U37BS6J6N : Ello all looking for some spa structure advice.I've seen examples with Components (similar to react) that have single files containing the model, view, and update in a single file. (the webpack starter does this a bit)

I"ve also seen spas written into individual folders for views, models,
<https://github.com/rtfeldman/elm-spa-example>

I've done both in other js spa's with varying degrees of success and am open to either.

Is there a "preferred best practice" way to do this?

Also, is there a "Redux" type single "source of truth" storage solution for elm that I should look into?

Or if there is a better channel to ask please point me to it

U3SJEDR96 : Alright, so; components don't work that well in Elm, since they sort of urge you to put the default abstraction boundary at the "TEA triplet", and result in having a separate `Msg` type, `Model` type, `update` `view` and `init` function for every "reusable thing". As such, trying to keep things flat and making reusable _functions_ will get you better results. Of course, when it makes sense to abstract at the TEA boundary, that's when you do. For example in the elm-spa-example, every page has a separate msg type and model, and separate functions.
U3SJEDR96 : A good resource for that is
<https://www.reddit.com/r/elm/comments/5jd2xn/how_to_structure_elm_with_multiple_models/dbkpgbd/> this comment/thread
U3SJEDR96 : as for "single source of truth" - you actually have _no other option_ in Elm.
U3SJEDR96 : In the end, you hand a single `main` to the runtime, which has a `model` that contains all your state
U37BS6J6N : <@U3SJEDR96> thanks for the info
U4872964V : <@U37BS6J6N> I know this sounds like cheesy advice, but my preferred best practice is not to worry about it, and refactor when the code becomes uncomfortable. That is, I just write the functions that I think I need at the moment :slightly_smiling_face:
U3SJEDR96 : Yeah. Also don't be "guilt tripped" into overabstracting and encapsulating things that - realistically - don't need to be encapsulated
U3SJEDR96 : best practices from other paradigms don't transfer perfectly to a functional language
U4872964V : I just do `exposing (..)` until I get conflicts, then I refine. Unless I'm writing a package of course, then the API is important. But that's another matter.
U3LT1UTPF : ```26| UpdateCurrentProduct product -&gt;
27| { model | currentProduct = product } ! []
28|
29| UpdateCurrentCycle newCycle -&gt;
30|&gt;   let
31|&gt;   currentProduct =
32|&gt;   model.currentProduct.product
33|&gt;
34|&gt;   findProductInNewCycle =
35|&gt;   find (\product -&gt; product.product == currentProduct) newCycle.productList
36|&gt;   in
37|&gt;   case findProductInNewCycle of
38|&gt;   Nothing -&gt;
39|&gt;   ({ model | currentCycle = newCycle }, UpdateCurrentProduct initialCurrentProduct)
40|&gt;
41|&gt;   Just product -&gt;
42|&gt;   ({ model | currentCycle = newCycle }, UpdateCurrentProduct product)

The 3rd branch has this type:
    ({ currentCycle : Cycle   , currentProduct : Product   , cycleList : List Cycle   , error : Maybe String   }   , *Cmd msg*   )

But the 4th is:
    ({ currentCycle : Cycle   , currentProduct : Product   , cycleList : List Cycle   , error : Maybe String   }   , *Msg*   )

Hint: All branches in a `case` must have the same type. So no matter which one we take, we always get back the same type of value.

```

I'm sure this is a very silly mistake, but I don't get what I'm doing wrong...

U41NK9BM4 : You miss a `! []` on UpdateCurrentCycle
U41NK9BM4 : See UpdateCurrentProduct for a comparison
U41NK9BM4 : <@U3LT1UTPF> ^^^^
U41NK9BM4 : Basically you don't pack your Msg into a Cmd like you do above
U3SJEDR96 : Well, no, it's that you seem to be trying to return a `Msg` from `update` so that it will be called again, rather than actually doing what needs to be done
U3SJEDR96 : `{ model | currentCycle = newCycle, currentProduct = findProdictInNewCycle |&gt; Maybe.withDefault initialCurrentProduct } ! []`instead
U3LT1UTPF : Thank you <@U3SJEDR96> and <@U41NK9BM4> :smile:
U3SJEDR96 : (which would replace that entire `case findProductInNewCycle of` :slightly_smiling_face: )
U3LT1UTPF : Great!
U3LT1UTPF : Then, one must not call a Msg from another Msg, right, <@U3SJEDR96>?
U3SJEDR96 : Well, you can, but if you find yourself doing that, it _usually_ makes more sense to take the functionality from that branch that you actually want, make it into a separate function (like `withCurrentProduct : Product -&gt; Model -&gt; Model`) and call that from both places
U3SJEDR96 : so then you'd have `(model |&gt; withCurrentProduct product) ! []` in one place, and `( model |&gt; withCurrentCycle newCycle |&gt; withCurrentProduct (productInNewCycle |&gt; withDefault initialProduct) ) ! []` in the other place
U3SJEDR96 : the alternatives are:- calling `update` from `update` directly, i.e. making it recursive, which can lead to nasty bugs and doesn't seem necessary at all
- forcing `Msg` into a `Cmd Msg` and letting the runtime call `update` instead, which should make you wonder "why do I need to asynchronously call a function I defined myself?" and "what happens in between?", since your model will essentially be in an invalid state between those calls

U3LT1UTPF : Wowww... I get it :smile::bananadance: Thank you so much, <@U3SJEDR96>
U0CLDU8UB : My favorite alternative is to rethink what I am trying to do. Many times I can use the same Msg in a couple of places, instead of having two separate `update` cases.
U5P4FLYLE : Hi all, I am working with elm-mdl with card. And I am adding action block like below:```      , Card.actions
      [ Card.border, css "vertical-align" "center", css "text-align" "right", backgroundColor ]
      [ Button.render Mdl [8,1] model.mdl
          [ Button.icon, Button.ripple ]
          [ Icon.i "favorite_border" ]
      , Button.render Mdl [8,2] model.mdl
          [ Button.icon, Button.ripple ]
          [ Icon.i "event_available" ]
      ]    ```
And it is added *below* the other blocks. What would you change to add it to the *right side* of already existing blocks? I bet it is more css question than elm-mdl one...

U0CLDU8UB : Of course sometimes I do need the two cases, but even then the separation might change with the rethinking.
U3LT1UTPF : Good advice, <@U0CLDU8UB> :smile:
U5H8JJP24 : Hi, I have this weird problem. If I write my functions like this:
```
newLocation : Result Http.Error Location -&gt; Model -&gt; (Model, Cmd Msg)
newLocation result model =
    case result of
        Err error -&gt;
            handleHttpError error model

        Ok location -&gt;
            model |&gt; updateLocation location |&gt; fetchRoute

handleHttpError : Http.Error -&gt; Model -&gt; (Model, Cmd msg)
handleHttpError error model =

    ({ model | error = Just (toString error) } |&gt; Debug.log "Error") ! []
```
I get the error:

```
Function `handleHttpError` is expecting the 1st argument to be:

    Http.Error

But it is:

    String
```

If I change the function annotation to:
```
handleHttpError : String -&gt; Model -&gt; ( Model, Cmd msg )
```
I get the error:

```
Function `handleHttpError` is expecting the 1st argument to be:

    String

But it is:

    Http.Error
```

U3SJEDR96 : are you calling that `handleHttpError` function from anywhere else? You first attempt looks correct to me...
U4872964V : yes, check the location of the error
U5H8JJP24 : oumph, thx <@U3SJEDR96> <@U4872964V>. I was searching at the wrong place. There was another call which caused the error... This took me 30 min to realise xD
U62R599PU : so new 'beginner' may be overstating it ... been trying to wrap my head around the concepts in Elm vs JavaScript. I have some basic framework (object ... record, some similarities, many differences ...that kind of thing). At a high level I understand subscriptions in the time clock sense or even keyboard input. The one area I can't seem to figure out is the equivalent approach/style to deal with observables.
U62R599PU : Can someone point me in the right direction. In the past I would have used Flyd observable streams.
U4872964V : in Elm, the concept of observable corresponds to a Msg
```