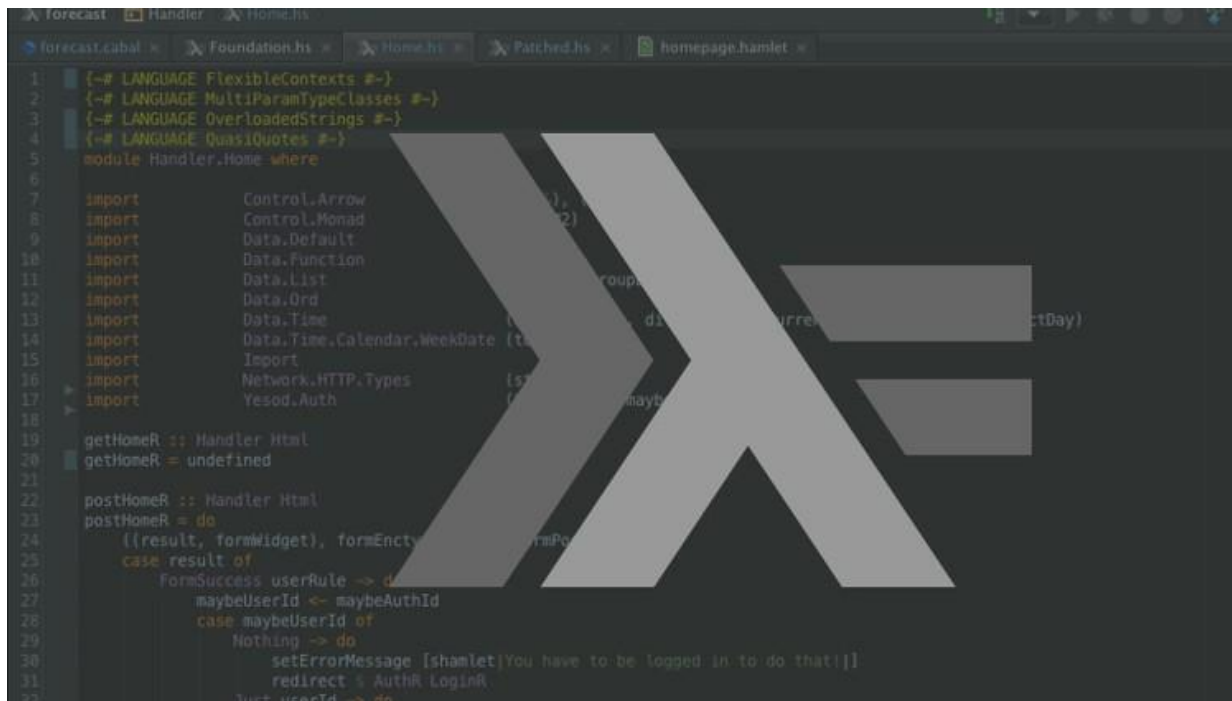
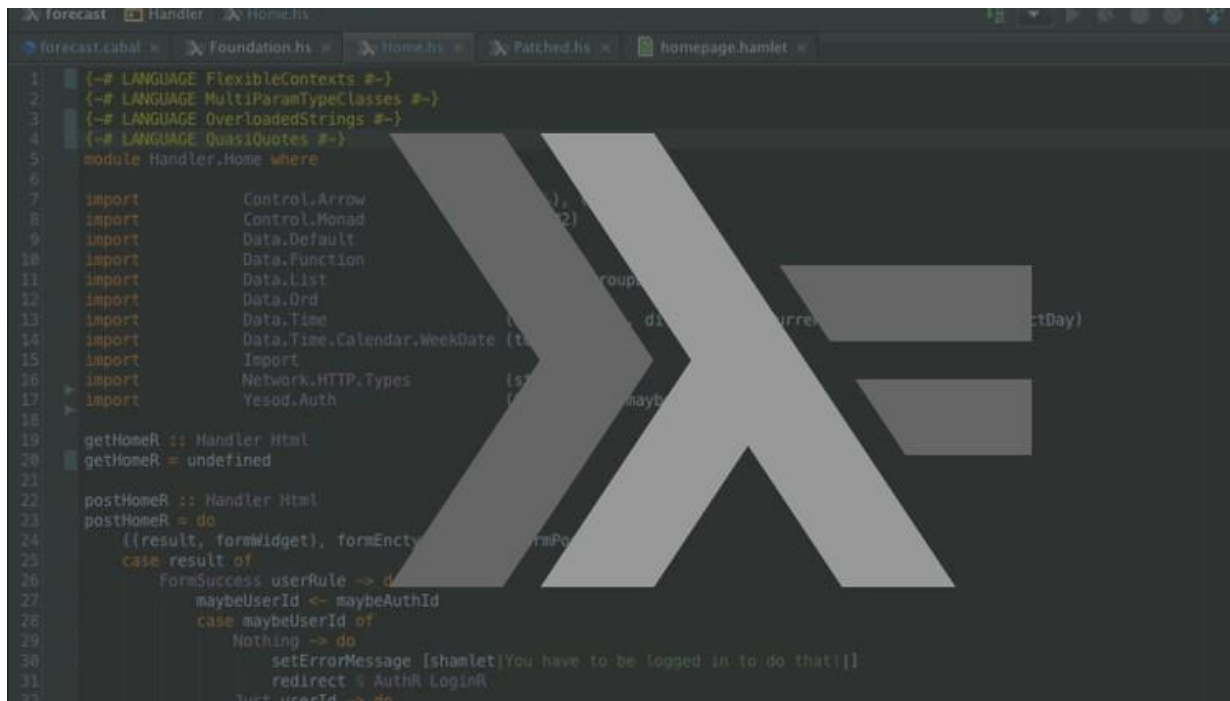


Advanced Types



```
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), formEnctype) <- runPost
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
```

Algebraic Types



The image shows a screenshot of a Haskell code editor with a dark theme. The editor has several tabs at the top: 'forecast.cabal', 'Foundation.hs', 'Home.hs', 'Patched.hs', and 'homepage.hamlet'. The 'Home.hs' tab is active, displaying Haskell code. The code includes language extensions, module declarations, imports, and function definitions. A large, semi-transparent watermark is overlaid on the code.

```
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), formEnctype, formPost)
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
```

Algebraic Types



```
data Shape
= Rectangle Float Float -- height, width
| Square Float -- size
| Circle Float -- radius
| Point
```

Algebraic Types



```
data Shape
  = Rectangle Float Float      -- height, width
  | Square Float               -- size
  | Circle Float               -- radius
  | Point
```

Algebraic types can be deconstructed using patterns:

```
area :: Shape -> Float

area (Rectangle width height) = width * height
area (Square size) = area (Rectangle size size)
area (Circle radius) = pi * radius * radius
area Point = 0
```

```
λ> area (Rectangle 3 4)
👉 12

λ> c = Circle 2.0
λ> area c
👉 12.566370614359172
```