Dev Lab: Node + Express

What is Node?

Node.js = JavaScript + File I/O + A Package Manager

or: Node.js = JavaScript - A Web Browser

What’s it like?

- Single-threaded
- Asynchronous & non-blocking
- Great for real-time applications (like maybe a web app, perhaps)
- Good community support

Adding Packages

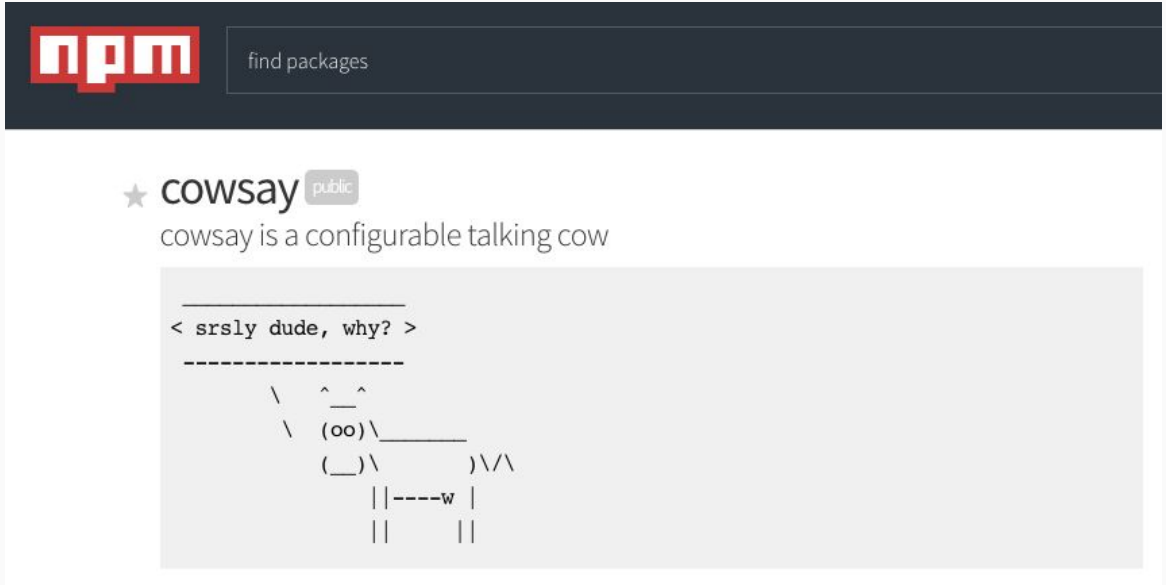
In your terminal:

```
npm init
```

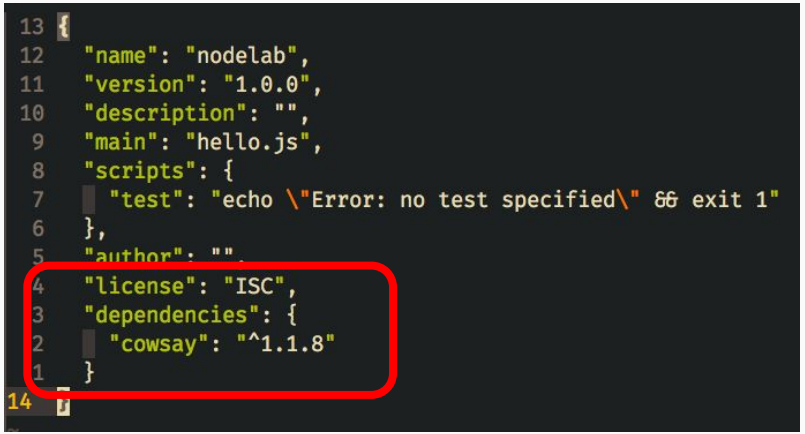
```
npm install --save <pkgname>
```

In your script:

```
require('<pkgname>')
```



package.json



```
{
  "name": "nodelab",
  "version": "1.0.0",
  "description": "",
  "main": "hello.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "cowsay": "^1.1.8"
  }
}
```

Adding Packages

In your terminal:

```
npm init
```

```
npm install --save <pkgname>
```

In your script:

```
require('<pkgname>')
```

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

console.log(cowsay.say({
  text: 'Node is pretty neat!'
}));
```

```
 ----------------------
< Node is pretty neat! >
 ----------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Creating Your Own Modules

Whatever you assign `module.exports` to is what you get when you `require` it later.

secrets.js

```
// I can do whatever work I want up here!

module.exports = {
  password: 'hunter2'
};
```

Requiring Your Own Modules

Works just like adding an npm package.

`require(‘./relative/path/to/file’)`

Note: don’t add “.js” at the end.

```
1 var secrets = require('./secrets');
2
3 console.log('The secret password is ' + secrets.password);
4
```

```
The secret password is hunter2
```

Debugging

`node debug <filename>`

Set breakpoints by adding

`debugger;`

in your script

Also try: node-inspector, WebStorm debugger

Stepping

- `cont`, `c` - Continue execution
- `next`, `n` - Step next
- `step`, `s` - Step in
- `out`, `o` - Step out
- `pause` - Pause running code (like pause button in Developer Tools)

Information

- `backtrace`, `bt` - Print backtrace of current execution frame
- `list(5)` - List scripts source code with 5 line context (5 lines before and after)
- `watch(expr)` - Add expression to watch list
- `unwatch(expr)` - Remove expression from watch list
- `watchers` - List all watchers and their values (automatically listed on each breakpoint)
- `repl` - Open debugger's repl for evaluation in debugging script's context
- `exec expr` - Execute an expression in debugging script's context

What is Express?

express

A lightweight, extensible node module that lets you make web servers with very little code.

A simple server with Express

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

app.use(express.static(__dirname + '/public'));

var port = process.env.PORT || 3000;
console.log("Express server running on " + port);
app.listen(process.env.PORT || port);
```

Creating an instance of express

A simple server with Express

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

app.use(express.static(__dirname + '/public'));

var port = process.env.PORT || 3000;
console.log("Express server running on " + port);
app.listen(process.env.PORT || port);
```

`app.use` - adds middleware to your server
`express.static` - built-in middleware. Lets you define the root directory from which you will serve files

A simple server with Express

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

app.use(express.static(__dirname + '/public'));

var port = process.env.PORT || 3000;
console.log("Express server running on " + port);
app.listen(process.env.PORT || port);
```

`app.listen` - listen for requests on a given port

Routers

- A way to specify which function should handle requests for each URL
- Chainable & abstractable
- You can add route-specific middleware if you want.

Using Routes

```
// Get the packages we need
var express = require('express');
var mongoose = require('mongoose');
var Llama = require('./models/llama');
var bodyParser = require('body-parser');
var router = express.Router();
```

```

// All our routes will start with /api
app.use('/api', router);

//Default route here
var homeRoute = router.route('/');

homeRoute.get(function(req, res) {
  res.json({ message: 'Hello World!' });
});

//Llama route
var llamaRoute = router.route('/llamas');

llamaRoute.get(function(req, res) {
  res.json([{ "name": "alice", "height": 12 }, { "name": "jane", "height": 13 }]);
});

```

GET requests to /api/llamas go here!

Addendum #1: Solving Callback Hell with Promises

Promise me, Ned.

```
return Promise.resolve()
```

```
var handleVote = function(body, from, res) {
  body = body.trim().toUpperCase();

  getActiveQuestion(function(err, question) {
    if (err || !question || !question.voting) {
      console.log('Something went wrong');
    } else {
      getStudent(from, function(err, student) {
        if (err || !student || alreadyVoted(question, student)
          console.log('Something went wrong');
        } else {
          castVote(question, student, body, function() {
            getActiveQuestion(function(err, question) {
              if (err) {
                console.log('Something went wrong');
              } else {
                io.emit('update', question);
                return sendMessage(res, "Thanks for voting, " +
              }
            });
          });
        }
      });
    }
  });
};
```

Old & Busted

```
var handleVote = function(body, from, res) {
  body = body.trim.toUpperCase();

  Promise.all([getActiveQuestion(), getStudent(from)])
    .then(function(question, student) {
      if (!question || !question.voting ||
        !student || alreadyVoted(question, student) ||
        !isValidVote(question, body)) {
        return Promise.reject();
      }

      return castVote(question, student, body);
    }).then(function() {
      return getActiveQuestion();
    }).then(function(question) {
      io.emit('update', question);
      sendMessage(res, "Thanks for voting, " + student.netid +
        "! Your vote was: " + body);
    }).catch(function(err) {
      console.log('Something went wrong!');
    });
};
```

New Hotness

Asynchronously fetch a question and a student from the DB.

getActiveQuestion() and getStudent() return promises.

When they're both ready, they get passed into our then() function.

```
var handleVote = function(body, from, res) {
  body = body.trim.toUpperCase();

  Promise.all([getActiveQuestion(), getStudent(from)])
    .then(function(question, student) {
      if (!question || !question.voting ||
        !student || alreadyVoted(question, student) ||
        !isValidVote(question, body)) {
        return Promise.reject();
      }

      return castVote(question, student, body);
    }).then(function() {
      return getActiveQuestion();
    }).then(function(question) {
      io.emit('update', question);
      sendMessage(res, "Thanks for voting, " + student.netid +
        "! Your vote was: " + body);
    }).catch(function(err) {
      console.log('Something went wrong!');
    });
};
```

We can chain then() functions.

Each then() function must return either a promise or a value.

If it’s a promise, it waits for THAT promise to resolve, then passes its resolved value into the next then() function in the chain.

```
1
2  var handleVote = function(body, from, res) {
3    body = body.trim.toUpperCase();
4
5    Promise.all([getActiveQuestion(), getStudent(from)])
6      .then(function(question, student) {
7        if (!question || !question.voting ||
8          !student || alreadyVoted(question, student) ||
9          !isValidVote(question, body)) {
10         return Promise.reject();
11       }
12
13       return castVote(question, student, body);
14     }).then(function() {
15       return getActiveQuestion();
16     }).then(function(question) {
17       io.emit('update', question);
18       sendMessage(res, "Thanks for voting, " + student.netid +
19         "! Your vote was: " + body);
20     }).catch(function(err) {
21       console.log('Something went wrong!');
22     });
23 };
24
```

If at any point the promise gets rejected (i.e. there was an error), it stops executing then() functions and executes the catch() function.

```
1
2  var handleVote = function(body, from, res) {
3    body = body.trim.toUpperCase();
4
5    Promise.all([getActiveQuestion(), getStudent(from)])
6      .then(function(question, student) {
7        if (!question || !question.voting ||
8          !student || alreadyVoted(question, student) ||
9          !isValidVote(question, body)) {
10         return Promise.reject();
11       }
12
13       return castVote(question, student, body);
14     }).then(function() {
15       return getActiveQuestion();
16     }).then(function(question) {
17       io.emit('update', question);
18       sendMessage(res, "Thanks for voting, " + student.netid +
19         "! Your vote was: " + body);
20     }).catch(function(err) {
21       console.log('Something went wrong!');
22     });
23 };
24
```

Addendum #2: Mongoose.js

What is Mongoose.js?

mongoose

- Node module for interacting with MongoDB
- Makes it easy(ish) to create schemas, validate data, and run DB queries.

A Simple App with Mongoose: Setup

Mongoose schemas let you define:

- Fields
- Types
- Validation requirements
- Error messages

```
var readlineSync = require('readline-sync');
var mongoose = require('mongoose');
mongoose.Promise = require('bluebird');
var Schema = mongoose.Schema;

mongoose.connect('mongodb://localhost/lab');
var db = mongoose.connection;

var llamaSchema = new Schema({
  name: { type: String, required: true },
  age: { type: Number, required: true, min: [18, 'Adult llamas only!'] },
  dateCreated: { type: Date, default: Date.now }
});

var Llama = mongoose.model('Llama', llamaSchema);

```

A Simple App with Mongoose: Main Loop

Create a new llama and then try to save it to the DB.

Mongoose will automatically try to validate the llama.

```
16
17 function loop() {
18   var name = readlineSync.question('What is the llama\'s name? ');
19   var age = readlineSync.questionInt('How old is the llama? ');
20
21   var llama = new Llama({
22     name: name,
23     age: age
24   });
25
26   llama.save()
27     .then(function(result) { // We did it!
28       console.log(result);
29     }).catch(function(err) { // Aw beans, something's goofed!
30       var errors = err.errors;
31       for (var key in errors) {
32         console.log(errors[key].message);
33       }
34     }).finally(loop);
35 }
36
37 db.once('open', loop);
```

The error messages you defined show up here.

Tips for MP4

- Read the Mongoose Quick Start guide.
- Learn how Promises work and use them to write your DB code.
- Use the node debugger.
- Use Postman to test your HTTP calls.