Express Typing

BuckleScript bindings for the express framework

https://github.com/BuckleTypes/bs-express

Maxime Ransan

ExpressJs

- Web Application Framework in Node.js
- Lightweight and highly customizable
- Has a large ecosystem of plugins called Middleware which can be chained together when processing a request

```
var express = require('express')
var app = express()

var myLogger = function (req, res, next) {
  console.log('LOGGED')
  next()
}

app.use(myLogger)

app.get('/', function (req, res) {
  res.send('Hello World!')
})

app.listen(3000)
```

Source: https://expressjs.com/en/guide/writing-middleware.html

Why starting the project

- I wanted to learn about JavaScript ecosystem and produce something useful along the way
- Disclaimer I don’t use Express in production or in any of my project but I do like to test my software ☺
- ExpressJs relies on dynamic feature of JavaScript which is always an interesting challenge for bindings in general

The ’Hanging’ problem

“If the current middleware function does not end the request-response cycle, it must call next() to pass control to the next middleware function. Otherwise, the request will be left hanging.”

Source: https://expressjs.com/en/guide/writing-middleware.html

```
var express = require('express')
var app = express()

app.get('/', function (req, res, next) {
  // Either next()
  // or
  // res.send(...)
})

app.listen(3000)
```

Can we use OCaml type system to enforce this invariant ?

The ’Hanging’ problem

- Abstract type
  - Simply declare a type in your binding without any any definition
  - Use the type declaration in external declaration

```
type done_;

module Next : {
  type content;
  type t = (Js.undefined content => done_);
};

module Response = {
 type t;
 external sendString : t => string => done_ = "send" [@@bs.send];
};

Module App = {
   type t;
  external get : t => path::string => (Request.t => Response.t => Next.t => done_ ) => unit = "" [@@bs.send];
};
```

Compiler will enforce that either next is called or a response is sent!

The ’Overload’ problem

next(err) will skip all remaining handlers in the chain except for those that are set up to handle errors as described above.

Source: https://expressjs.com/en/guide/error-handling.html

call next('route') to pass control to the next route.

https://expressjs.com/en/guide/using-middleware.html

next() // pass control to the next handler

https://expressjs.com/en/guide/routing.html

The ‘Overload’ problem

- OCaml does not support overloading
- Solution 1: **multiple** external declaration

```
external f: unit -> unit  = ”” [@@bs.val]
external f2: string -> unit = “f” [@@bs.val]
```

  - Combinatorial explosion with multiple function parameters
- Solution 2: **abstract type** and identity functions

```
module Next = {
  type content;
  type t = (Js.undefined content => done_);
  let middleware = Js.undefined;
  external castToContent : 'a => content = "%identity";
  let route = Js.Undefined.return (castToContent "route");
  let error (e:Error.t) => Js.Undefined.return (castToContent e);
};
```

The ’Overload’ problem

- Problem with solution 2:

```
let array array_of_content => Js.Undefined.return (castToContent e);
```

- You can now create arrays of arrays of content which is actually not supported by the JS function

Questions ?