Functional Reactive Programming with Highland.js

Functional Reactive Programming ~~with Highland.js~~

Functional Programming

Streams

Functional Programming

What is functional programming?

Seriously, though.

What is functional programming?

What is a function?

A function is a relation that uniquely associates members of one set with members of another set.
– Wolfram MathWorld

$$\sin\left(\frac{\pi}{2}\right) = 1$$

Features of Functional Languages

First-Class Functions

```
var a = function(x) {
    return 'Hello, ' + x + '!';
};

var b = a;
```

Higher-Order Functions

```
function b(fn, x) {
    return fn(x);
}
```

Map

```
var square = function(n) {
    return n * n;
};

[1, 2, 3, 4].map(square);
// [1, 4, 9, 16]
```

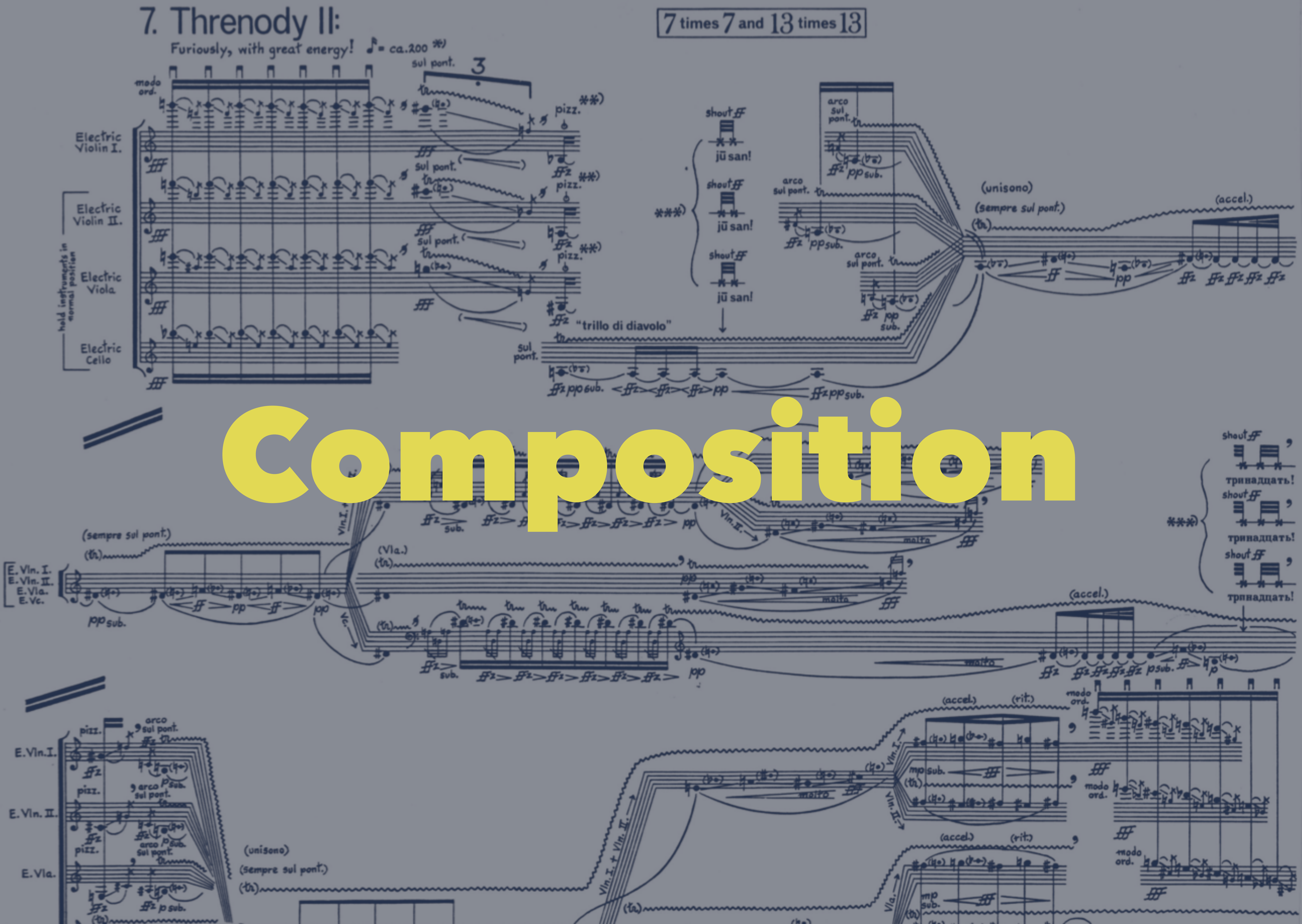
Reduce

```
var product = function(a, b) {
    return a * b;
};

[1, 2, 3, 4].reduce(product, 1);
// 24
```

Filter

```
var even = function(n) {
    return n % 2 === 0;
};

[0, 1, 2, 3, 4].filter(even);
// [0, 2, 4]
```

Referential transparency



Composition

Composition

$$(f \circ g)(x) = f(g(x))$$

$f \circ g$ with Ramda

Product of the squares of even numbers.

$$product \circ square \circ even$$

```
var R = require('ramda');
var evenSquaresProduct = R.compose(
  R.reduce(product, 1),
  R.map(square),
  R.filter(even)
);

var result = evenSquaresProduct([1, 2, 3, 4]);
// [2, 4] -> [4, 16] -> 64
```

Lazy evaluation

Streams

Why streams?

- Lower memory overhead
- Throughput
- Deal with data when it’s available

Performance

20 users, 100,000 documents, 1 minute

Callbacks

Transactions: 1

Max Memory: 1.4 GB

Availability: 5%

Response Time: 44.32 s

Performance

20 users, 100,000 documents, 1 minute

Callbacks

Transactions: 2,191
Max Memory: 94 MB
Availability: 100%
Response Time: 0.54 s

What are Streams?

What are Streams?

```
# Make your clipboard shouty in OS X.
$ pbpaste | tr '[:lower:]' '[:upper:]' | pbcopy
```

Pipe

```
var fs = require('fs');
var file = fs.createReadStream('./path/to/file.txt');
file.pipe(process.stdout);
```

Pipe

```
var AWS = require('aws-sdk');
var s3 = new AWS.S3();

function requestHandler(request, response) {
    var params = {} // Bucket, key, etc.
    var downloadStream = s3.getObject(params).createReadStream();
    downloadStream.pipe(response);
}
```

Highland.js

The high-level streams library for Node.js and the browser.

Alternatives

- RxJS
- Bacon.js

$f \circ g$ with Highland.js

```
var _ = require('highland');
var evenSquaresProduct = _.compose(
  _.reduce(1, product),
  _.map(square),
  _.filter(even)
);

var result = evenSquaresProduct([1, 2, 3, 4]);
// Not actually a result, but a lazy stream.

result.invoke('toString', [10]).pipe(process.stdout);
```

Thunk

- each
- done
- apply
- toArray
- pipe
- resume

What's with the _()?

Highland Stream Constructor _()

- Array
- Generator
- Node Readable Stream
- EventEmitter
- Promise
- Iterator
- Iterable

Highland Stream Constructor _()

```
var myStream = _();
myStream.write(1);
myStream.write(2);
myStream.write(3);
myStream.end();
```

Generator to Stream

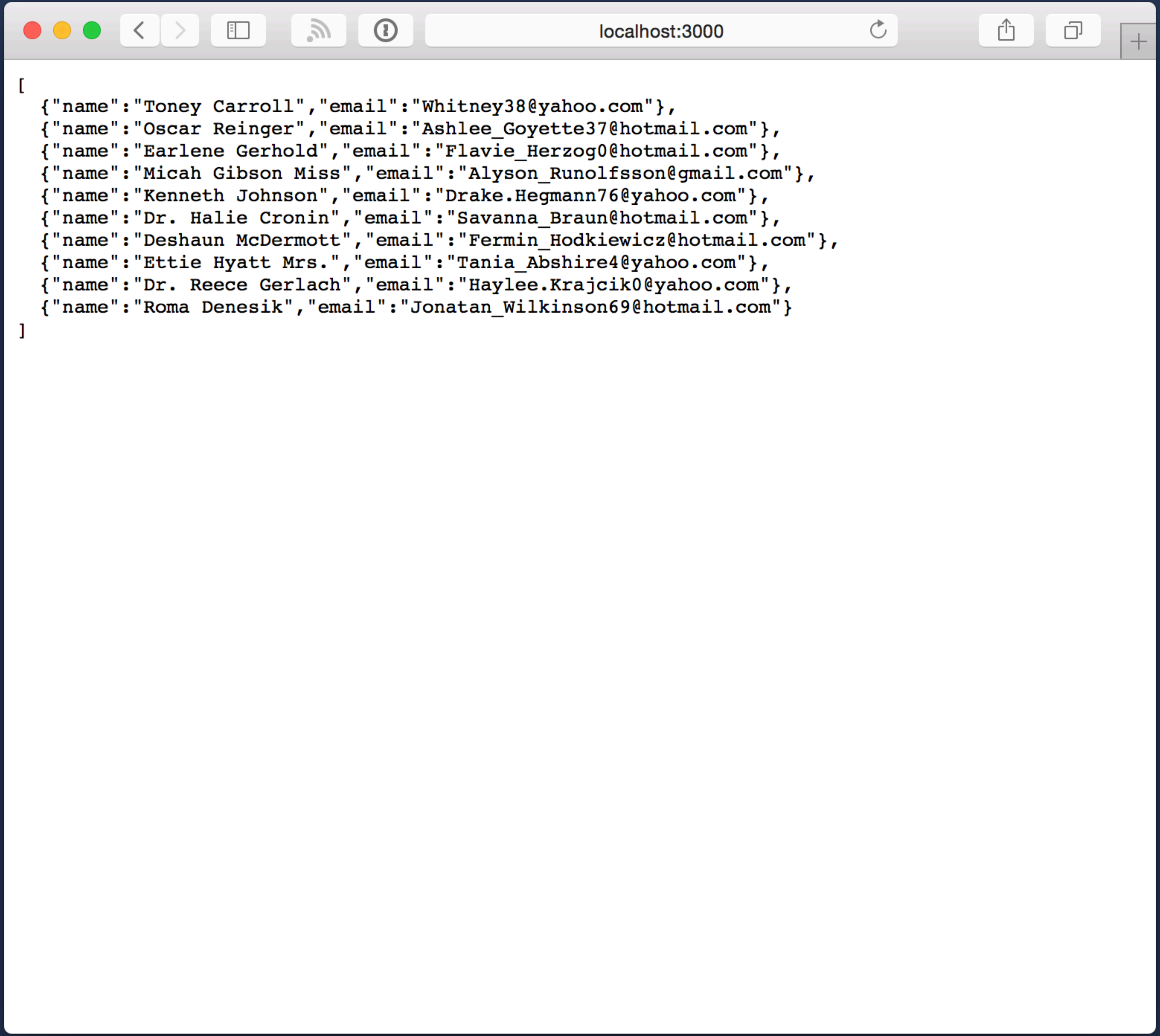
```
function* numberGenerator() {
    yield 1;
    yield 2;
    yield 3;
    yield 4;
}

var result = evenSquaresProduct(numberGenerator());
result.invoke('toString', [10]).pipe(process.stdout);
```

Real-World Example

- Get data from MongoDB
- Map objects to view model
- Serialize to JSON
- Output HTML

Real-World Example (JSON)

```
var JSONStream = require('JSONStream');

function nameEmail(person) {
    return {
        name: person.firstName + ' ' + person.lastName,
        email: person.email
    };
}

function requestHandler(request, response) {
    response.writeHead(200, {'Content-Type': 'application/json'});
    var people = db.collection('people').find({}).stream();
    var json = JSONStream.stringify();
    _(people).map(nameEmail).pipe(json).pipe(response);
}
```



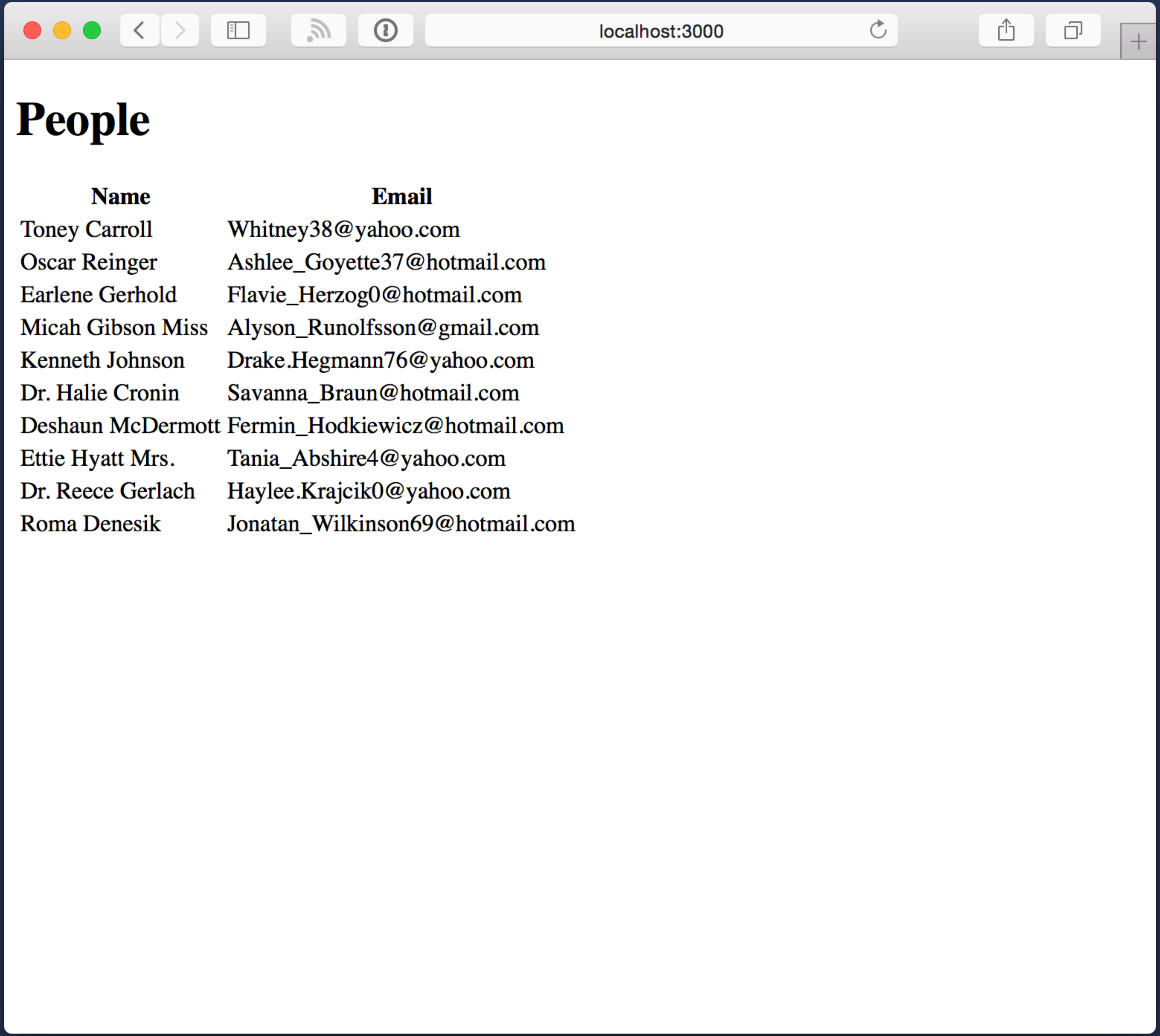
```
[
  {"name":"Toney Carroll","email":"Whitney38@yahoo.com"},
  {"name":"Oscar Reinger","email":"Ashlee_Goyette37@hotmail.com"},
  {"name":"Earlene Gerhold","email":"Flavie_Herzog0@hotmail.com"},
  {"name":"Micah Gibson Miss","email":"Alyson_Runolfsson@gmail.com"},
  {"name":"Kenneth Johnson","email":"Drake.Hegmann76@yahoo.com"},
  {"name":"Dr. Halie Cronin","email":"Savanna_Braun@hotmail.com"},
  {"name":"Deshaun McDermott","email":"Fermin_Hodkiewicz@hotmail.com"},
  {"name":"Ettie Hyatt Mrs.","email":"Tania_Abshire4@yahoo.com"},
  {"name":"Dr. Reece Gerlach","email":"Haylee.Krajcik0@yahoo.com"},
  {"name":"Roma Denesik","email":"Jonatan_Wilkinson69@hotmail.com"}
]
```

Real-World Example (HTML)

```
var Dust = require('dustjs-linkedin');

var compiled = Dust.compile(templateSrc, 'template');
Dust.loadSource(compiled);
var template = _.partial(Dust.stream, 'template');

function requestHandler(request, response) {
    response.writeHead(200, {'Content-Type': 'text/html'});
    var findStream = db.collection('people').find({}).stream();
    var people = _.map(nameEmail, findStream);
    var context = {people: people};
    template(context).pipe(response);
}
```

Dust Template

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>OKC.js Lightning Talk, Round 10</title>
    </head>
    <body>
        <h1>People</h1>
        <table>
            <thead>
                <tr>
                    <th>Name</th>
                    <th>Email</th>
                </tr>
            </thead>
            <tbody>
                {#people}<tr><td>{name}</td><td>{email}</td></tr>{/people}
            </tbody>
        </table>
    </body>
</html>
```



People

Name	Email
Toney Carroll	Whitney38@yahoo.com
Oscar Reinger	Ashlee_Goyette37@hotmail.com
Earlene Gerhold	Flavie_Herzog0@hotmail.com
Micah Gibson Miss	Alyson_Runolfsson@gmail.com
Kenneth Johnson	Drake.Hegmann76@yahoo.com
Dr. Halie Cronin	Savanna_Braun@hotmail.com
Deshaun McDermott	Fermin_Hodkiewicz@hotmail.com
Ettie Hyatt Mrs.	Tania_Abshire4@yahoo.com
Dr. Reece Gerlach	Haylee.Krajcik0@yahoo.com
Roma Denesik	Jonatan_Wilkinson69@hotmail.com

@nlindley