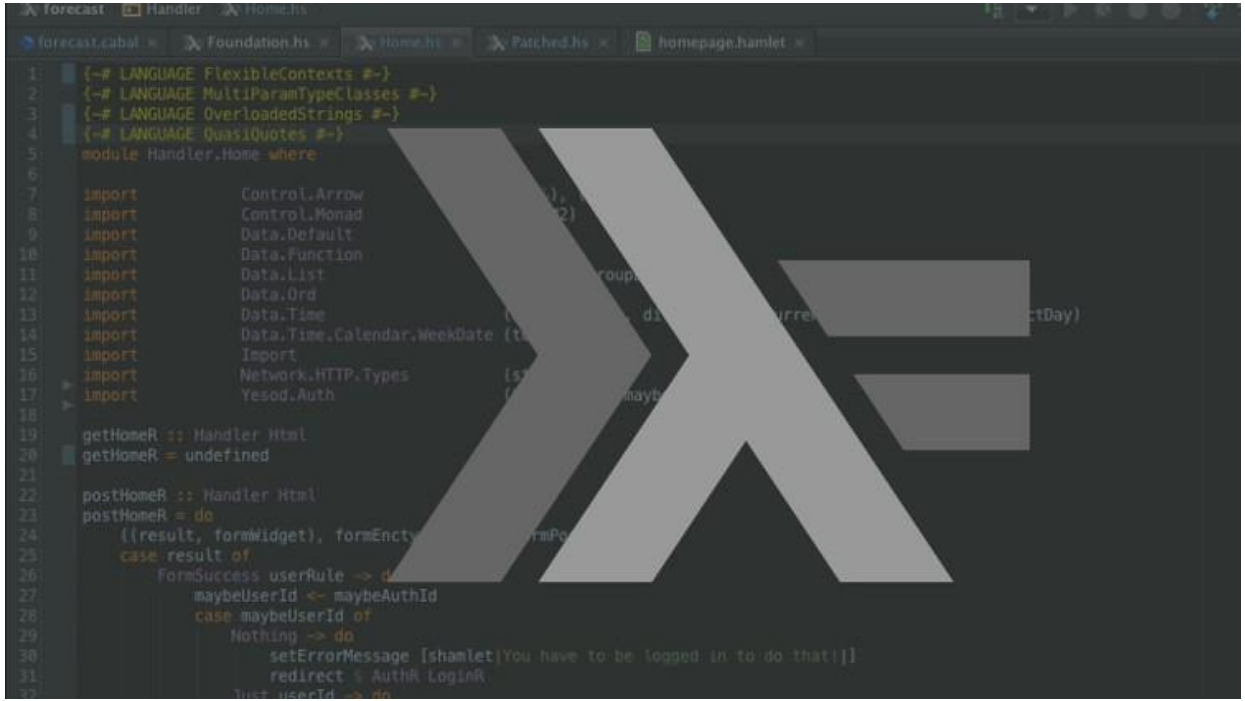
Implementation of fmap



Functors

We already know how to apply functions:

```
λ> (+3) 2                    👉  5
```

But...

```
λ> (+3) (Just 2)             ❌
```

In this case, we can use `fmap`!

```
λ> fmap (+3) (Just 2)        👉  Just 5
λ> fmap (+3) Nothing         👉  Nothing
```

And it also works with `Either`, lists, tuples and functions:

```
λ> fmap (+3) (Right 2)       👉  Right 5
λ> fmap (+3) (Left "error")  👉  Left "error"

λ> fmap (+3) [1, 2, 3]       👉  [4, 5, 6]        -- same as map

λ> fmap (+3) (1, 6)          👉  (1, 9)           -- because (,) is a type

λ> (fmap (*2) (+1)) 3        👉  8                -- same as (.)
```

Implementation of fmap

`fmap` applies a function to the elements of a generic container `f a` returning a container of the same type.

`fmap` is a function of the instances of the class `Functor`:

```
λ> :type fmap
fmap :: Functor f => (a -> b) -> (f a -> f b)
```

Where

```
λ> :info Functor
class Functor f where
    fmap :: (a -> b) -> (f a -> f b)
```