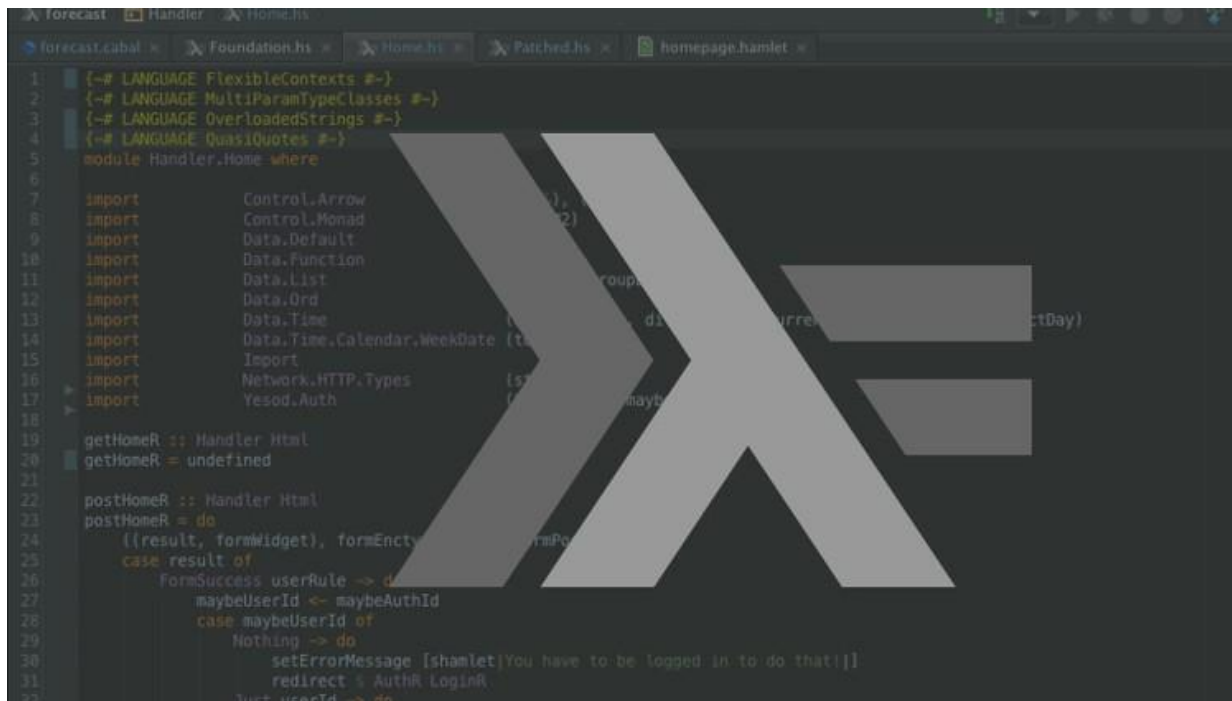
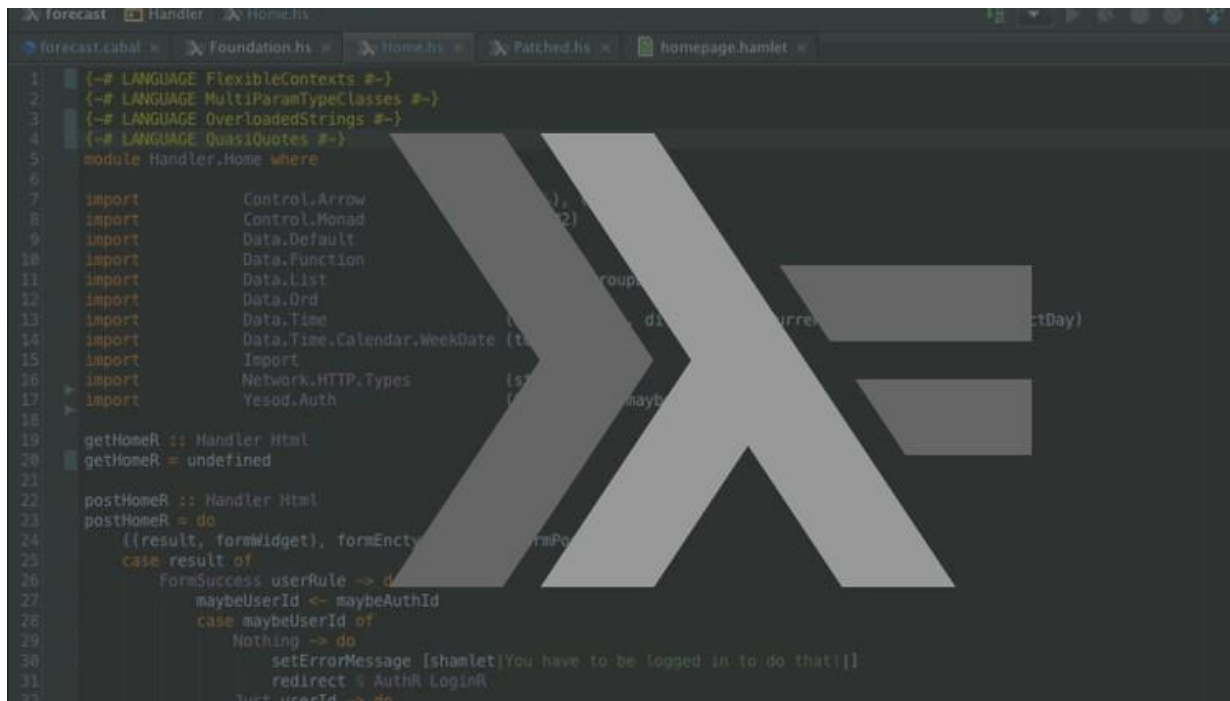


Advanced Types



```
1 {-# LANGUAGE FlexibleContexts #-}
2 {-# LANGUAGE MultiParamTypeClasses #-}
3 {-# LANGUAGE OverloadedStrings #-}
4 {-# LANGUAGE QuasiQuotes #-}
5 module Handler.Home where
6
7 import Control.Arrow
8 import Control.Monad
9 import Data.Default
10 import Data.Function
11 import Data.List
12 import Data.Ord
13 import Data.Time
14 import Data.Time.Calendar.WeekDate
15 import Import
16 import Network.HTTP.Types
17 import Yesod.Auth
18
19 getHomeR :: Handler Html
20 getHomeR = undefined
21
22 postHomeR :: Handler Html
23 postHomeR = do
24   ((result, formWidget), formEncrypt)
25   case result of
26     FormSuccess userRule -> do
27       maybeUserId <- maybeAuthId
28       case maybeUserId of
29         Nothing -> do
30           setErrorMessage [shamlet|You have to be logged in to do that!|]
31           redirect % AuthR.LoginR
32         Just userId -> do
```

Maybe Data Type

A screenshot of a Haskell code editor with a dark theme. The editor shows a file named 'Handler.hs' with several tabs open: 'forecast.cabal', 'Foundation.hs', 'Handler.hs', 'Patched.hs', and 'homepage.hamlet'. The code in 'Handler.hs' includes language extensions, imports, and function definitions. A large, semi-transparent 'X' watermark is overlaid on the code. The code defines a 'Handler' module with imports from 'Control.Monad', 'Data.Default', 'Data.Function', 'Data.List', 'Data.Ord', 'Data.Time', 'Data.Time.Calendar.WeekDate', 'Network.HTTP.Types', and 'Yesod.Auth'. It defines 'getHomeR' and 'postHomeR' functions. 'postHomeR' uses 'maybeAuthId' to handle authentication, with a 'Nothing' case that sets an error message and redirects to the login page.

```
1 {-# LANGUAGE FlexibleContexts #-}
2 {-# LANGUAGE MultiParamTypeClasses #-}
3 {-# LANGUAGE OverloadedStrings #-}
4 {-# LANGUAGE QuasiQuotes #-}
5 module Handler.Home where
6
7 import Control.Monad
8 import Control.Monad
9 import Data.Default
10 import Data.Function
11 import Data.List
12 import Data.Ord
13 import Data.Time
14 import Data.Time.Calendar.WeekDate
15 import Network.HTTP.Types
16 import Yesod.Auth
17
18 getHomeR :: Handler Html
19 getHomeR = undefined
20
21 postHomeR :: Handler Html
22 postHomeR = do
23   ((result, formWidget), formEnctype) <- runFormPost
24   case result of
25     FormSuccess userRule -> do
26       maybeUserId <- maybeAuthId
27       case maybeUserId of
28         Nothing -> do
29           setErrorMessage [shamlet|You have to be logged in to do that!|]
30           redirect % AuthR.LoginR
31       Just userId -> do
```

Maybe a



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

```
data Maybe a = Just a | 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`).

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.