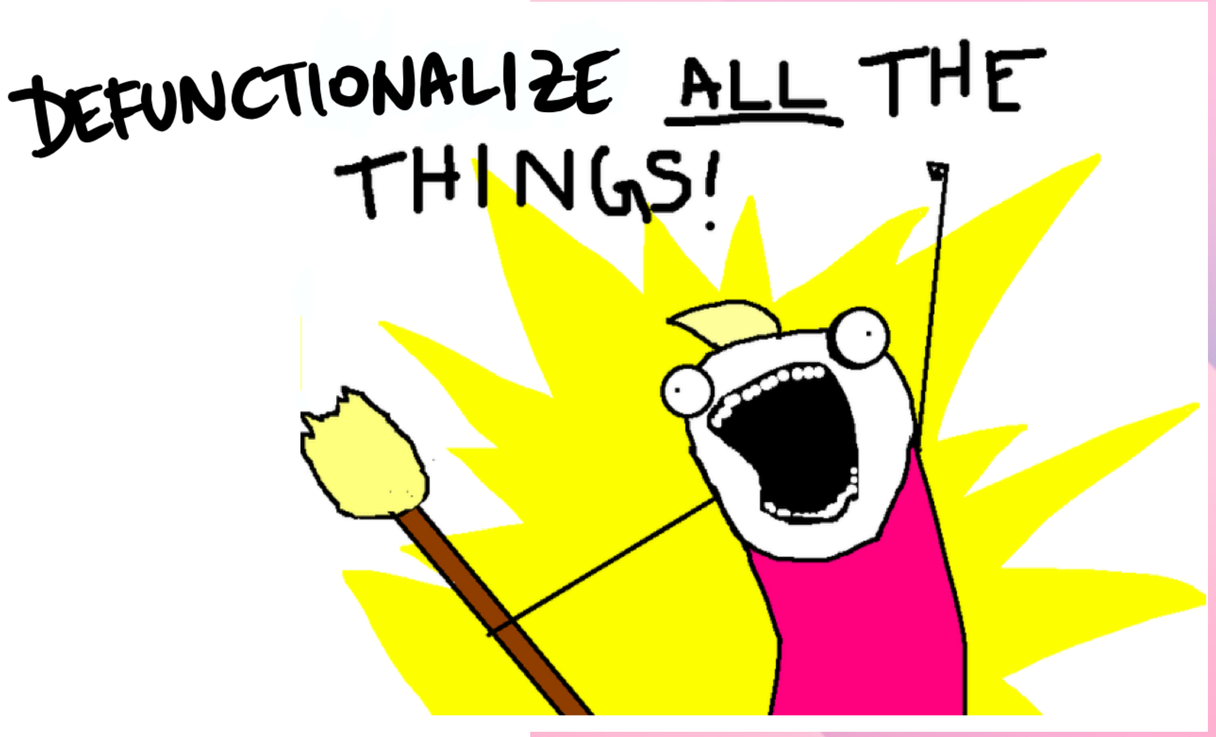
Permutive

Store pattern in Elm

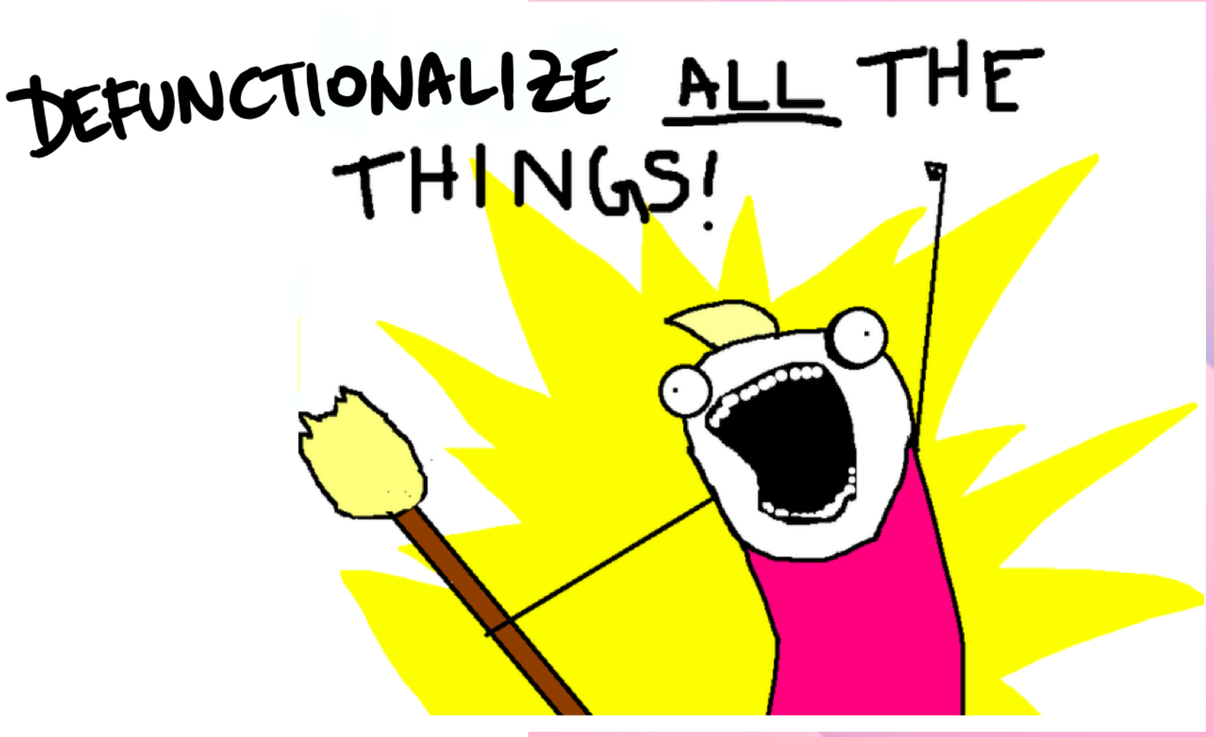
Permutive

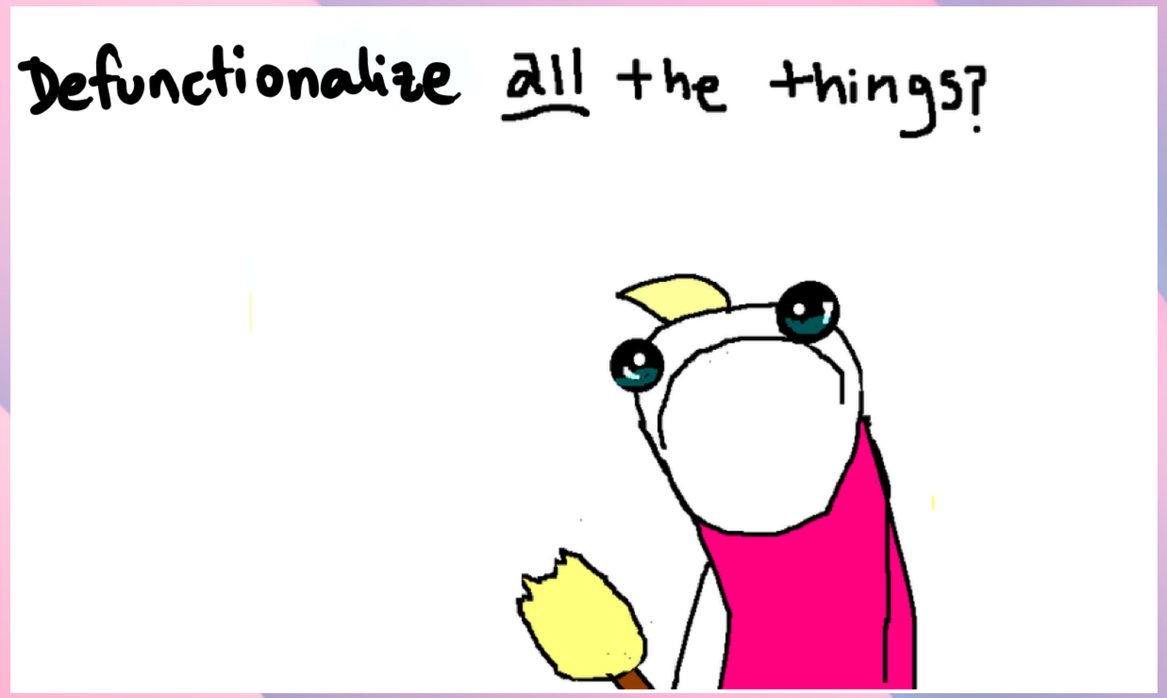
Store pattern in Elm



Permutive

Store pattern in Elm





Agenda

What it is

What it gives us

Example + demo

Nice extras

What it is

```
type alias Store =
    { -- we're loading all posts at once
      -- GET /api/posts
      posts : WebData (Dict PostId Post)

    , -- we're loading all users at once
      -- GET /api/users/
      users : WebData (Dict UserId User)

    , -- we're lazy loading images as needed
      -- GET /api/images/<ID>
      images : Dict ImageId (WebData Image)
    }

type Action -- maps to HTTP request
    = GetPosts
    | GetUsers
    | GetImage ImageId
    | CreatePost PostCreateData

type Msg -- maps to HTTP response
    = HttpError Action Http.Error
    | GotPosts (List Post)
    | GotUsers (List User)
    | GotImage Image
    | CreatedPost Action Post
```

What it is

```elm
type alias Store =
    { -- we're loading all posts at once
      -- GET /api/posts
      posts : WebData (Dict PostId Post)

    , -- we're loading all users at once
      -- GET /api/users/
      users : WebData (Dict UserId User)

    , -- we're lazy loading images as needed
      -- GET /api/images/<ID>
      images : Dict ImageId (WebData Image)
    }

type Action -- maps to HTTP request
    = GetPosts
    | GetUsers
    | GetImage ImageId
    | CreatePost PostCreateData

type Msg -- maps to HTTP response
    = HttpError Action Http.Error
    | GotPosts (List Post)
    | GotUsers (List User)
    | GotImage Image
    | CreatedPost Action Post
```

```elm
module Store exposing
    ( Store, init
    , Action(..), runAction
    , Msg(..), update
    )

init : Store
runAction : Action -> Store -> (Store, Cmd Msg)
update    : Msg    -> Store -> (Store, Cmd Msg)
```

What it is

```
type alias Store =
    { -- we're loading all posts at once
      -- GET /api/posts
      posts : WebData (Dict PostId Post)

    , -- we're loading all users at once
      -- GET /api/users/
      users : WebData (Dict UserId User)

    , -- we're lazy loading images as needed
      -- GET /api/images/<ID>
      images : Dict ImageId (WebData Image)
    }

type Action -- maps to HTTP request
    = GetPosts
    | GetUsers
    | GetImage ImageId
    | CreatePost PostCreateData

type Msg -- maps to HTTP response
    = HttpError Action Http.Error
    | GotPosts (List Post)
    | GotUsers (List User)
    | GotImage Image
    | CreatedPost Action Post
```

```
module Store exposing
    ( Store, init
    , Action(..), runAction
    , Msg(..), update
    )

init : Store
runAction : Action -> Store -> (Store, Cmd Msg)
update    : Msg    -> Store -> (Store, Cmd Msg)
```

```
type alias WebData a =
    RemoteData Http.Error a

type RemoteData x a
    = NotAsked
    | Loading
    | Failure x
    | Success a
```

What it is

```elm
type alias Store =
    { -- we're loading all posts at once
      -- GET /api/posts
      posts : WebData (Dict PostId Post)

    , -- we're loading all users at once
      -- GET /api/users/
      users : WebData (Dict UserId User)

    , -- we're lazy loading images as needed
      -- GET /api/images/<ID>
      images : Dict ImageId (WebData Image)
    }

type Action -- maps to HTTP request
    = GetPosts
    | GetUsers
    | GetImage ImageId
    | CreatePost PostCreateData

type Msg -- maps to HTTP response
    = HttpError Action Http.Error
    | GotPosts (List Post)
    | GotUsers (List User)
    | GotImage Image
    | CreatedPost Action Post
```

```elm
module Store exposing
    ( Store, init
    , Action(..), runAction
    , Msg(..), update
    )

init : Store
runAction : Action -> Store -> (Store, Cmd Msg)
update    : Msg    -> Store -> (Store, Cmd Msg)
```

```elm
type alias WebData a =
    RemoteData Http.Error a

type RemoteData x a
    = NotAsked
    | Loading
    | Failure x
    | Success a
```

```elm
type alias Model =
    { store : Store
    -- , ...
    }
```

How it’s used

```
type alias Post =
    { id : PostId
    , title : String
    , authorId : UserId
    , content : String
    , imageIds : List ImageId
    }
```

How it’s used

```
type alias Post =
    { id : PostId
    , title : String
    , authorId : UserId
    , content : String
    , imageIds : List ImageId
    }
```

```
module Page.Posts exposing (Config, dataRequests, view)

dataRequests : List Store.Action
dataRequests =
    [ Store.GetPosts
    , Store.GetUsers
    ]
```

How it’s used

```
type alias Post =
    { id : PostId
    , title : String
    , authorId : UserId
    , content : String
    , imageIds : List ImageId
    }
```

```
module Page.Posts exposing (Config, dataRequests, view)

dataRequests : List Store.Action
dataRequests =
    [ Store.GetPosts
    , Store.GetUsers
    ]
```

```
module Page.Post exposing (dataRequests, view)

dataRequests : Store -> PostId -> List Store.Action
dataRequests store postId =
    let
        staticRequests : List Store.Action
        staticRequests =
            [ Store.GetPosts
            , Store.GetUsers
            ]

        dynamicRequests : List Store.Action
        dynamicRequests =
            RemoteData.get postId store.posts
                |> RemoteData.map .imageIds
                |> RemoteData.withDefault []
                |> List.map Store.GetImage
    in
    staticRequests ++ dynamicRequests
```

How it's used

```
type alias Post =
    { id : PostId
    , title : String
    , authorId : UserId
    , content : String
    , imageIds : List ImageId
    }
```

```
module Page.Posts exposing (Config, dataRequests, view)

dataRequests : List Store.Action
dataRequests =
    [ Store.GetPosts
    , Store.GetUsers
    ]
```

```
dataRequests : Store -> Route -> List Store.Action
dataRequests store route =
    case route of
        PostsRoute ->
            Page.Posts.dataRequests

        PostRoute postId ->
            Page.Post.dataRequests store postId

        UserRoute _ ->
            Page.User.dataRequests
```

```
module Page.Post exposing (dataRequests, view)

dataRequests : Store -> PostId -> List Store.Action
dataRequests store postId =
    let
        staticRequests : List Store.Action
        staticRequests =
            [ Store.GetPosts
            , Store.GetUsers
            ]

        dynamicRequests : List Store.Action
        dynamicRequests =
            RemoteData.get postId store.posts
                |> RemoteData.map .imageIds
                |> RemoteData.withDefault []
                |> List.map Store.GetImage
    in
    staticRequests ++ dynamicRequests
```

How it's used

```
type alias Post =
    { id : PostId
    , title : String
    , authorId : UserId
    , content : String
    , imageIds : List ImageId
    }
```

```
module Page.Posts exposing (Config, dataRequests, view)

dataRequests : List Store.Action
dataRequests =
    [ Store.GetPosts
    , Store.GetUsers
    ]
```

```
dataRequests : Store -> Route -> List Store.Action
dataRequests store route =
    case route of
        PostsRoute ->
            Page.Posts.dataRequests

        PostRoute postId ->
            Page.Post.dataRequests store postId

        UserRoute _ ->
            Page.User.dataRequests
```

```
module Page.Post exposing (dataRequests, view)

dataRequests : Store -> PostId -> List Store.Action
dataRequests store postId =
    let
        staticRequests : List Store.Action
        staticRequests =
            [ Store.GetPosts
            , Store.GetUsers
            ]

        dynamicRequests : List Store.Action
        dynamicRequests =
            RemoteData.get postId store.posts
                |> RemoteData.map .imageIds
                |> RemoteData.withDefault []
                |> List.map Store.GetImage
    in
    staticRequests ++ dynamicRequests
```

```
CreatePost data ->
    let
        request : Store.Action
        request =
            Store.CreatePost data
    in
    model
        |> sendDataRequest request
```

Demo time!

What it gives us

A convention for common need

What it gives us

A convention for common need

Commands as data (defunctionalization)

What it gives us

A convention for common need

Commands as data (defunctionalization)

Declarative data dependencies

What it gives us

A convention for common need

Commands as data (defunctionalization)

Declarative data dependencies

Data reuse across pages

What it gives us

A convention for common need

Commands as data (defunctionalization)

Declarative data dependencies

Data reuse across pages

(With elbow grease) Great error messages

Thank you!
Questions?