# Server-Side Development with NodeJS

Week 1: Intro to Server-side Development

Intro to Node.js and NPM

* Node.js and NPM
  + Uses event-driven, non-blocking I/O model
    - Makes it lightweight and efficient
  + Utilities written in JS for web development:
    - Bower, Grunt, Gulp, Yeoman
  + Server-side Development:
    - Web server, Business logic, Database access
  + Node Package Manager:
    - Manages ecosystem of node modules/packages

Node Modules

* Node Modules
  + JS Modules:
    - JS does not define a standard library
    - CommonJS fills in cap by defining APIs for common application needs, which Node follows
  + Node Modules: Each file in Node is its own module
    - *module* variable: gives access to the current module definition in a file
    - *module.exports* variable: determines the export from current module
    - *require* function: used to import a module
  + Module Ex.:
    - rectangle Module:

module.exports = function(){

return {perimeter:function(x,y){

return (2\*(x+y));},

area: function(x,y)

{return(x\*y)));

} }; }

* + - Using this module:

var rect = require(‘./rectangle’);

* + Module Ex:
    - rectangle Module:

exports.perimeter = function (x, y){

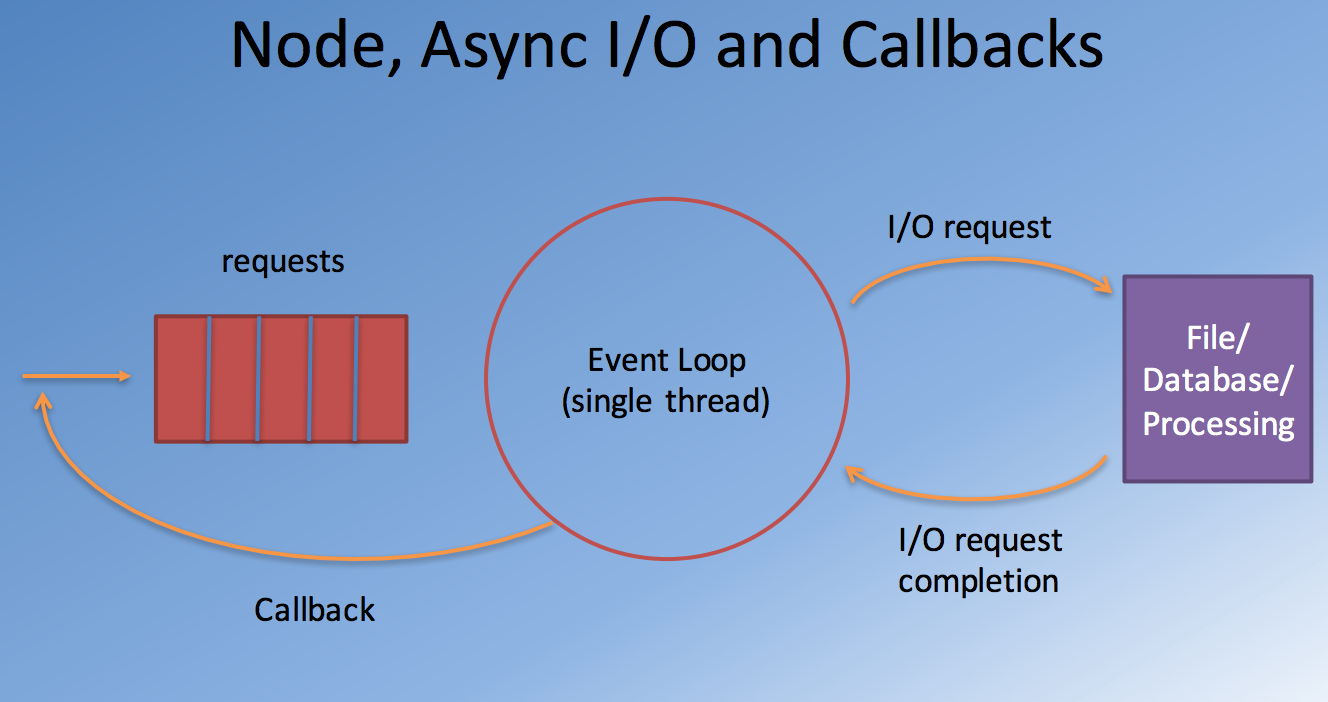
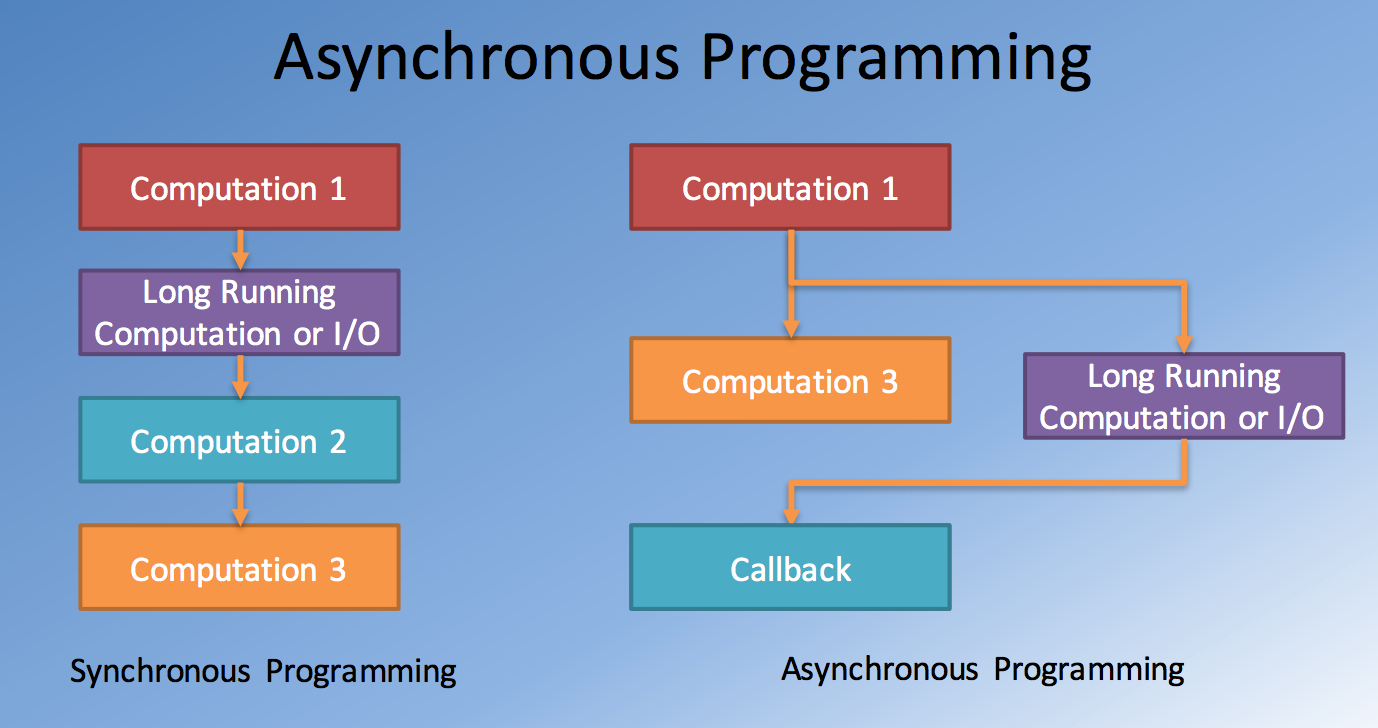
return (2\*(x+y));

}

exports.area = function (x, y) {

return (x\*y);

}

* + - (exports is alias for module.exports)
* Node Modules: Callbacks and Error Handling
  + Features of JS:
    - First-class functions: functions can be treated as any other variable
    - Closures:
      * A function defined inside another function has access to all other variables in outer function
      * Inner function will continue to have access to the variables from the outer scope even after the outer function has returned
  + Asynchronous Programming:
  + Callbacks and Error Handling
    - rectangle module:

module.exports = function(x,y,callback) {

try { if (x < 0 || y < 0) {

throw new Error("Rectangle dimensions should be greater than zero: l = " + x + ", and b = " + y);

} else

callback(null,

{ perimeter: function () { return (2\*(x+y)); }, area:function () { return (x\*y); }

});

}

catch (error) { callback(error,null); }

}

* + - Calling the function:

rect(l,b, function(err, rectangle) {

if(err) {

Console.log(err);

}

else {

…

}

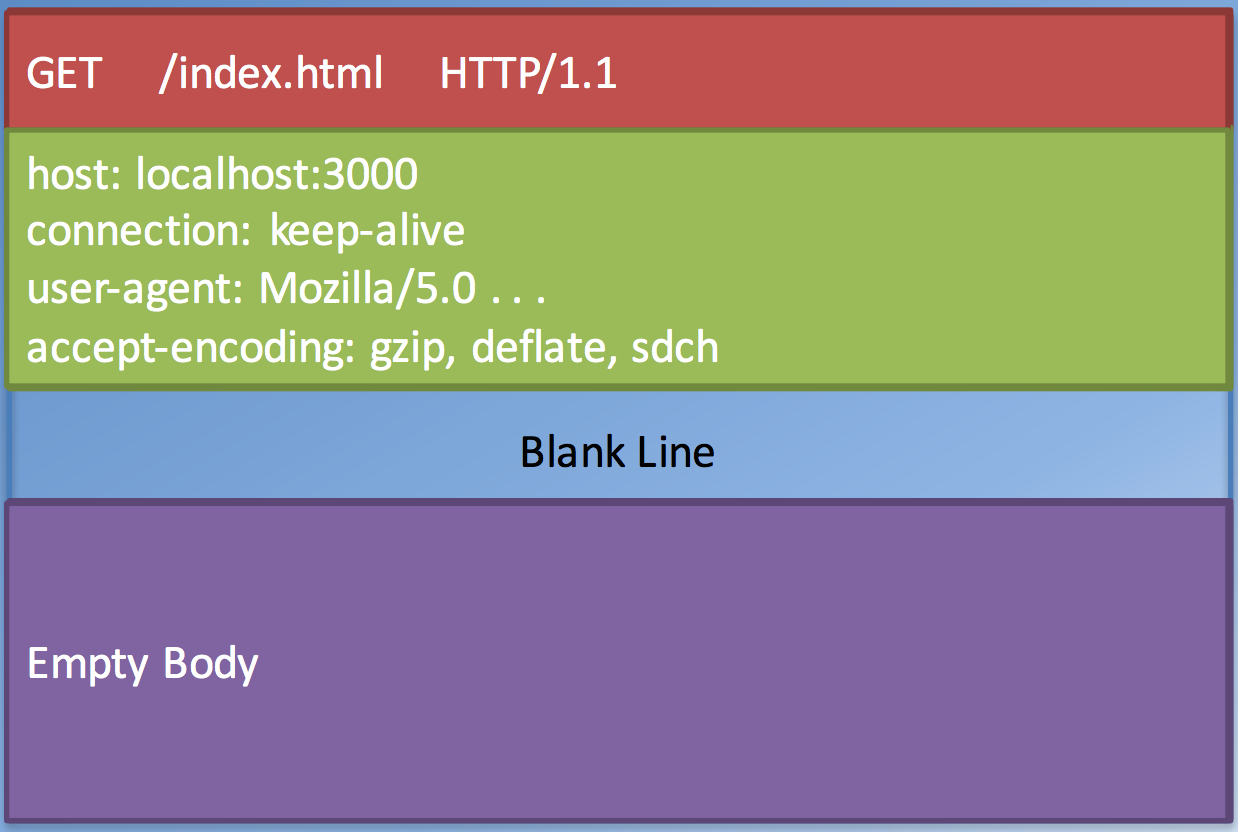
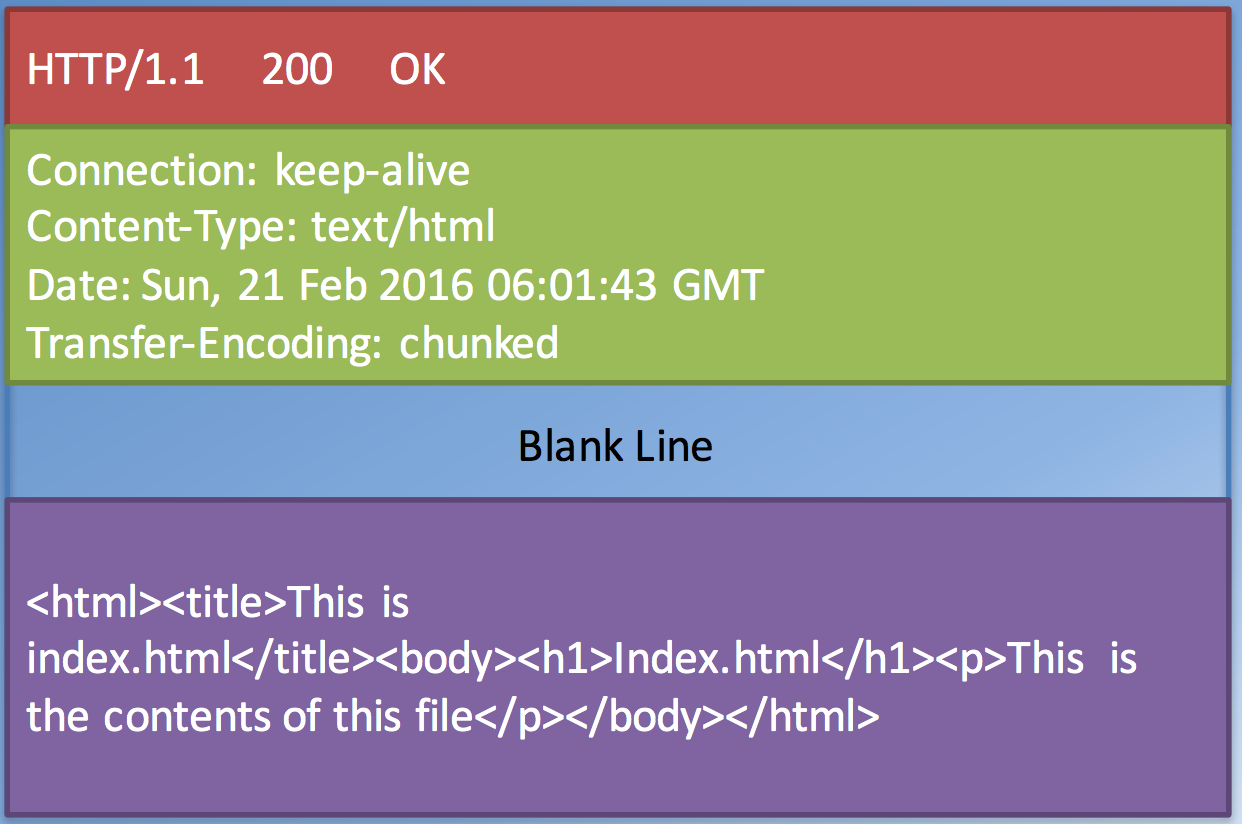
});

* Node Modules: Further Details
  + Node Modules:
    - File-based Modules
    - Core Modules ex. path, fs, os, util
    - External Node modules: third-party modules installed using NPM in node\_modules folder in your Node app
  + Using Node Modules:
    - Include them using require function:
      * File-based modules:
        + require(‘./module\_name’)
        + Specify the relative path to the file
      * Core and External modules:
        + Require(‘module\_name’)
        + Looks for external modules in ./node\_modules
  + A Brief Tour of a Node Module

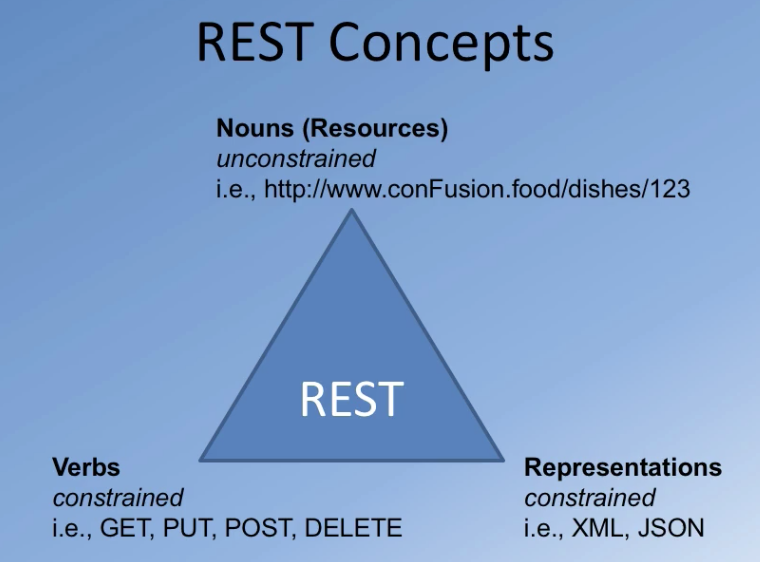
Semantic Versioning –

* + <major version>.<minor version>.<patch>
  + npm install can specify the acceptable package version:
    - Exact: npminstall express@4.0.0
    - Patch acceptable: npminstall express@”~4.0.0”
    - Minor version acceptable: npm install express@”^4.0.0”

Node and HTTP

* Client and Server
* Client-Server Communication
  + Network operations cause unexpected delays
  + Need to write apps recognizing the asynchronous nature of communication
* Hypertext Transfer Protocol (HTTP)
  + Client-server communications protocol
  + Allows retrieving interlinked text documents (hypertext)
    - WWW
  + HTTP Verbs:
    - HEAD
    - GET
    - POST
    - PUT
    - DELETE
    - TRACE
    - OPTIONS
    - CONNECT
* HTTP Request Message
  + 
* HTTP Response Message
  + 
* HTTP Response:
  + XML or JSON
* Node HTTP Module:
  + Core networking module supporting a high-performance for a HTTP stack
  + Using the module:
    - var http = require(‘http’)
  + Creating a server:
    - var server = http.createServer(function(req, res){…});
  + Starting the server:
    - server.listen(port,…)
  + Incoming request message information available through the first parameter “req”
    - req.headers, req.body, . . .
  + Response message is constructed on the second parameter “res”
    - res.setHeader("Content-Type", "text/html");
    - res.statusCode = 200;
    - res.writeHead(200, { 'Content-Type': 'text/html' });
    - res.write(’Hello World!');
    - res.end(‘<html><body><h1>Hello World</h1></body></html>’);
  + Node path Module:
    - Using path Module:
      * Var path = require(‘path’);
    - Some example path methods:
      * Path.resolve(‘./public’+fileUrl);
      * Path.extname(filePath)
  + Node fs Module:
    - Var fs = require(‘fs’);
    - Fs.exists(filePath, function (exists) {…});
    - Fs.createReadStream(filePath).path(res);

Intro to Express

* Intro to Express:
  + Web application framework that provides a robust set of features, with many third-party middleware to extend functionality
  + Install with: npm install express - -save
  + Express Middleware: provide a lot of plug-in functionality that can be used within your Express application
    - Ex. morgan for logging:
      * Var morgan = require(‘morgan’);
      * App.use(morgan(‘dev’));
* Representational State Transfer (REST):
  + Web Services:
    - A system designed to support interoperability of systems connected over a network
      * Service oriented architecture (SOA)
      * Standardized way of integrating web-based apps using open standards operating over the internet
    - Two common approaches used in practice:
      * SOAP
      * REST
  + REST:
    - A style of software architecture for distributed hypermedia systems such as the WWW
    - Verbs:
      * HTTP GET --- READ
      * HTTP POST --- CREATE
      * HTTP PUT ---- UPDATE
      * HTTP DELETE --- DELETE
    - Representations: JSON and XML
  + Express Application Routes
    - We examined REST in the previous lecture
    - Identify an end point with URI
    - App Routes:
      * App.all(‘/dishes’, function( req, res, next) {…});
      * App.get(‘/dishes’, function( req, res, next) {…});
      * App.post (‘/dishes’, function( req, res, next) {…});
      * App.put (‘/dishes’, function( req, res, next) {…});
      * App.deletel(‘/dishes’, function( req, res, next) {…});
  + Body Parser:
    - Middleware to parse the body of the message
    - Using body parser:
      * Var bodyParser = require(‘body-parser’)
      * App.use(bodyParser.json()); //parse the JSON body
    - Parses the body of the messages and populalates the req.body
  + Express Router: creates a mini-Express application
    - Var dishRouter = express.Router()
    - DishRouter.use(bodyParser.json());
    - dishRouter.route(‘/’)

Week 2: Data & MongoDB

Express Generator

* Express Generator
  + Quick scaffolding tool to generate an Express application skeleton
  + Installing Express generator
    - npm install express-generator –g
  + generating an Express application: generates a folder under the current folder named App name

express <App Name>

* + app.js: starting application
  + package.json
  + public: static resources
  + routes application routes
  + views template: template engine templates

Intro to MongoDB

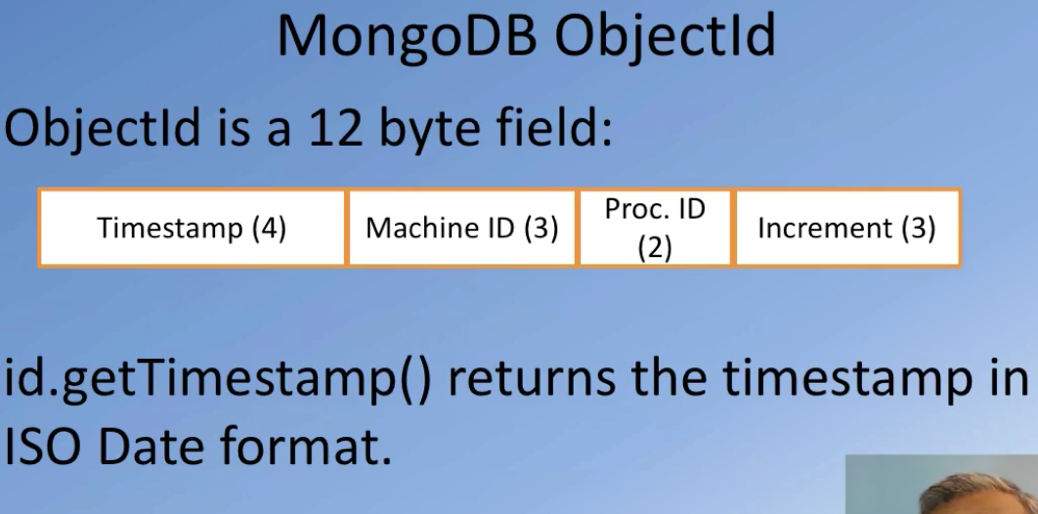
* Intro to MongoDB
  + Databases:
    - Document databases(MongoDB)
      * Self-contained piece of info
      * Collection: collection of documents
      * Database: a set of collections
    - Key-Value databases
    - Column-family databases
    - Graph databases
  + MongoDB:
    - Server can support multiple databases
    - Database consists of a set of collections
    - A collection is a set of documents
    - Document is effectively a JSON document with some additional features
    - Stores documents in BSON format (Binary JSON)
      * Supports length prefix on each value
      * Information about the type of a field value
      * Additional primitive types not supported by raw JSON
  + MongoDB ObjectId
    - Every document in Mongo must have an “\_id” field that is unique
    - Default ObjectId created by Mongo when you insert a document
    - {

"\_id" : ObjectId("56ce74c0b02806eff4558f1f"),

"name" : "Uthapizza",

"description" : "Test"

}

* + - 

Node and MongoDB

* Node MongoDB Driver
  + Provides a high-level API for a Node app to interact with MongoDB Server
  + npm install mongodb –save
  + Can connect to MongoDB, insert, delete, update and query documents
  + Supports both callback based and promise based interactions

Mongoose ODM

* MongoDB
  + No pre-specified structure
* Mongoose:
  + Mongoose ODM
    - Object Data Model
    - Object Document Mapping
    - Object relation mapping (ORM)
  + Connecting to MongoDB

var mongoose = require('mongoose’);

var url = 'mongodb://localhost:27017/conFusion'; mongoose.connect(url);

var db = mongoose.connection;

db.on('error', console.error.bind(console, 'connection error:'));

db.once(‘open’, function () { console.log(“Connected correctly to server”); . . . });

* + Mongoose Schema:
    - Schema: structure of the data to be stored:
    - Defines all the fields of your document and their types
    - Schema types: String, Number, Date, Buffer, Boolean, Mixed, ObjectId, Array
    - Schema used to create Model function
  + Schema example:

var mongoose = require('mongoose');

var Schema = mongoose.Schema;

var dishSchema = new Schema({ name: { type: String, required: true, unique: true}, description: { type: String, required: true}}, {timestamps: true});

var Dishes = mongoose.model('Dish', dishSchema);

* + Mongoose-based Database Operations

var Dishes = require('./models/dishes');

var newDish = Dishes({ name: 'Uthapizza', description: 'Test' });

newDish.save(function (err) {

if (err) throw err;

Dishes.find({}, function (err, dishes) {

if (err) throw err;

db.collection('dishes').drop(function () {

db.close();

});

});

});