HI, I'M CHRIS DICKINSON
REGISTRY HUMAN AT NPM, INC.
TWITTER: @ISNTITVACANT
GITHUB: CHRISDICKINSON

hi, I'm chris.
thank you for having me!
I work on the npm registry remotely
from sunny Portland.
Well, Portland.

WHY NPM BUILT A FRAMEWORK

today I'm here to talk to you about "why npm built a framework"
^ did you know...

THE NPM REGISTRY SERVES

11MM USERS

we serve 11MM people.
something like 20-30K requests per second.
(less now that npm5 is a thing. you're using npm5, right?)
but did you know!

13 ENGINEERS SERVE

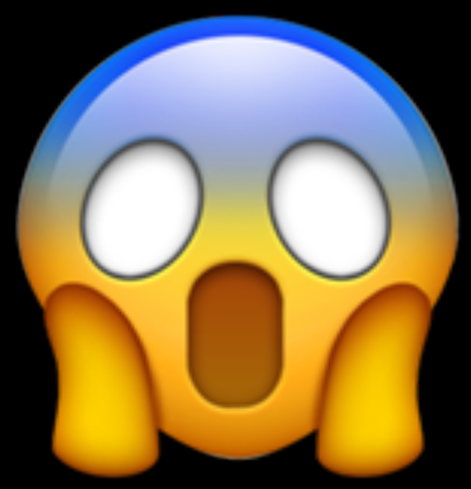
11MM USERS

there's only thirteen people at npm who work on services.
and...

5 SUPPORT TECHS SERVE

11MM USERS

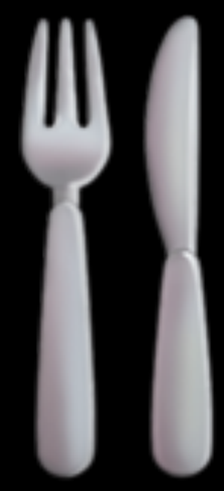
five support techs.



that's a pretty tough ratio to navigate.

HOW

so, how do we do it?



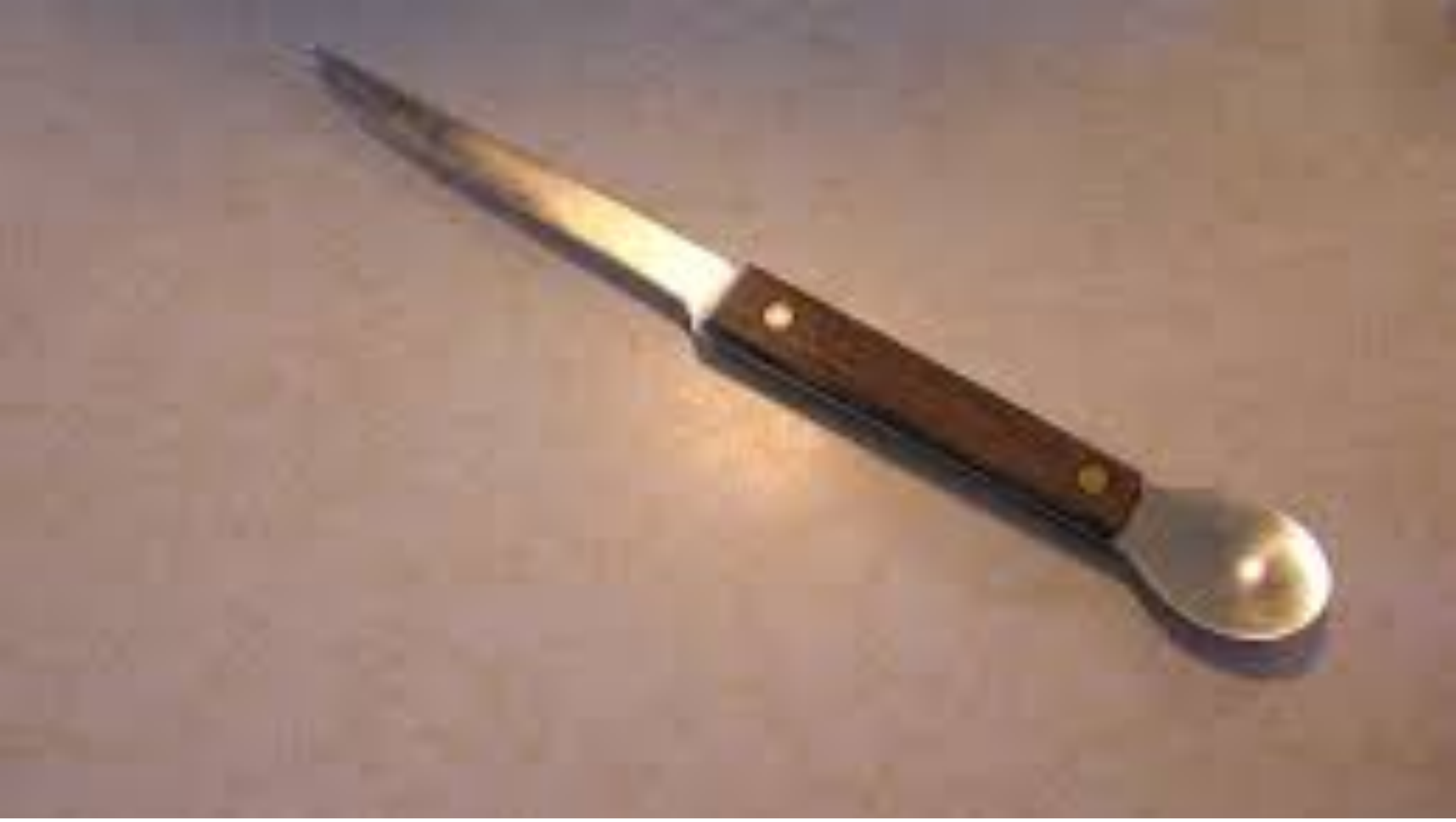
we cut the problem up into services that talk REST-ful HTTP
now, as you know, there's a lot of room for interpretation there. and
a lot of this interpretation is to the side of the problem at hand.

SPIFE

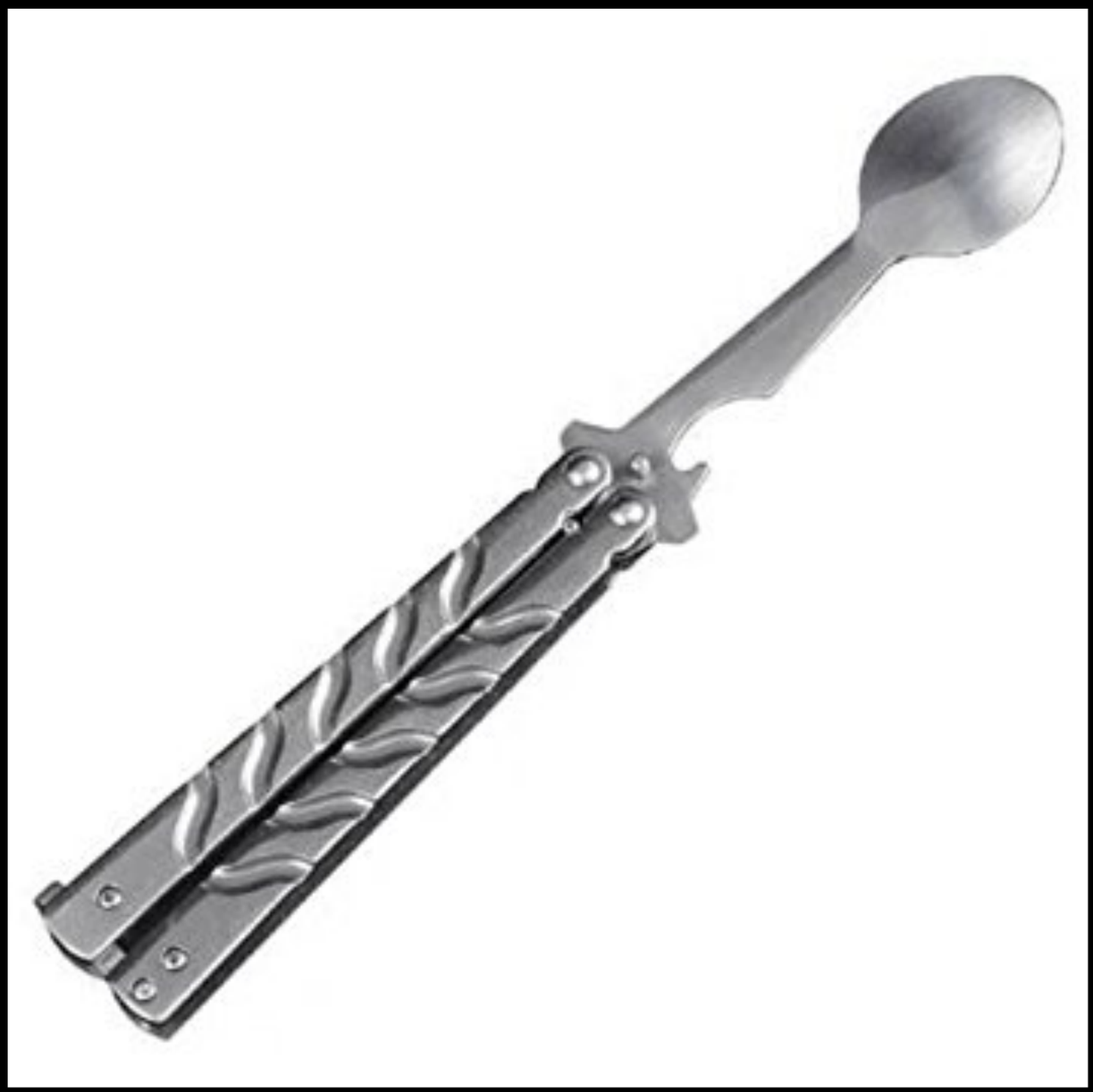
🍴 COMBINATION KNIFE + SPOON 🍴

(SOMETHING NO ONE ASKED FOR)

we built a framework.
it's called spife! not a knife, not a spoon,
but *both*. self-deprecatingly
named after something *nobody asked for*.



for example: in the best case you get soup on your hands,
in the worst, well
it doesn't bear mentioning, especially if you like
gesticulating wildly like I do.



we want something more like this. A switchblade spoon. A switchspoon.

WHY WRITE A FRAMEWORK?

you will instinctively ask "why did you write a framework?"

RESTIFY IS GREAT
HAPI IS AWESOME
EXPRESS IS WONDERFUL

we use (or used) all of these. other frameworks are great. But they were all built under different language constraints, for companies facing different problems than the
ones npm faces.

OUR PLATFORM IS CHANGING

Node is changing.

- Native promise support has been introduced.
- New V8 versions are being picked up more frequently, making new language features available.
- and the LTS schedule makes it easier to know what to support.

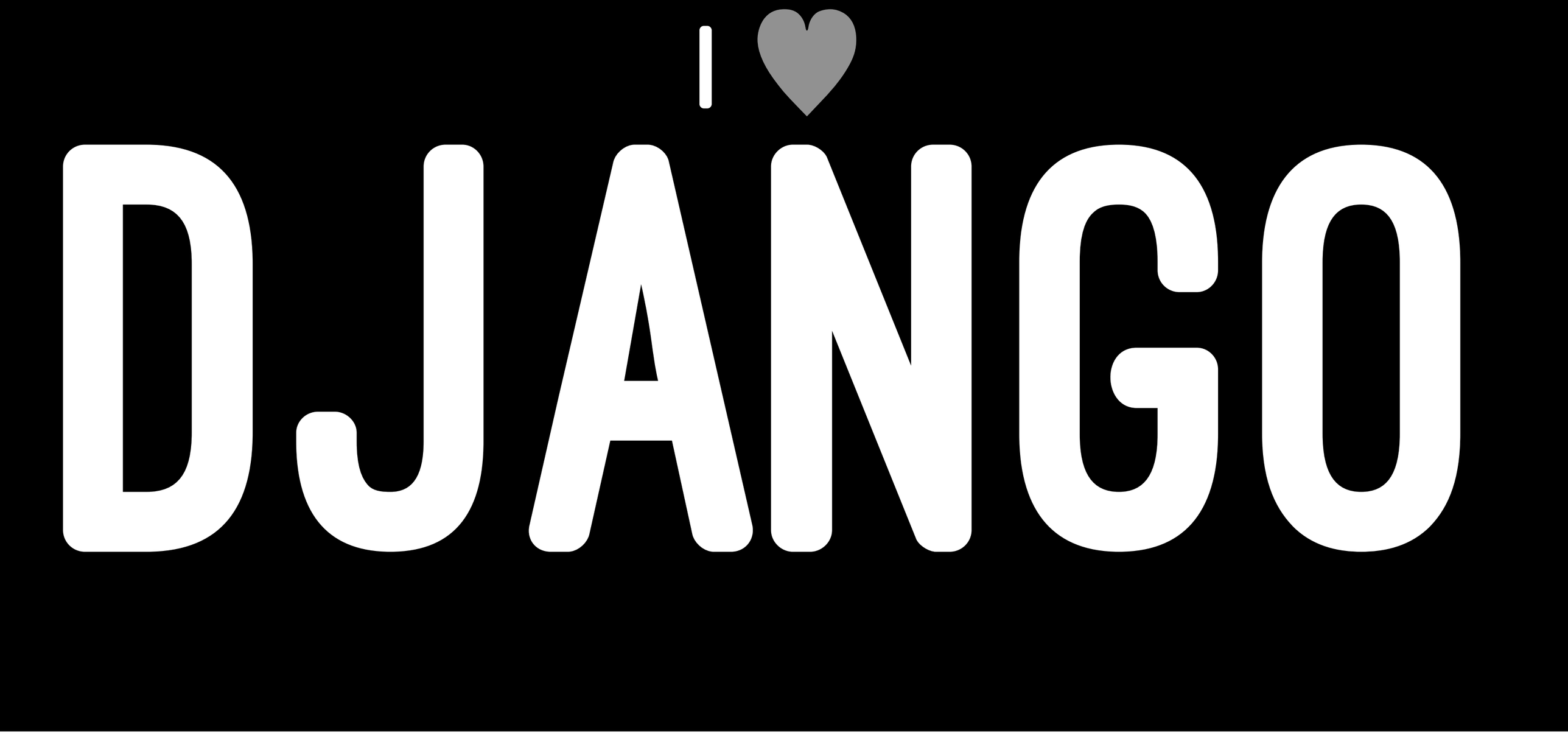
OUR LANGUAGE IS CHANGING

- JS in 2018 is a much more expressive language with better primitives than it has ever had before
- this means new APIs are possible.

OUR ECOSYSTEM IS GROWING

over the last few years there have been great efforts around building reusable framework packages
express (under "jshttp") and hapi both make their constituent parts available
for reuse.

and another reason lurks...

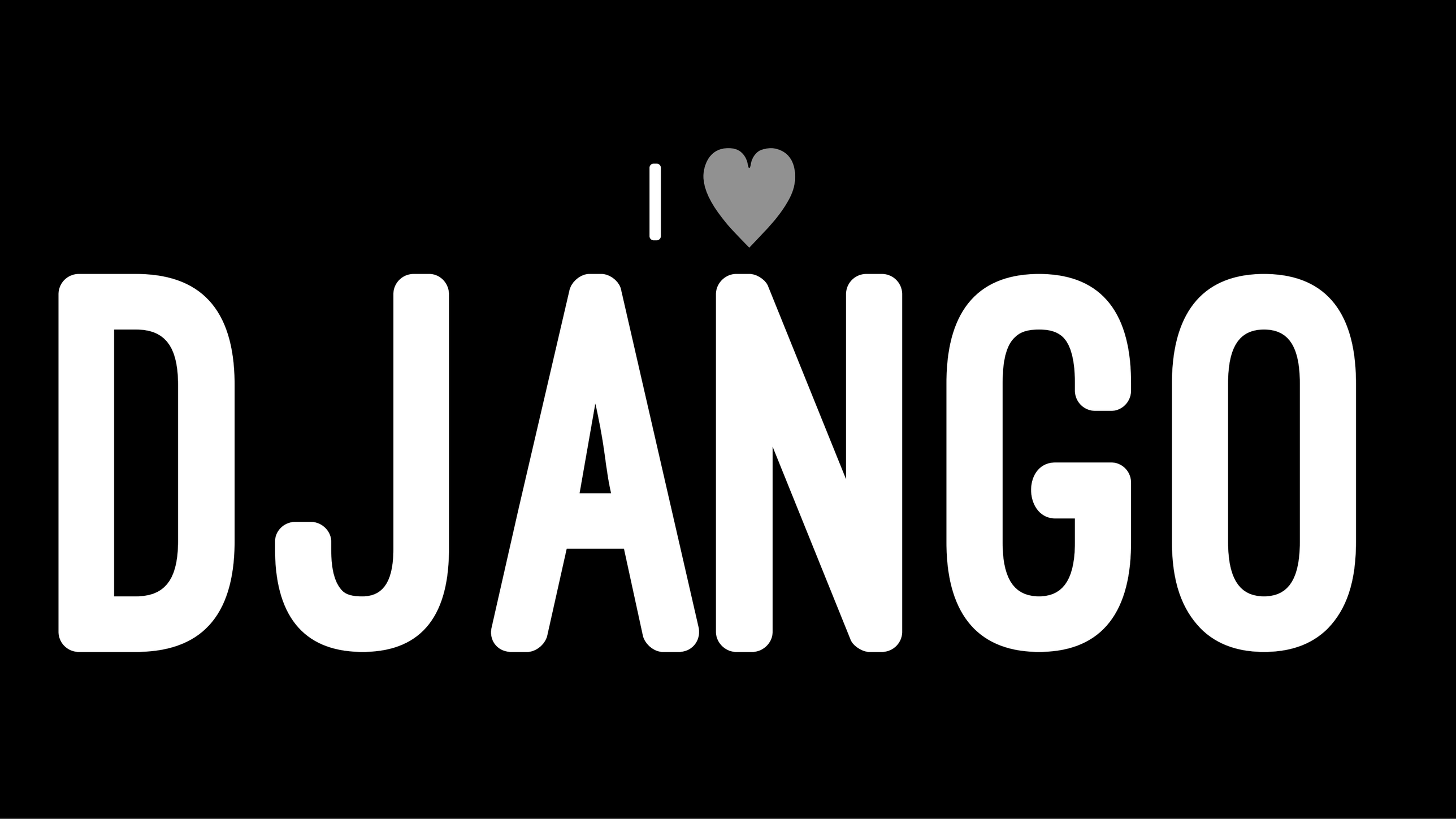


I grew up on Django.

I ♥ DJANGO

If you're not familiar with Django, it is a Python web framework that was
built by a newspaper company in Lawrence, Kansas. A small team of engineers

Django's motto is:



"The framework for perfectionists with deadlines." And I think Django lived up to its motto.

I DON'T MISS DJANGO IN PARTICULAR

I don't miss Django in particular — if you're more familiar with Rails you may be able to comiserate.

- I miss how easy it was to build a site.
- I miss the clarity of knowing what functionality went where.
- And I really miss being able to respond to a request just by returning a value from a function.

SO WHAT ARE SPIFE'S PRIORITIES?

MAKE DOING THE EASY THING THE SAME AS DOING THE RIGHT THING

identify the intersection of your most common tasks with your
most involved tasks. Seek to make those tasks easy.

RESTRICT EXPRESSIVENESS TO GAIN FUNCTIONALITY

The most flexible materials aren't the easiest materials to work with.
Building your own cabinet out of materials you source is flexible. IKEA is less expressive, but you'll have a cabinet faster.

A GOOD IDEA IS SOMETHING THAT DOES NOT SOLVE JUST ONE SINGLE PROBLEM. BUT RATHER CAN SOLVE MULTIPLE PROBLEMS AT ONCE.

- SHIGERU MIYAMOTO

Shigeru Miyamoto, the designer of the Mario series (among others), said this in 2010.

DO NOT MIND BEING THE CONSISTENT FOOLISH LITTLE HOBGOBLIN

🍴 HELLO SPIFE 🍴

So what does spife look like?

```
$ npx @npm/spife init project my-first-project
```

Use npx to start a project.

```
.
├── bin
|   └── manage.js
├── lib
|   ├── apps
|   ├── routes.js
|   ├── settings.js
|   └── views.js
├── package-lock.json
├── package.json
└── test
```

It'll scaffold a directory.

```
bin/manage.js runserver
```

Run the application like so.

ROUTING

```
const routes = require('@npm/spife/routing')

module.exports = routes`
  GET / myView
`(require('./views.js'))
```

```
const routes = require('@npm/spife/routing')

const numbers = routes.param('numbers', Number)

module.exports = routes`
  GET     /             myView
  GET     /${numbers}   myNumbers
`(require('./views.js'))
```

```
const routes = require('@npm/spife/routing')

const numbers = routes.param('numbers', value => {
  value = Number(value)
  if (value % 2 === 0) {
    throw new Error('not a valid number')
  }
  return value
})

module.exports = routes`
  GET     /             myView
  GET     /${numbers}   myNumbers
`(require('./views.js'))
```

```
bin/manage.js init app books lib/apps/books
```

```
├── bin
|   └── manage.js
├── lib
|   ├── apps
|   |   └── blogs
|   |       ├── index.js
|   |       ├── routes.js <---- new routes!
|   |       └── views.js
|   ├── routes.js
|   ├── settings.js
|   └── views.js
├── package-lock.json
├── package.json
└── test
```

```
const routes = require('@npm/spife/routing')

const numbers = routes.param('numbers', Number)

module.exports = routes`
  *       /books          books     # include the book routes
  GET     /               myView
  GET     /${numbers}     myNumbers
`({
  ...require('./views.js'),
  books: require('./apps/books/routes')
})
```

VIEW FUNCTIONS

JS Modules are your controllers and views are your controller methods.

```
module.exports = {myView}

function myView (req, context) {
  return {hello: 'world'}
}
```

- return objects for json, strings for, well, strings
- return nothing for a `204 No Content`

```
module.exports = {myView}

async function myView (req, context) {
  const target = await req.body.get('target')
  return {hello: target}
}
```

- return a promise for any of the above, that's fine too
- `req.body` returns a promise for the parsed request body

```
const reply = require('@npm/spife/reply')

module.exports = {myView}

async function myView (req, context) {
  const target = await req.body.get('target')
  return reply.status({hello: target}, 203)
}
```

- control headers and statuses using `reply`
- `reply` adds a well-known `Symbol` to your response

```
const reply = require('@npm/spife/reply')

module.exports = {myView}

async function myView (req, context) {
  const target = await req.body.get('target')
  if (target !== 'world') {
    throw new Error('oh no')
  }

  return reply.status({hello: target}, 203)
}
```

- throw errors! that's okay

```
const reply = require('@npm/spife/reply')

module.exports = {myView}

async function myView (req, context) {
  const target = await req.body.get('target')
  if (target !== 'world') {
    throw reply(new Error('oh no'), 400, {
      'cache-control': 'private'
    })
  }

  return reply.status({hello: target}, 203)
}
```

- you can control the status/headers of errors, too

MIDDLEWARE

```
// lib/middleware/redis.js
module.exports = myMiddleware

const redis = require('redis')

function myMiddleware ({redisUrl}) {
  return {
    async processServer (server, next) {
    },
    async processRequest (req, next) {
    }
  }
}
```

```
module.exports = myMiddleware

const redis = require('redis')

function myMiddleware ({redisUrl}) {
  let client = null
  return {
    async processServer (server, next) {
      client = redis.createClient(redisUrl)
      await next()
      client.end({flush: false})
    },
    async processRequest (req, next) {
    }
  }
}
```

```
module.exports = myMiddleware

const redis = require('redis')

function myMiddleware ({redisUrl}) {
  let client = null
  return {
    async processServer (server, next) {
      client = redis.createClient(redisUrl)
      await next()
      client.end({flush: false})
    },
    async processRequest (req, next) {
      req.redisClient = client
      return next(req)
    }
  }
}
```

4 AVAILABLE LIFECYCLES:

- processServer
- processRequest
  - processView
  - processBody

ORM

```
// lib/apps/books/models/book.js
const orm = require('@npm/spife/db/orm')
const joi = require('@npm/spife/joi')
const Author = require('./author')

class Book {
  constructor ({id, author_id, author, title}) {
    this.id = id
    this.author_id = author_id
    this.author = author
    this.title = title
  }
}

Book.objects = orm(Book, {
  id: joi.number().integer().greater(-1).required(),
  author: orm.fk(Author),
  title: joi.string()
})

module.exports = Book
```

```
// lib/apps/books/views.js
const Book = require('./models/book')

module.exports = myView

async function myView (req, context) {
  let books = Book.objects.all()
  return books
}
```

```
const Book = require('./models/book')

module.exports = myView

async function myView (req, context) {
  let books = Book.objects.all()

  if (req.query.author) {
    books = books.filter({
      'author.name:contains': req.query.author
    })
  }

  return books
}
```

```
const Book = require('./models/book')
const pagination = require('@npm/spife/views/paginate')

module.exports = myView

async function myView (req, context) {
  let books = Book.objects.all()

  if (req.query.author) {
    books = books.filter({
      'author.name:contains': req.query.author
    })
  }

  return pagination(req, context, {
    queryset: books
  })
}
```

OTHER STUFF:

- HOT RELOADING
- WEBPACK MIDDLEWARE (SPIFE-WEBPACK)
- TOKEN BUCKET RATELIMITING (SPIFE-RATELIMIT)
- CSRF & CSP MIDDLEWARE
- ETC ETC

BUT WHAT DO WE USE

SPIEF

FOR. THESE DAYS?

- A FILE PROCESSING QUEUE
- A LOT OF OUR CACHING SERVICES
- OUR SAAS BILLING SYSTEM
- OUR USER DATABASE
- AND. WELL

preview.npmjs.com

TO SUMMARIZE

MAKE TOOLS THAT ALIGN WITH YOUR PRIORITIES

figure out what you want to prioritize!
^ this will help you make a thousand-and-one decisions

MAKE TOOLS THAT ARE COMFORTABLE FOR YOU

practice this! it is easy to get discouraged.
^ spife is the fifth or sixth rewrite of this framework.
^ find folks that will encourage you.

SPIFE IS A THING. TRY IT OUT MAYBE

NPM IS A COMPANY THAT SELLS GOODS AND SERVICES

```
npx @npm/spife init project spork
```

CHRIS DICKINSON
TWITTER: @ISNTITVACANT
GITHUB: CHRISDICKINSON
HTTPS://GITHUB.COM/NPM/SPIFE