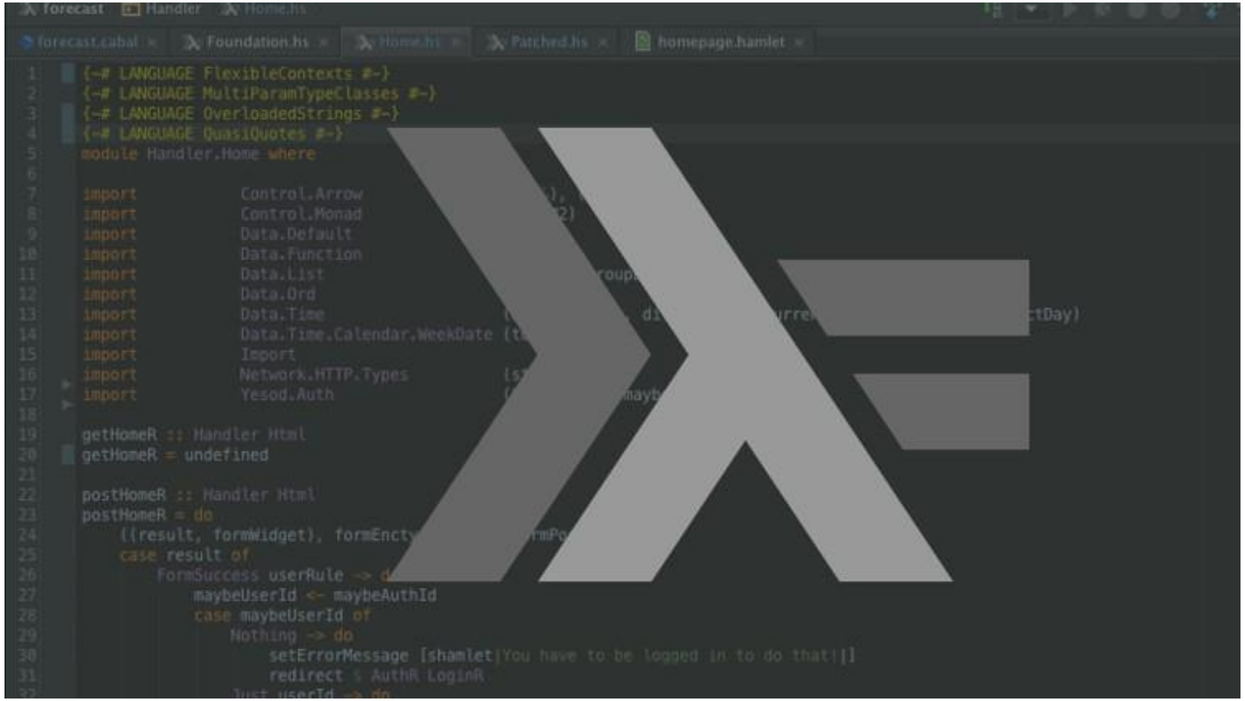
Advanced Types



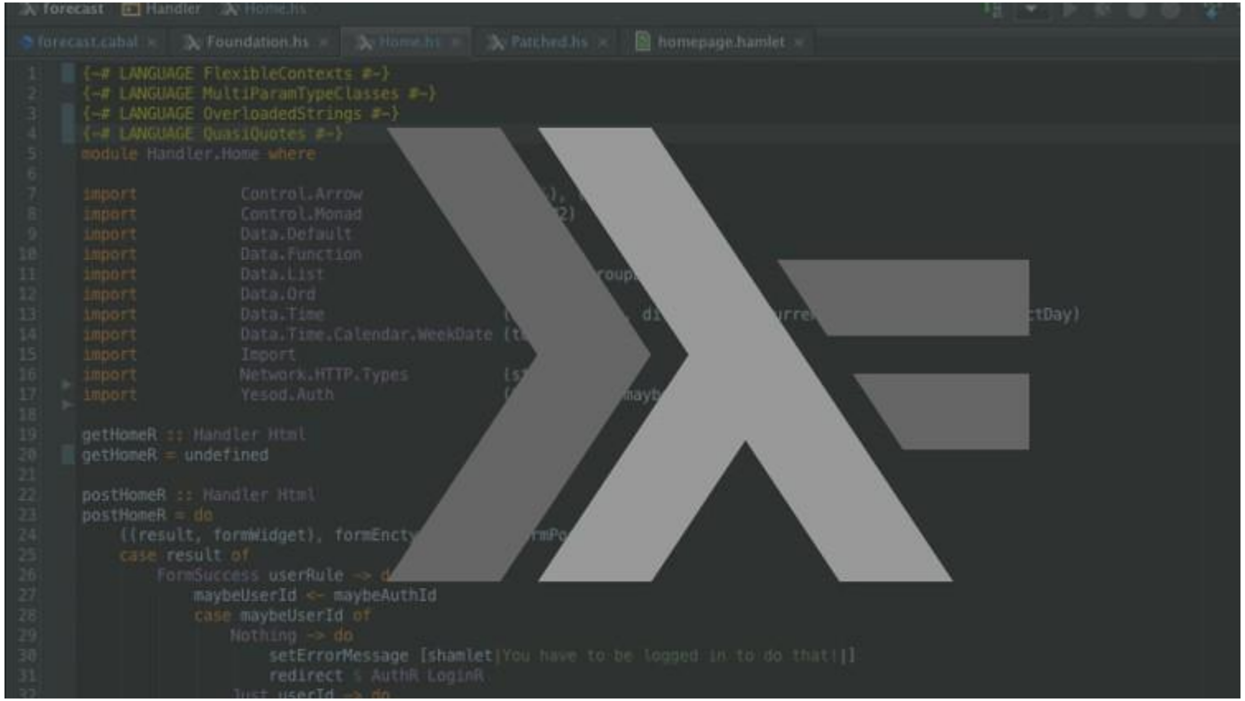
```
{-# LANGUAGE FlexibleContexts #-}
{-# LANGUAGE MultiParamTypeClasses #-}
{-# LANGUAGE OverloadedStrings #-}
{-# LANGUAGE QuasiQuotes #-}
module Handler.Home where

import           Control.Arrow                         ), (
import           Control.Monad                          2)
import           Data.Default
import           Data.Function
import           Data.List                                 roup
import           Data.Ord
import           Data.Time                   (        , di       rre                    ctDay)
import           Data.Time.Calendar.WeekDate (t
import           Import
import           Network.HTTP.Types          (s
import           Yesod.Auth                  (          mayb

getHomeR :: Handler Html
getHomeR = undefined

postHomeR :: Handler Html
postHomeR = do
    ((result, formWidget), formEncty          rmPo
    case result of
        FormSuccess userRule -> d
            maybeUserId <- maybeAuthId
            case maybeUserId of
                Nothing -> do
                    setErrorMessage [shamlet|You have to be logged in to do that!|]
                    redirect $ AuthR LoginR
                Just userId -> do
```

Maybe Data Type



```
{-# LANGUAGE FlexibleContexts #-}
{-# LANGUAGE MultiParamTypeClasses #-}
{-# LANGUAGE OverloadedStrings #-}
{-# LANGUAGE QuasiQuotes #-}
module Handler.Home where

import           Control.Arrow                  ), (
import           Control.Monad                  2)
import           Data.Default
import           Data.Function
import           Data.List                      roup
import           Data.Ord
import           Data.Time                  (       , di     rre                ctDay)
import           Data.Time.Calendar.WeekDate (t
import           Import
import           Network.HTTP.Types          (s
import           Yesod.Auth                  (           mayb

getHomeR :: Handler Html
getHomeR = undefined

postHomeR :: Handler Html
postHomeR = do
    ((result, formWidget), formEncty            rmPo
    case result of
        FormSuccess userRule -> d
            maybeUserId <- maybeAuthId
            case maybeUserId of
                Nothing -> do
                    setErrorMessage [shamlet|You have to be logged in to do that!|]
                    redirect $ AuthR LoginR
                Just userId -> do
```

Maybe a

The polymorphic type `Maybe a` is predefined like this:

```haskell
data Maybe a = Just a | Nothing
```

Maybe a

The polymorphic type `Maybe a` is predefined like this:

```haskell
data Maybe a = Just a | Nothing
```

It Expresses two possibilities:

- the presence of a value (of type`a` with the constructor `Just`), or
- its absence (with the empty constructor `Nothing`).

Maybe a

The polymorphic type `Maybe a` is predefined like this:

```
data Maybe a = Just a | Nothing
```

It Expresses two possibilities:

- the presence of a value (of type`a` with the constructor `Just`), or
- its absence (with the empty constructor `Nothing`).

Applications:

- Indicate possible null values.
- Indicate absence of a result.
- Report an error.

Examples

Let’s see some examples in practise.