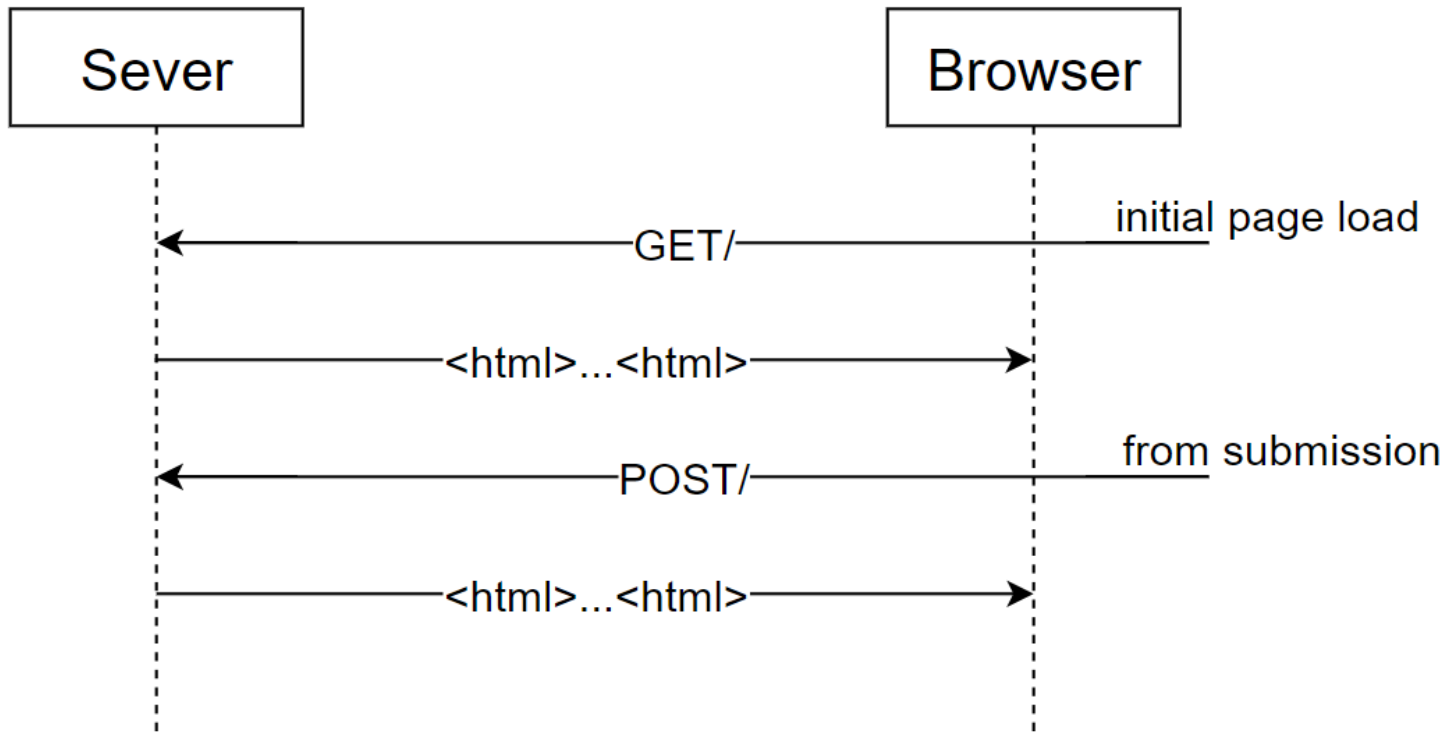
Scala Web and API Servers

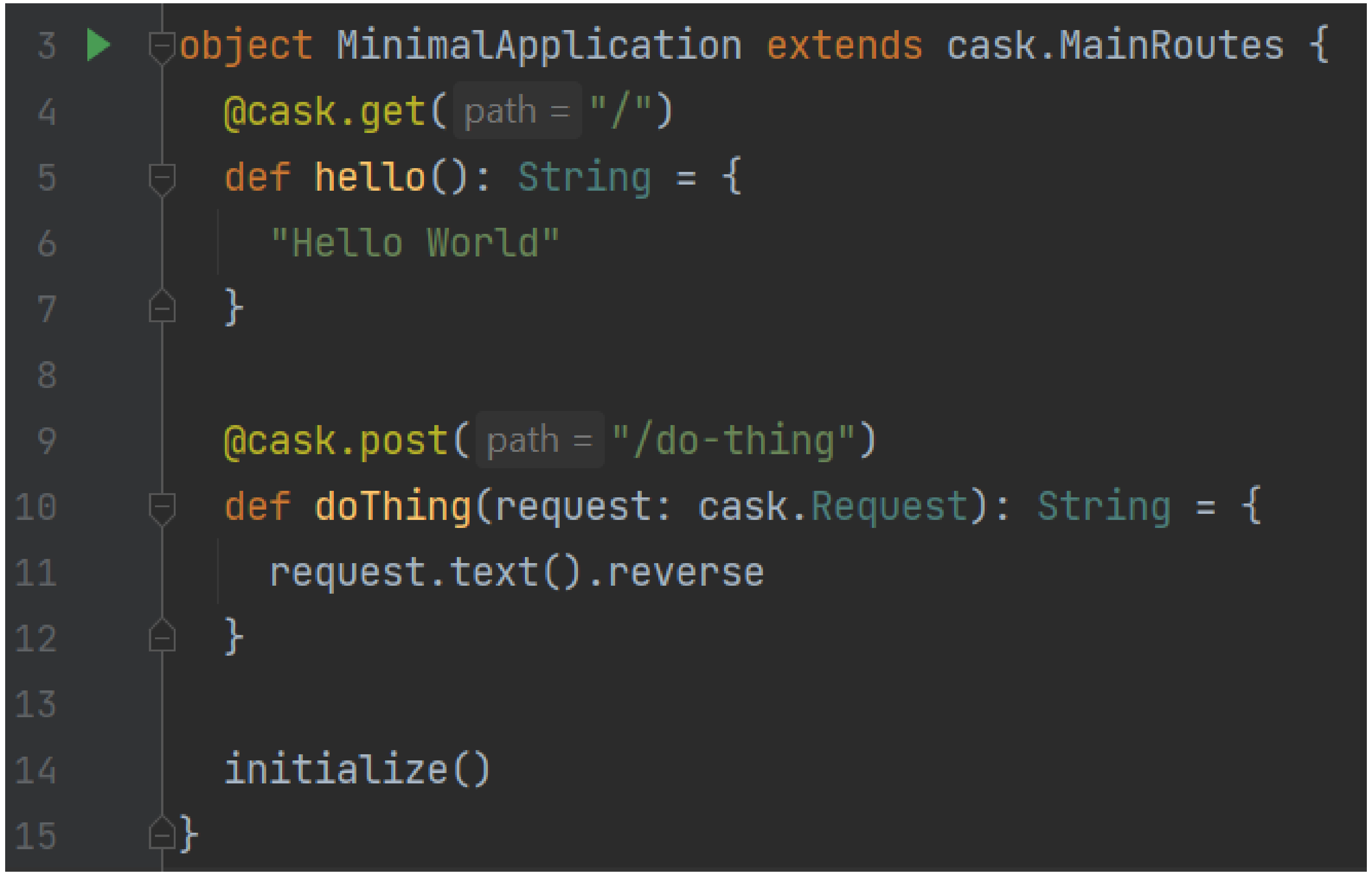
Восьмая лекция

Web Server

- HTTP/HTTPS
- Routes

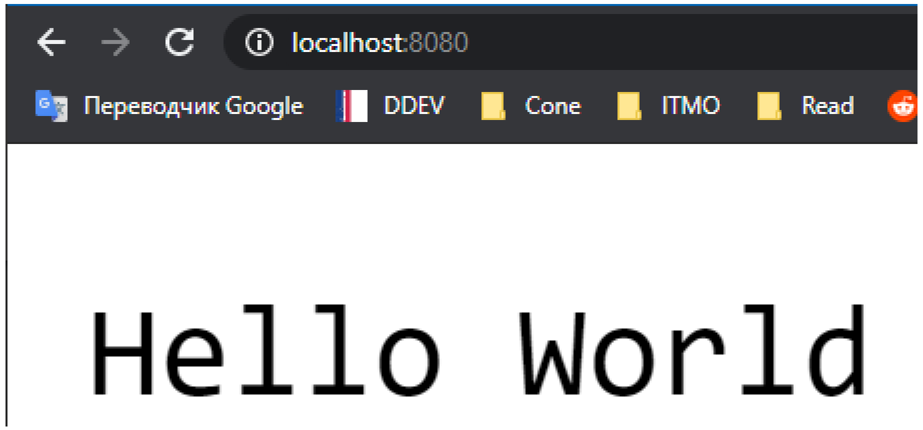


Minimal Application

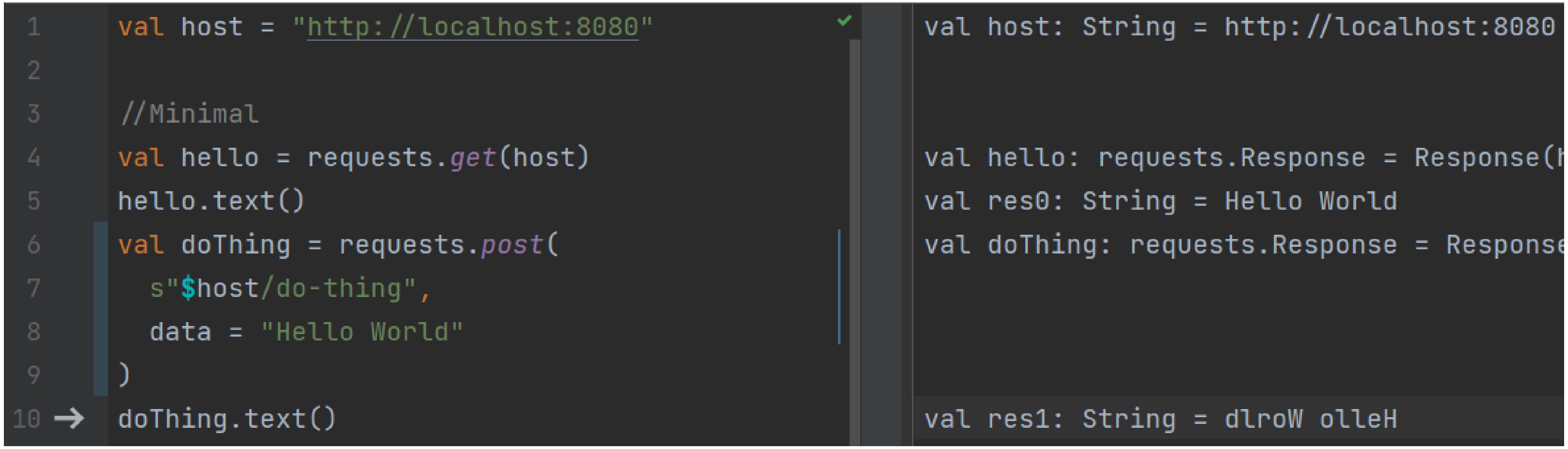


```
object MinimalApplication extends cask.MainRoutes {
  @cask.get( path = "/")
  def hello(): String = {
    "Hello World"
  }

  @cask.post( path = "/do-thing")
  def doThing(request: cask.Request): String = {
    request.text().reverse
  }

  initialize()
}
```

Minimal Application test



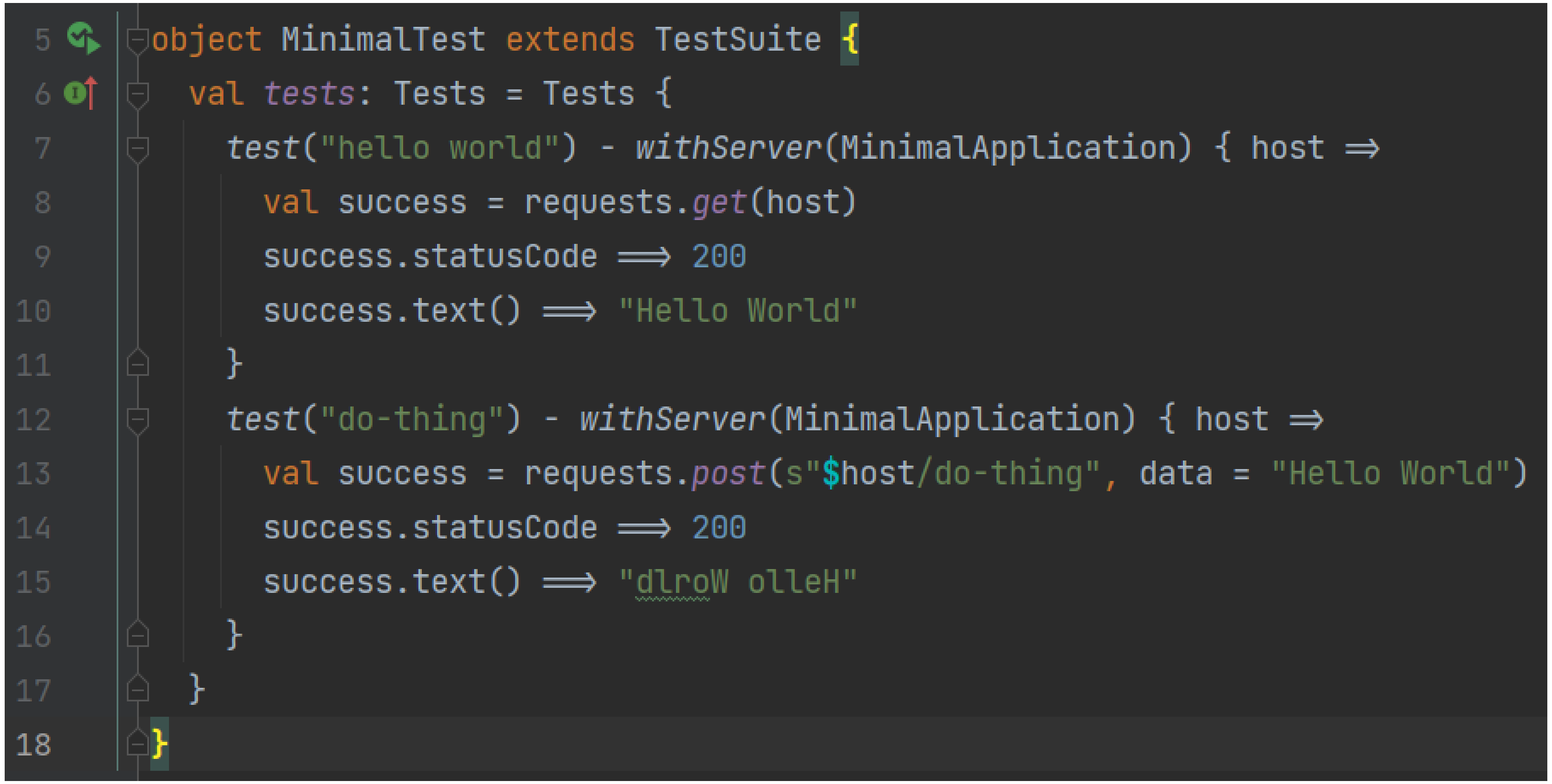
```
Hello World
```



```
val host = "http://localhost:8080"

//Minimal
val hello = requests.get(host)
hello.text()
val doThing = requests.post(
  s"$host/do-thing",
  data = "Hello World"
)
doThing.text()
```

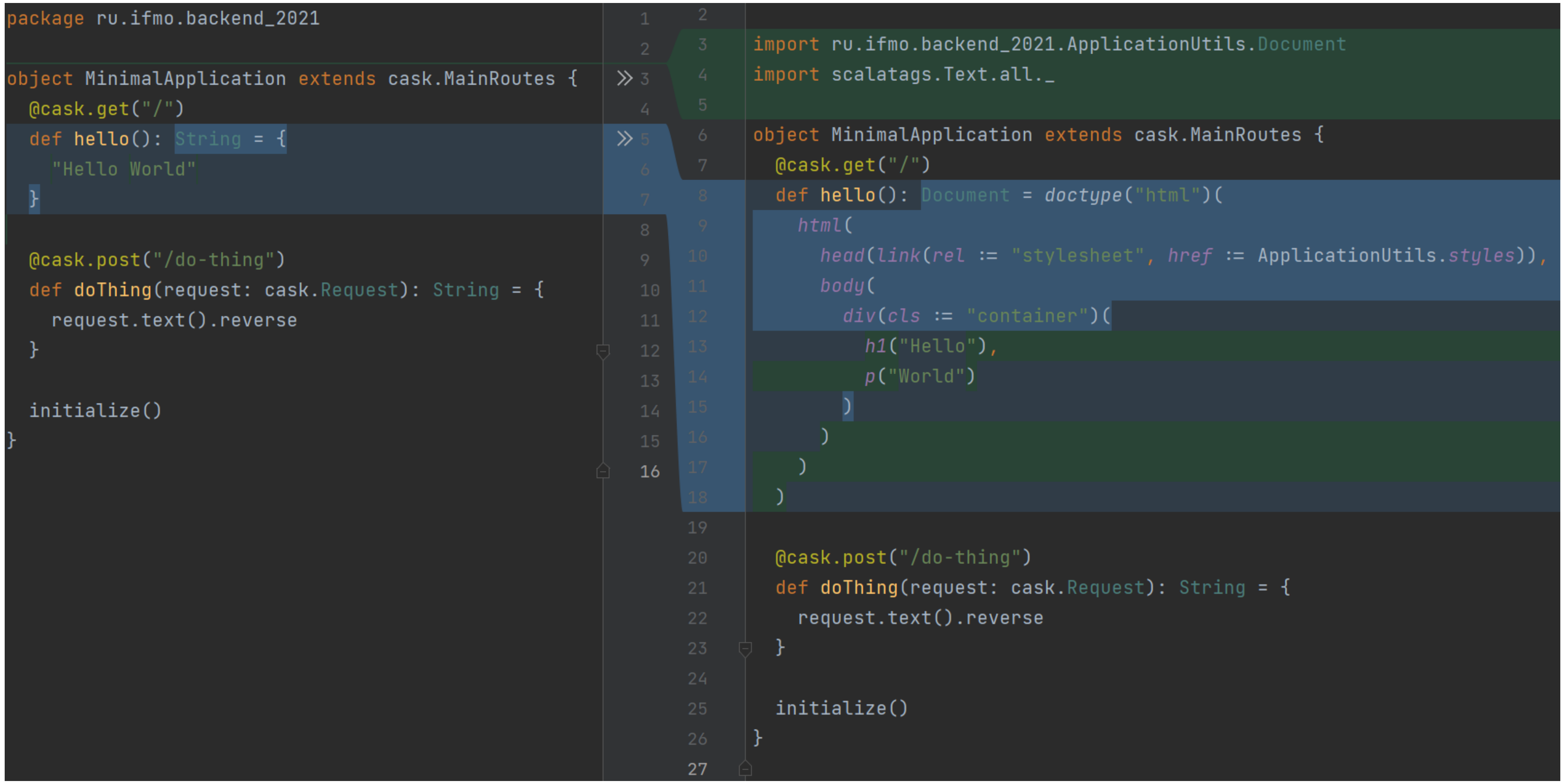
```
val host: String = http://localhost:8080


val hello: requests.Response = Response(
val res0: String = Hello World
val doThing: requests.Response = Response




val res1: String = dlroW olleH
```

Minimal Application tests



```scala
object MinimalTest extends TestSuite {
  val tests: Tests = Tests {
    test("hello world") - withServer(MinimalApplication) { host ⇒
      val success = requests.get(host)
      success.statusCode ==> 200
      success.text() ==> "Hello World"
    }
    test("do-thing") - withServer(MinimalApplication) { host ⇒
      val success = requests.post(s"$host/do-thing", data = "Hello World")
      success.statusCode ==> 200
      success.text() ==> "dlroW olleH"
    }
  }
}
```

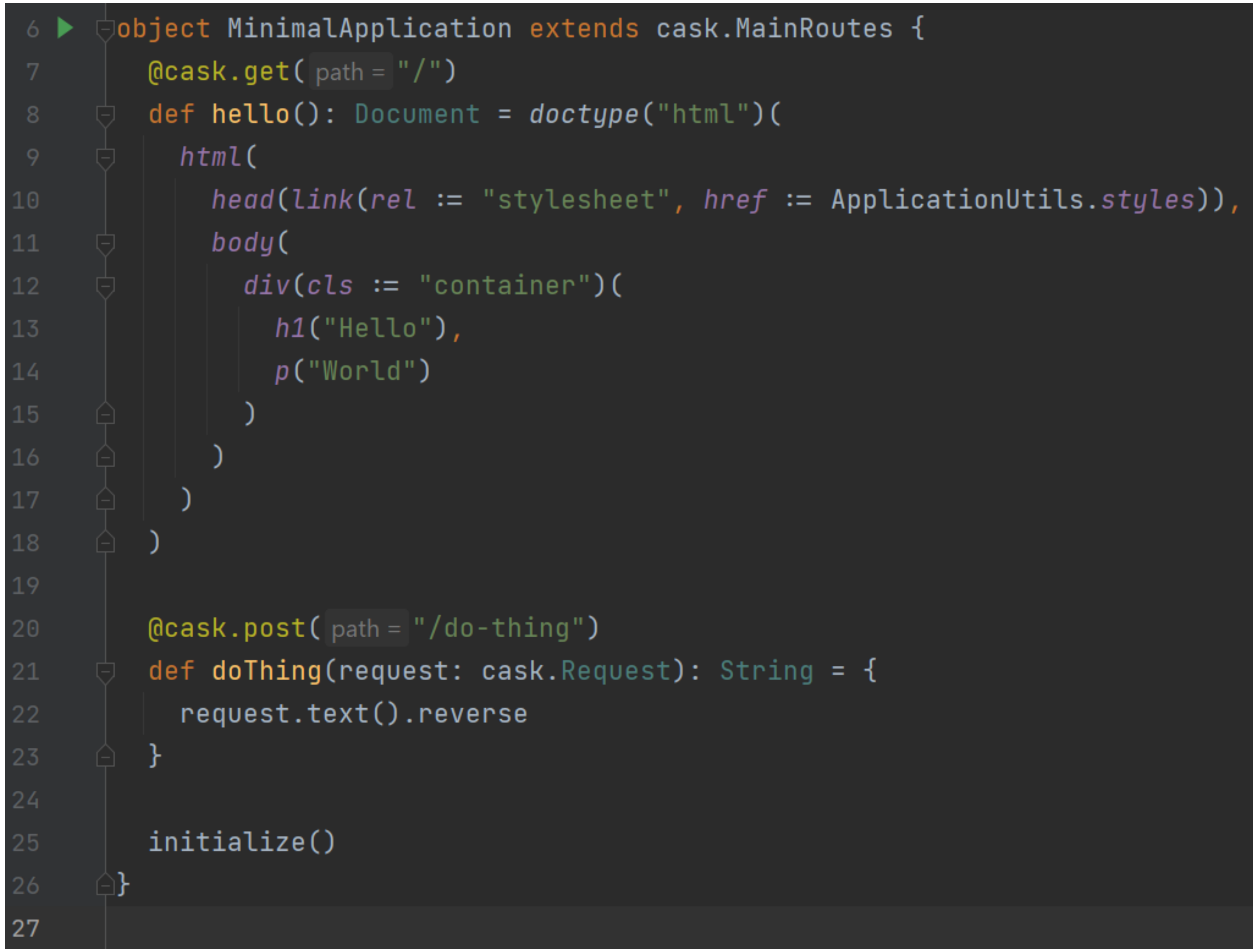
Minimal with tags



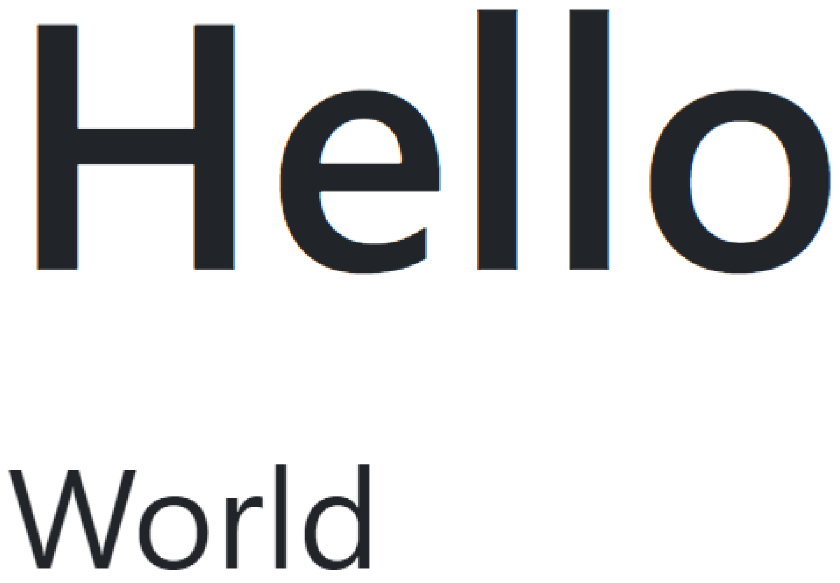
```scala
package ru.ifmo.backend_2021

object MinimalApplication extends cask.MainRoutes {
  @cask.get("/")
  def hello(): String = {
    "Hello World"
  }

  @cask.post("/do-thing")
  def doThing(request: cask.Request): String = {
    request.text().reverse
  }

  initialize()
}
```

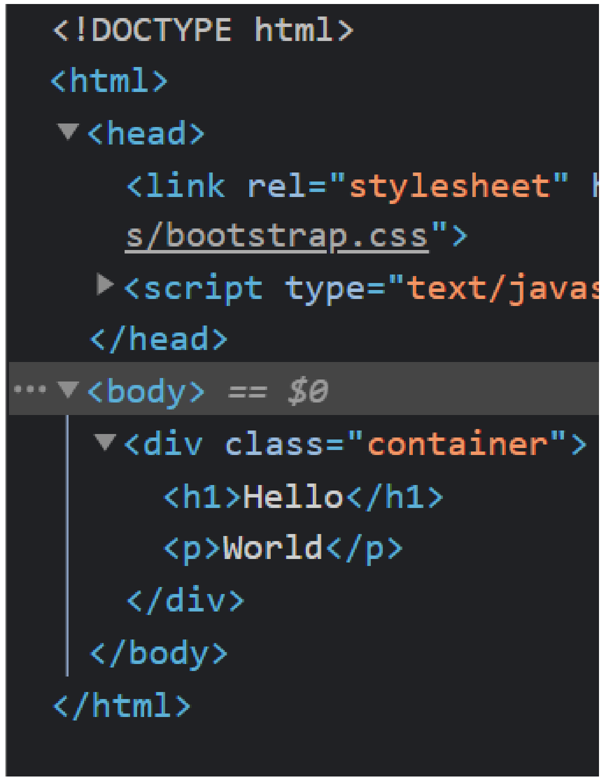
```scala

import ru.ifmo.backend_2021.ApplicationUtils.Document
import scalatags.Text.all._

object MinimalApplication extends cask.MainRoutes {
  @cask.get("/")
  def hello(): Document = doctype("html")(
    html(
      head(link(rel := "stylesheet", href := ApplicationUtils.styles)),
      body(
        div(cls := "container")(
          h1("Hello"),
          p("World")
        )
      )
    )
  )

  @cask.post("/do-thing")
  def doThing(request: cask.Request): String = {
    request.text().reverse
  }

  initialize()
}
```

Minimal with tags

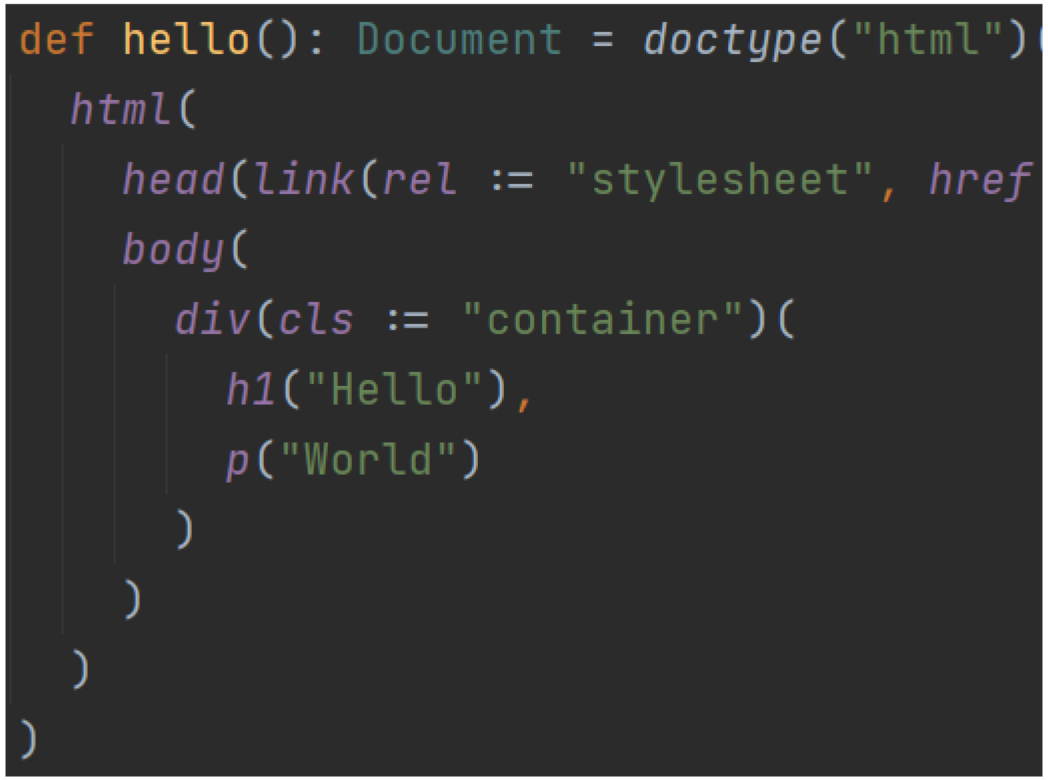


```scala
object MinimalApplication extends cask.MainRoutes {
  @cask.get( path = "/")
  def hello(): Document = doctype("html")(
    html(
      head(link(rel := "stylesheet", href := ApplicationUtils.styles)),
      body(
        div(cls := "container")(
          h1("Hello"),
          p("World")
        )
      )
    )
  )

  @cask.post( path = "/do-thing")
  def doThing(request: cask.Request): String = {
    request.text().reverse
  }

  initialize()
}
```



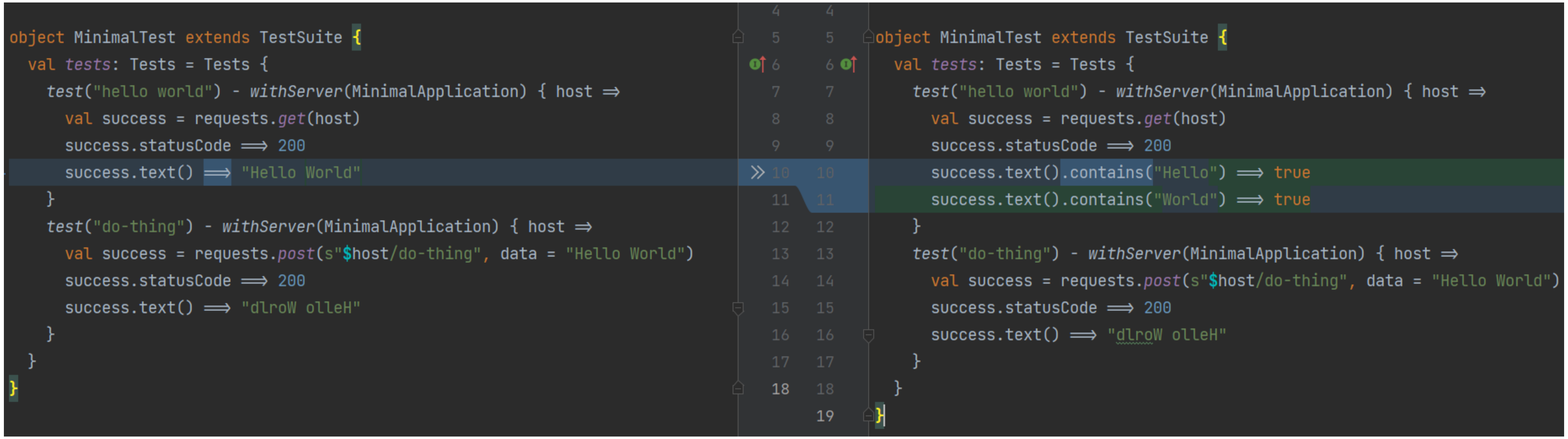


```
<!DOCTYPE html>
<html>
 ▼<head>
    <link rel="stylesheet" 
    s/bootstrap.css">
   ▶<script type="text/javas
   </head>
••• ▼<body> == $0
   ▼<div class="container">
      <h1>Hello</h1>
      <p>World</p>
    </div>
   </body>
</html>
```



```
def hello(): Document = doctype("html")
  html(
    head(link(rel := "stylesheet", href
    body(
      div(cls := "container")(
        h1("Hello"),
        p("World")
      )
    )
  )
)
```

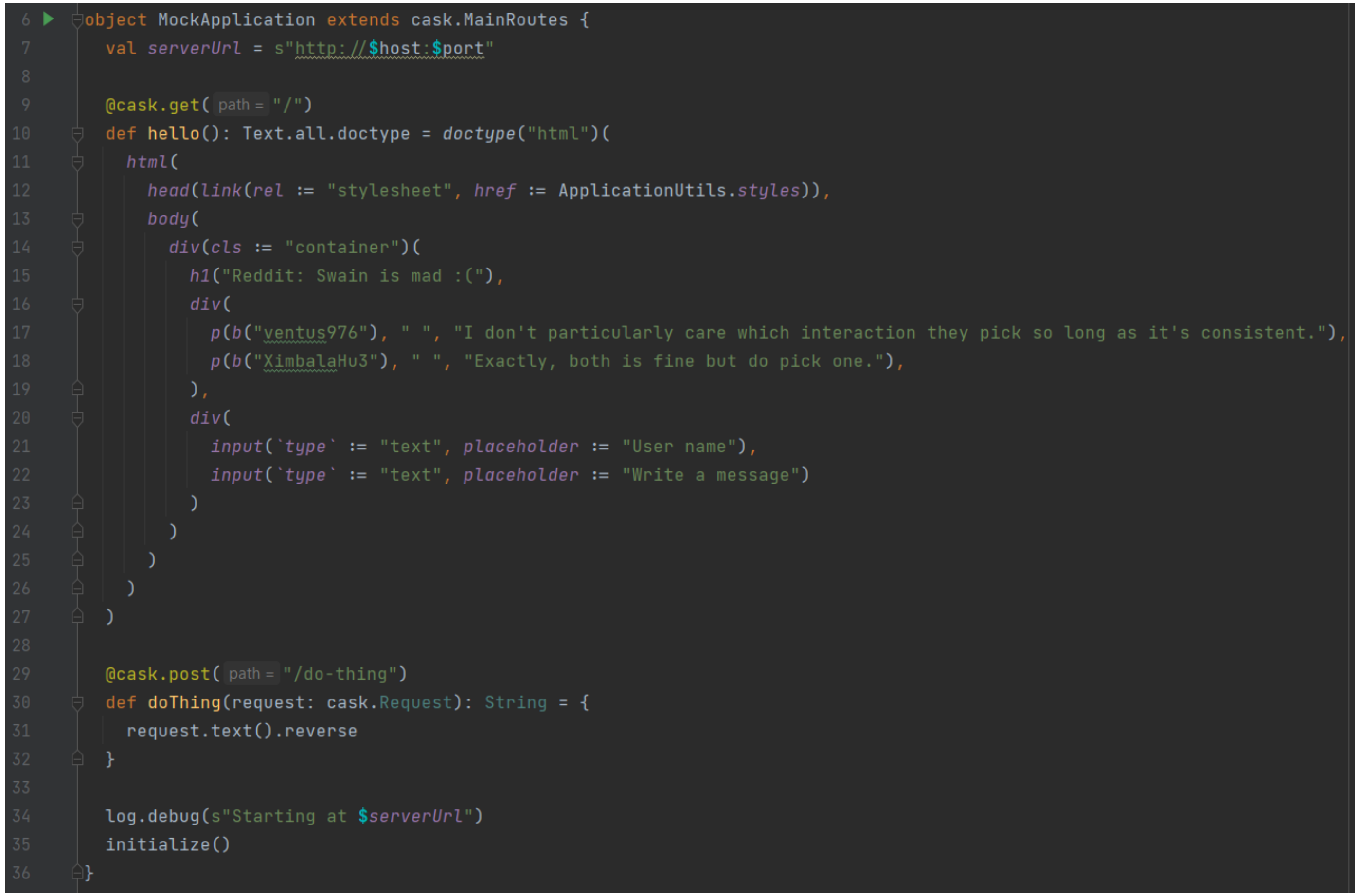
Fix tests



```
object MinimalTest extends TestSuite {
  val tests: Tests = Tests {
    test("hello world") - withServer(MinimalApplication) { host ⇒
      val success = requests.get(host)
      success.statusCode ⟹ 200
      success.text() ⟹ "Hello World"
    }
    test("do-thing") - withServer(MinimalApplication) { host ⇒
      val success = requests.post(s"$host/do-thing", data = "Hello World")
      success.statusCode ⟹ 200
      success.text() ⟹ "dlroW olleH"
    }
  }
}
```

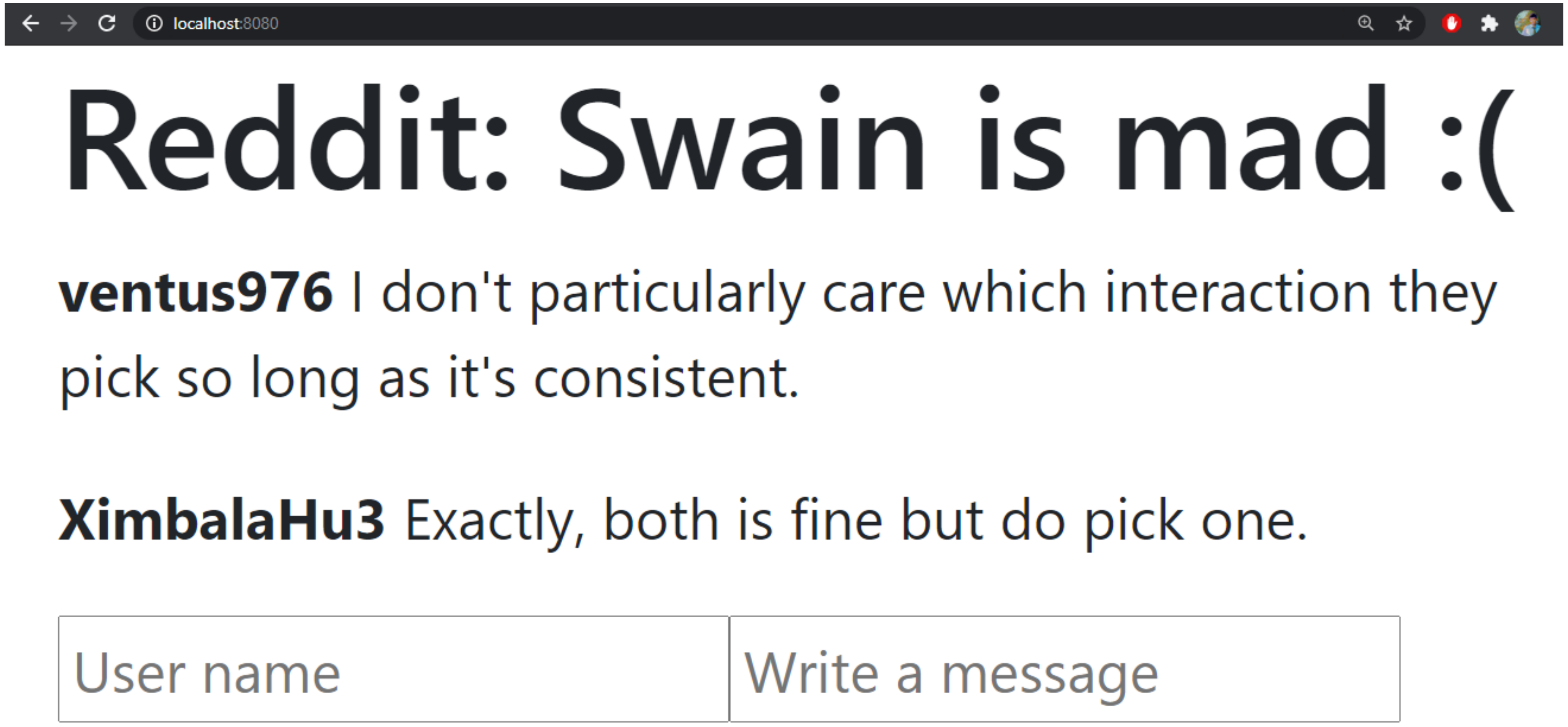
```
object MinimalTest extends TestSuite {
  val tests: Tests = Tests {
    test("hello world") - withServer(MinimalApplication) { host ⇒
      val success = requests.get(host)
      success.statusCode ⟹ 200
      success.text().contains("Hello") ⟹ true
      success.text().contains("World") ⟹ true
    }
    test("do-thing") - withServer(MinimalApplication) { host ⇒
      val success = requests.post(s"$host/do-thing", data = "Hello World")
      success.statusCode ⟹ 200
      success.text() ⟹ "dlroW olleH"
    }
  }
}
```

Mock Reddit



```scala
object MockApplication extends cask.MainRoutes {
  val serverUrl = s"http://$host:$port"

  @cask.get( path = "/")
  def hello(): Text.all.doctype = doctype("html")(
    html(
      head(link(rel := "stylesheet", href := ApplicationUtils.styles)),
      body(
        div(cls := "container")(
          h1("Reddit: Swain is mad :("),
          div(
            p(b("ventus976"), " ", "I don't particularly care which interaction they pick so long as it's consistent."),
            p(b("XimbalaHu3"), " ", "Exactly, both is fine but do pick one."),
          ),
          div(
            input(`type` := "text", placeholder := "User name"),
            input(`type` := "text", placeholder := "Write a message")
          )
        )
      )
    )
  )

  @cask.post( path = "/do-thing")
  def doThing(request: cask.Request): String = {
    request.text().reverse
  }

  log.debug(s"Starting at $serverUrl")
  initialize()
}
```

Mock Reddit Page

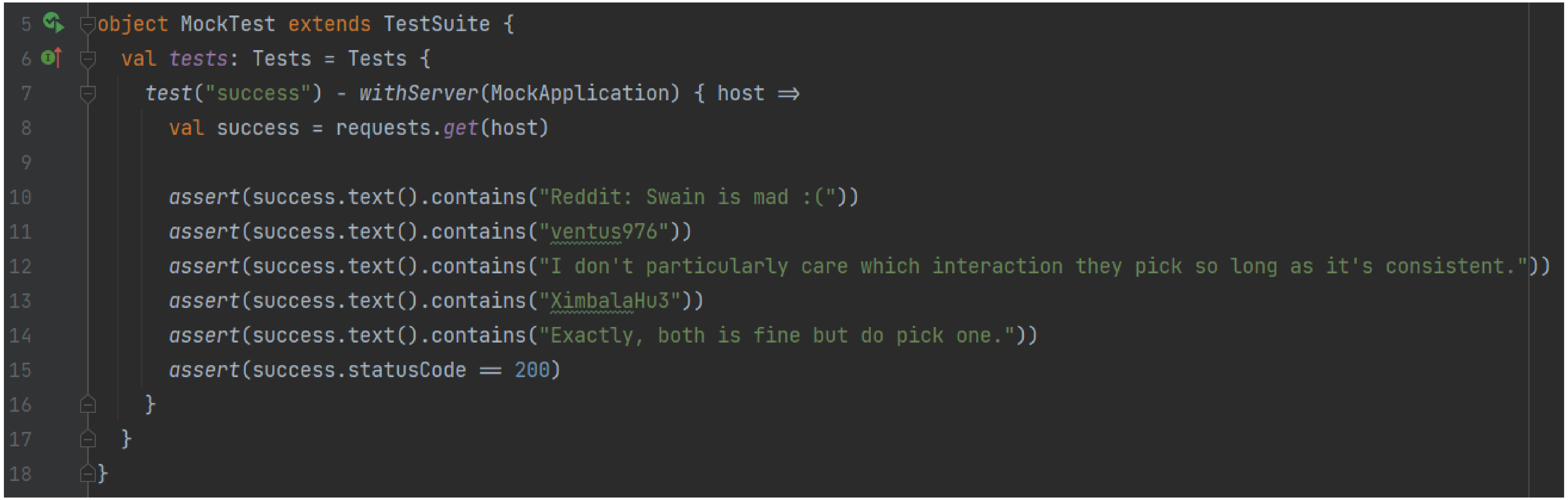

localhost:8080

Reddit: Swain is mad :(

ventus976 I don't particularly care which interaction they pick so long as it's consistent.

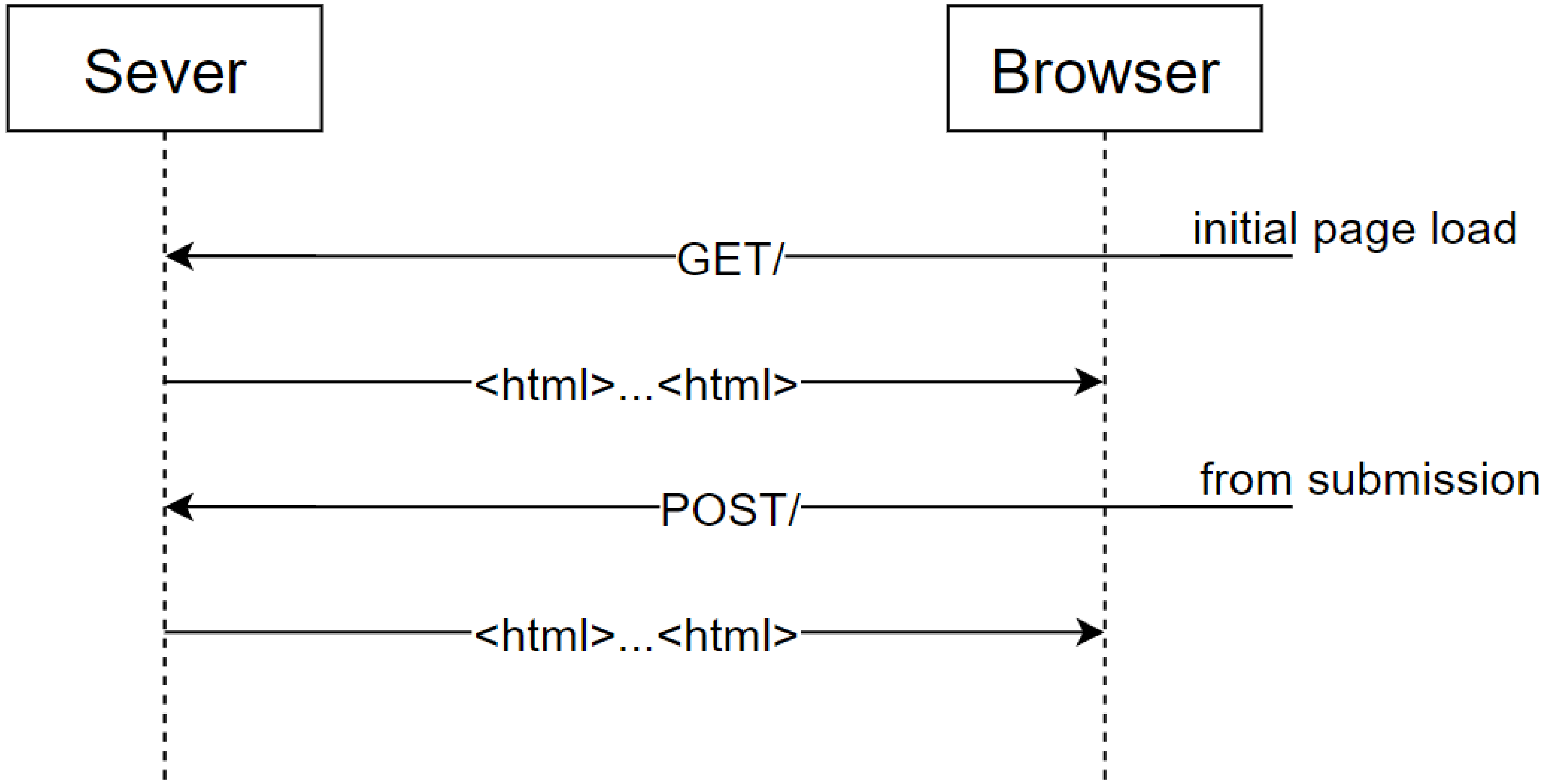
XimbalaHu3 Exactly, both is fine but do pick one.

Mock Reddit Tests

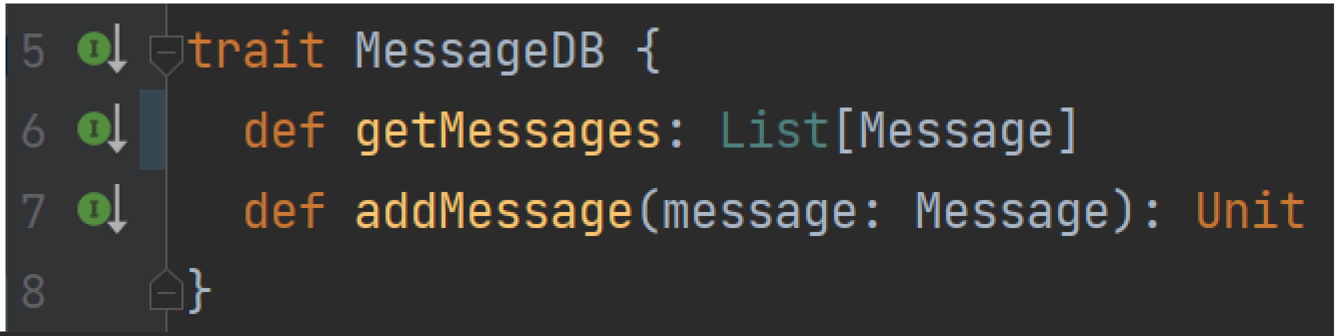


```scala
object MockTest extends TestSuite {
  val tests: Tests = Tests {
    test("success") - withServer(MockApplication) { host =>
      val success = requests.get(host)

      assert(success.text().contains("Reddit: Swain is mad :("))
      assert(success.text().contains("ventus976"))
      assert(success.text().contains("I don't particularly care which interaction they pick so long as it's consistent."))
      assert(success.text().contains("XimbalaHu3"))
      assert(success.text().contains("Exactly, both is fine but do pick one."))
      assert(success.statusCode == 200)
    }
  }
}
```

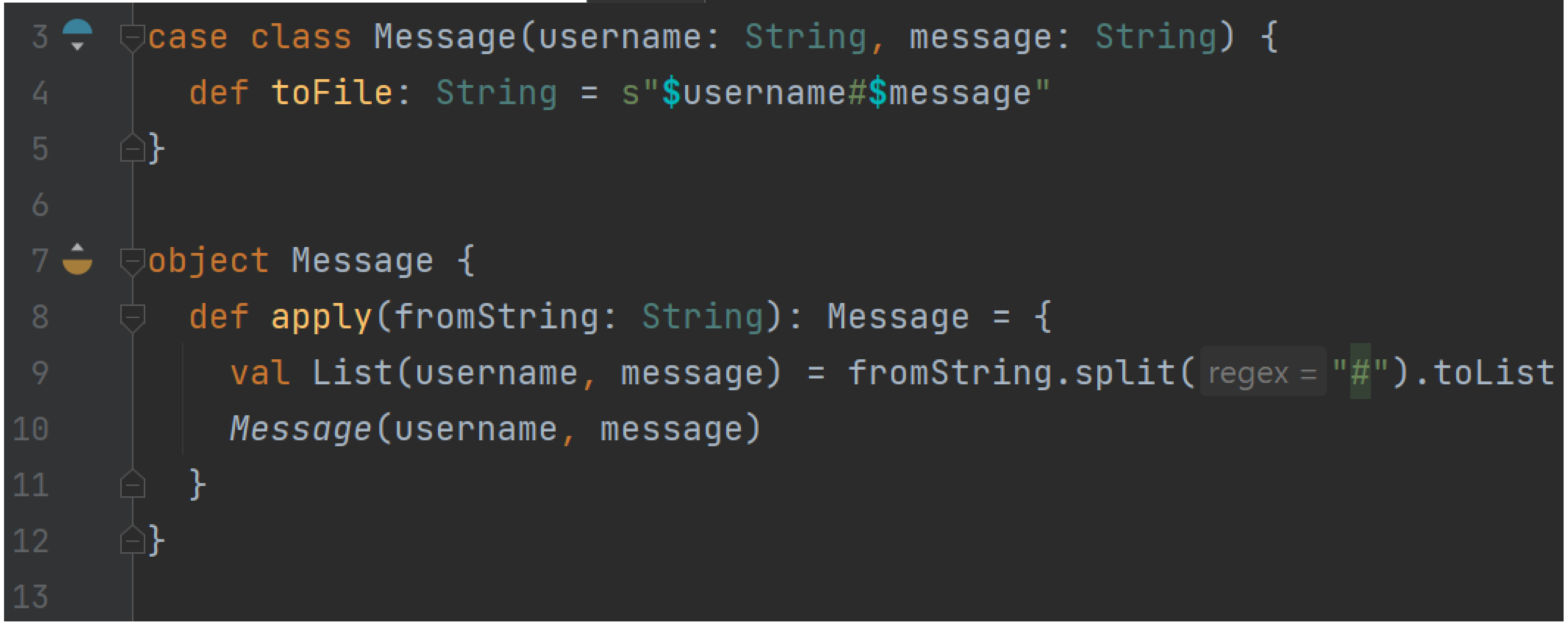
Webserver workflow



MessageDB

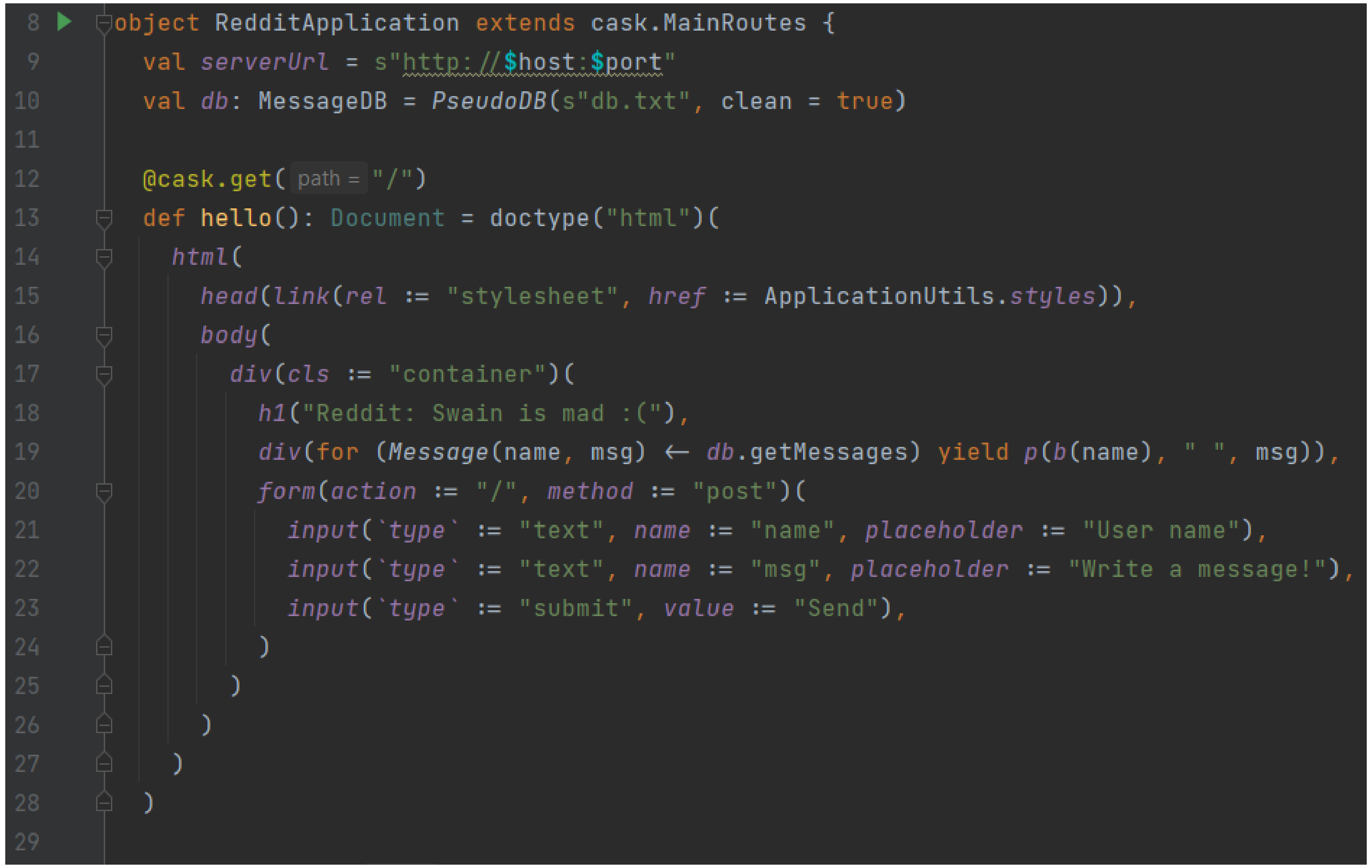


```scala
trait MessageDB {
  def getMessages: List[Message]
  def addMessage(message: Message): Unit
}
```



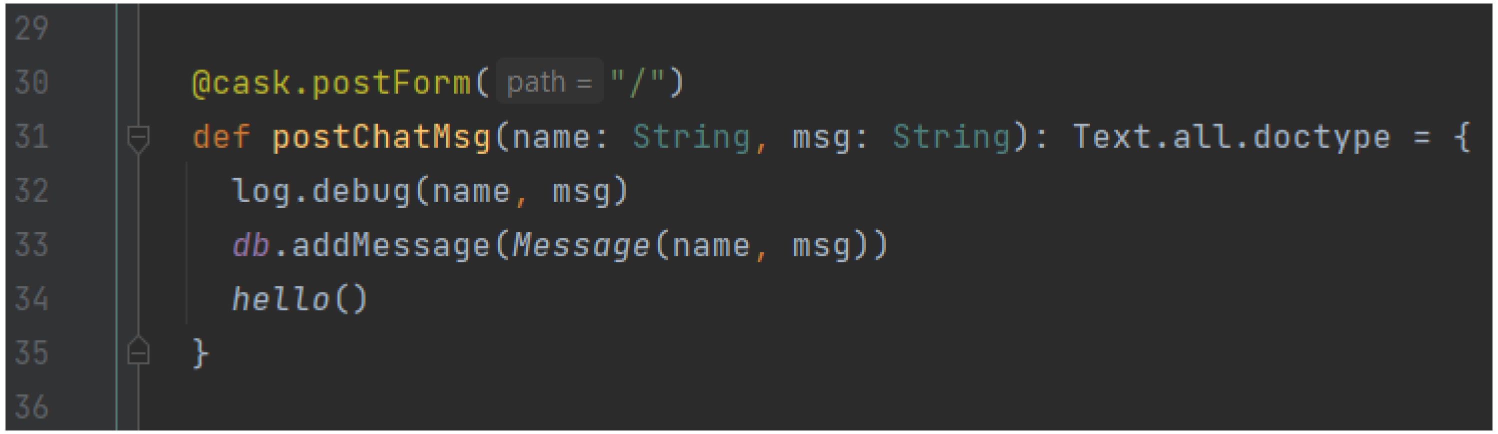
```scala
case class Message(username: String, message: String) {
  def toFile: String = s"$username#$message"
}

object Message {
  def apply(fromString: String): Message = {
    val List(username, message) = fromString.split( regex = "#").toList
    Message(username, message)
  }
}

```

Reddit Application



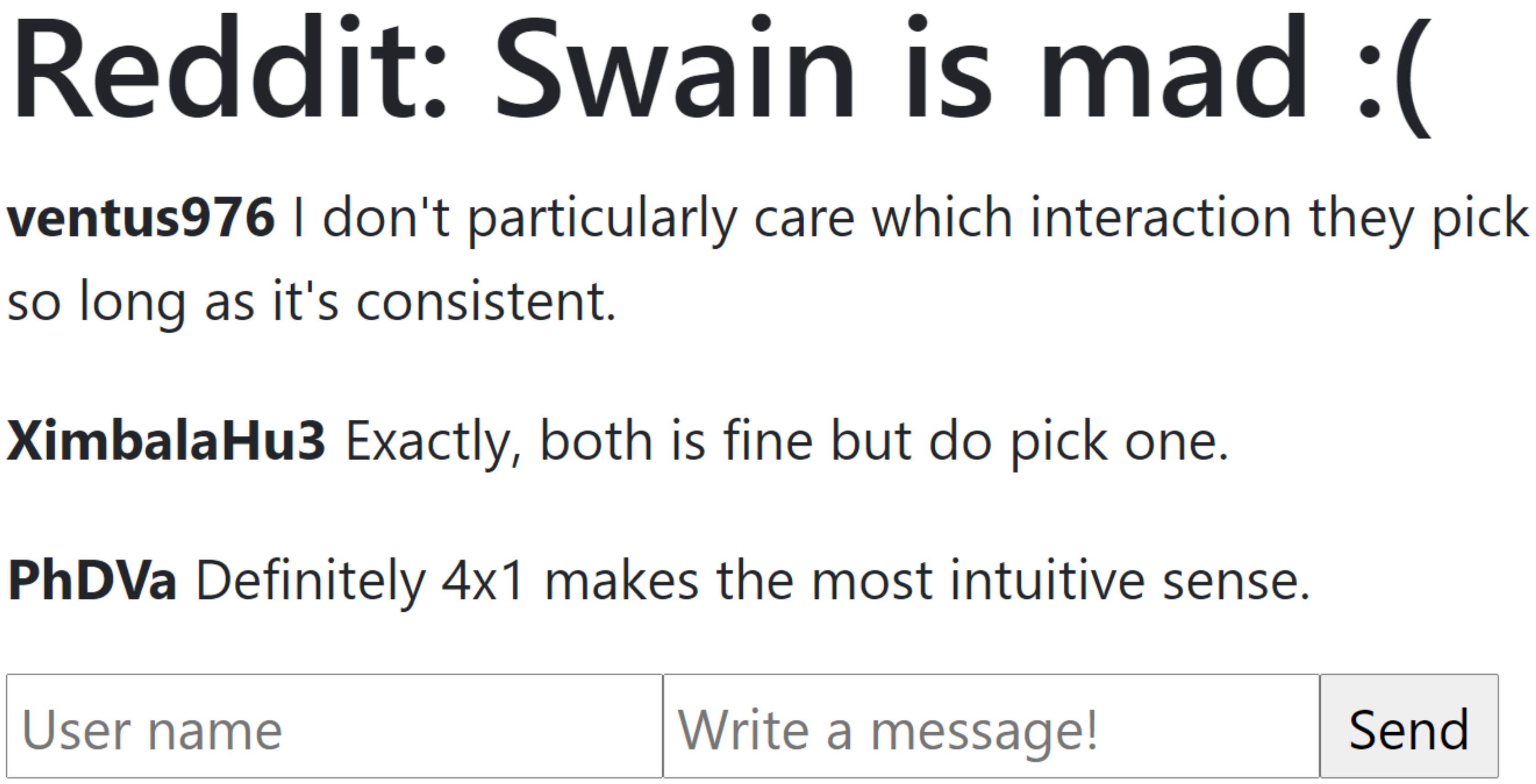
```scala
object RedditApplication extends cask.MainRoutes {
  val serverUrl = s"http://$host:$port"
  val db: MessageDB = PseudoDB(s"db.txt", clean = true)

  @cask.get("/")
  def hello(): Document = doctype("html")(
    html(
      head(link(rel := "stylesheet", href := ApplicationUtils.styles)),
      body(
        div(cls := "container")(
          h1("Reddit: Swain is mad :("),
          div(for (Message(name, msg) ← db.getMessages) yield p(b(name), " ", msg)),
          form(action := "/", method := "post")(
            input(`type` := "text", name := "name", placeholder := "User name"),
            input(`type` := "text", name := "msg", placeholder := "Write a message!"),
            input(`type` := "submit", value := "Send"),
          )
        )
      )
    )
  )

```

Form Handling



```
@cask.postForm( path = "/")
def postChatMsg(name: String, msg: String): Text.all.doctype = {
  log.debug(name, msg)
  db.addMessage(Message(name, msg))
  hello()
}
```

Result

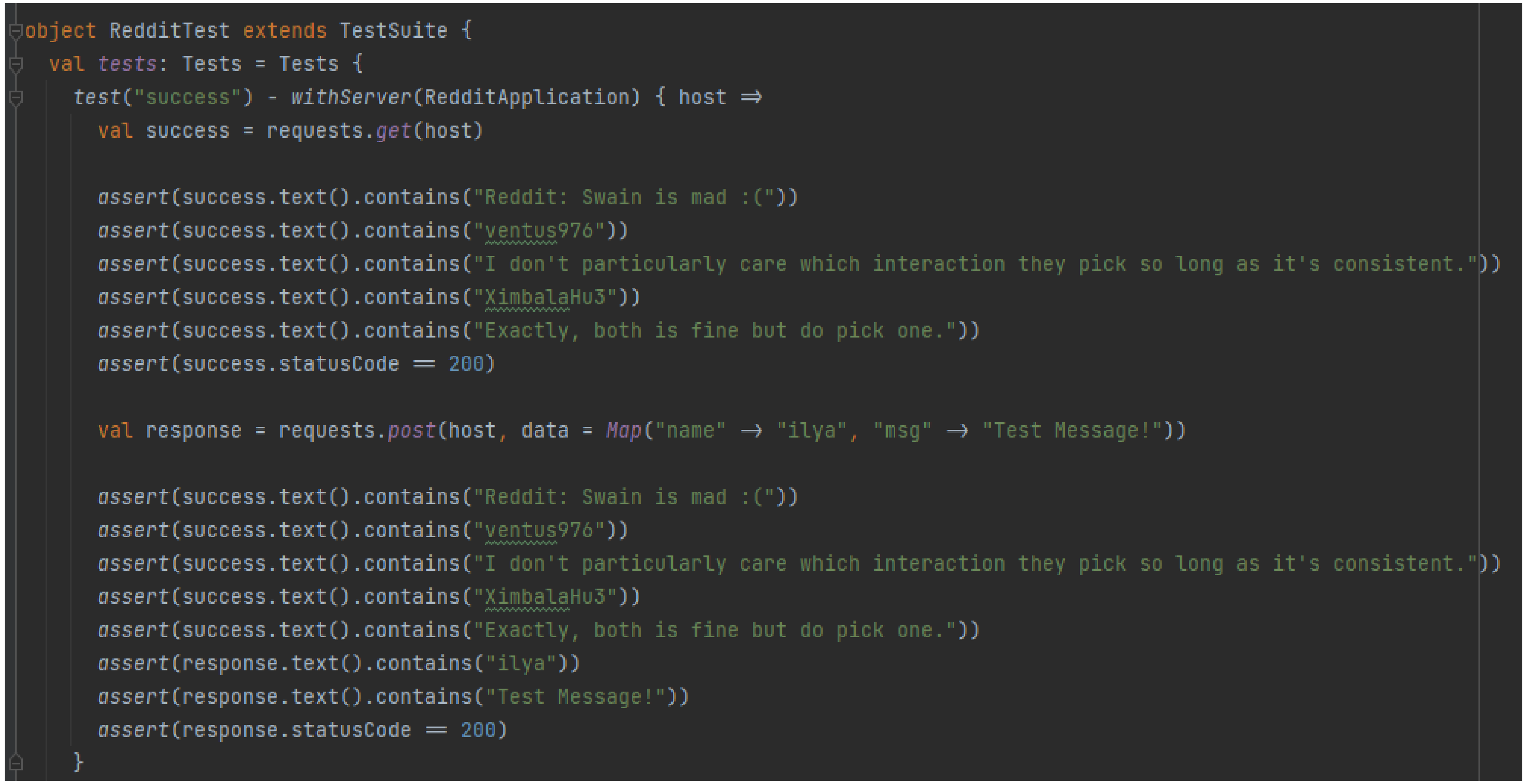

Reddit: Swain is mad :(

ventus976 I don't particularly care which interaction they pick so long as it's consistent.

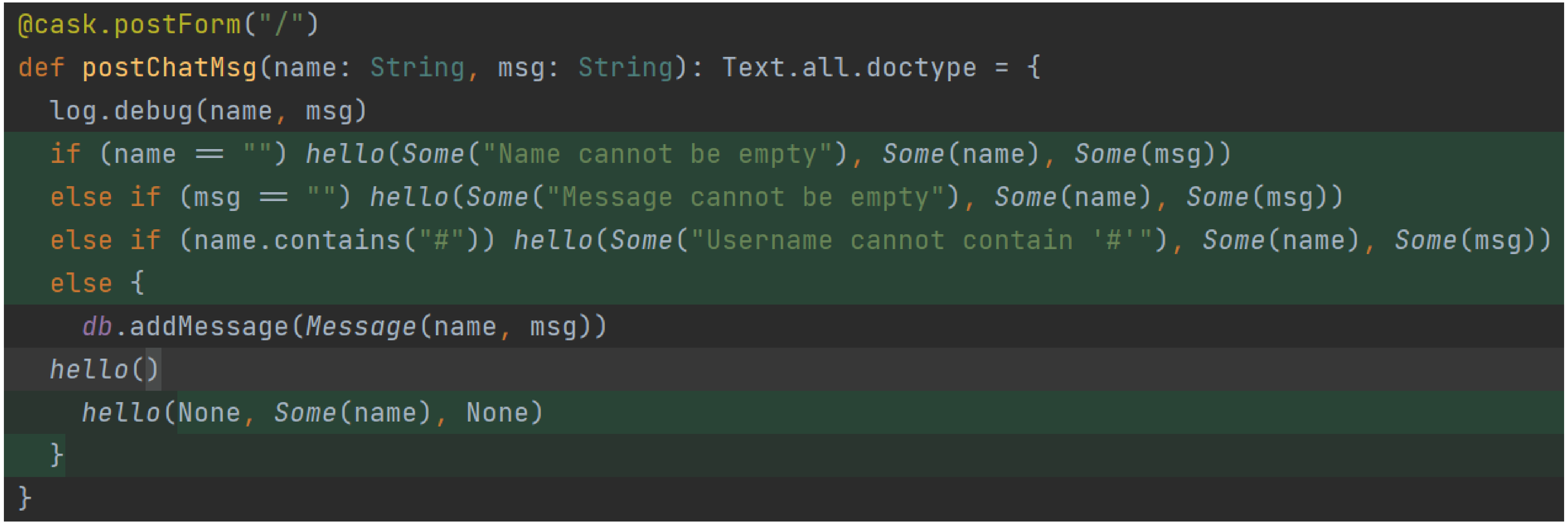
XimbalaHu3 Exactly, both is fine but do pick one.

PhDVa Definitely 4x1 makes the most intuitive sense.

More tests

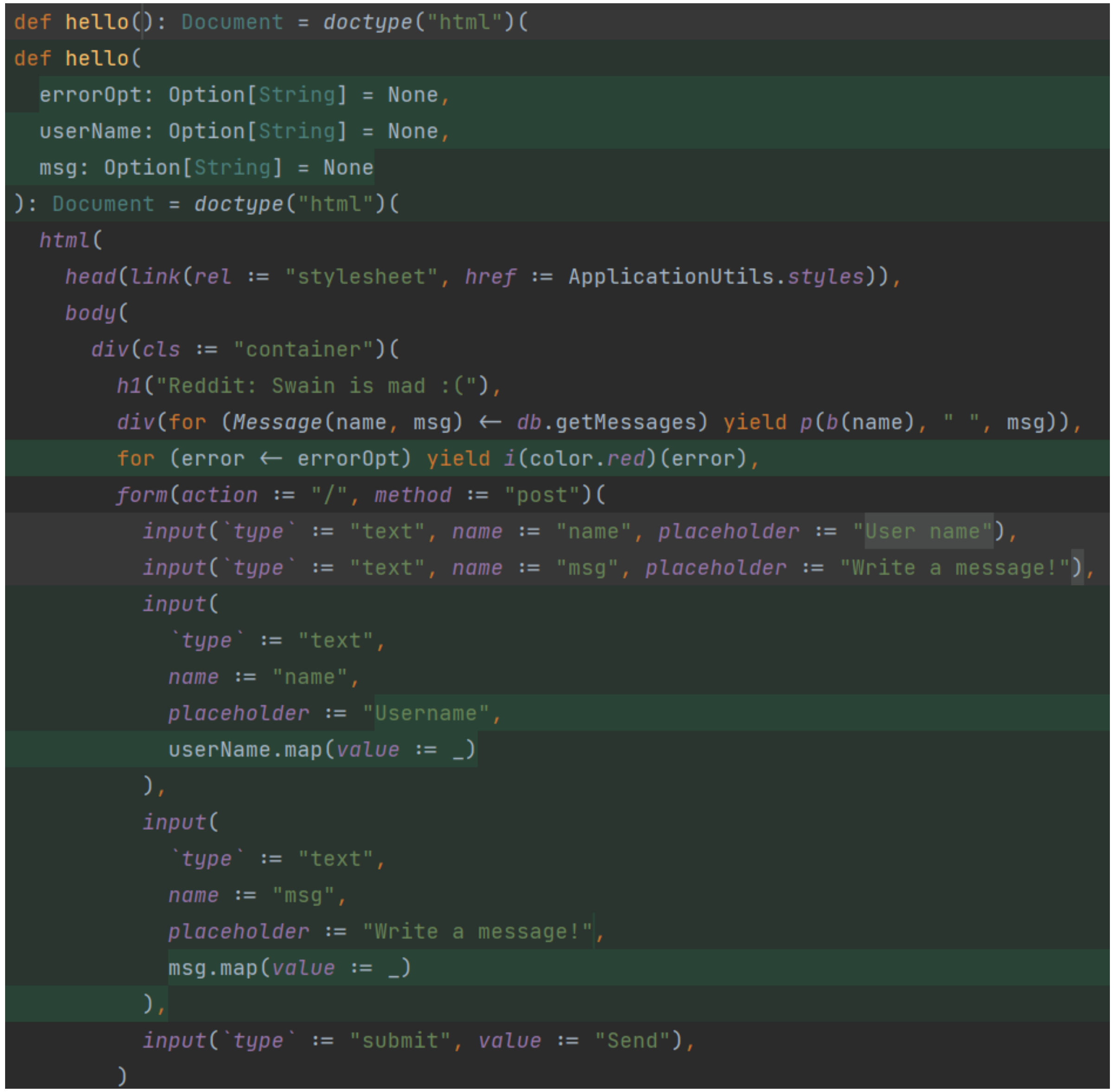

```scala
object RedditTest extends TestSuite {
  val tests: Tests = Tests {
    test("success") - withServer(RedditApplication) { host =>
      val success = requests.get(host)

      assert(success.text().contains("Reddit: Swain is mad :("))
      assert(success.text().contains("ventus976"))
      assert(success.text().contains("I don't particularly care which interaction they pick so long as it's consistent."))
      assert(success.text().contains("XimbalaHu3"))
      assert(success.text().contains("Exactly, both is fine but do pick one."))
      assert(success.statusCode == 200)

      val response = requests.post(host, data = Map("name" -> "ilya", "msg" -> "Test Message!"))

      assert(success.text().contains("Reddit: Swain is mad :("))
      assert(success.text().contains("ventus976"))
      assert(success.text().contains("I don't particularly care which interaction they pick so long as it's consistent."))
      assert(success.text().contains("XimbalaHu3"))
      assert(success.text().contains("Exactly, both is fine but do pick one."))
      assert(response.text().contains("ilya"))
      assert(response.text().contains("Test Message!"))
      assert(response.statusCode == 200)
    }
```

Error handling and usability



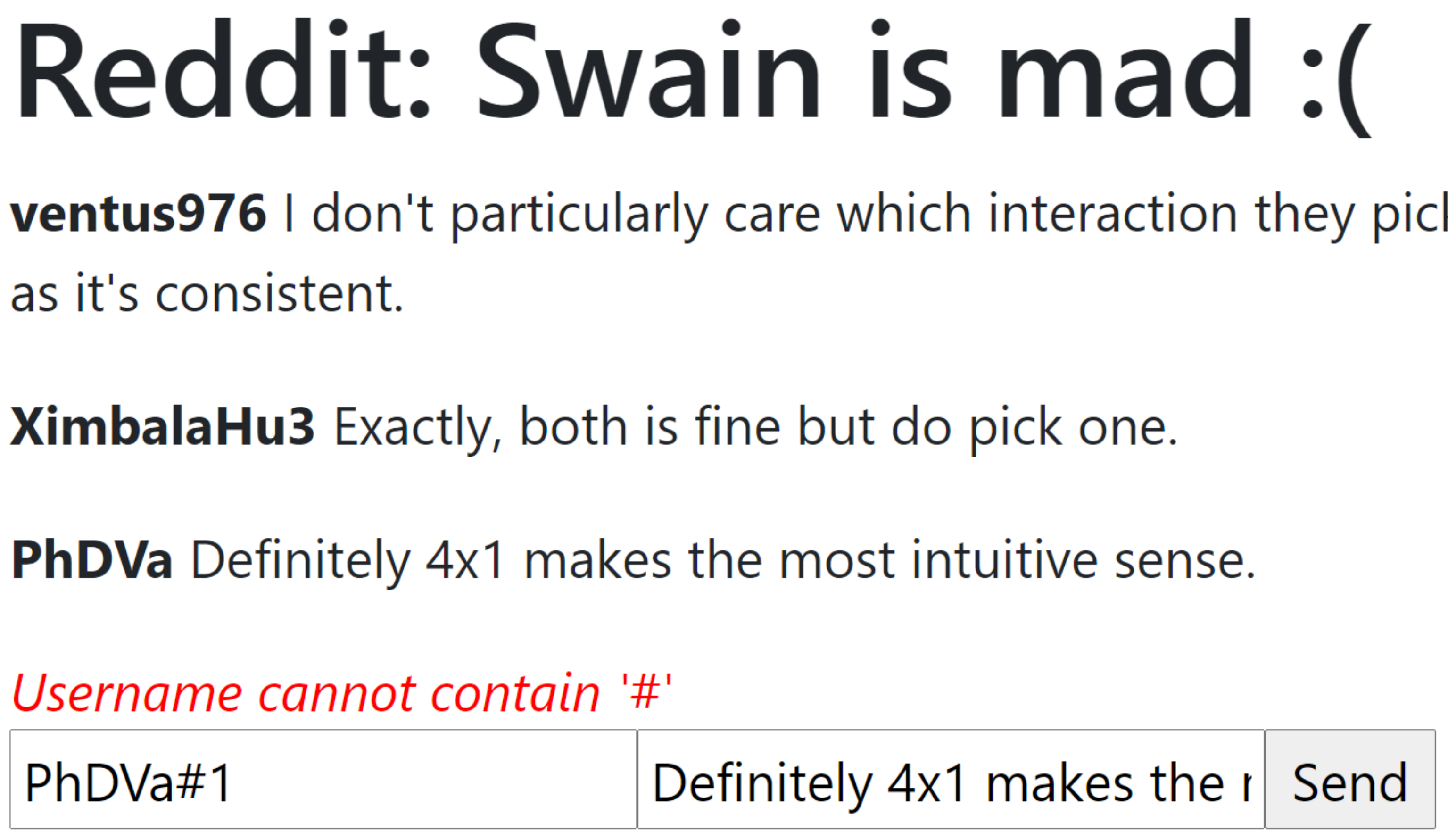
```scala
@cask.postForm("/")
def postChatMsg(name: String, msg: String): Text.all.doctype = {
  log.debug(name, msg)
  if (name == "") hello(Some("Name cannot be empty"), Some(name), Some(msg))
  else if (msg == "") hello(Some("Message cannot be empty"), Some(name), Some(msg))
  else if (name.contains("#")) hello(Some("Username cannot contain '#'"), Some(name), Some(msg))
  else {
    db.addMessage(Message(name, msg))
  hello()
    hello(None, Some(name), None)
  }
}
```

Errors display



```
def hello(): Document = doctype("html")(
def hello(
  errorOpt: Option[String] = None,
  userName: Option[String] = None,
  msg: Option[String] = None
): Document = doctype("html")(
  html(
    head(link(rel := "stylesheet", href := ApplicationUtils.styles)),
    body(
      div(cls := "container")(
        h1("Reddit: Swain is mad :("),
        div(for (Message(name, msg) ← db.getMessages) yield p(b(name), " ", msg)),
        for (error ← errorOpt) yield i(color.red)(error),
        form(action := "/", method := "post")(
          input(`type` := "text", name := "name", placeholder := "User name"),
          input(`type` := "text", name := "msg", placeholder := "Write a message!"),
          input(
            `type` := "text",
            name := "name",
            placeholder := "Username",
            userName.map(value := _)
          ),
          input(
            `type` := "text",
            name := "msg",
            placeholder := "Write a message!",
            msg.map(value := _)
          ),
          input(`type` := "submit", value := "Send"),
        )
```

Error display result



Reddit: Swain is mad :(

ventus976 I don't particularly care which interaction they pick so long as it's consistent.

XimbalaHu3 Exactly, both is fine but do pick one.

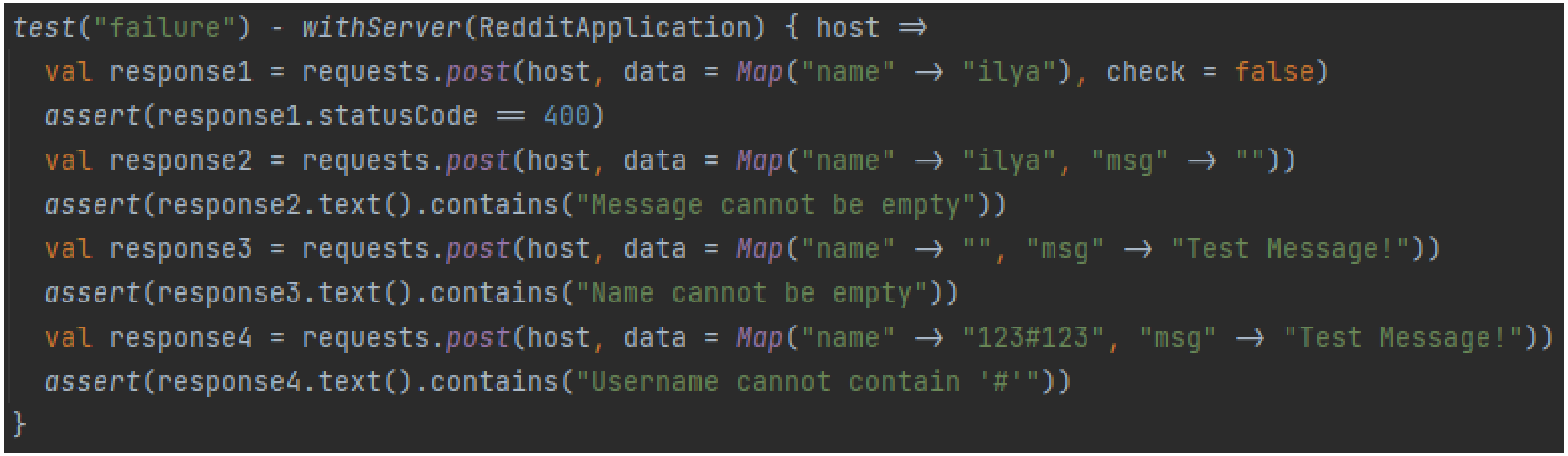
PhDVa Definitely 4x1 makes the most intuitive sense.

Username cannot contain '#'

PhDVa#1

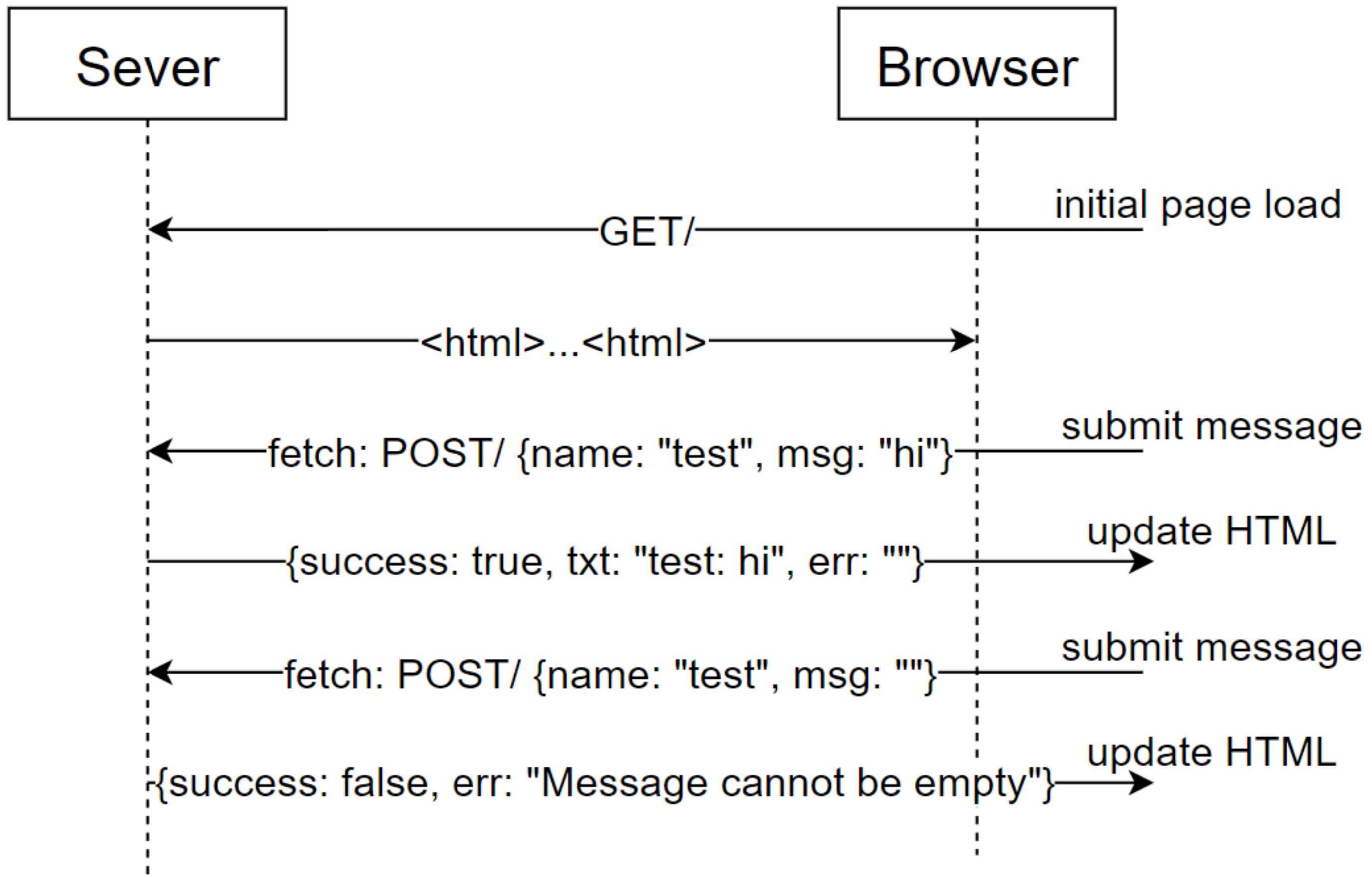
Definitely 4x1 makes the r

Test error handling

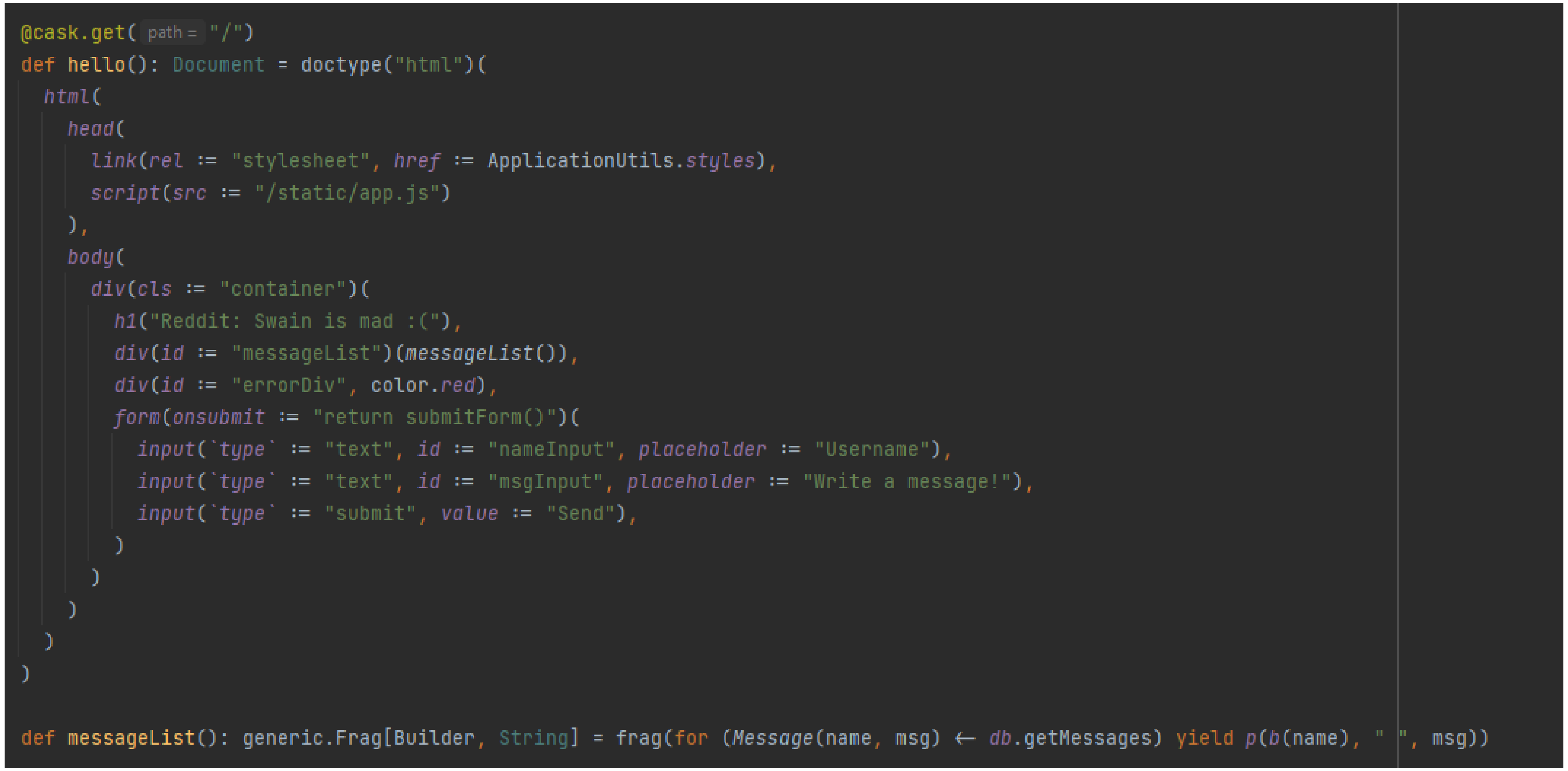


```
test("failure") - withServer(RedditApplication) { host =>
  val response1 = requests.post(host, data = Map("name" -> "ilya"), check = false)
  assert(response1.statusCode == 400)
  val response2 = requests.post(host, data = Map("name" -> "ilya", "msg" -> ""))
  assert(response2.text().contains("Message cannot be empty"))
  val response3 = requests.post(host, data = Map("name" -> "", "msg" -> "Test Message!"))
  assert(response3.text().contains("Name cannot be empty"))
  val response4 = requests.post(host, data = Map("name" -> "123#123", "msg" -> "Test Message!"))
  assert(response4.text().contains("Username cannot contain '#'"))
}
```

Updating page via API request

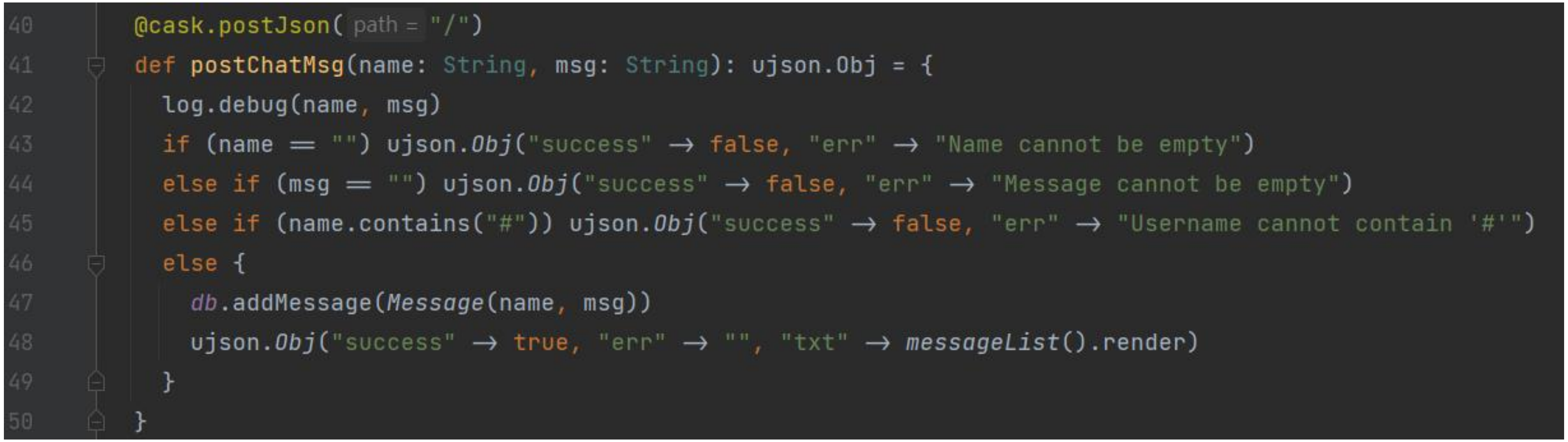

Sever
Browser
initial page load
GET/
<html>...<html>
submit message
fetch: POST/ {name: "test", msg: "hi"}
update HTML
{success: true, txt: "test: hi", err: ""}
submit message
fetch: POST/ {name: "test", msg: ""}
update HTML
{success: false, err: "Message cannot be empty"}

Change main page



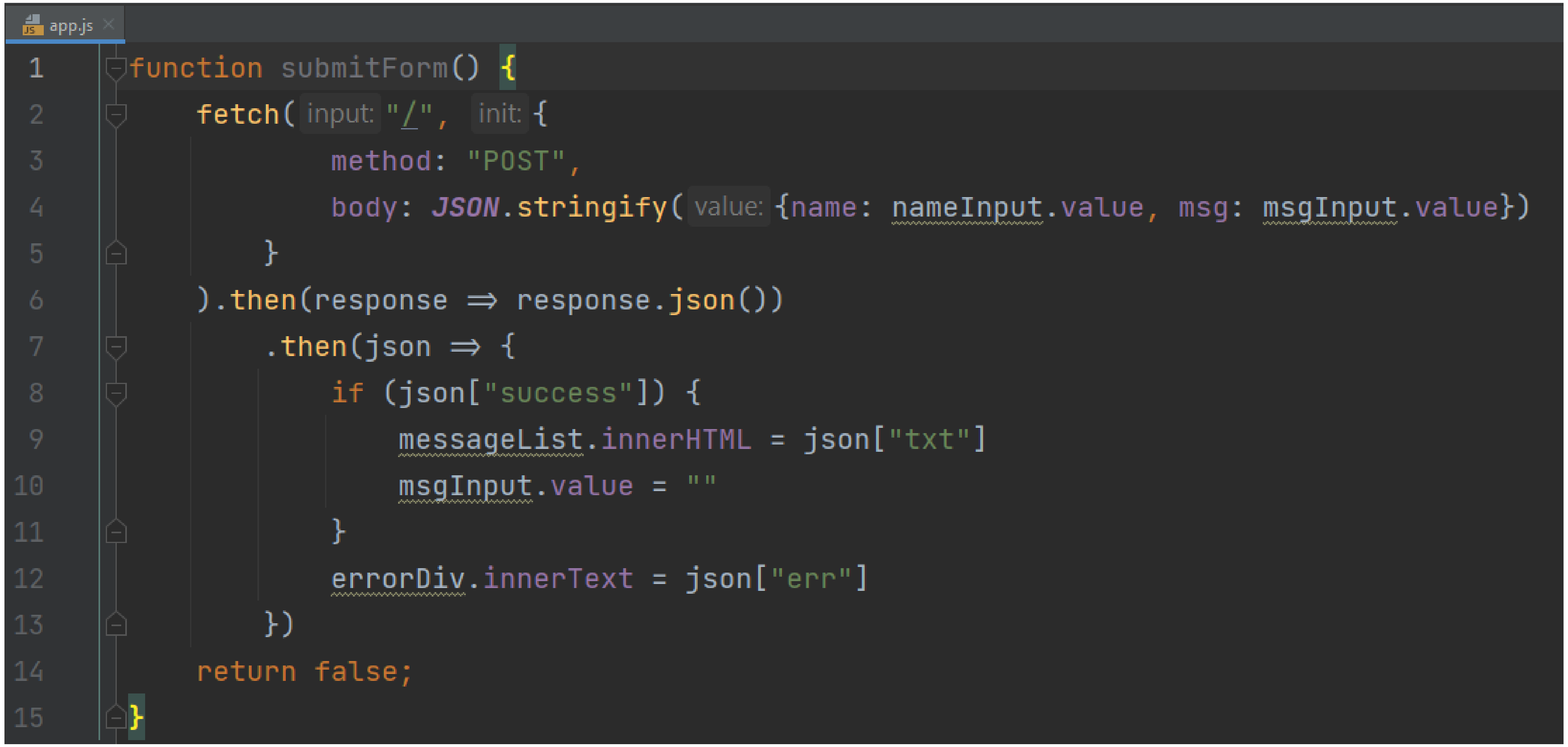
```
@cask.get( path = "/")
def hello(): Document = doctype("html")(
  html(
    head(
      link(rel := "stylesheet", href := ApplicationUtils.styles),
      script(src := "/static/app.js")
    ),
    body(
      div(cls := "container")(
        h1("Reddit: Swain is mad :("),
        div(id := "messageList")(messageList()),
        div(id := "errorDiv", color.red),
        form(onsubmit := "return submitForm()")(
          input(`type` := "text", id := "nameInput", placeholder := "Username"),
          input(`type` := "text", id := "msgInput", placeholder := "Write a message!"),
          input(`type` := "submit", value := "Send"),
        )
      )
    )
  )
)

def messageList(): generic.Frag[Builder, String] = frag(for (Message(name, msg) ← db.getMessages) yield p(b(name), " ", msg))
```

From handling to API handling



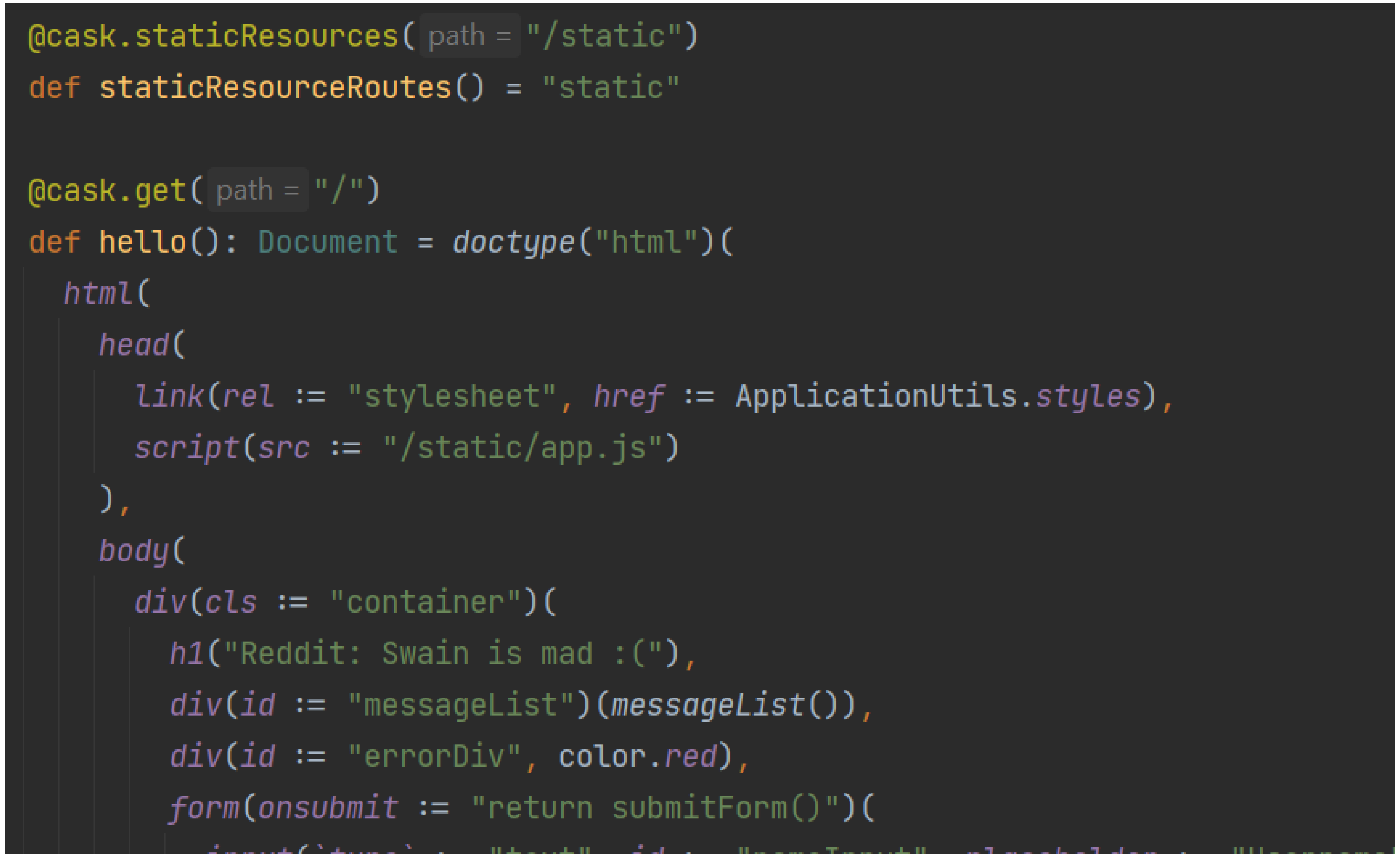
```scala
@cask.postJson( path = "/")
def postChatMsg(name: String, msg: String): ujson.Obj = {
  log.debug(name, msg)
  if (name == "") ujson.Obj("success" -> false, "err" -> "Name cannot be empty")
  else if (msg == "") ujson.Obj("success" -> false, "err" -> "Message cannot be empty")
  else if (name.contains("#")) ujson.Obj("success" -> false, "err" -> "Username cannot contain '#'")
  else {
    db.addMessage(Message(name, msg))
    ujson.Obj("success" -> true, "err" -> "", "txt" -> messageList().render)
  }
}
```

JS on Form submit



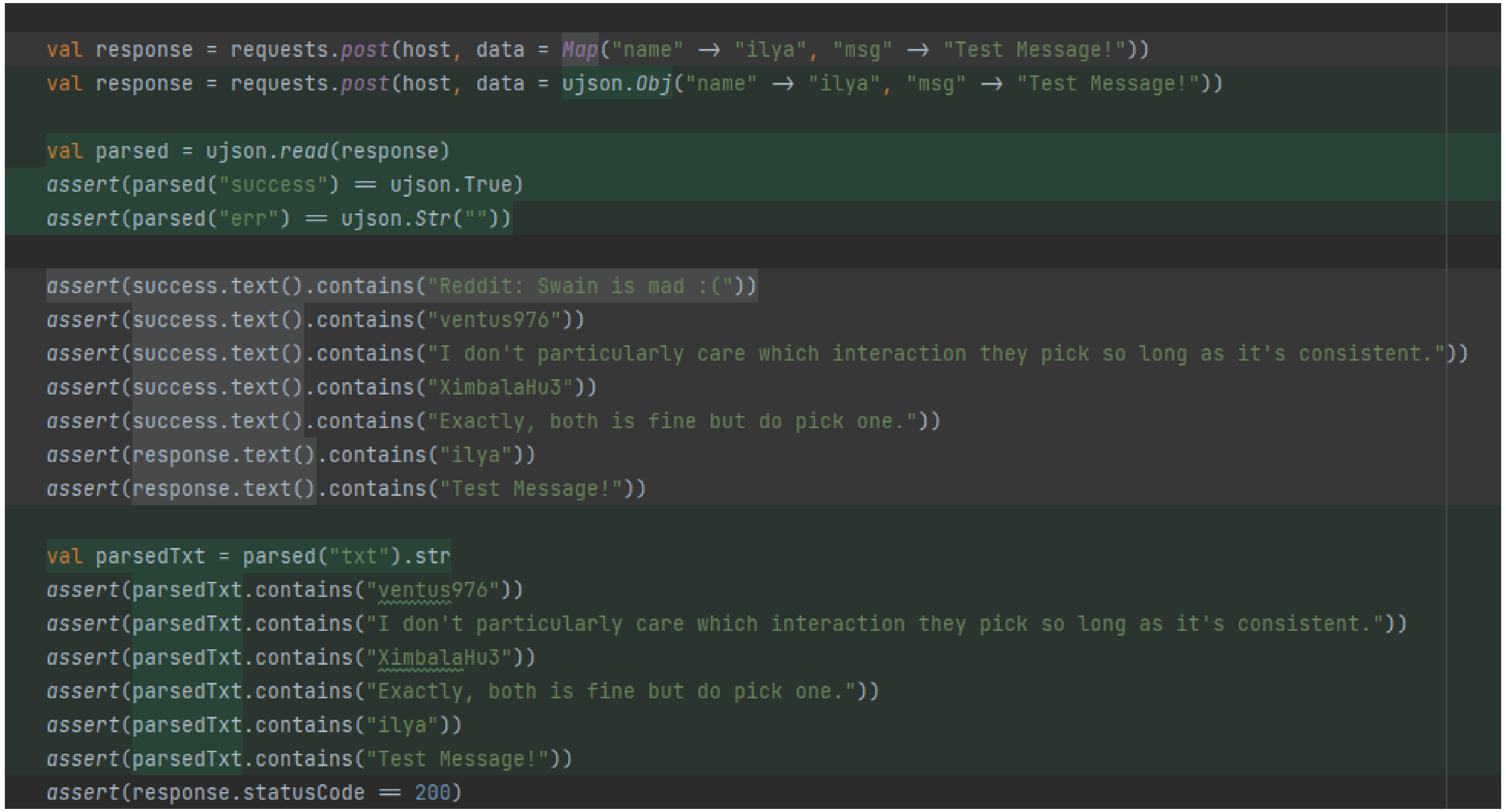
```
function submitForm() {
    fetch( input: "/",  init: {
            method: "POST",
            body: JSON.stringify( value: {name: nameInput.value, msg: msgInput.value})
        }
    ).then(response => response.json())
        .then(json => {
            if (json["success"]) {
                messageList.innerHTML = json["txt"]
                msgInput.value = ""
            }
            errorDiv.innerText = json["err"]
        })
    return false;
}
```

Static resources



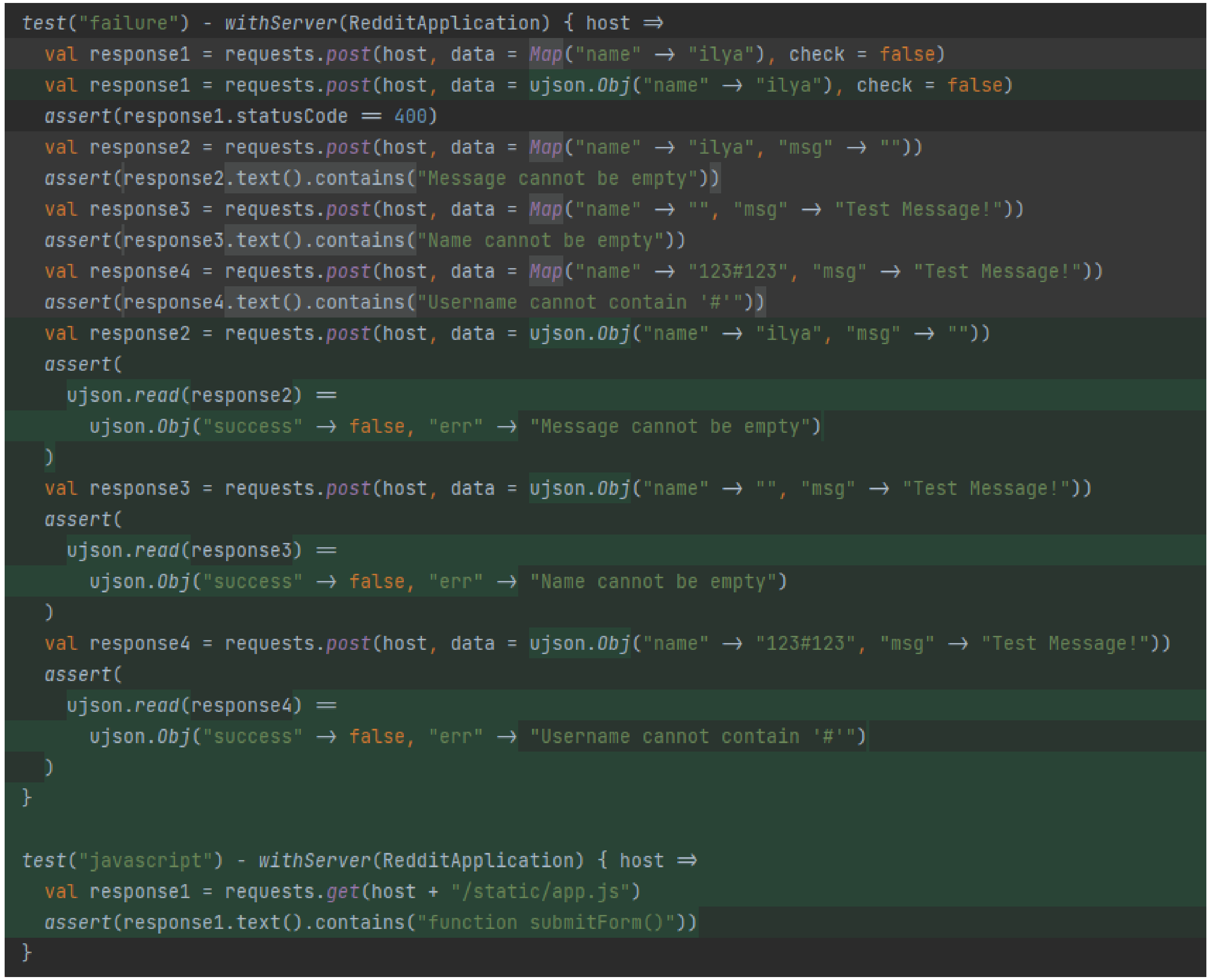
```
@cask.staticResources( path = "/static")
def staticResourceRoutes() = "static"

@cask.get( path = "/")
def hello(): Document = doctype("html")(
  html(
    head(
      link(rel := "stylesheet", href := ApplicationUtils.styles),
      script(src := "/static/app.js")
    ),
    body(
      div(cls := "container")(
        h1("Reddit: Swain is mad :("),
        div(id := "messageList")(messageList()),
        div(id := "errorDiv", color.red),
        form(onsubmit := "return submitForm()")(
```

Changing tests



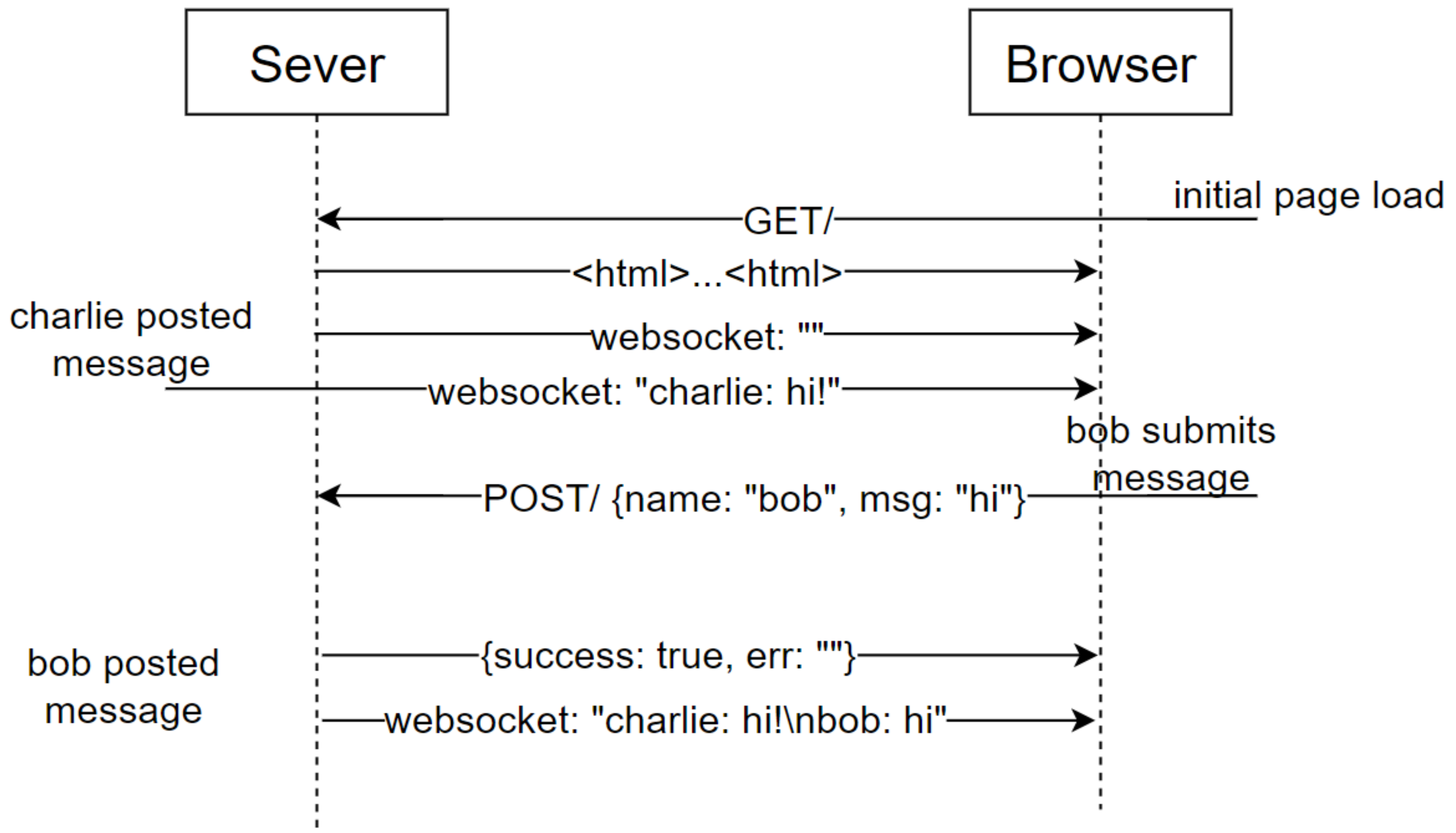
```
val response = requests.post(host, data = Map("name" -> "ilya", "msg" -> "Test Message!"))
val response = requests.post(host, data = ujson.Obj("name" -> "ilya", "msg" -> "Test Message!"))

val parsed = ujson.read(response)
assert(parsed("success") == ujson.True)
assert(parsed("err") == ujson.Str(""))

assert(success.text().contains("Reddit: Swain is mad :("))
assert(success.text().contains("ventus976"))
assert(success.text().contains("I don't particularly care which interaction they pick so long as it's consistent."))
assert(success.text().contains("XimbalaHu3"))
assert(success.text().contains("Exactly, both is fine but do pick one."))
assert(response.text().contains("ilya"))
assert(response.text().contains("Test Message!"))

val parsedTxt = parsed("txt").str
assert(parsedTxt.contains("ventus976"))
assert(parsedTxt.contains("I don't particularly care which interaction they pick so long as it's consistent."))
assert(parsedTxt.contains("XimbalaHu3"))
assert(parsedTxt.contains("Exactly, both is fine but do pick one."))
assert(parsedTxt.contains("ilya"))
assert(parsedTxt.contains("Test Message!"))
assert(response.statusCode == 200)
```

Changing tests

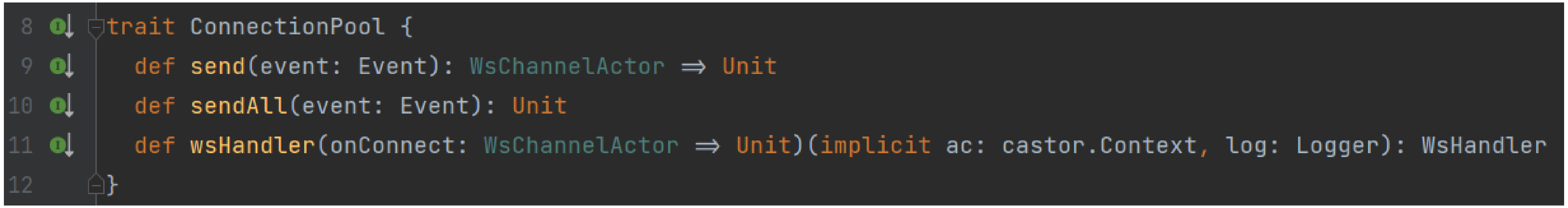


```scala
test("failure") - withServer(RedditApplication) { host =>
  val response1 = requests.post(host, data = Map("name" -> "ilya"), check = false)
  val response1 = requests.post(host, data = ujson.Obj("name" -> "ilya"), check = false)
  assert(response1.statusCode == 400)
  val response2 = requests.post(host, data = Map("name" -> "ilya", "msg" -> ""))
  assert(response2.text().contains("Message cannot be empty"))
  val response3 = requests.post(host, data = Map("name" -> "", "msg" -> "Test Message!"))
  assert(response3.text().contains("Name cannot be empty"))
  val response4 = requests.post(host, data = Map("name" -> "123#123", "msg" -> "Test Message!"))
  assert(response4.text().contains("Username cannot contain '#'"))
  val response2 = requests.post(host, data = ujson.Obj("name" -> "ilya", "msg" -> ""))
  assert(
    ujson.read(response2) ==
      ujson.Obj("success" -> false, "err" -> "Message cannot be empty")
  )
  val response3 = requests.post(host, data = ujson.Obj("name" -> "", "msg" -> "Test Message!"))
  assert(
    ujson.read(response3) ==
      ujson.Obj("success" -> false, "err" -> "Name cannot be empty")
  )
  val response4 = requests.post(host, data = ujson.Obj("name" -> "123#123", "msg" -> "Test Message!"))
  assert(
    ujson.read(response4) ==
      ujson.Obj("success" -> false, "err" -> "Username cannot contain '#'")
  )
}

test("javascript") - withServer(RedditApplication) { host =>
  val response1 = requests.get(host + "/static/app.js")
  assert(response1.text().contains("function submitForm()"))
}
```

Updating page via WebSocket

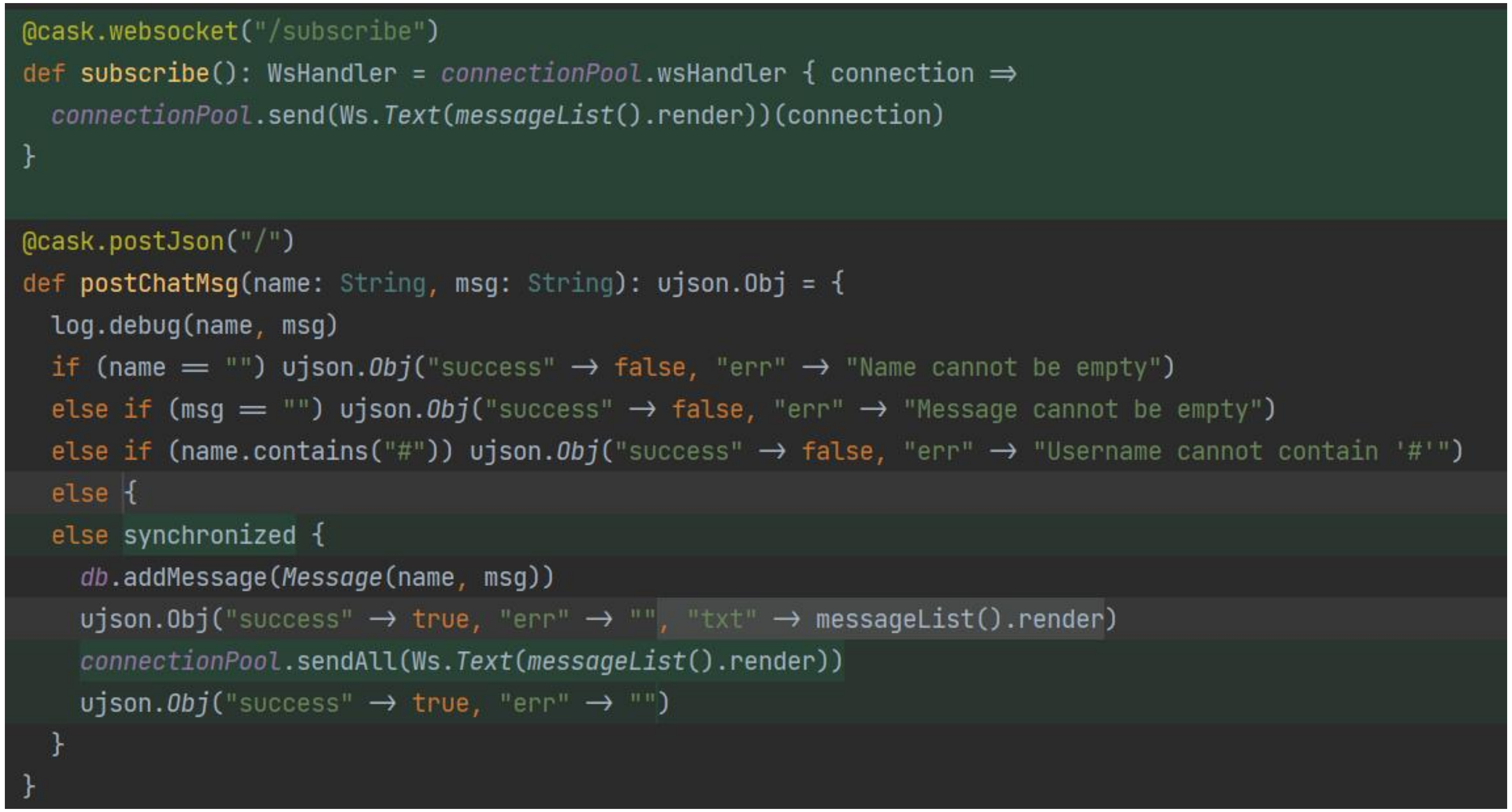


Ws Connection Pool



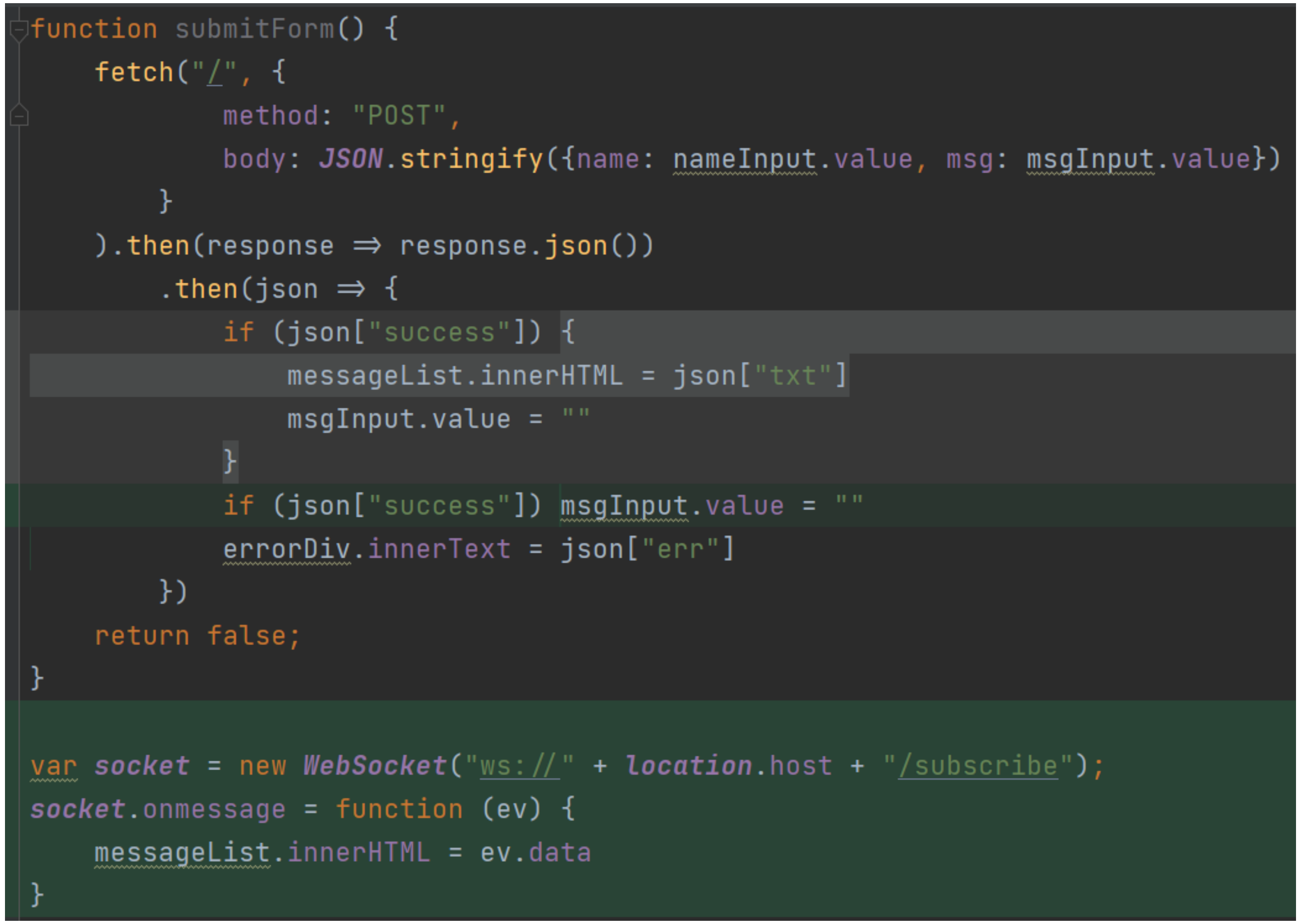
```scala
trait ConnectionPool {
  def send(event: Event): WsChannelActor ⇒ Unit
  def sendAll(event: Event): Unit
  def wsHandler(onConnect: WsChannelActor ⇒ Unit)(implicit ac: castor.Context, log: Logger): WsHandler
}
```

Using Connection Pool



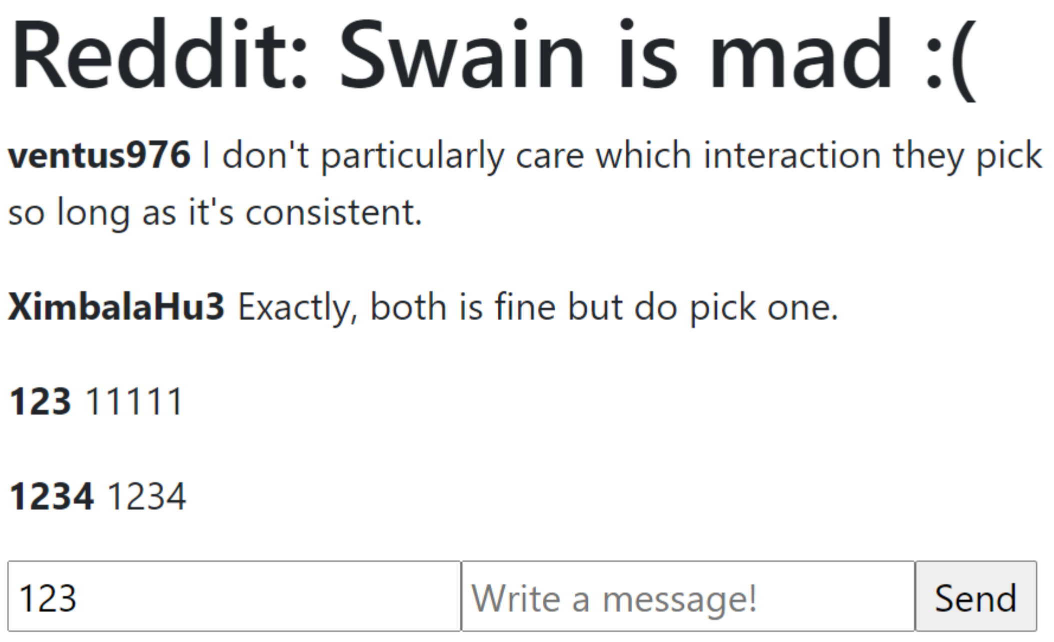
```
@cask.websocket("/subscribe")
def subscribe(): WsHandler = connectionPool.wsHandler { connection =>
  connectionPool.send(Ws.Text(messageList().render))(connection)
}

@cask.postJson("/")
def postChatMsg(name: String, msg: String): ujson.Obj = {
  log.debug(name, msg)
  if (name == "") ujson.Obj("success" -> false, "err" -> "Name cannot be empty")
  else if (msg == "") ujson.Obj("success" -> false, "err" -> "Message cannot be empty")
  else if (name.contains("#")) ujson.Obj("success" -> false, "err" -> "Username cannot contain '#'")
  else {
  else synchronized {
    db.addMessage(Message(name, msg))
    ujson.Obj("success" -> true, "err" -> "", "txt" -> messageList().render)
    connectionPool.sendAll(Ws.Text(messageList().render))
    ujson.Obj("success" -> true, "err" -> "")
  }
}
```

Using WebSocket



```
function submitForm() {
    fetch("/", {
            method: "POST",
            body: JSON.stringify({name: nameInput.value, msg: msgInput.value})
        }
    ).then(response => response.json())
        .then(json => {
            if (json["success"]) {
                messageList.innerHTML = json["txt"]
                msgInput.value = ""
            }
            if (json["success"]) msgInput.value = ""
            errorDiv.innerText = json["err"]
        })
    return false;
}

var socket = new WebSocket("ws://" + location.host + "/subscribe");
socket.onmessage = function (ev) {
    messageList.innerHTML = ev.data
}
```

Result



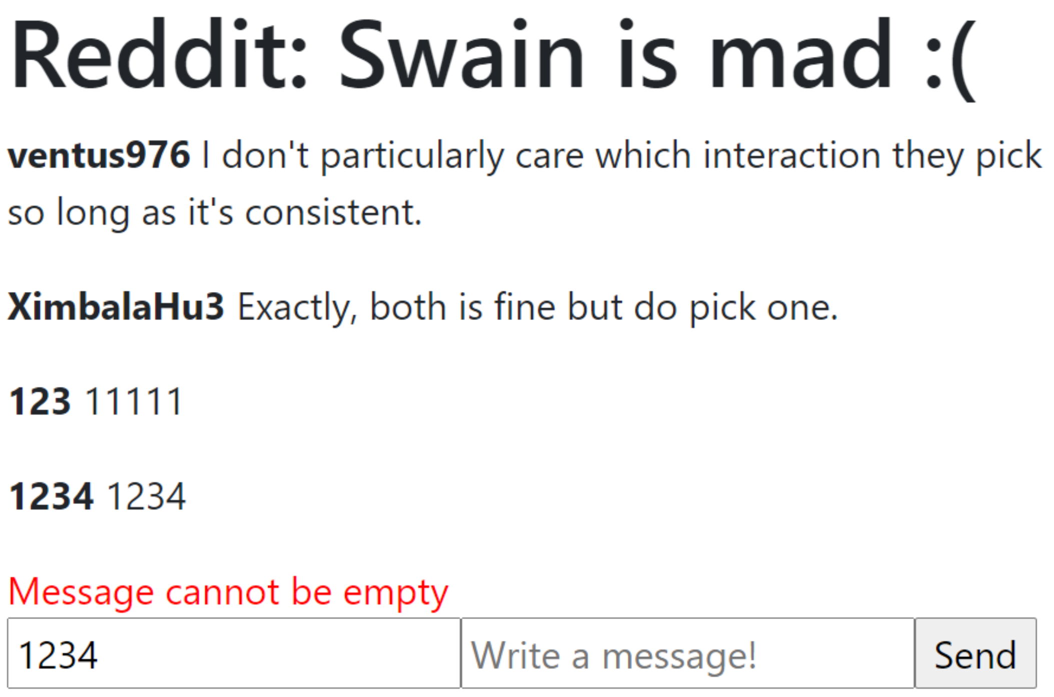
Reddit: Swain is mad :(

ventus976 I don't particularly care which interaction they pick so long as it's consistent.

XimbalaHu3 Exactly, both is fine but do pick one.

123 11111

1234 1234



Reddit: Swain is mad :(

ventus976 I don't particularly care which interaction they pick so long as it's consistent.

XimbalaHu3 Exactly, both is fine but do pick one.

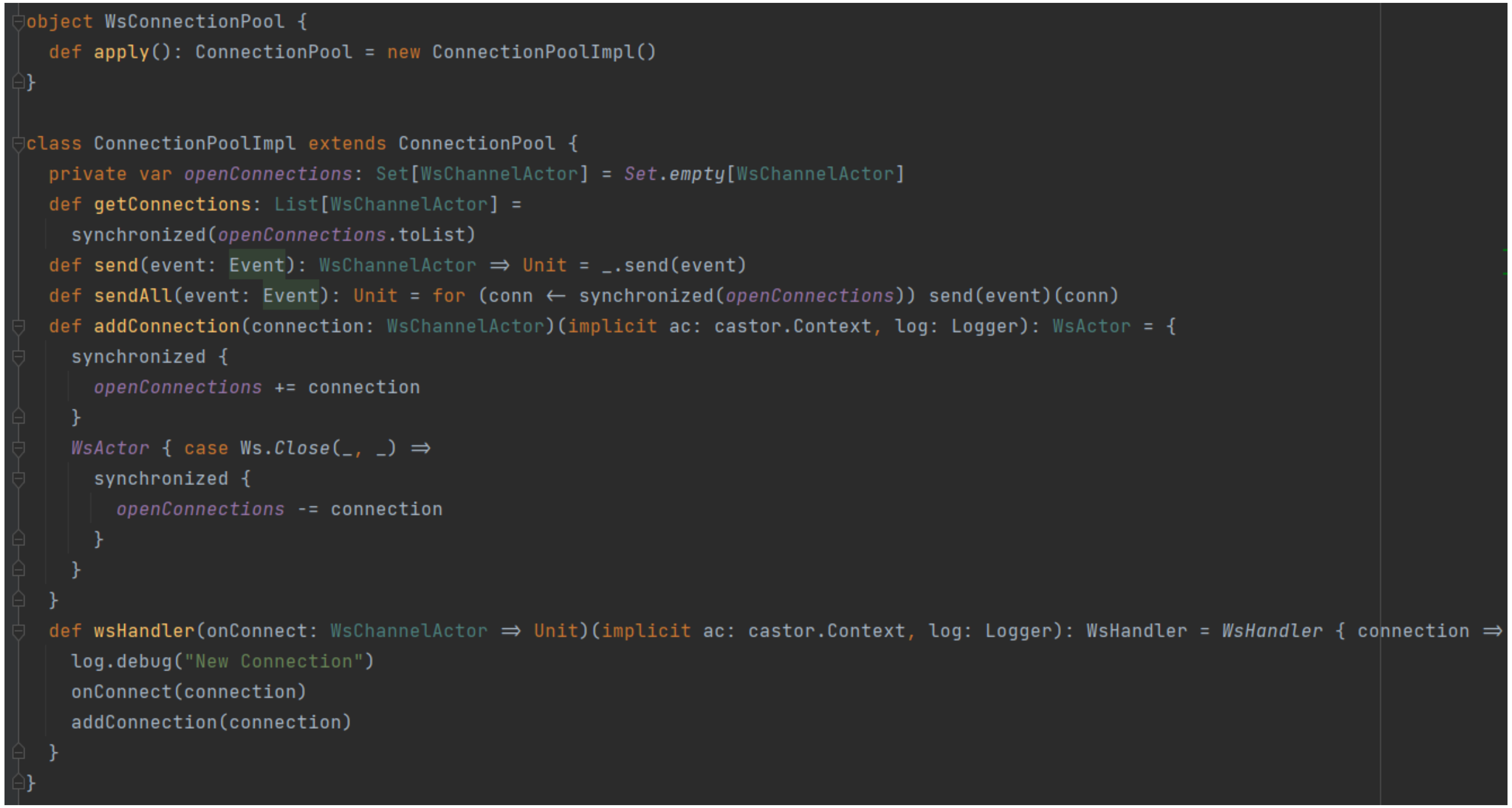
123 11111

1234 1234

Message cannot be empty

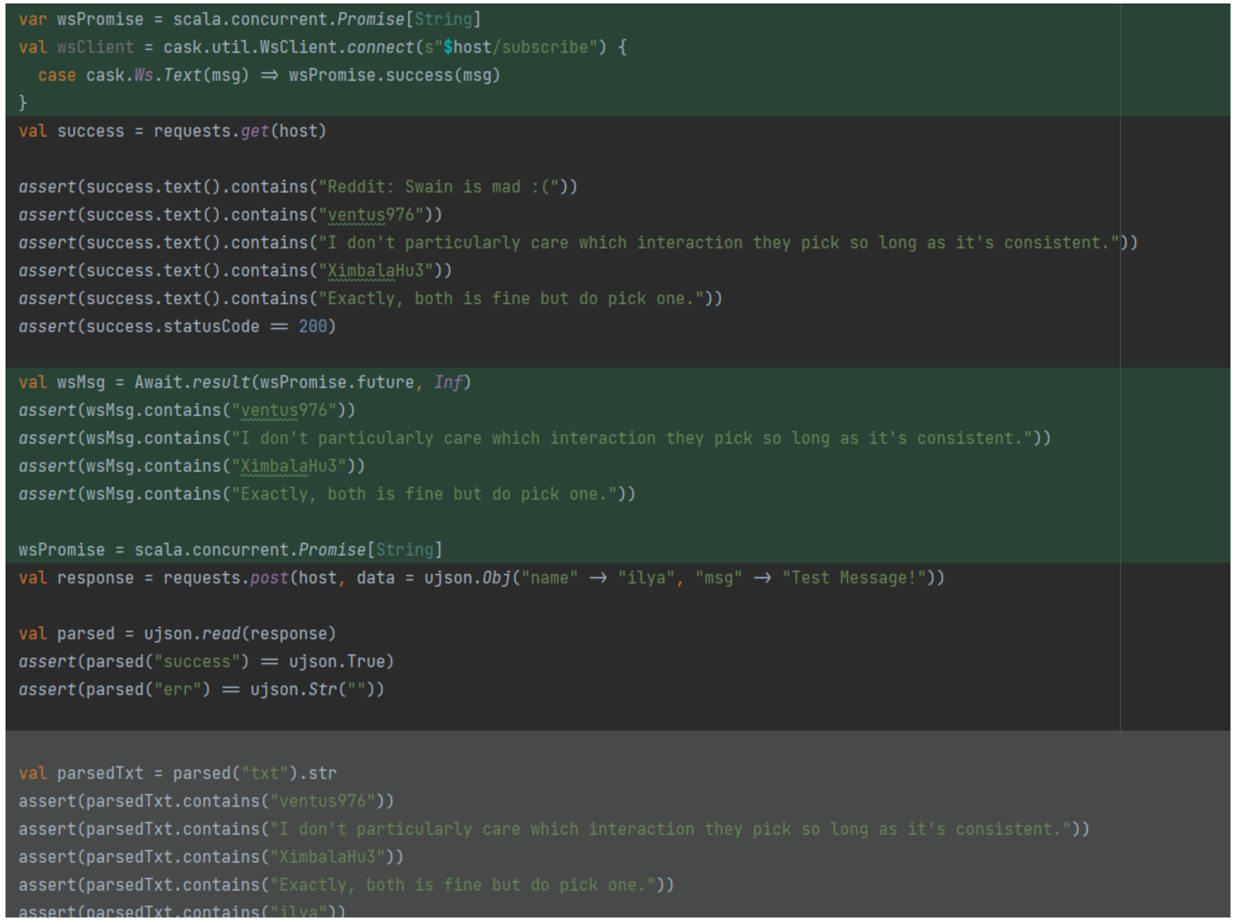
Concurrency

- syncronized
- Future
- Promise
- Actors



```scala
object WsConnectionPool {
  def apply(): ConnectionPool = new ConnectionPoolImpl()
}

class ConnectionPoolImpl extends ConnectionPool {
  private var openConnections: Set[WsChannelActor] = Set.empty[WsChannelActor]
  def getConnections: List[WsChannelActor] =
    synchronized(openConnections.toList)
  def send(event: Event): WsChannelActor ⇒ Unit = _.send(event)
  def sendAll(event: Event): Unit = for (conn ← synchronized(openConnections)) send(event)(conn)
  def addConnection(connection: WsChannelActor)(implicit ac: castor.Context, log: Logger): WsActor = {
    synchronized {
      openConnections += connection
    }
    WsActor { case Ws.Close(_, _) ⇒
      synchronized {
        openConnections -= connection
      }
    }
  }
  def wsHandler(onConnect: WsChannelActor ⇒ Unit)(implicit ac: castor.Context, log: Logger): WsHandler = WsHandler { connection ⇒
    log.debug("New Connection")
    onConnect(connection)
    addConnection(connection)
  }
}
```

Reddit tests



```scala
var wsPromise = scala.concurrent.Promise[String]
val wsClient = cask.util.WsClient.connect(s"$host/subscribe") {
  case cask.Ws.Text(msg) => wsPromise.success(msg)
}
val success = requests.get(host)

assert(success.text().contains("Reddit: Swain is mad :("))
assert(success.text().contains("ventus976"))
assert(success.text().contains("I don't particularly care which interaction they pick so long as it's consistent."))
assert(success.text().contains("XimbalaHu3"))
assert(success.text().contains("Exactly, both is fine but do pick one."))
assert(success.statusCode == 200)

val wsMsg = Await.result(wsPromise.future, Inf)
assert(wsMsg.contains("ventus976"))
assert(wsMsg.contains("I don't particularly care which interaction they pick so long as it's consistent."))
assert(wsMsg.contains("XimbalaHu3"))
assert(wsMsg.contains("Exactly, both is fine but do pick one."))

wsPromise = scala.concurrent.Promise[String]
val response = requests.post(host, data = ujson.Obj("name" -> "ilya", "msg" -> "Test Message!"))

val parsed = ujson.read(response)
assert(parsed("success") == ujson.True)
assert(parsed("err") == ujson.Str(""))


val parsedTxt = parsed("txt").str
assert(parsedTxt.contains("ventus976"))
assert(parsedTxt.contains("I don't particularly care which interaction they pick so long as it's consistent."))
assert(parsedTxt.contains("XimbalaHu3"))
assert(parsedTxt.contains("Exactly, both is fine but do pick one."))
assert(parsedTxt.contains("ilya"))
```

Useful links

- Cask example - https://github.com/Backend-ITMO-2021/cask-example
- ScalaTags - https://com-lihaoyi.github.io/scalatags/
- Cask - https://com-lihaoyi.github.io/cask/
- uTest - https://github.com/com-lihaoyi/utest#getting-started
- Play Framework - https://www.playframework.com/
- Akka HTTP - https://doc.akka.io/docs/akka-http/current/introduction.html
- HTTP4S - https://http4s.org/v1.0/service/