Shakras and Hot Stone Therapy (SHST)

Modern techniques for relaxation using ancient methods

(a hands-on approach)

Server Architectures with Node

Why should the browser folk have all the fun?

Your friendly presenter

Jeffrey Labonski
jlabonski@chariotsolutions.com
Consultant, Chariot Solutions
Somewhat Recent Javascript Convert

Javascript Convert?

Javascript Convert?

Actually, yes

However, there's beauty in there

However, there's beauty in there

This may take a bit of time to find.

Efficient use of resources makes me happy.

Efficient use of resources makes me happy.

(hapi?)

This thing is *thin*

Customer production server from pm2:

```
$ pm2 status

│ SERVER (PROD) │ online │ 12D     │ 0%   │ 148.7 MB │
```

Customer production server from pm2:

```
$ pm2 status

│ SERVER (PROD) │ online │ 12D      │ 0%  │ 148.7 MB │
```

12 days of uptime

Customer production server from pm2:

```
$ pm2 status

│ SERVER (PROD) │ online │ 12D      │ 0%   │ 148.7 MB │
```

12 days of uptime

150MB of RAM!

```
java -Xmx4G -Xms4G -jar app.jar
```

```
 :: Spring Boot ::        (v2.0.3.RELEASE)

Started TestApp in 24.351 seconds (JVM running for 24.811)
```

Single threaded

Code isomorphism between client and server

If you use JS on the client...

Fun to write, to be honest

JS is a gateway drug to Lambda

Node architecture makes me happy

Node architecture makes me happy

(hapi?)

Node was born from two neat pieces of technology

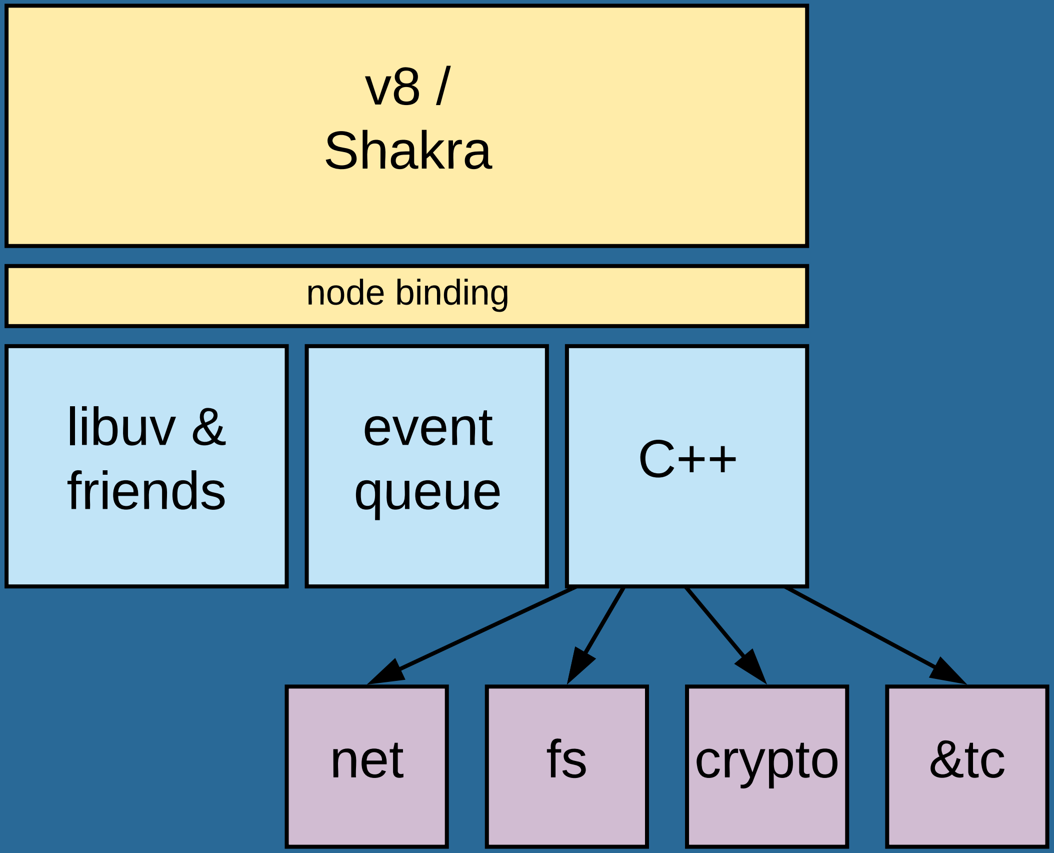
Node was born from two neat pieces of technology

`libuv`

Node was born from two neat pieces of technology

`libuv`

`v8`


v8 /
Shakra
node binding
libuv &
friends
event
queue
C++
net
fs
crypto
&tc

v8 & shakra

libuv

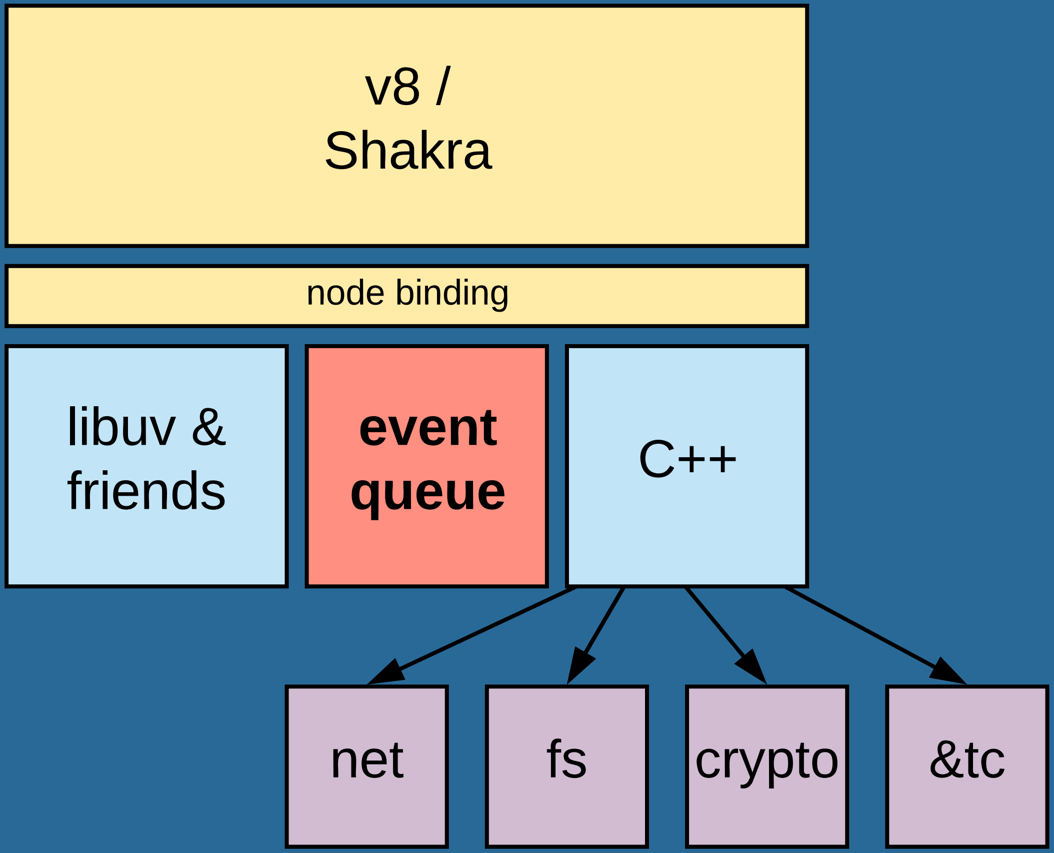
Only a matter of time

What, exactly, is it good for?

IO

IO

Like, *tons* of IO


v8 /
Shakra
node binding
libuv &
friends
**event
queue**
C++
net
fs
crypto
&tc

Try not to do too much

All CPU use on node is crimethink

Get in, get out

A humorous vingette...

`bcrypt(3)`*!!!!!*

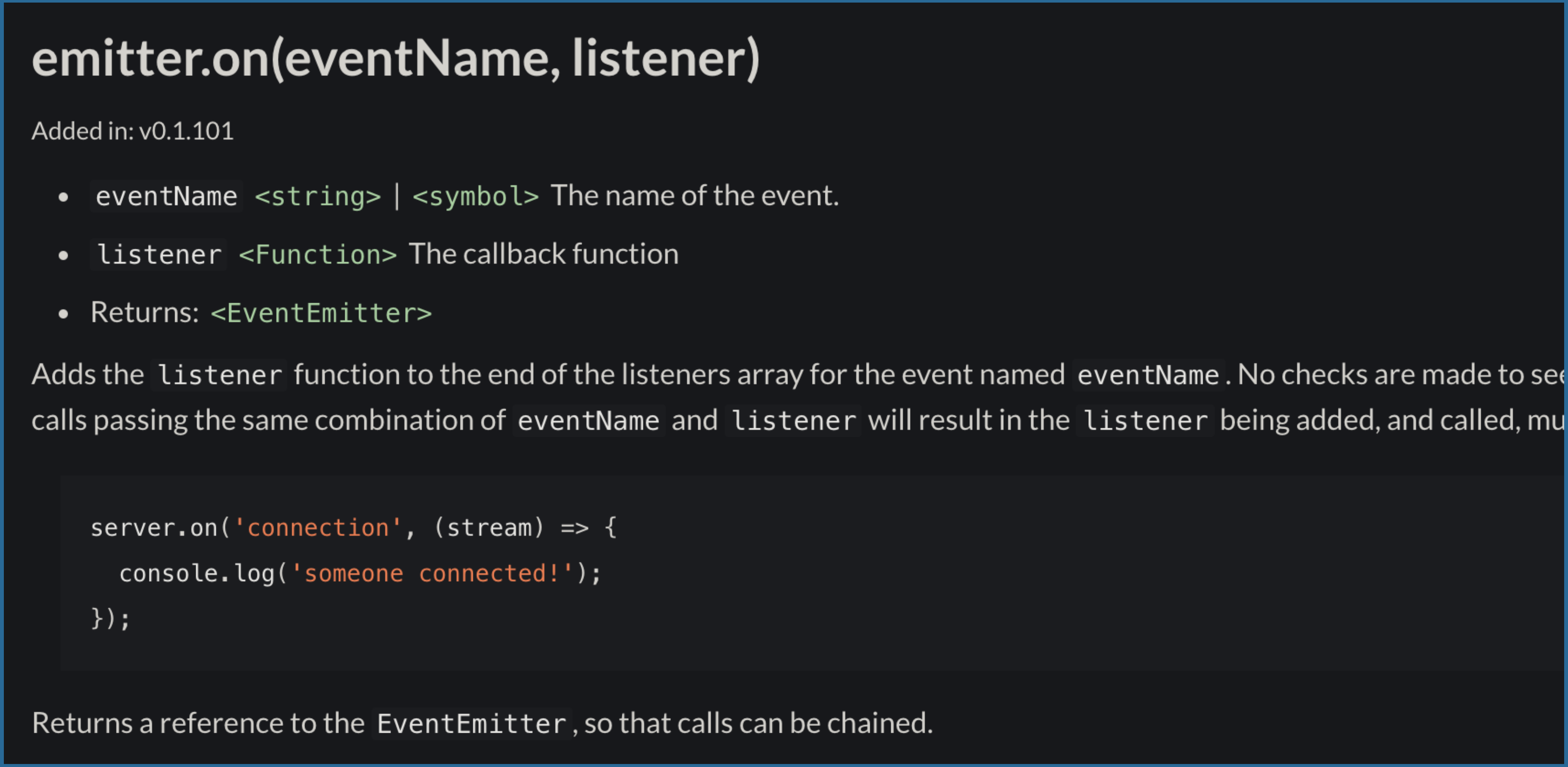
WARNING:

Handrails are not available

There really is only one real method in node that matters.

There really is only one real method in node that matters.

`on()`



emitter.on(eventName, listener)

Added in: v0.1.101

- `eventName` `<string>` | `<symbol>` The name of the event.
- `listener` `<Function>` The callback function
- Returns: `<EventEmitter>`

Adds the `listener` function to the end of the listeners array for the event named `eventName`. No checks are made to see
calls passing the same combination of `eventName` and `listener` will result in the `listener` being added, and called, mu

```
server.on('connection', (stream) => {
  console.log('someone connected!');
});
```

Returns a reference to the `EventEmitter`, so that calls can be chained.

Looks familiar?

```
server.on('request', (req, res) => {
    // node HTTP server
})

observer.subscribe((x) => {
    // Angular HttpClient
})
```

The fundamental unit of javascript is the *continuation*.

Javascript is a *continuation oriented* language.

```
whenItIsDone(thing, (result) => {
    doSomethingElse(result)
})
```

```
const fs = require('fs')

fs.readFile('./foo.txt', 'utf8', (err, contents) => {
    console.log(contents)
});
```

The *continuation* is a lambda

The *continuation* is a lambda

It executes later.

The *continuation* is a lambda

It executes later.

You can never get here from there.

```
                                            (err, contents) => {
    console.log(contents)
});
```

```
fs.readFile('./foo.txt', 'utf8', (err, contents) => {
    console.log(contents)
});

fs.readFile('./bar.txt', 'utf8', (err, contents) => {
    console.log(contents)
});
```

This is the beauty of nodejs

Handle your result

This is the beauty of nodejs

Handle your result *later*

Man, it can get ugly fast

```
const fs = require('fs')
const dns = require('net')

fs.readFile('./foo.txt', 'utf8', (err, contents) => {
    fs.writeFile('./bar.txt', 'utf8', (err, contents) => {
        dns.lookup(contents, (err, address, family) => {
            console.log(address)
        })
    })
})
```

```
do(foo, bar => {
    some(bar, baz => {
        thing(baz, quux => {
            complex(quux, quuz => {
                eventually(quuz, waldo => {
                    go(waldo, fred => {
                        crazy(fred, thud => {
                            console.log(thud)
                        })
                    })
                })
            })
        })
    })
})
```

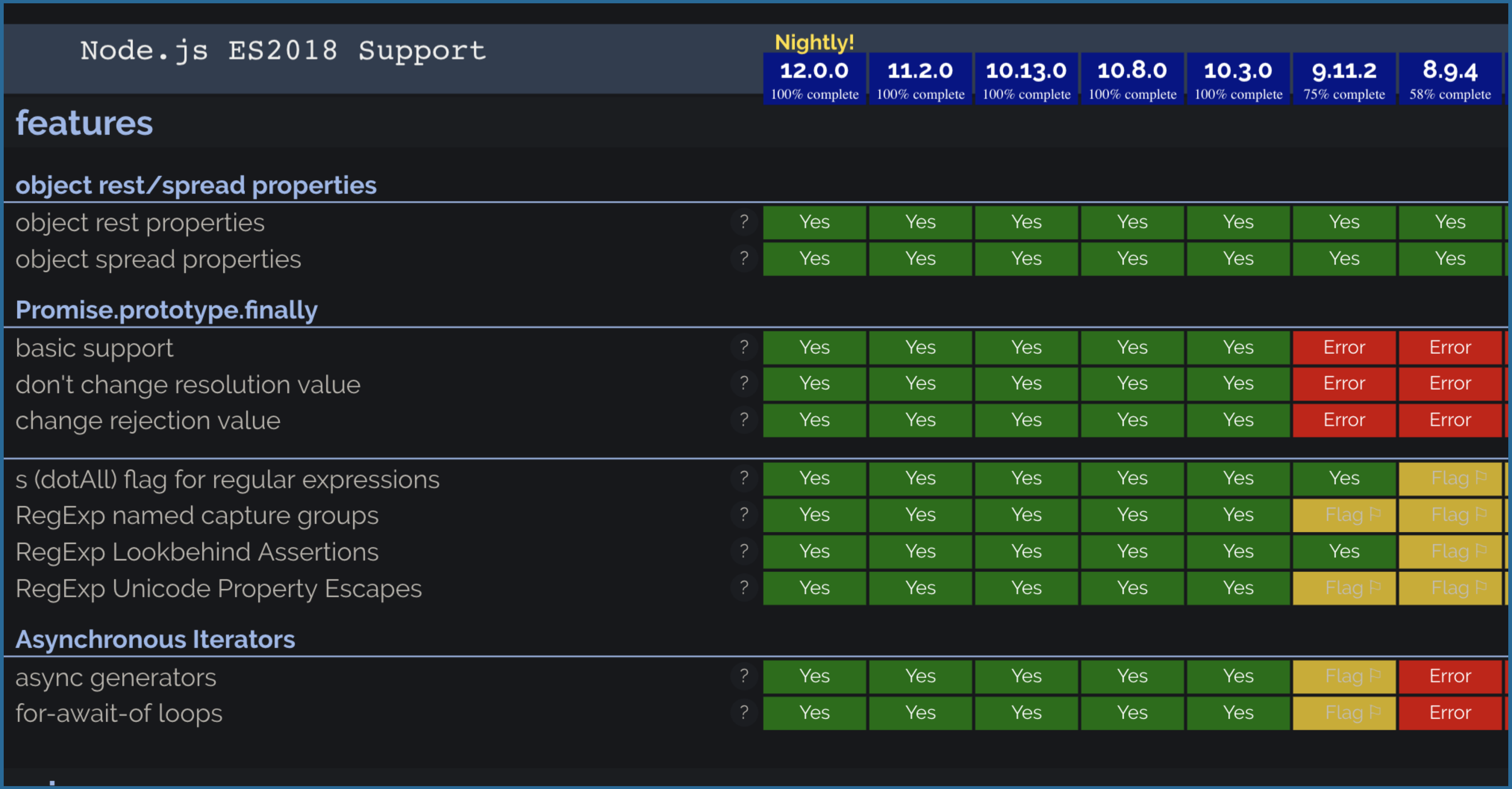
Promises are syntactic sugar

```
do(foo)
    .then(bar => {
        some(bar)
    }).then(baz => {
        thing(baz)
    }).then(quux => {
        complex(quux)
    }).then(quuz => {
        eventually(quuz)
    }).then(waldo => {
        go(waldo)
    }).then(fred => {
        crazy(fred)
    }).then(thud => {
        console.log(thud)
```

`async` / `await` are the new hotness

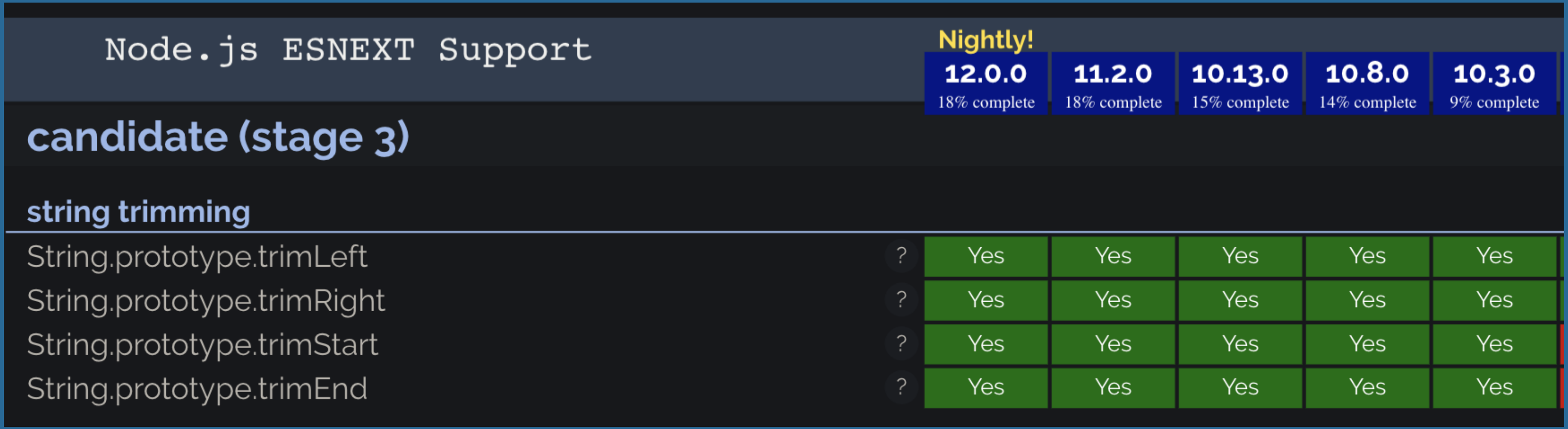
```
try{
    const bar   = await do(foo)
    const baz   = await some(bar)
    const quux  = await thing(baz)
    const quuz  = await complex(quux)
    const waldo = await eventually(quuz)
    const fred  = await go(waldo)
    const thud  = await crazy(fred)
    console.log(thud)
}catch(err){
    // whoops
}
```

```
const p = do(foo);

const bar = await p;
const util = require('util')

const readFileAsync = util.promisify(fs.readFile)

const contents = await readFileAsync('./foo.txt', 'utf8')
```



Node.js ES2018 Support

features	Nightly! 12.0.0 100% complete	11.2.0 100% complete	10.13.0 100% complete	10.8.0 100% complete	10.3.0 100% complete	9.11.2 75% complete	8.9.4 58% complete
object rest/spread properties							
object rest properties ?	Yes	Yes	Yes	Yes	Yes	Yes	Yes
object spread properties ?	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Promise.prototype.finally							
basic support ?	Yes	Yes	Yes	Yes	Yes	Error	Error
don't change resolution value ?	Yes	Yes	Yes	Yes	Yes	Error	Error
change rejection value ?	Yes	Yes	Yes	Yes	Yes	Error	Error
s (dotAll) flag for regular expressions ?	Yes	Yes	Yes	Yes	Yes	Yes	Flag ⚐
RegExp named capture groups ?	Yes	Yes	Yes	Yes	Yes	Flag ⚐	Flag ⚐
RegExp Lookbehind Assertions ?	Yes	Yes	Yes	Yes	Yes	Yes	Flag ⚐
RegExp Unicode Property Escapes ?	Yes	Yes	Yes	Yes	Yes	Flag ⚐	Flag ⚐
Asynchronous Iterators							
async generators ?	Yes	Yes	Yes	Yes	Yes	Flag ⚐	Error
for-await-of loops ?	Yes	Yes	Yes	Yes	Yes	Flag ⚐	Error



Node.js ESNEXT Support

candidate (stage 3)

string trimming		Nightly! 12.0.0 18% complete	11.2.0 18% complete	10.13.0 15% complete	10.8.0 14% complete	10.3.0 9% complete
String.prototype.trimLeft	?	Yes	Yes	Yes	Yes	Yes
String.prototype.trimRight	?	Yes	Yes	Yes	Yes	Yes
String.prototype.trimStart	?	Yes	Yes	Yes	Yes	Yes
String.prototype.trimEnd	?	Yes	Yes	Yes	Yes	Yes

```
const f = (a, b) => {
    return a + b
}

const p = f('a', ?)

assert(p('b') === 'ab')
```

Let's do some actual *work*.

REST servers

- Express
- HAPI
- Restify
- Loopback
- Sails
- ... probably a baker's dozen more by now

Express.js is the gold standard

Does whatcha need

```
const express = require('express')
const app = express()
const port = 80

app.get('/', (req, res) => res.send('Hello World!'))

app.listen(port, () => console.log(`Example app listening on po
```

C'mon get HAPI

git@bitbucket.org:chariotspaday/server.git

Things to like

In this desolate wasteland of `~0.0.9`

Vibrant plugins, easy API to write your own

- good
- boom
- nes
- confidence
- poop
- bell
- blipp

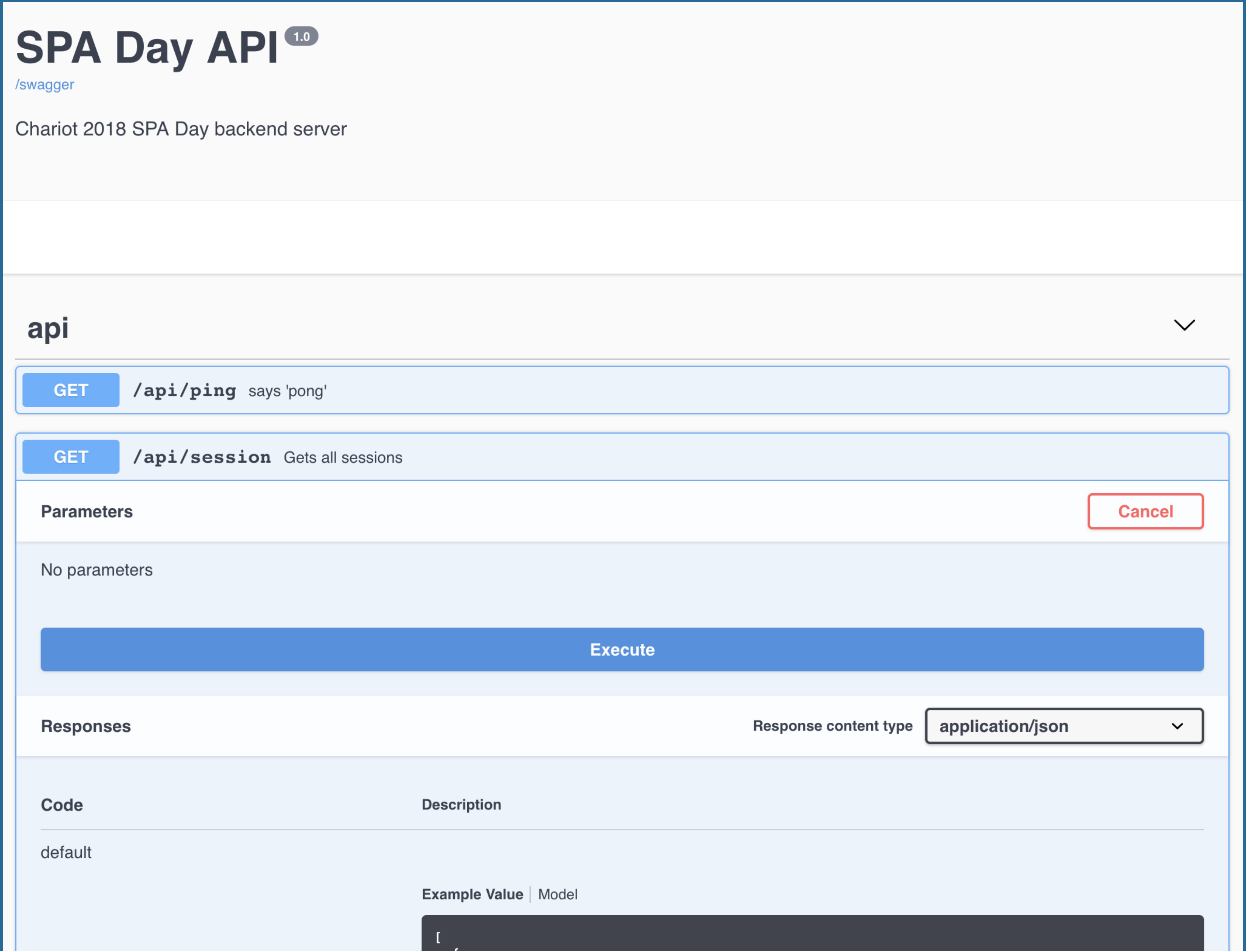
```
server.route({
    method: 'GET',
    path: '/',
    handler: function (request, h) {
        return 'Hello!'
    }
})
```

hapi ❤️ joi

```
const session = joi.object().keys({
    id: joi.number().integer().min(1).required(),
    name: joi.string().trim(true).min(1).required(),
    date: joi.string().trim(true).isoDate().required()
})
```

```
method: 'GET',
options: {
    tags: ['api'],
    description: 'Gets all subscriptions for a session',
    path: '/session/{id}/subscriptions',
    validate: {
        params: {
            id: joi.number().integer().min(1)
        }
    },
response: {
        options: {
            abortEarly: false
        },
        sample: 100,
        schema: joi.array().items(joiSchemas.registration)
    }
```

```
"/api/session": {
    "get": {
        "responses": {
            "default": {
                "description": "",
                "schema": {
                    "type": "array",
                    "items": {
                        "$ref": "#/definitions/IdNameDateModel"
                    }
                }
            }
        },
        "produces": ["application/json"],
        "tags": ["api"],
```



SPA Day API 1.0

/swagger

Chariot 2018 SPA Day backend server

api

GET /api/ping says 'pong'

GET /api/session Gets all sessions

Parameters

Cancel

No parameters

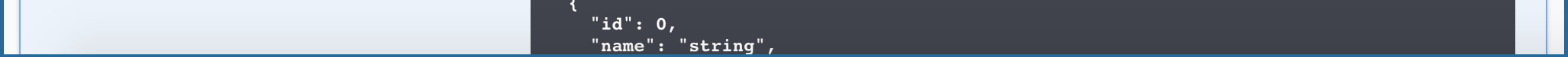
Execute

Responses

Response content type application/json

Code	Description
default	**Example Value** | Model

```
[
```



```
  {
    "id": 0,
    "name": "string",
```

Stupid node tricks with hapi, express, and websockets

```
const hapi = require('hapi')
const ws = require('ws')

const server = new hapi.Server({port: 8000})
const listener = server.listener

const wss = new WebSocket.Server({
    server: listener,
    path: '/api/chat'
})

await server.start()
```

```
this._removeListeners = addListeners(this._server, {
    listening: this.emit.bind(this, 'listening'),
    error: this.emit.bind(this, 'error'),
    upgrade: (req, socket, head) => {
        this.handleUpgrade(req, socket, head, (ws) => {
        this.emit('connection', ws, req);
        });
    }
});
```

```
server.events.on('log', (event, tags) => {
server.events.on('request', (request, event, tags) => {
server.events.on('response', (request) => {
```

Takeaway (whew!)

- Understand on()
- Grok continuations, how code moves through node
- Never eat CPU.
- Use cool language features
- Be Javascripty on the server