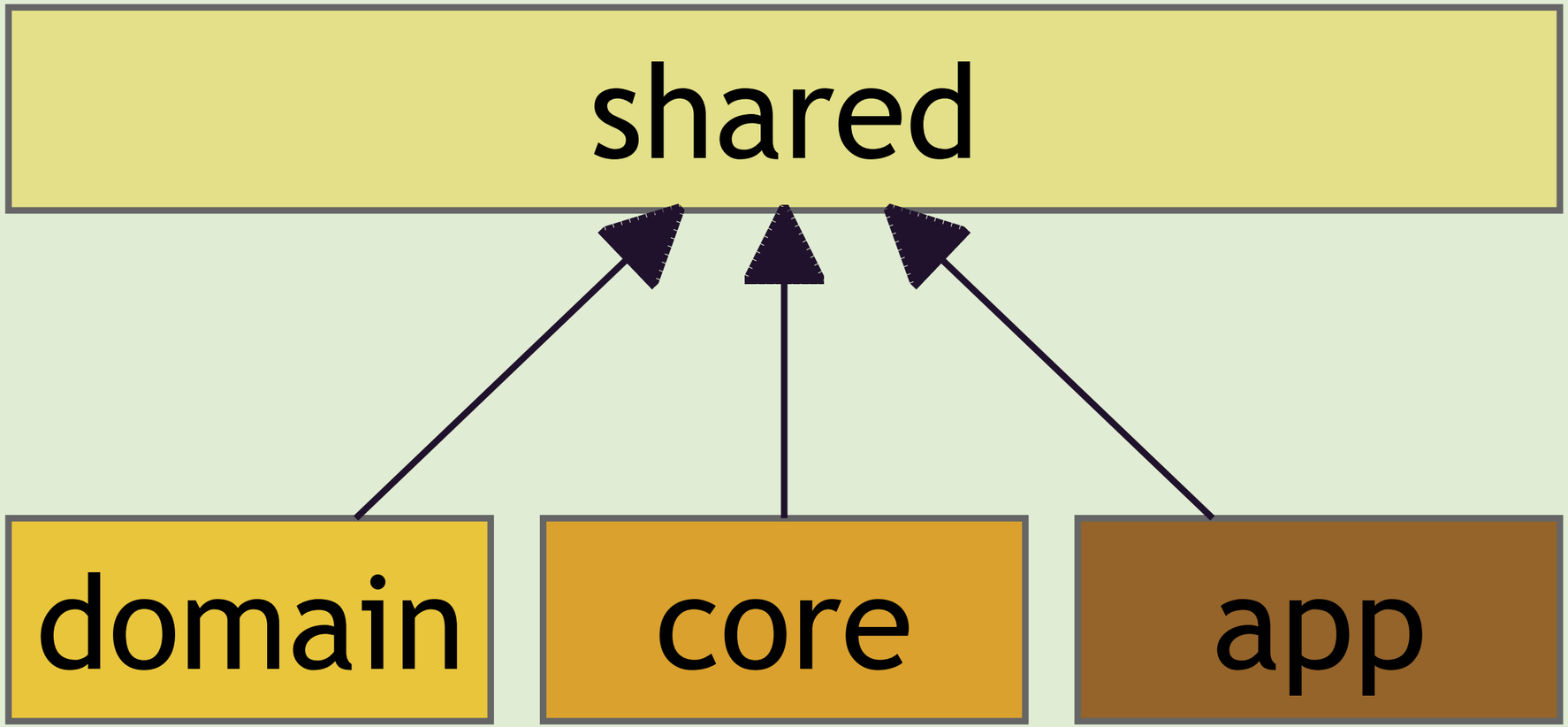

```
db:
  name: exomap
  instanceType: db.t3.large
  postgresVersion: '16.6'
...
pulumiConfig:
  aws:region: ${aws.region}
  ecsConfig: ${ecs}
  ecrUrl: ${stackRefs.domainInfra.ecrUrlForPatcher}
  dbConfig: ${db}
  datadogConfig: ${datadog}
  networking: ${networking}
```

→ exposed in the program as **`config.require()`**

→ ESC is just a **convenience**, for config you can also use files, environment vars, etc.

integration

`/infra/ ... /build.sbt` **structure**



Take-away 🍔

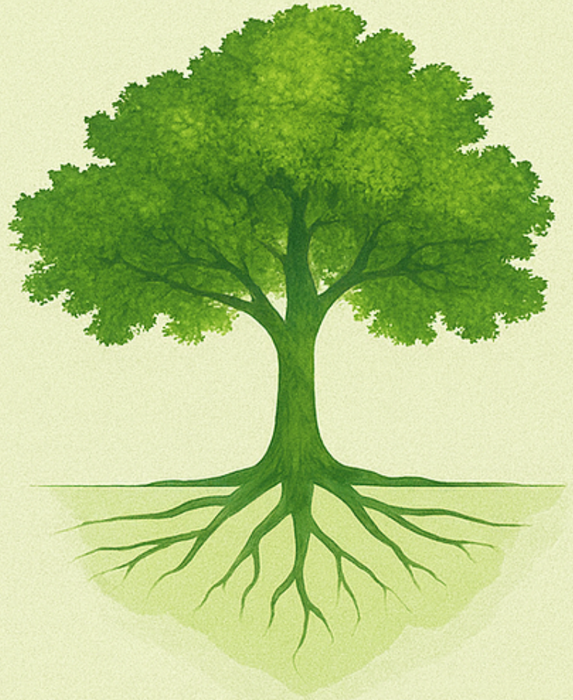
→ 🦸 You *really* can do everything with Scala

→ ♦️ Compact, efficient, safe, uniform codebase from front to back plus ops

→ 👬 Great for product teams or solo players

→ 🌟 Your Scala skills can shine everywhere

→ 🥲 Programming joy FTW



THANKS 🙏
QUESTIONS?

FULL-STACK SCALA

JONAS CHAPUIS, PH.D.
TERASOL