Node.js

Server responsibilities in SPA world

- Expose API for resources
- Store and fetch data
- Security and data safety & consistency
- Performance

REST APIs

Representational
State
Transfer

Usually transferred as JSON

```
{
    “id”: 42,
    “name”: “Project”,
    “milestones”: [
        “Alpha”,
        “Beta”,
        “Release”
    ]
}
```

Resources

List Resources

- http://example.com/api/projects/
- GET - get all projects (or partials of them)
- POST - create a new project
- PUT - replace the entire list of projects
- DELETE - delete all projects

Object Resources

- http://example.com/api/projects/123
- GET - retrieve the project
- PUT - update the project
- DELETE - delete the project

Server is still responsible for validation and consistency

Why is Node getting so popular?

Unique architecture

The dream of sharing server and client code

JSON serialization is trivial

Easy integration with other JS based systems (e.g. MongoDB)

Great for high throughput real-time services

Non blocking IO

What you’re used to:

```
var file = fs.readFile(‘file’);
```

Requires lots of threads

and thus many servers

Node = 1 active thread

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

fs.readFile('myFile.txt', function(err, data) {
    if (err) throw err;

    data = data.toString().replace('\t', '    ');

    fs.writeFile('myFile.txt', data, function(err) {
        if (err) throw err;

        console.log('converted tabs to spaces');
    });
});
```

npm

Comes bundled with Node

Lots of libraries

package.json

Just remember to run:

npm install

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

Express

Simple HTTP library

A lot like Ruby's Sinatra & Python's Flask

Express boilerplate

```
var express = require('express');
var bodyParser = require('body-parser');
var app = express();

app.use(express.static('public'));
app.use(bodyParser.json());

var server = app.listen(3000, function() {
    console.log('Server is up and running');
});
```

GET

```
app.get('/example', function(request, response) {
    response.send({success: true});
});
```

Reuse path

```
app.route('/resources')
    .get(function(request, response) {})
    .post(function(request, response) {});
```

POST

```
app.post('/example', function(request, response) {
    console.log('Request body', request.body);

    response.sendStatus(200);
});
```

Sending responses

- response.send(json)
- response.sendStatus(404)
- response.status(404).send({error: ‘Not found’})
- response.redirect(‘/example’)
- You have to respond with *something* or the caller will hang

Path params

```
app.post('/example/:id', function(request, response) {
    console.log('Request id', request.params.id);
    console.log('Request body', request.body);

    response.sendStatus(200);
});
```

Query params

```
app.post('/example', function(request, response) {
    console.log('Request query', request.query);

    response.sendStatus(200);
});
```

Generic Parameter Handling

```
app.param('id', function(request, response, next, id) {
    request.id = parseInt(id);
    next();
});
```

Reuse code for path

```
app.route('/projects')
    .all(function(request, response) {
        console.log('Accessing projects');
    })
    .get(function(request, response) {
        // Handle get
    });
```

Everything is possible

- Server side template rendering
- Authentication
- Caching
- Database integration
- Call external services
- … and much more. Google is still your friend!

HTTP in Angular

$http service

```
$http.get('/some/url').then(
    function(response) {
        console.log('got response', response.data);
    },
    function(rejection) {
        console.log('request failed');
    }
);
```

```
$http.post('/some/url', {request: 'body'}).then(
    function(response) {
        console.log('got response', response.data);
    }
);
```