Nodejs

Kayartaya Vinod

Who am I?

- Vinod Kumar K
- Graduated in 1994 (B.Sc, Sahyadri Science College, Shimoga)
- Learnt BASIC in 1989 and started teaching C and Foxpro in 1994
- Owned a training institute between 1999 and 2002
- Over 20 years of experience in Software Training and Development
- Mostly work with Java based technologies and web frameworks (both client and server)
  - Also code in PHP, C#, ASP.Net and more
- Share my training resources at http://vinod.co

Agenda

- Introduction
- Getting and installing nodejs
- Using modules
- Using the http module to serve
- Understanding the nodejs package manager (npm)
- Understanding package.json
- Using the express framework
- Build a REST API using nodejs, express and mongodb

Introduction

- Node is just JavaScript **with out** a browser
- Node is based on the **V8** JavaScript runtime engine used in the Google Chrome
- Allows JavaScript to be used in the **backend**
  - For example, you can use jQuery, File IO, or run web server at the backend
- Node makes it possible for you to take advantage of everything **JavaScript has to offer**, with out being tied to a browser

Introduction

- Node.js was created and first published for Linux use in **2009**
- Its development and maintenance was spearheaded by **Ryan Dahl** and sponsored by **Joyent**, the firm where Dahl worked

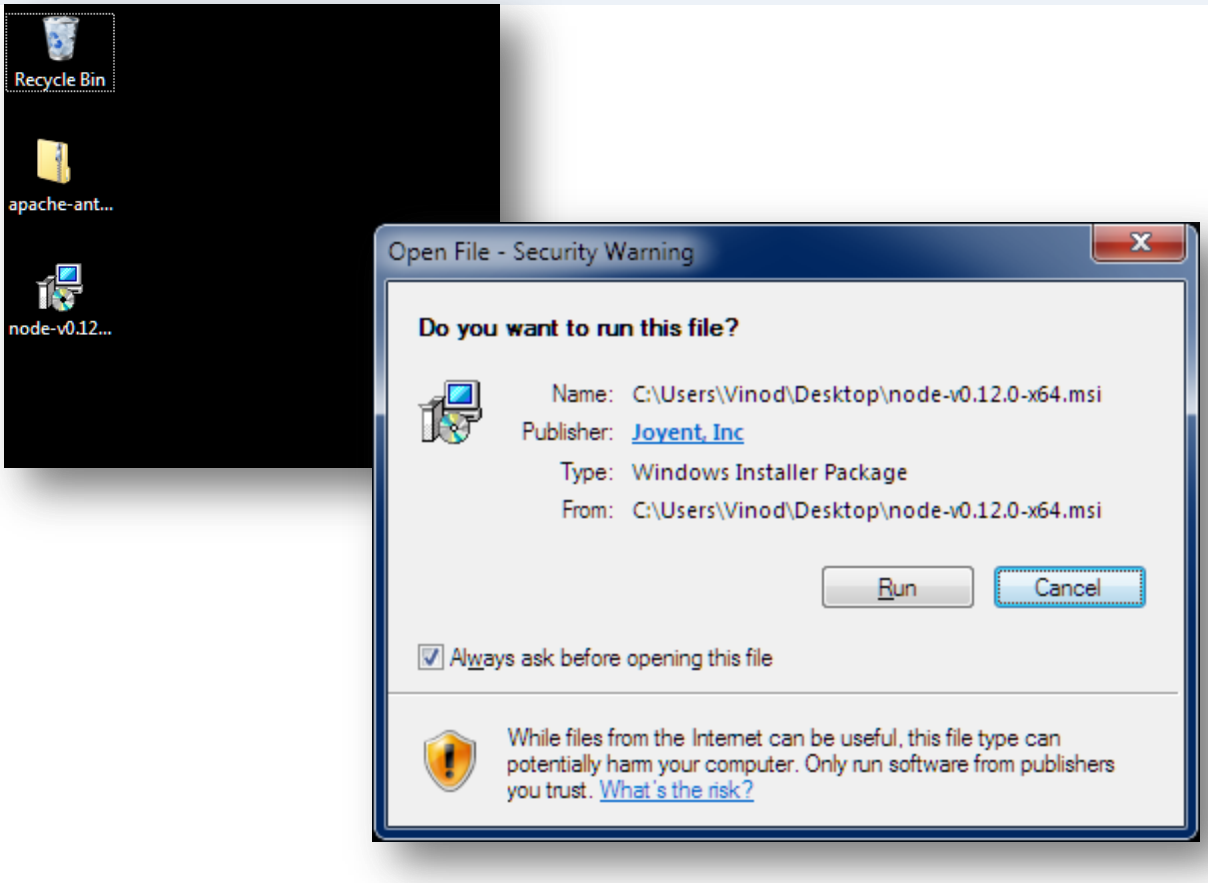
Introduction

- **npm**, a package manager for Node.js libraries, was introduced in **2011**
- In June 2011, Microsoft partnered with Joyent to create a **native Windows version** of Node.js
- The first Node.js build to support Windows was released in **July**

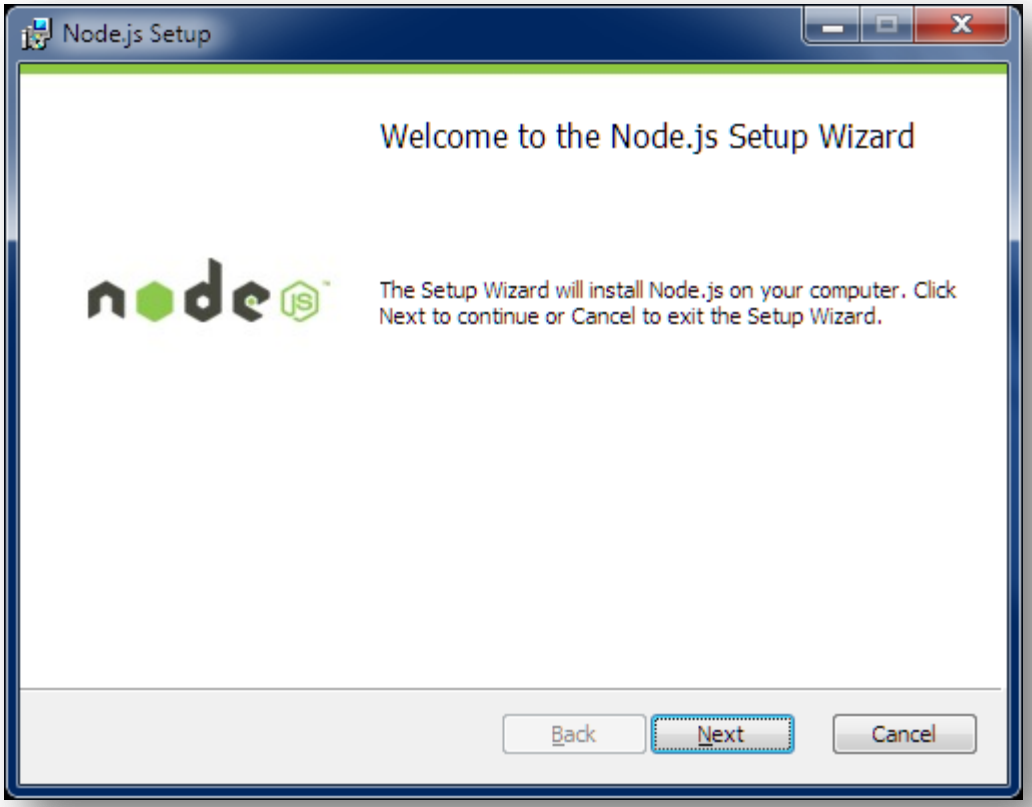
Getting and installing node.js

- Visit http://nodejs.org and click the **INSTALL** button

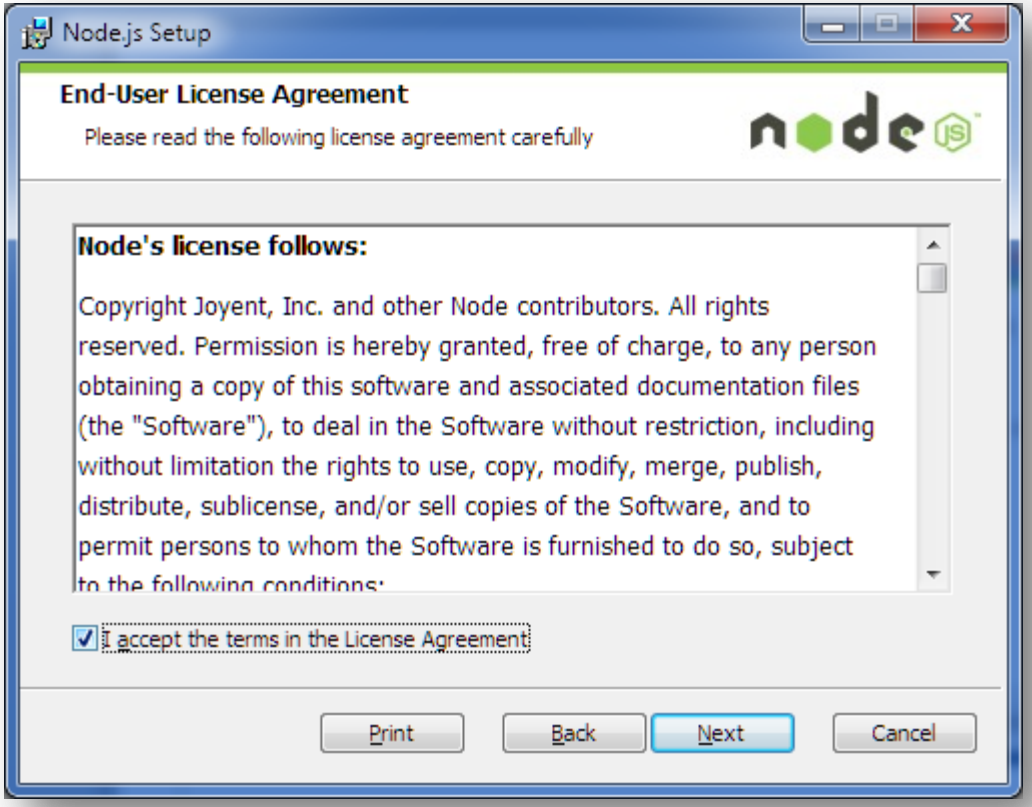
Getting and installing node.js



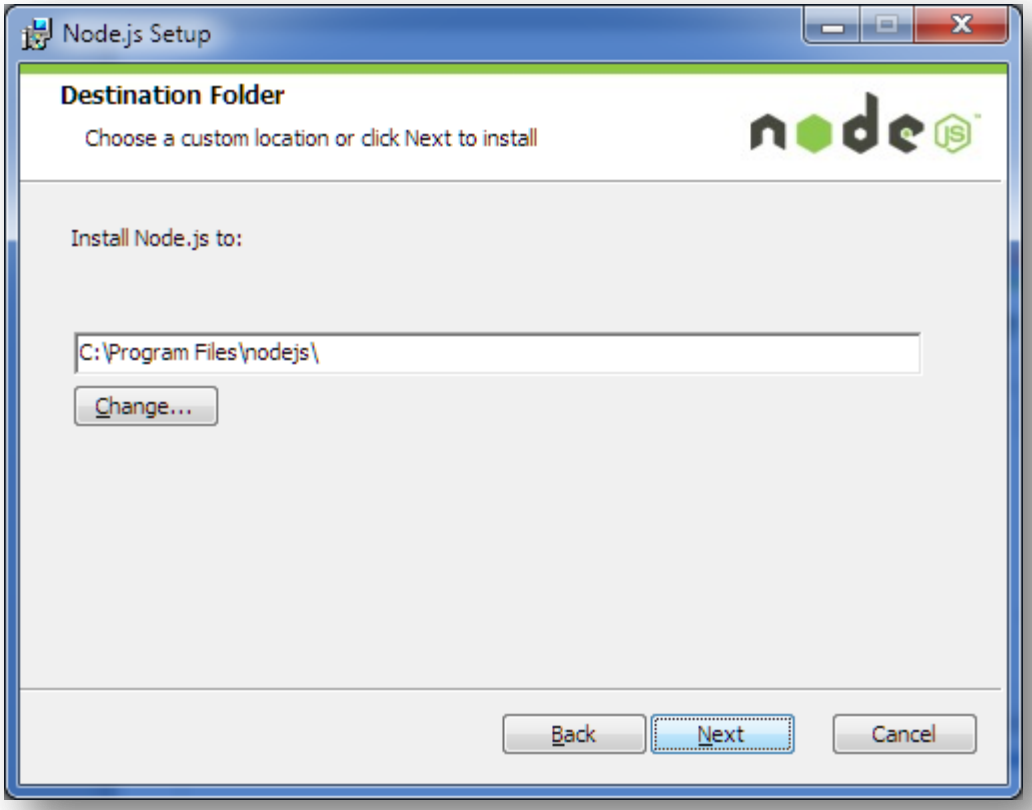
Getting and installing node.js



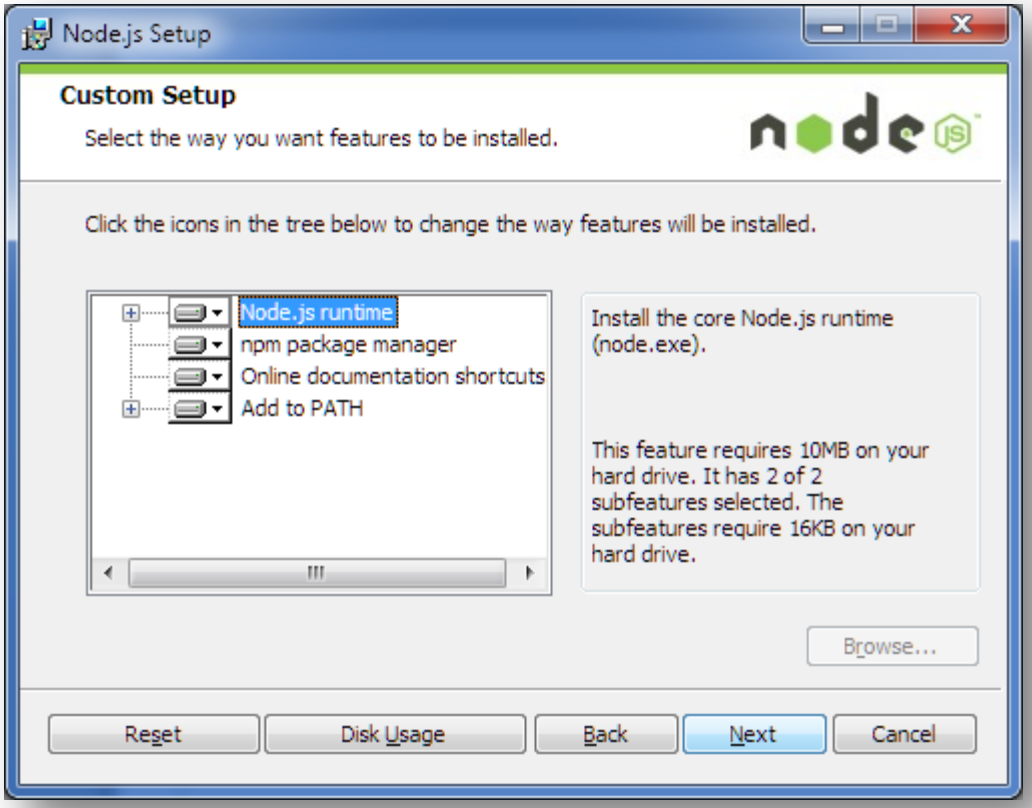
Getting and installing node.js



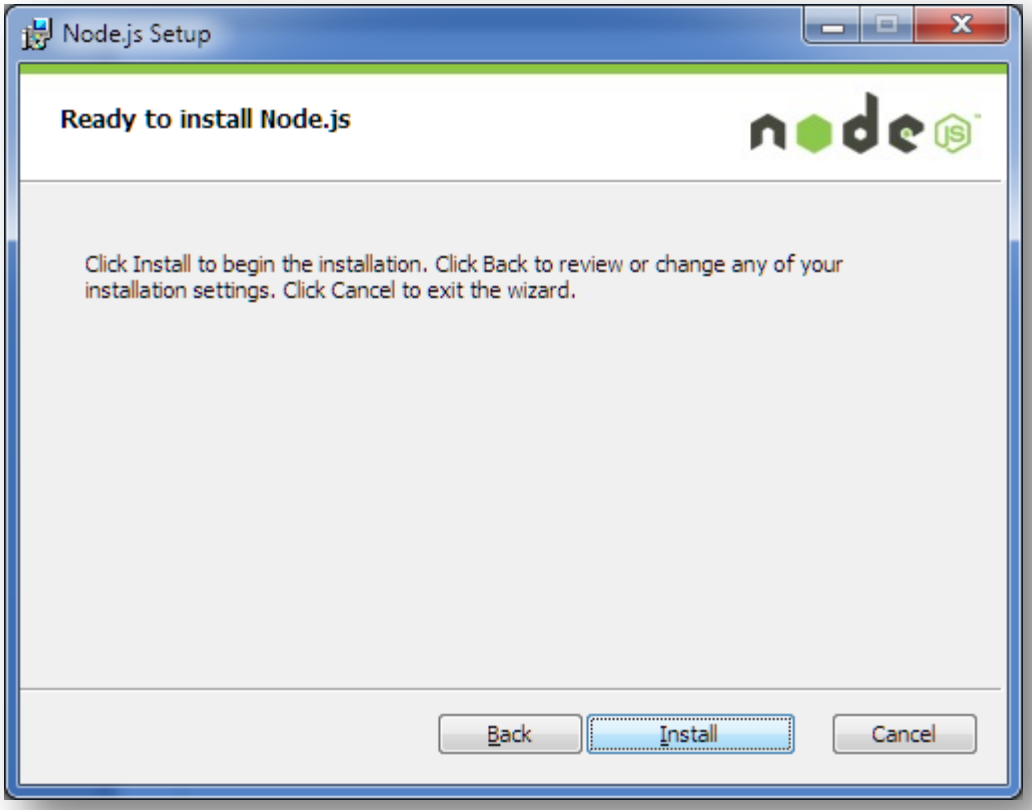
Getting and installing node.js



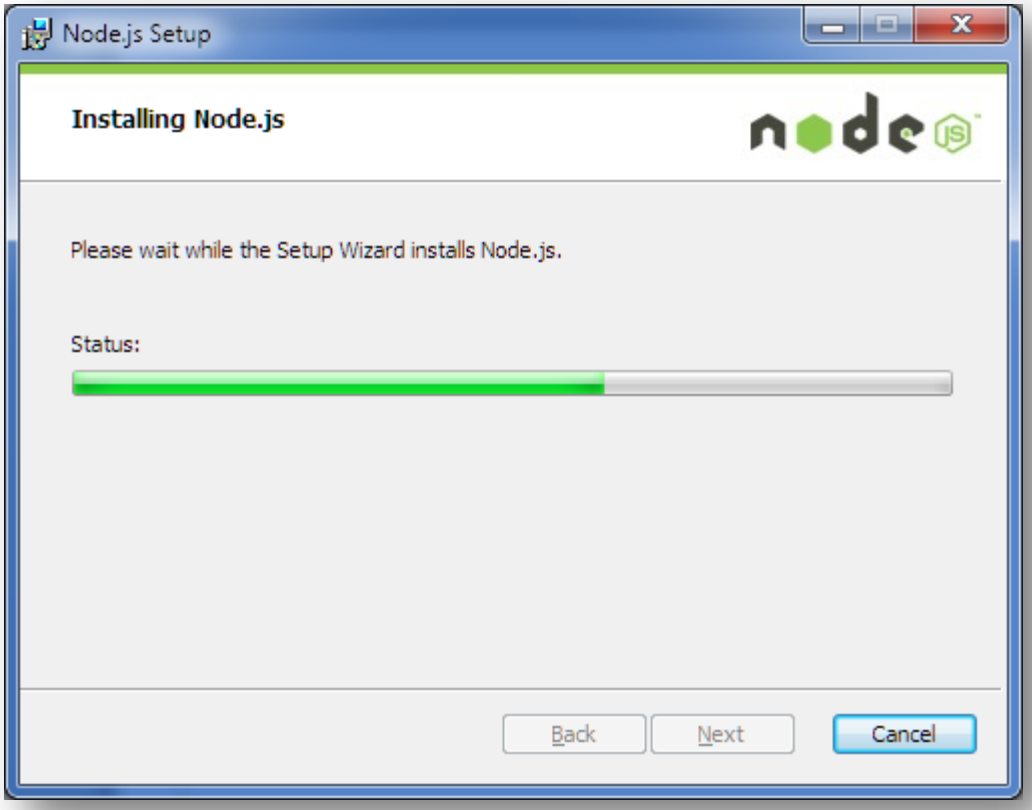
Getting and installing node.js



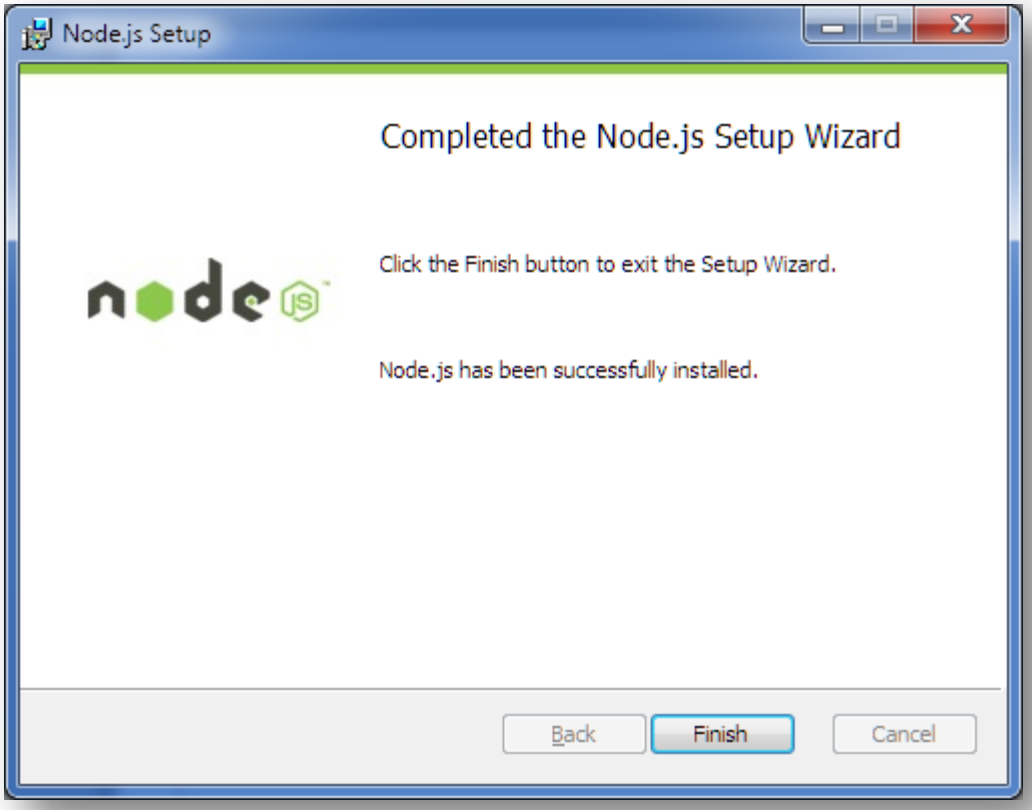
Getting and installing node.js



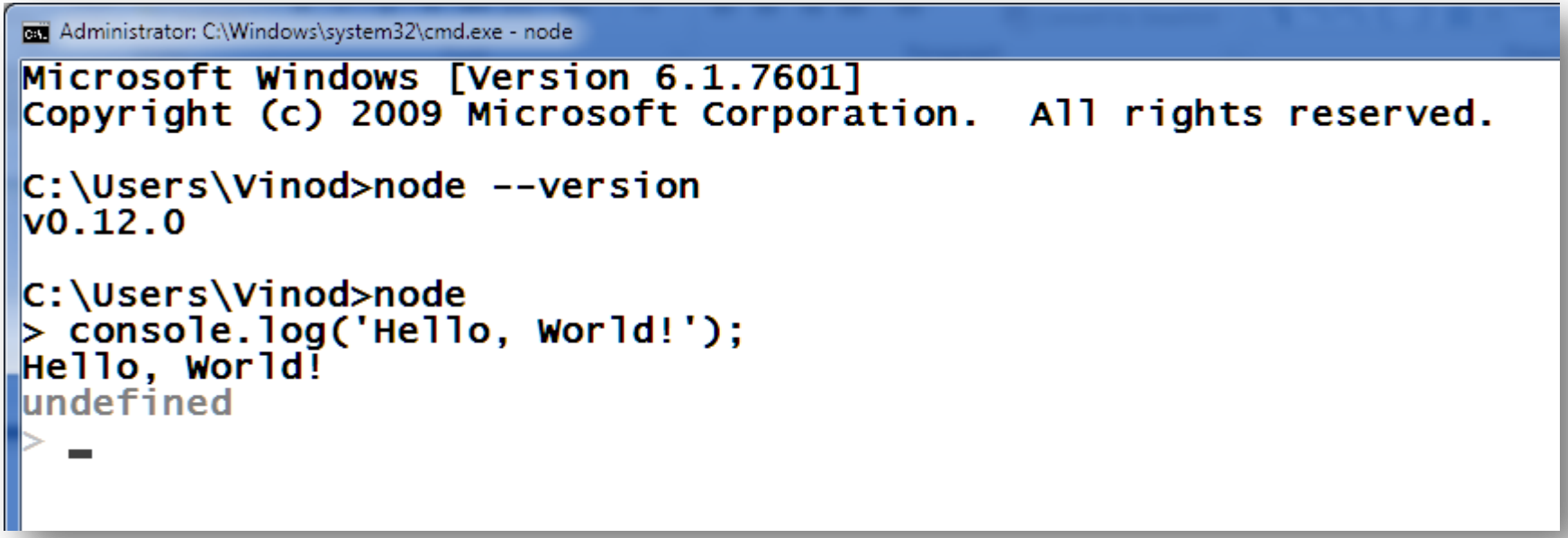
Getting and installing node.js



Getting and installing node.js



Getting and installing node.js



Overview of node.js

- Node adds few **functions** and **objects** typically not avaiable in web browsers - **global**, **require**, **module**, **process**, etc.
  - It also removes few that you may be familiar with in the browsers, such as window, document, location
- The '**console**' object is available in both node and chrome

Modules

- When you are working with a large project, you may want to **break it into smaller** logical pieces
- Node allows to do this using **'modules'**
- A module is just a JavaScript **source file**
- Assign something to **'module.exports'** object in the module
- Use the **'require'** method to obtain the content of a module

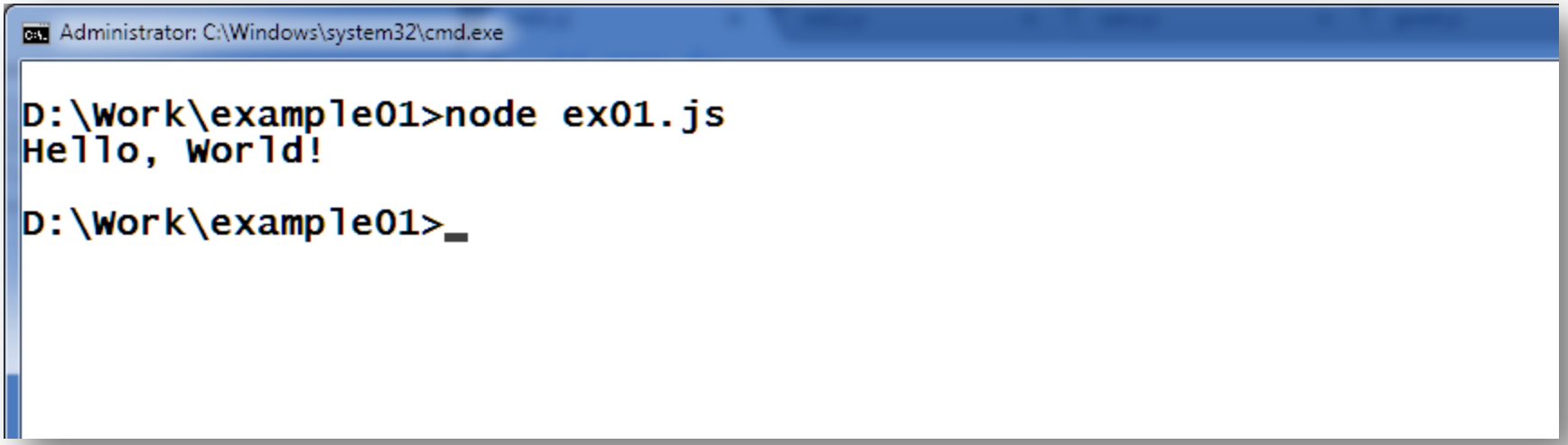
Module representing a function

ex01.js | greet.js

```
1  // greet.js
2
3  module.exports = function(msg){
4      console.log(msg);
5  }
```

ex01.js | greet.js

```
1  // ex01.js
2
3  var greet = require("./greet");
4  greet("Hello, World!");
```



Module representing an object

calci.js ex02.js

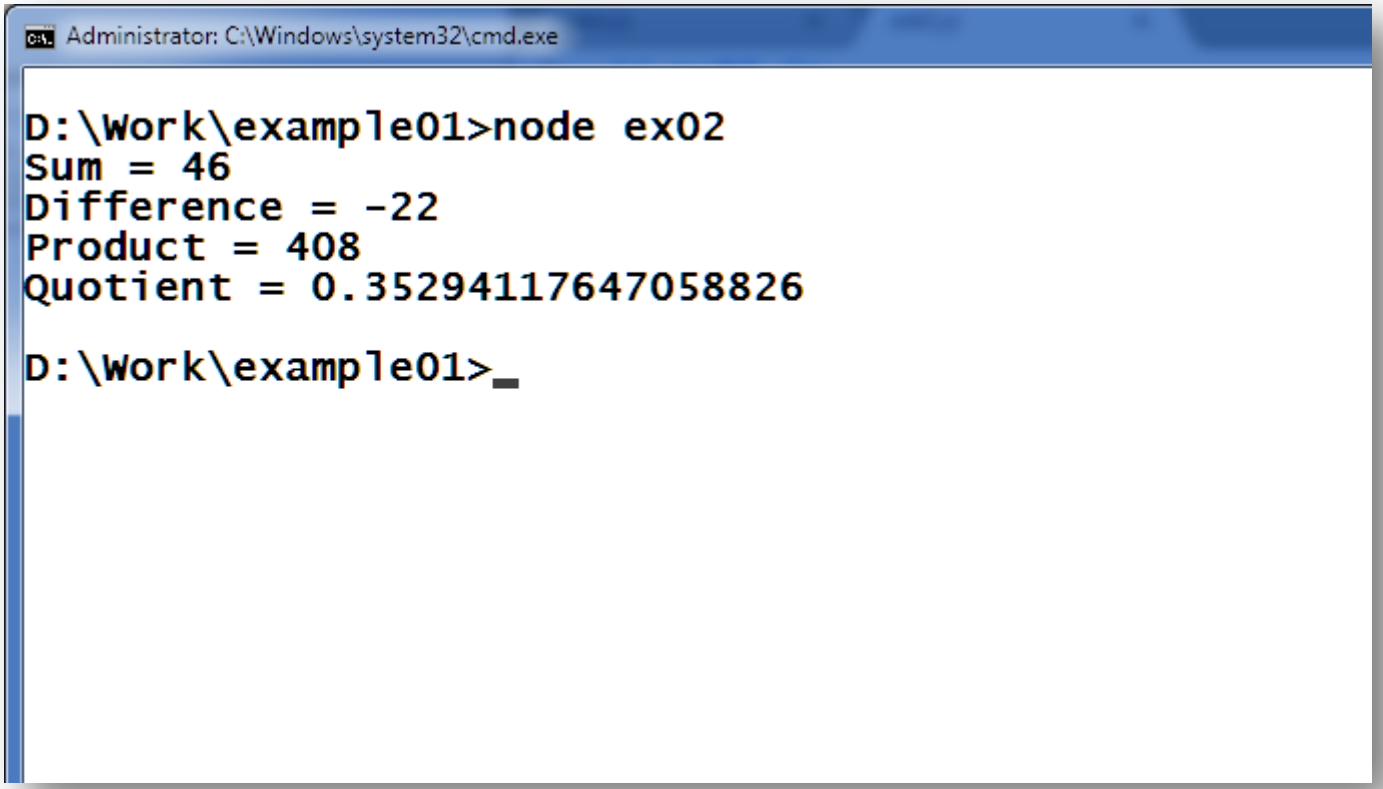
```
1  module.exports.add = function(num1, num2){
2      return num1+num2;
3  }
4  module.exports.subtract = function(num1, num2){
5      return num1-num2;
6  }
7  module.exports.multiply = function(num1, num2){
8      return num1*num2;
9  }
10 module.exports.divide = function(num1, num2){
11     return num1/num2;
12 }
13
```

Using the object from the 'calci' module

calci.js | ex02.js

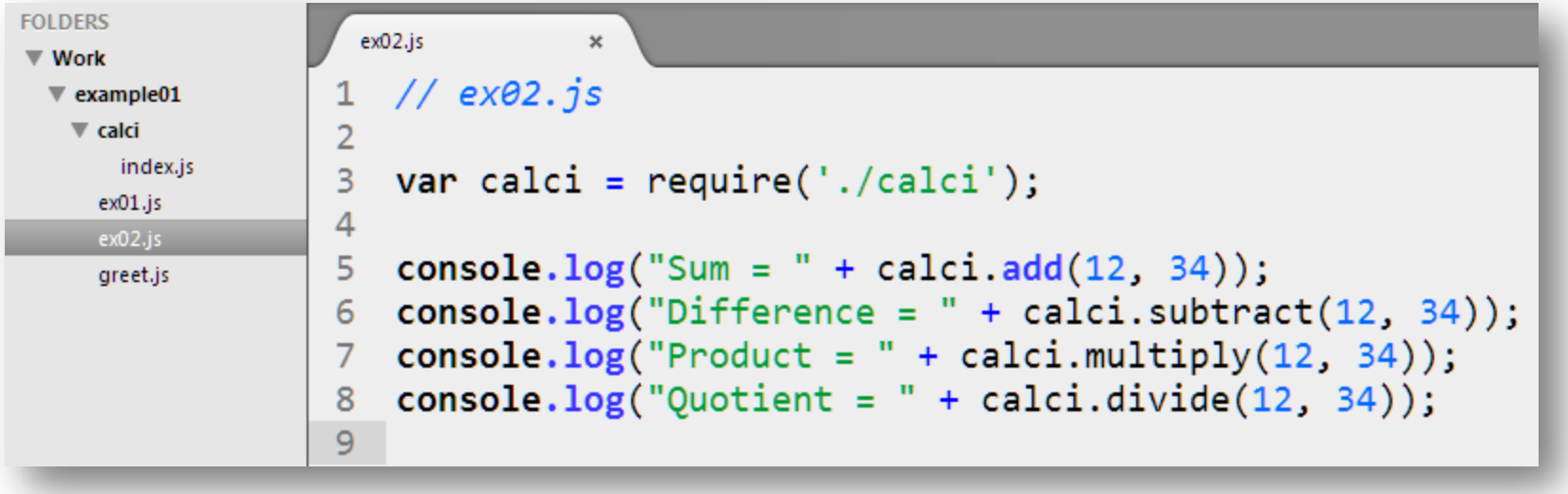
```
1  // ex02.js
2
3  var calci = require('./calci');
4
5  console.log("Sum = " + calci.add(12, 34));
6  console.log("Difference = " + calci.subtract(12, 34));
7  console.log("Product = " + calci.multiply(12, 34));
8  console.log("Quotient = " + calci.divide(12, 34));
9
```

Output


Administrator: C:\Windows\system32\cmd.exe
D:\Work\example01>node ex02
Sum = 46
Difference = -22
Product = 408
Quotient = 0.35294117647058826
D:\Work\example01>

Modules

- A module name also may represent a **folder** containing a source code file **'index.js'**



- Variables declarations inside the module with **'var'** keyword will be treated as a **local** variables for the module

Core modules

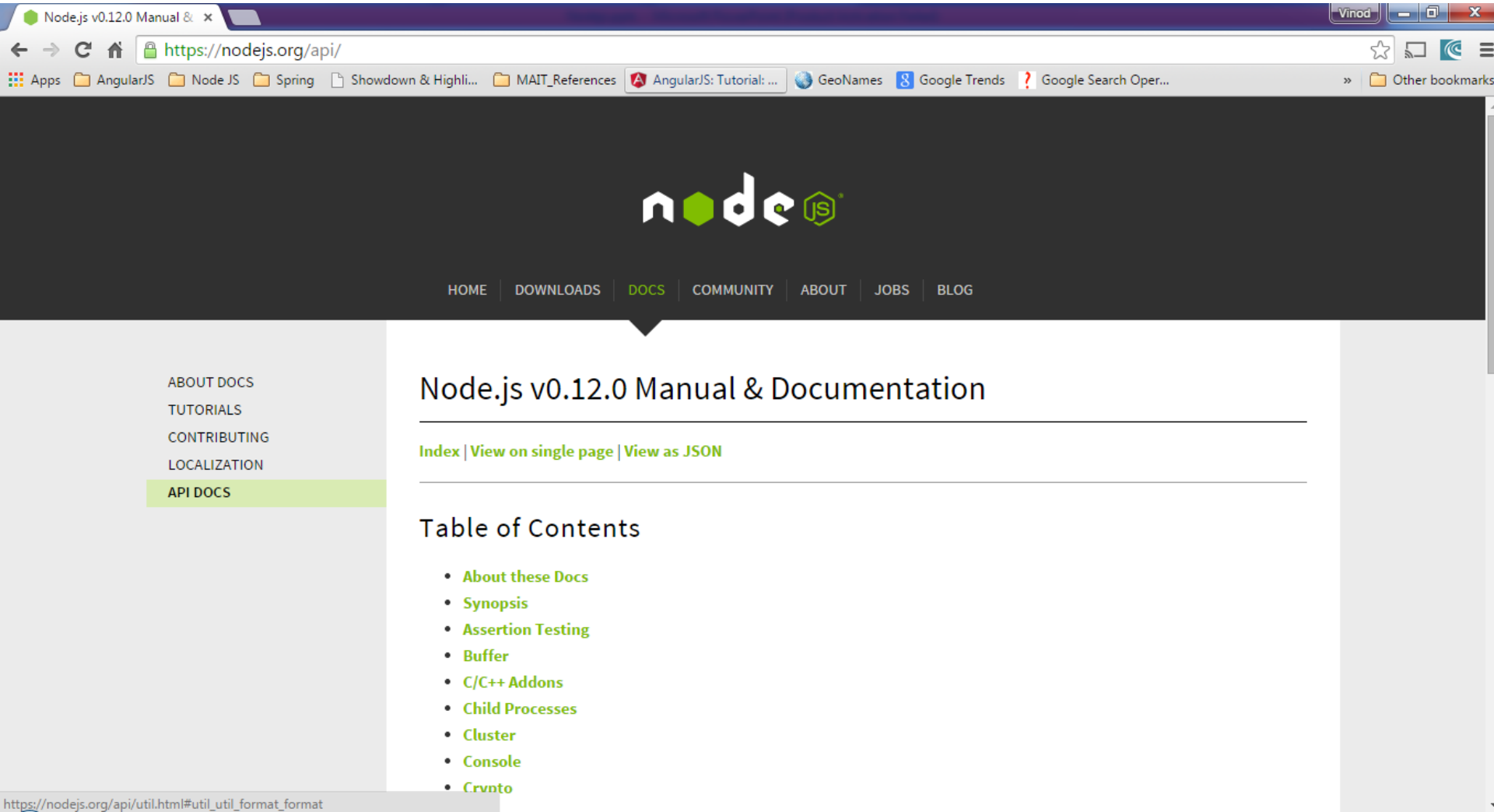
- Node has several modules **compiled** into the binary
- Core modules are always **preferentially loaded** if their identifier is passed to require()
  - For instance, require('http') will always return the built in HTTP module, even if there is a file by that name

Core modules

assert, buffer, child_process, cluster, crypto, dgram, dns, events, **fs**, **http**, https, net, **os**, path, punycode, **querystring**, readline, repl, string_decoder, tls, tty, url, **util**, vm, zlib

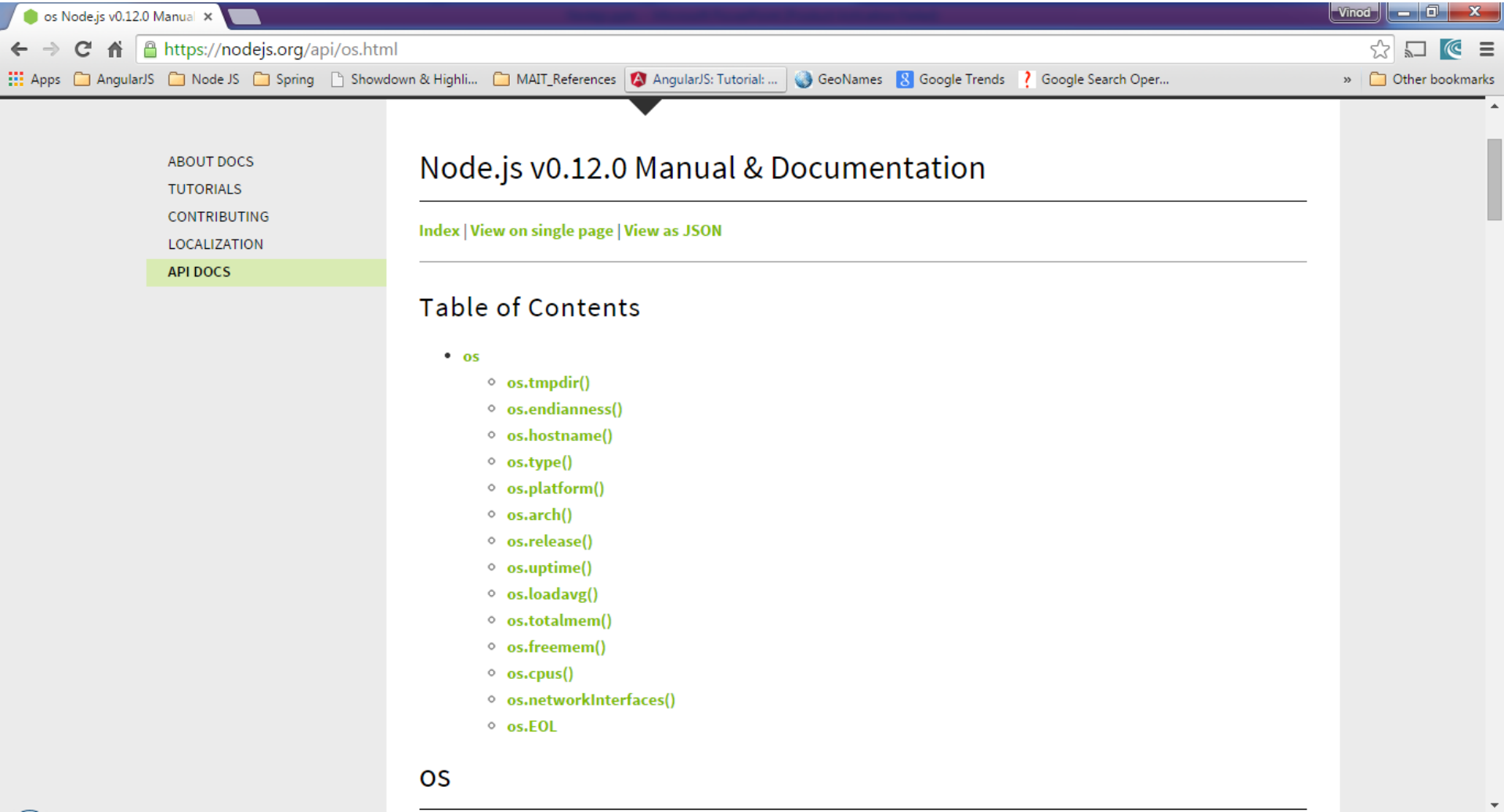
Online docs for core modules

- Visit https://nodejs.org/api/



Online docs for core modules

- The 'os' module



Using the core modules

ex03.js

```
1  var os = require("os");
2  var util = require("util");
3
4  console.log("os.type() = " + os.type());
5  console.log("os.uptime() = " + os.uptime()/60 + " minutes");
6
7  var name = "Vinod";
8  var age = 41;
9  var message = util.format("My name is %s and I am %d years old.", name, age);
10 console.log(message);
```

Administrator: C:\Windows\system32\cmd.exe

```
D:\Work\example01>node ex03
os.type() = Windows_NT
os.uptime() = 243.76922964833332 minutes
My name is Vinod and I am 41 years old.

D:\Work\example01>
```

Event loop

- Event-loops are the **core** of event-driven programming,
- Almost all the **UI programs** use event-loops to track the user event
  - For example, Button clicks, Ajax Requests etc.
- When a node program **starts**, the event-loop starts along with it **automatically**

Event loop

- In contrast to many other environments, all inputs and outputs in Node are **non-blocking**
  - For example, if you are making a call to a database, in many other environments,
    - that call is going to block
    - wait for the results to come back from the database
    - stop your program until the database responds
    - resume the program once those results are ready
- Node, **keeps running** the program

Event loop

- **Traditional** I/O

```
var result = db.query("select x from table_Y");
doSomethingWith(result); //wait for result!
doSomethingElse(); //execution is blocked!
```

- Non-traditional, **Non-blocking** I/O

```
db.query("select x from table_Y", function (result){
      doSomethingWith(result); //wait for result!
});
doSomethingElse(); // executes without blocking
```

Event loop

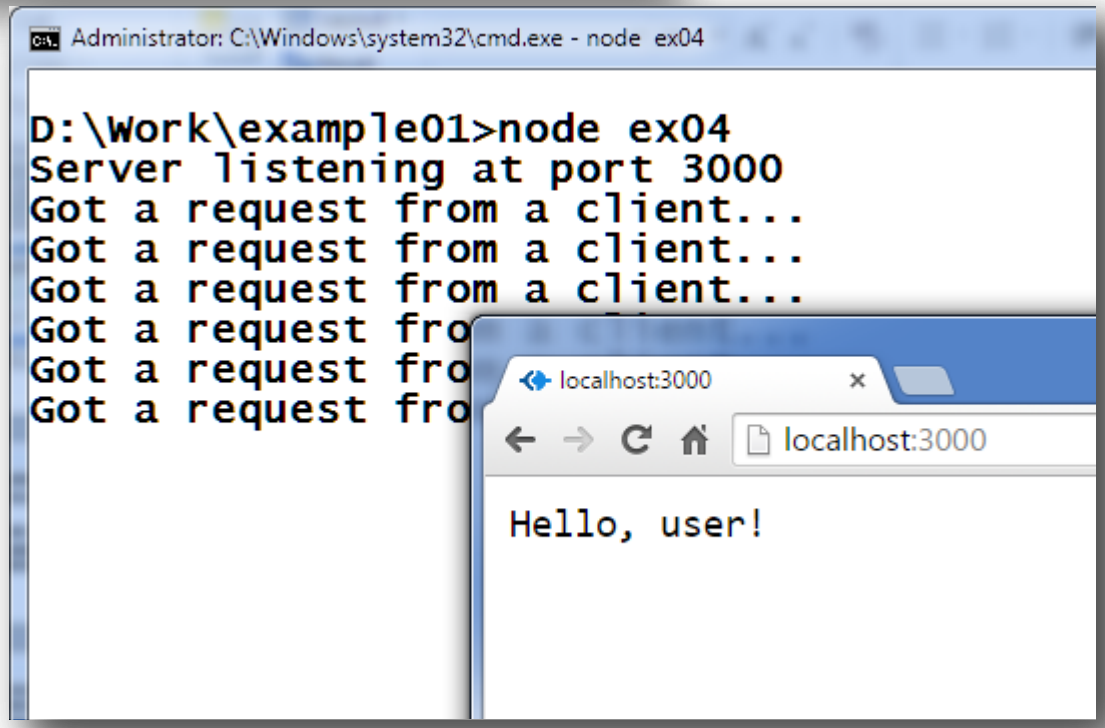
ex04.js

```
1  var http = require("http");
2  var server = http.createServer(function(req, resp){
3      console.log("Got a request from a client...");
4      resp.end("Hello, user!");
5  });
6
7  server.listen(3000);
8  console.log("Server listening at port 3000");
```

The server.listen(..) registers itself with the event loop in a non-blocking fashion.

Line 8 continues to execute as soon as line 7 is called

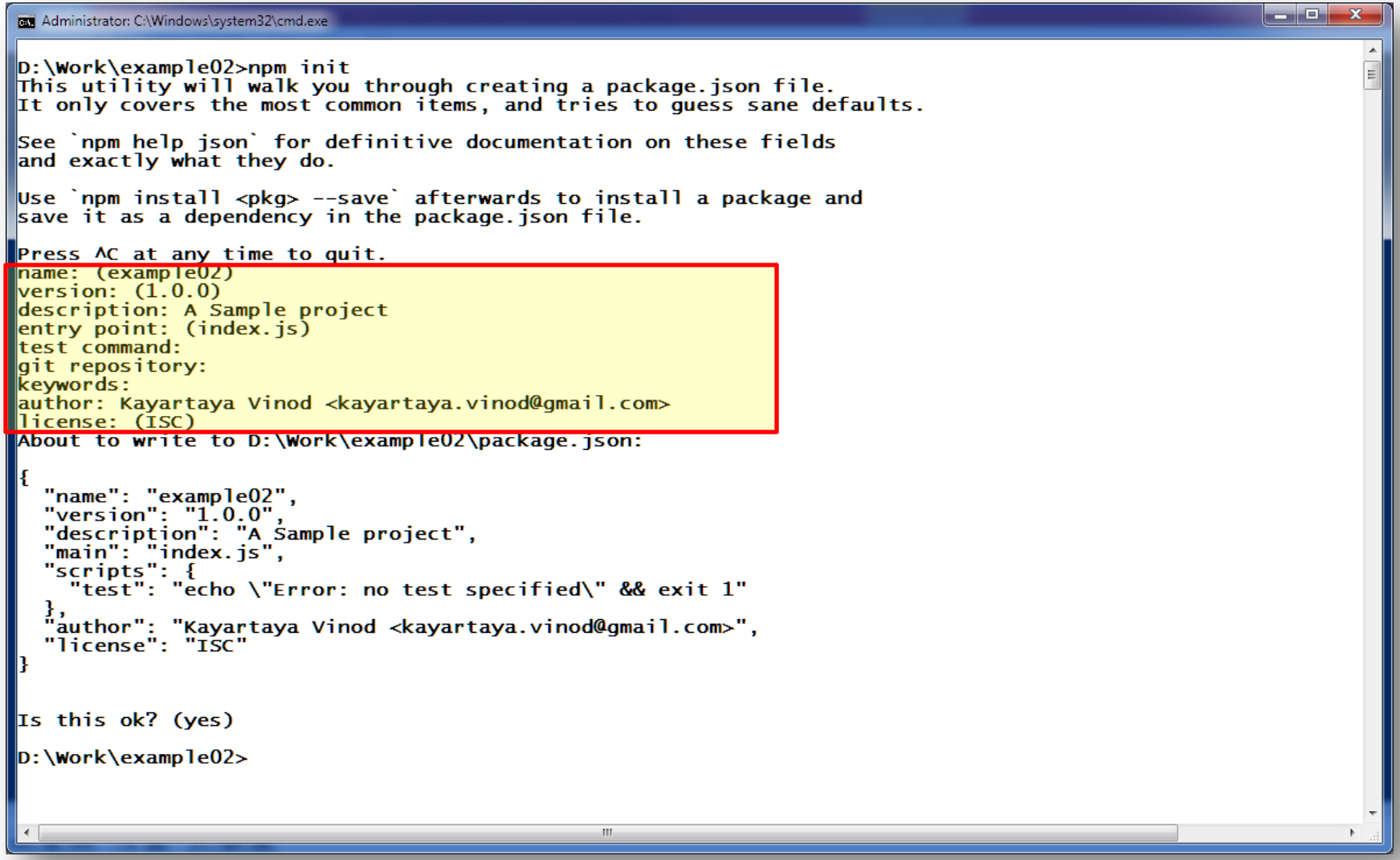
Each time a client makes a request, the callback function supplied to the server (line 2) is executed



npm

- npm is a **package manger** for JavaScript
- npm is **distributed** with node
- Helps in project configuration, **dependency management**, module installation and many other things
- To start **creating** a JavaScript (node) project, use the command
  - npm init

npm init


Administrator: C:\Windows\system32\cmd.exe
D:\Work\example02>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sane defaults.
See `npm help json` for definitive documentation on these fields
and exactly what they do.
Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.
Press ^C at any time to quit.
name: (example02)
version: (1.0.0)
description: A Sample project
entry point: (index.js)
test command:
git repository:
keywords:
author: Kayartaya Vinod <kayartaya.vinod@gmail.com>
license: (ISC)
About to write to D:\Work\example02\package.json:
{
"name": "example02",
"version": "1.0.0",
"description": "A Sample project",
"main": "index.js",
"scripts": {
"test": "echo \"Error: no test specified\" && exit 1"
},
"author": "Kayartaya Vinod <kayartaya.vinod@gmail.com>",
"license": "ISC"
}
Is this ok? (yes)
D:\Work\example02>

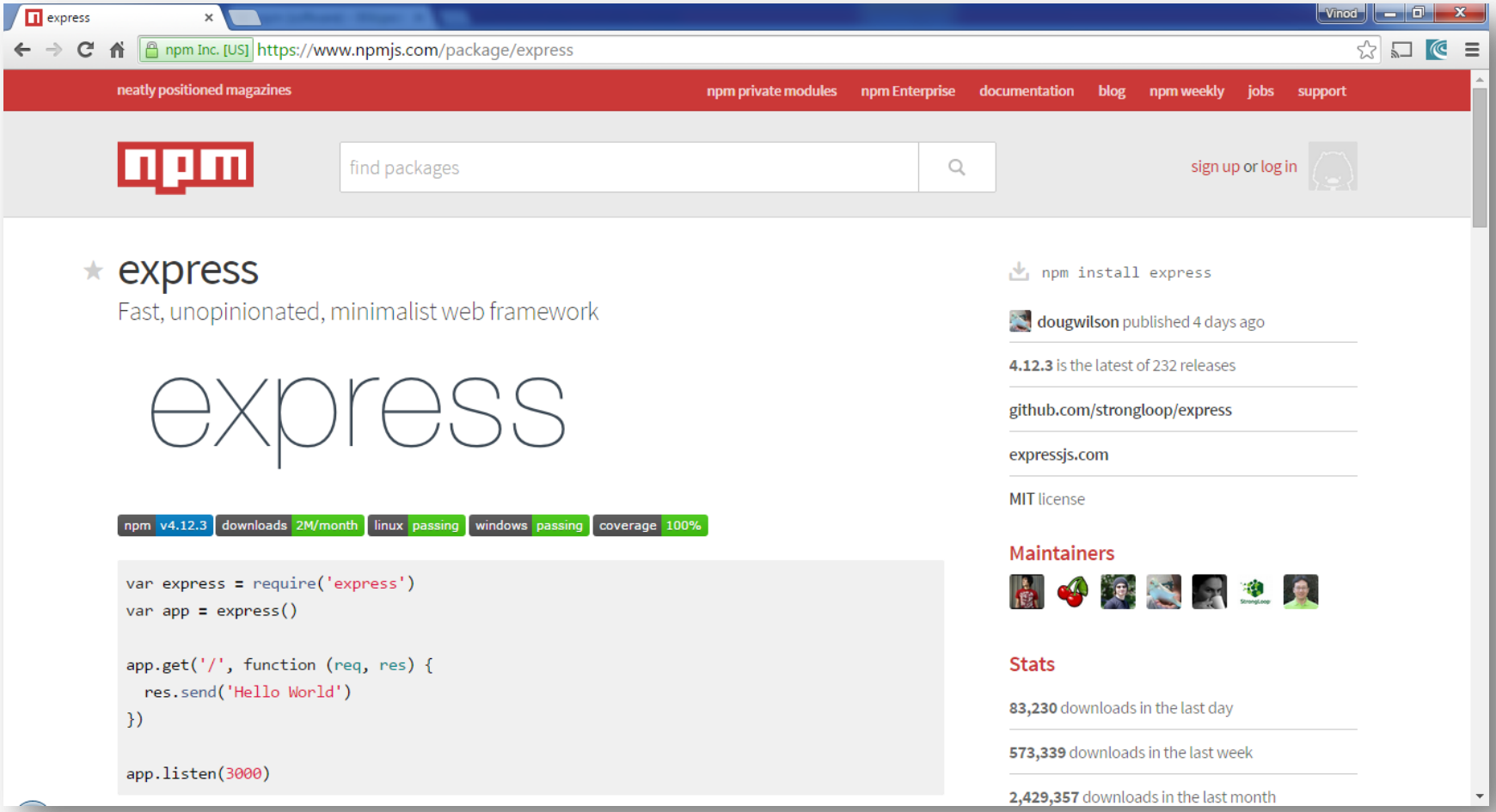
package.json

```
{
  "name": "example02",
  "version": "1.0.0",
  "description": "A Sample project",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Kayartaya Vinod <kayartaya.vinod@gmail.com>",
  "license": "ISC"
}
```

npm

- The **node community** provides thousands of modules apart from what is already available in the core module collection
- Available at http://npmjs.org

npm - finding a module



npm

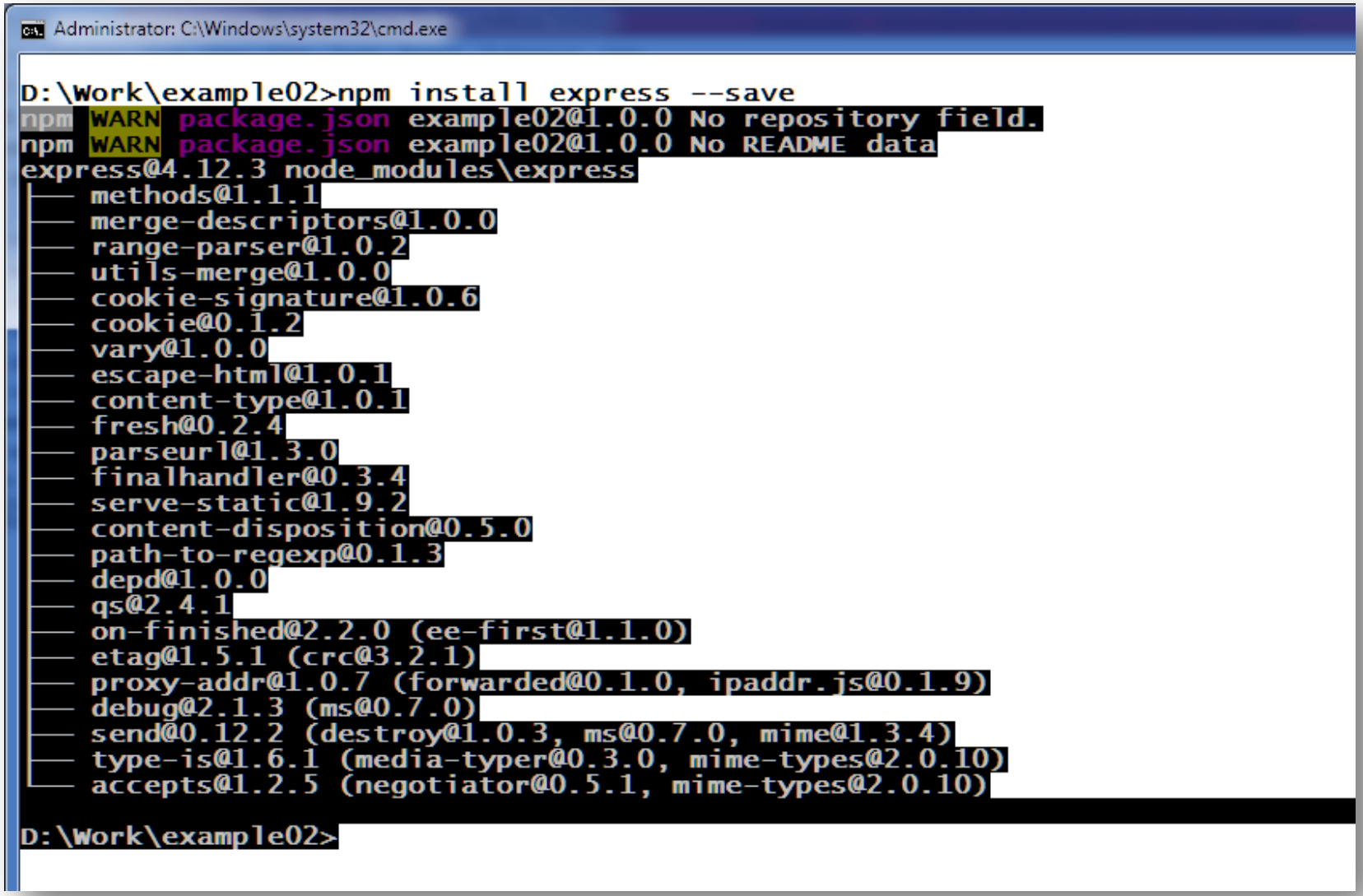
- To install a module to your project,

  use the **'npm install'** command

- Use the **'--save'** option to save the dependency

  information in the **package.json** file

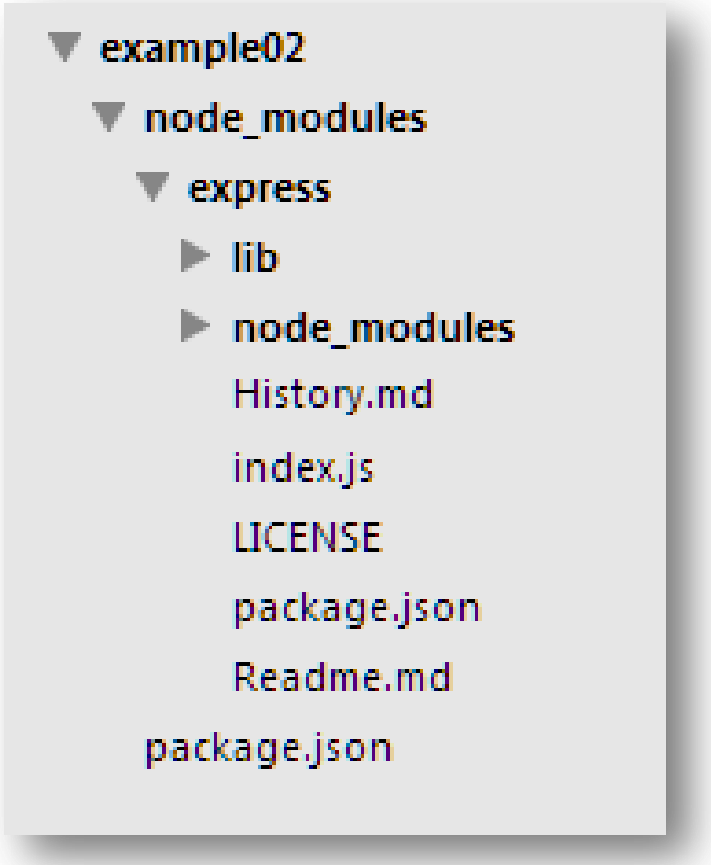
npm


Administrator: C:\Windows\system32\cmd.exe
D:\Work\example02>npm install express --save
npm WARN package.json example02@1.0.0 No repository field.
npm WARN package.json example02@1.0.0 No README data
express@4.12.3 node_modules\express
├── methods@1.1.1
├── merge-descriptors@1.0.0
├── range-parser@1.0.2
├── utils-merge@1.0.0
├── cookie-signature@1.0.6
├── cookie@0.1.2
├── vary@1.0.0
├── escape-html@1.0.1
├── content-type@1.0.1
├── fresh@0.2.4
├── parseurl@1.3.0
├── finalhandler@0.3.4
├── serve-static@1.9.2
├── content-disposition@0.5.0
├── path-to-regexp@0.1.3
├── depd@1.0.0
├── qs@2.4.1
├── on-finished@2.2.0 (ee-first@1.1.0)
├── etag@1.5.1 (crc@3.2.1)
├── proxy-addr@1.0.7 (forwarded@0.1.0, ipaddr.js@0.1.9)
├── debug@2.1.3 (ms@0.7.0)
├── send@0.12.2 (destroy@1.0.3, ms@0.7.0, mime@1.3.4)
├── type-is@1.6.1 (media-typer@0.3.0, mime-types@2.0.10)
└── accepts@1.2.5 (negotiator@0.5.1, mime-types@2.0.10)
D:\Work\example02>

Updated package.json

```
{
  "name": "example02",
  "version": "1.0.0",
  "description": "A Sample project",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Kayartaya Vinod <kayartaya.vinod@gmail.com>",
  "license": "ISC",
  "dependencies": {
    "express": "^4.12.3"
  }
}
```

Updated folder structure



npm

- npm is capable of **installing**, **starting** and **maintaining** node projects
- To install a **global** module, use the **-g** switch
  - npm install -g express
- To start the application, use the option **'start'**
  - npm start
  - This will look for the value of "scripts.start" in the package.json file

Updated package.json

npm

- To update the project, use the **'update'** option
  - npm update
  - Downloads and adds the dependencies and removes the one that are not applicable, based on the information in package.json

Using ‘express’

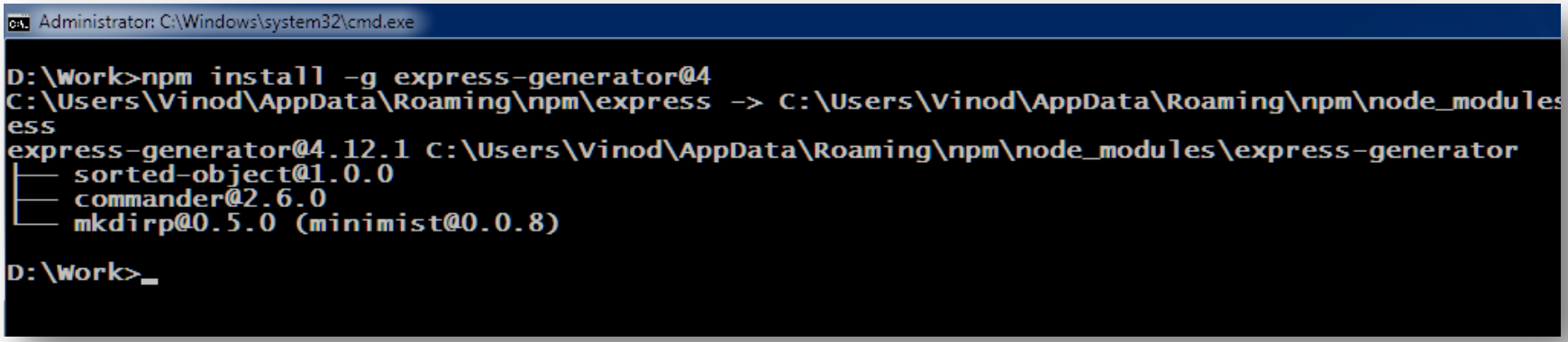
- Express is a minimal and flexible Node.js **web application framework** that provides a robust set of features for web and mobile applications
  - Robust routing
  - Focus on high performance
  - Super-high test coverage
  - HTTP helpers (redirection, caching, etc)
  - View system supporting 14+ template engines
  - Content negotiation
  - Executable for generating applications quickly

Using ‘express’

- **Install** the executable:
  - npm install -g express-generator@4
- **Create** the app:
  - express /tmp/foo && cd /tmp/foo
- Install **dependencies**:
  - npm install
- **Start** the server:
  - npm start

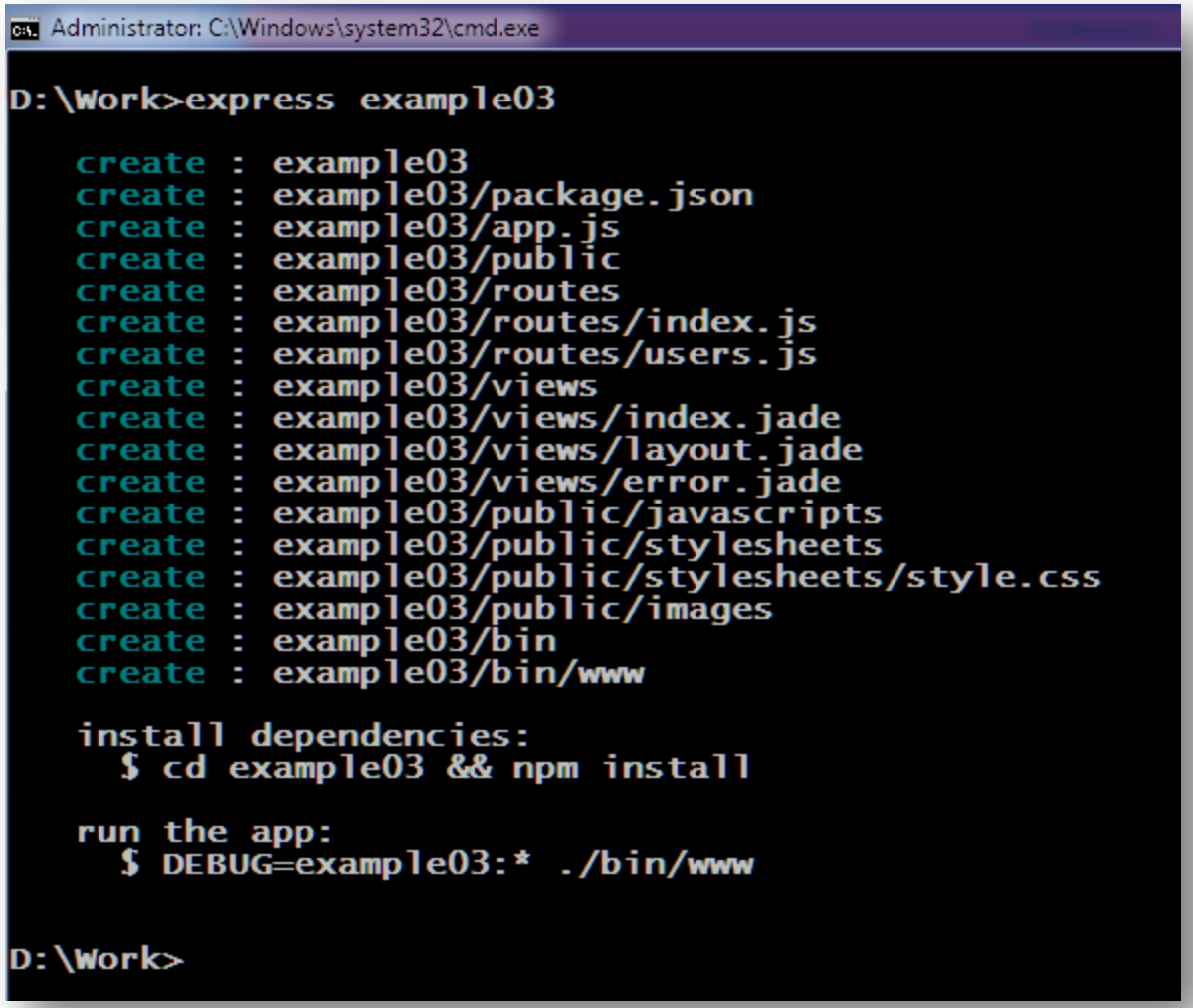
Install the executable

npm install -g express-generator@4



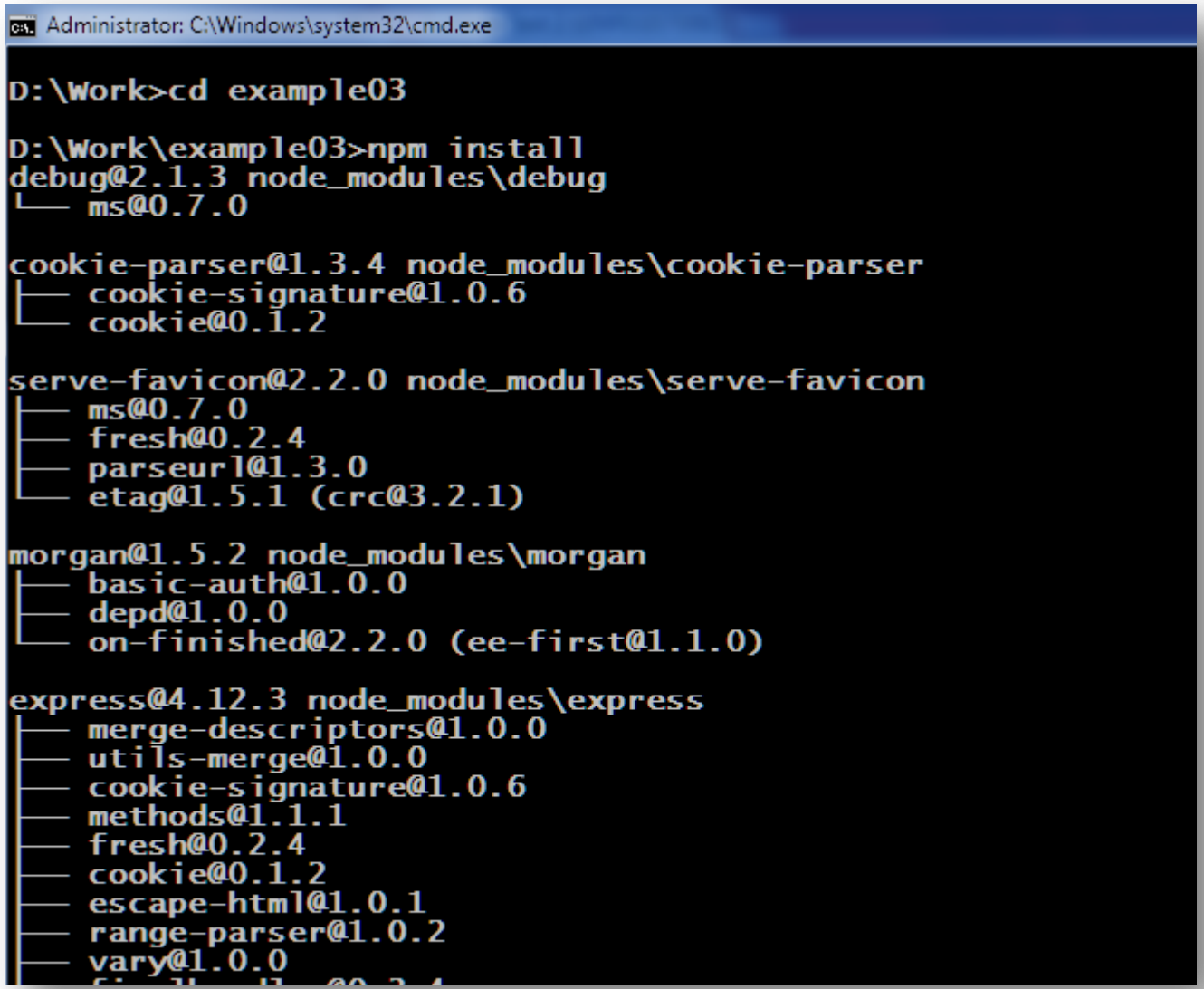
Create the app

express example03



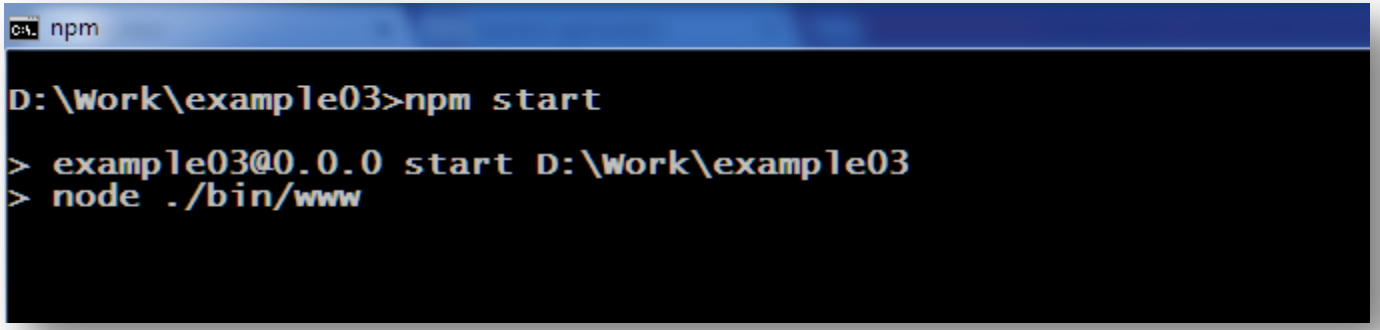
Install dependencies

npm install

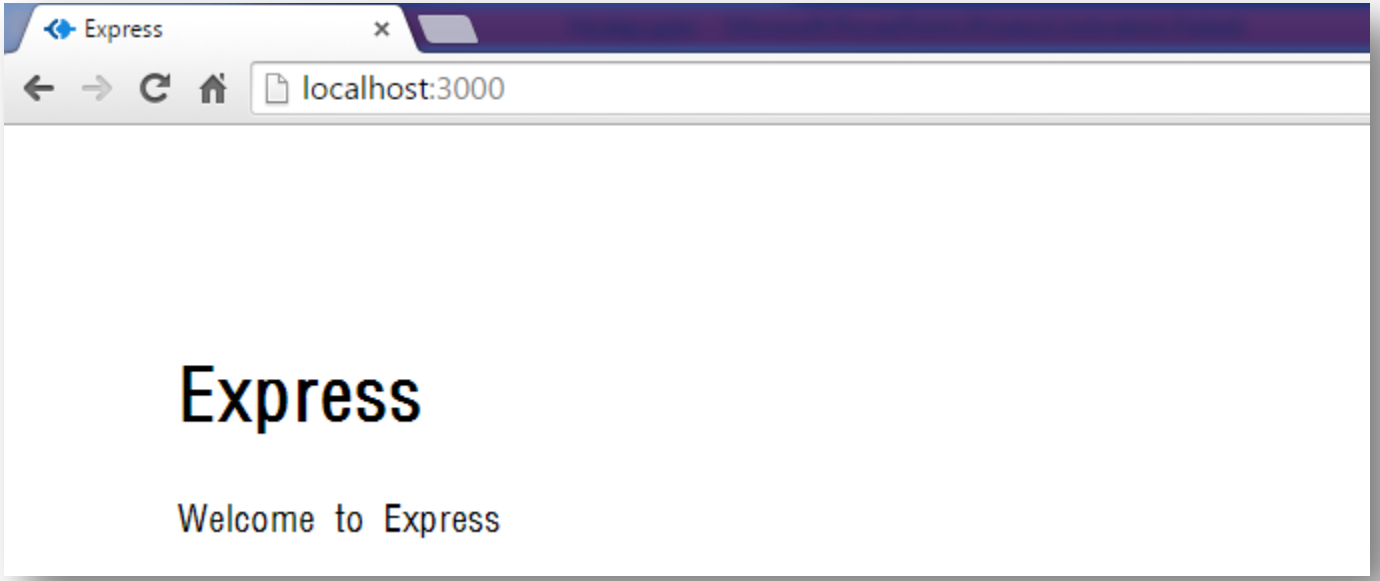


Start the server

npm start



Open http://localhost:3000 in a browser



Using the 'express' module

- The 'express' module is **built on top** of the http module
  - There is no need to use both 'express' and 'http'
- After including the 'express' module in your application, use it **as a function** to instantiate it:

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

var app = express();
```

Using the 'express' module

- The 'app' can be configured with **routing** information, and to **register** with the event loop, call the **'listen'** method

```
app.get("/", function(req, resp){
    resp.end("Hello, World!");
});
app.listen(3000);
```

Routing with express

- **Routing** refers to determining how an application responds to a client request to a particular **endpoint**
  - Endpoint is a URI (or path) and a specific HTTP request method (GET, POST, and so on)
- Each route can have one or more **handler** functions, which is / are executed when the route is matched

Routing with express

- Route definition takes the following structure

  app.METHOD(PATH, HANDLER), where

  - app is an instance of express,
  - METHOD is an HTTP request method,
  - PATH is a path on the server, and
  - HANDLER is the function executed when the route is matched.

Routing with express

- Duplicate mappings will be **ignored**
- For example,

```
app.get("/hello", function(req, resp){
        resp.send("Welcome, Friends!");
});
app.get("/hello", function(req, resp){
        resp.send("Hello, Friends!");
});
```

Call to http://localhost:3000/hello would get response as **"Welcome, Friends!"**

Creating a REST api

- Express supports the following routing methods corresponding to **HTTP methods**:
  - **get**, **post**, **put**, head, **delete**, options, trace, copy, lock, mkcol, move, purge, propfind, proppatch, unlock, report, mkactivity, checkout, merge, m-search, notify, subscribe, unsubscribe, patch, search, and connect

Creating a REST api

- The HTTP methods GET, POST, PUT, DELETE correspond to **server side actions** for getting, adding, updating and deleting a resource.
  - A resource could be a product, employee, user, etc.
- For example, a URI http://localhost:3000/users should return **a list of all users** when requested using HTTP GET method and the same thing should **add a new user** to the repository when requested using the POST method.
  - The data to be added is sent as part of the HTTP request payload

Creating a REST api

```
 6  app.get("/users", function(req, resp){
 7      resp.send("List of users will be sent.");
 8  });
 9  app.get("/users/:id", function(req, resp){
10      resp.send("Users for id "
11          + req.params.id + " will be sent.");
12  });
13  app.post("/users", function(req, resp){
14      resp.send("A user will be added to the db");
15  });
16  app.put("/users/:id", function(req, resp){
17      resp.send("The user with id "
18          + req.params.id + " will be updated");
19  });
20  app.delete("/users/:id", function(req, resp){
21      resp.send("The user with id "
22          + req.params.id + " will be deleted");
23  });
```

Creating a REST api

- Chainable route handlers for a route path can be created using **app.route()**

```
4  app.route("/users")
5      .get(function(req, resp){
6          resp.send("List of users will be sent.");
7      })
8      .post(function(req, resp){
9          resp.send("A user will be added to the db");
10     });
11
```

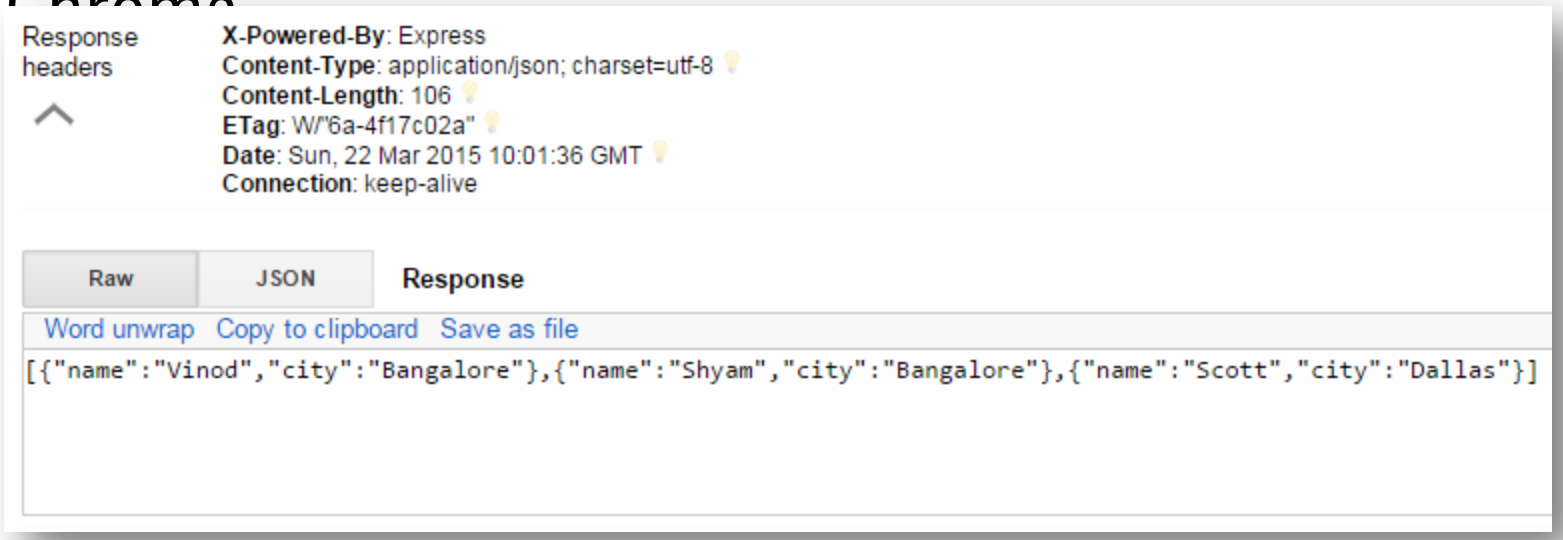
Creating a REST api

- To send **JSON** data for the GET methods, use the method `resp.json(obj)`

```
4  app.get("/users", function(req, resp){
5
6      var users = [
7          {name: "Vinod", city: "Bangalore"},
8          {name: "Shyam", city: "Bangalore"},
9          {name: "Scott", city: "Dallas"}
10     ];
11
12     resp.json(users);
13 });
```

Creating a REST api

- To check the REST api, use a tool like **Advanced Rest Client**, a very popular extension for Google Chrome

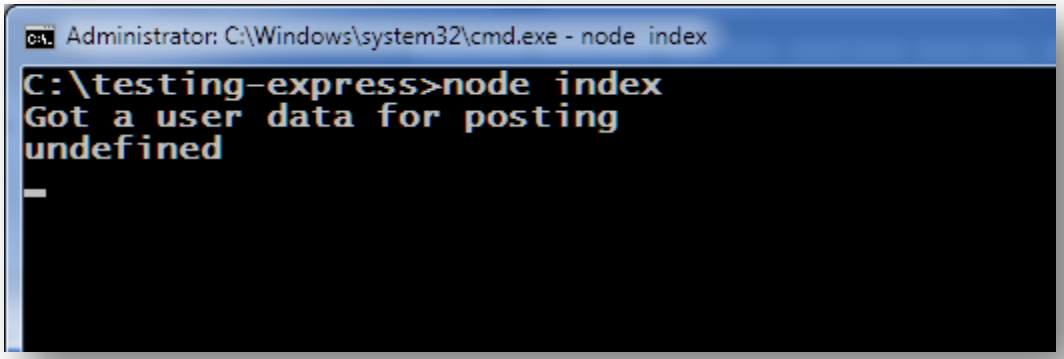


Creating a REST api

- To receive the data sent via **POST** or **PUT** methods, use the `req.body` property
- Contains **key-value** pairs of data submitted in the request body
- By default, it is **undefined**, and is populated when you use body-parsing middleware such as **body-parser**

Creating a REST api

```
1  var express = require("express");
2  var app = express();
3
4  app.post("/users", function(req, resp){
5      var user = req.body;
6      console.log("Got a user data for posting")
7      console.dir(user);
8      resp.send("User data posted successfully!");
9  });
10
11 app.listen(3000);
```



Creating a REST api

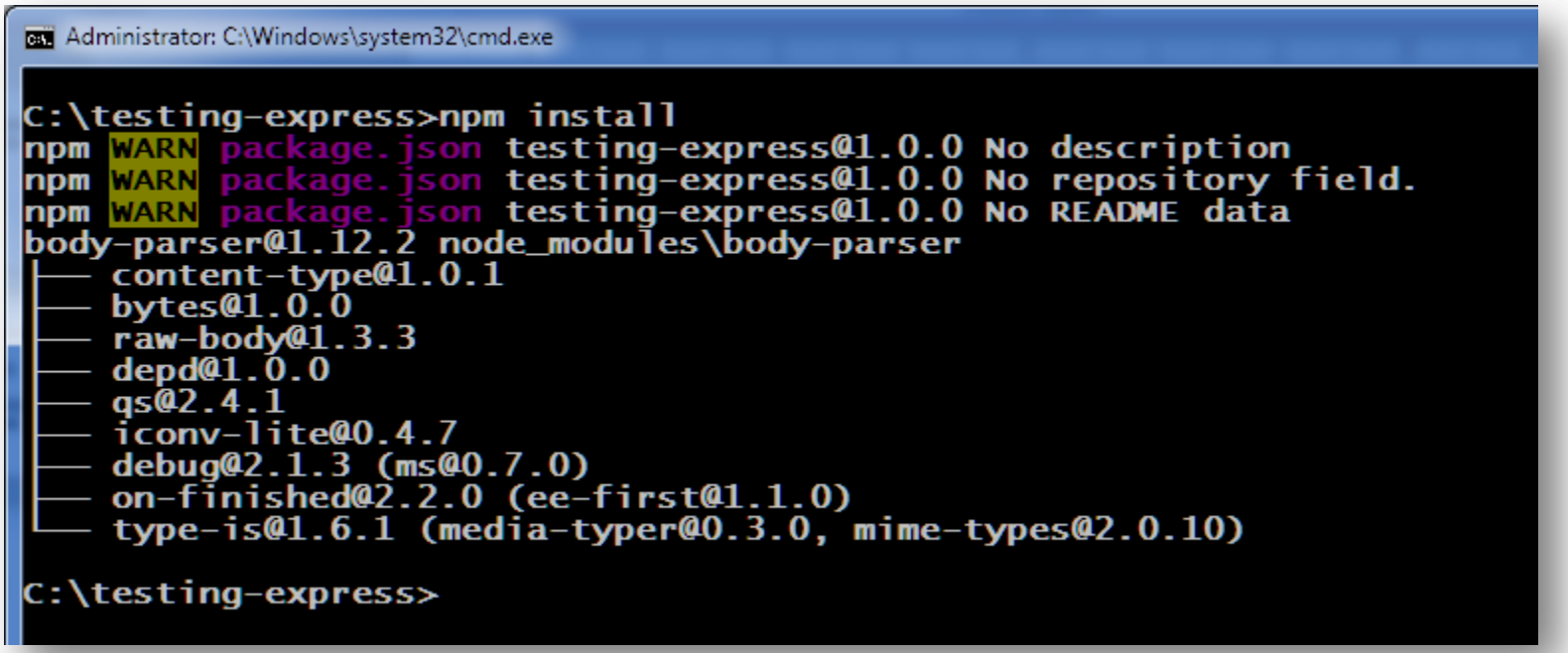
- Add the **dependency** in the package.json

```
{
  "name": "testing-express",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Vinod <kayartaya.vinod@gmail.com>",
  "license": "ISC",
  "dependencies": {
    "express": "^4.12.3",
    "body-parser": "*"
  }
}

```

Creating a REST api

- **Update** the project

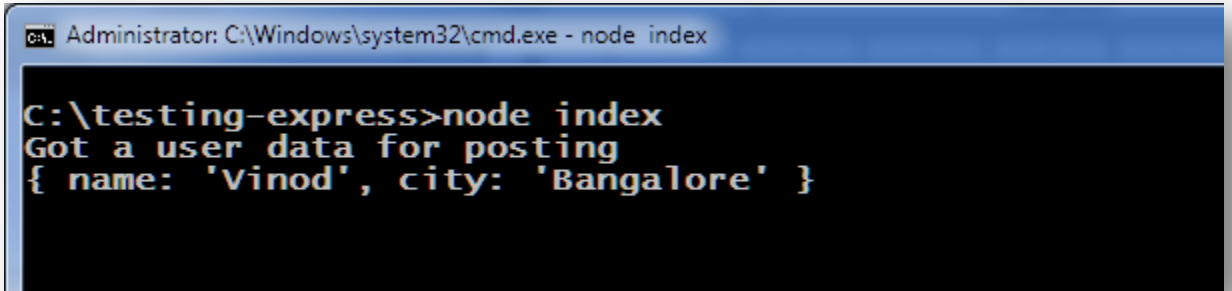


Creating a REST api

- **Modified** version of index.js

```
1  var express = require("express");
2  var bodyParser = require('body-parser');
3
4  var app = express();
5  app.use(bodyParser.json());
6
7  app.post("/users", function(req, resp){
8      var user = req.body;
9      console.log("Got a user data for posting")
10     console.dir(user);
11     resp.send("User data posted successfully!");
12 });
13
14 app.listen(3000);
```

- And the result of **POST request**



Serving static content

- Configure the “app” to use the static content located in a folder using the express.static() method

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

- The file “/public/index.html” will be sent automatically for the “/” request

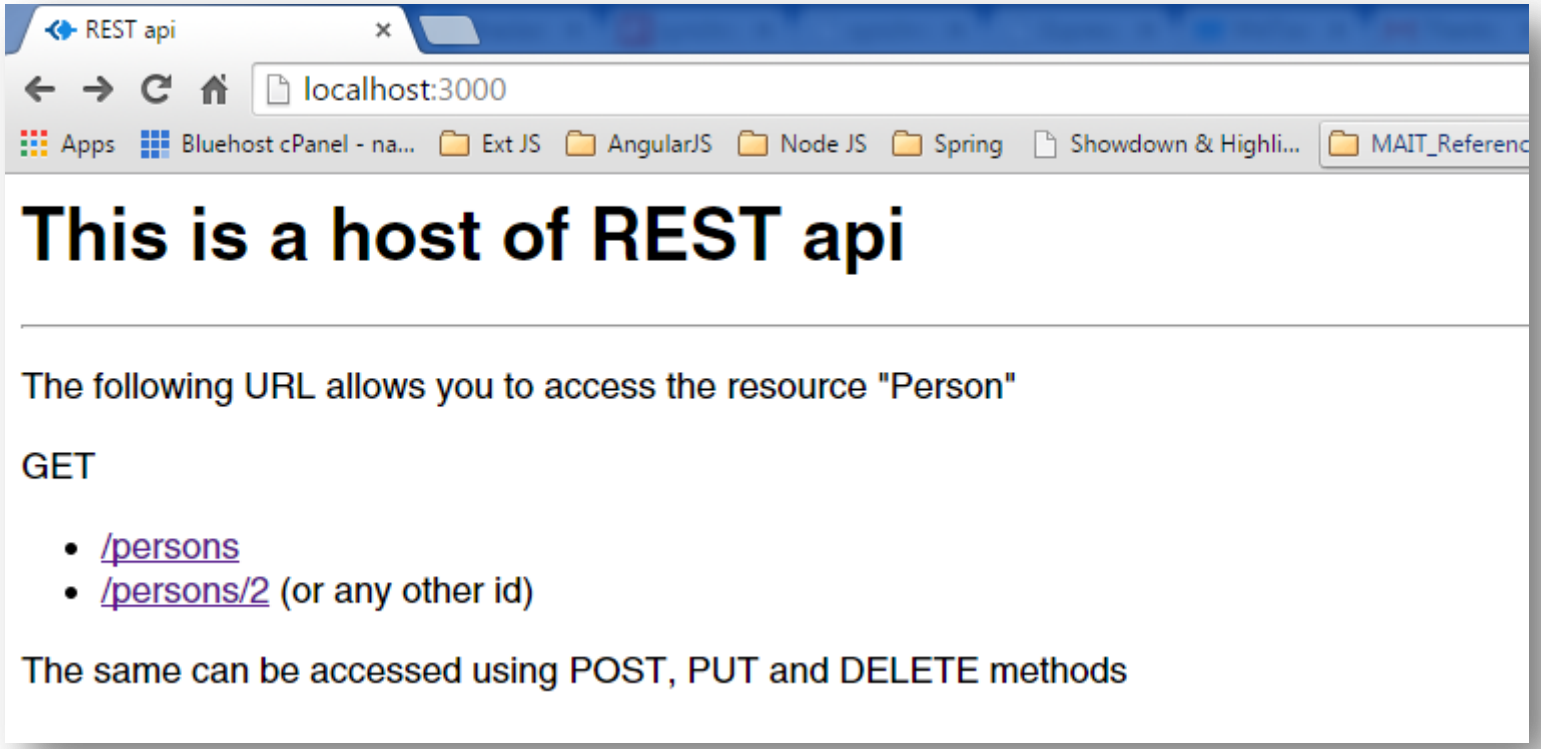
Static content

public/index.html

index.html

```
1  <html>
2  <head>
3      <title>REST api</title>
4  </head>
5  <body>
6  <h1>This is a host of REST api</h1>
7  <hr />
8  <p>The following URL allows you to access the resource "Person"</p>
9
10 <p>GET</p>
11 <ul>
12     <li><a href="/persons">/persons</a></li>
13     <li><a href="/persons/2">/persons/2</a> (or any other id)</li>
14 </ul>
15
16 <p>The same can be accessed using POST, PUT and DELETE methods</p>
17 </body>
18 </html>
```

Static content



OVERVIEW OF MONGO DB

A NOSQL (Not Only SQL) database

MongoDB

- MongoDB is an open-source **document database**
- Hu**mongo**us **DB**
- Falls into the category of **NOSQL** databases
- Stores the data in **un-normalized** format
- Data is stored as **collections** of documents

Document database

- A **record** in an RDBMS is equivalent to a **document** in MongoDB
  - A data structure composed of field and value pairs.
- MongoDB **documents** are similar to **JSON** objects.
  - The values of fields may include scalara data, other documents, arrays, and arrays of documents.
  - Internally, MongoDB stores these documents in the binary format, called BSON (Binary JSON)

Document

```
{
        id      : 7788,
        name    : "Vinod Kumar",
        phones  : [ "9731424784", "9844083934"],
        emails  : [
                {
                        type    : "personal",
                        address : "kayartaya.vinod@gmail.com"
                },
                {
                        type    : "official",
                        address : "vinod@knowledgeworksindia.com"
                }
        ]
}
```

Advantages

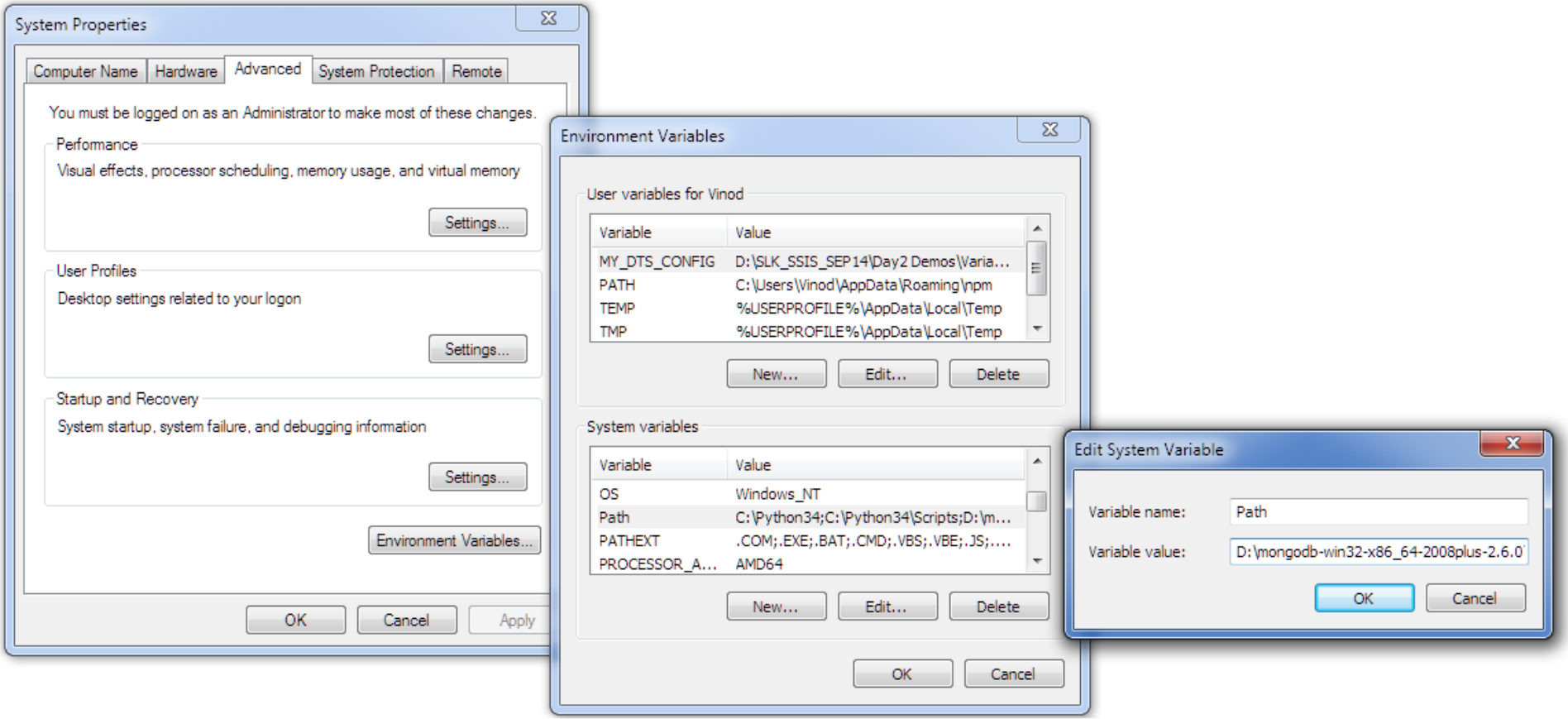
- Documents correspond to **native data** types in many programming languages
- Embedded documents and arrays **reduce need for expensive joins**
- **Dynamic schema** supports fluent polymorphism

Installation/setup

- Download the binary for your **operating system**
  - For Windows 7 64bit:
    - https://fastdl.mongodb.org/win32/mongodb-win32-x86_64-2008plus-2.6.0.zip
- **Unzip** to a drive
  - In my computer:
    - D:\mongodb-win32-x86_64-2008plus-2.6.0

Installation/setup

- Add the D:\mongodb-win32-x86_64-2008plus-2.6.0\bin to your **PATH variable**



Default DB location

- MongoDB requires a **data directory** to store all data
- MongoDB's default data directory path is **\data\db**
  - You can create this folder structure in the same drive as mongodb's installation drive
  - Or specify another location when starting the server

Starting the server

- The server can be started by running the

  mongod.exe executable

- Some of the useful options are:

  --dbpath=PATH-TO-YOUR-DB

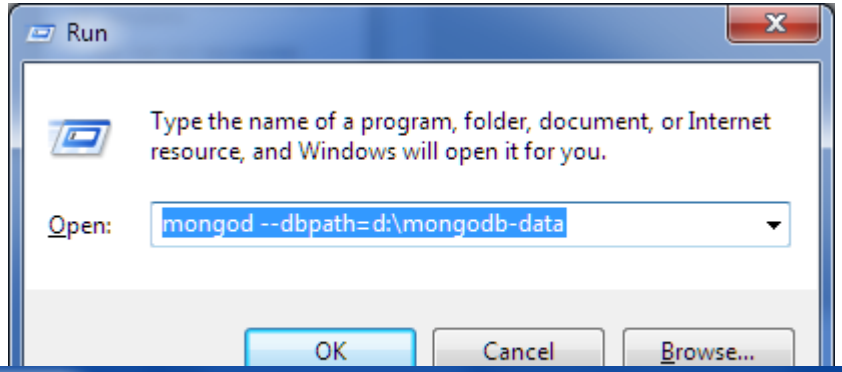
  --port=27017

  Example:

  mongod --dbpath=d:\mongodb-data

(Note: you must create the folder manually before running this command)

Starting the server

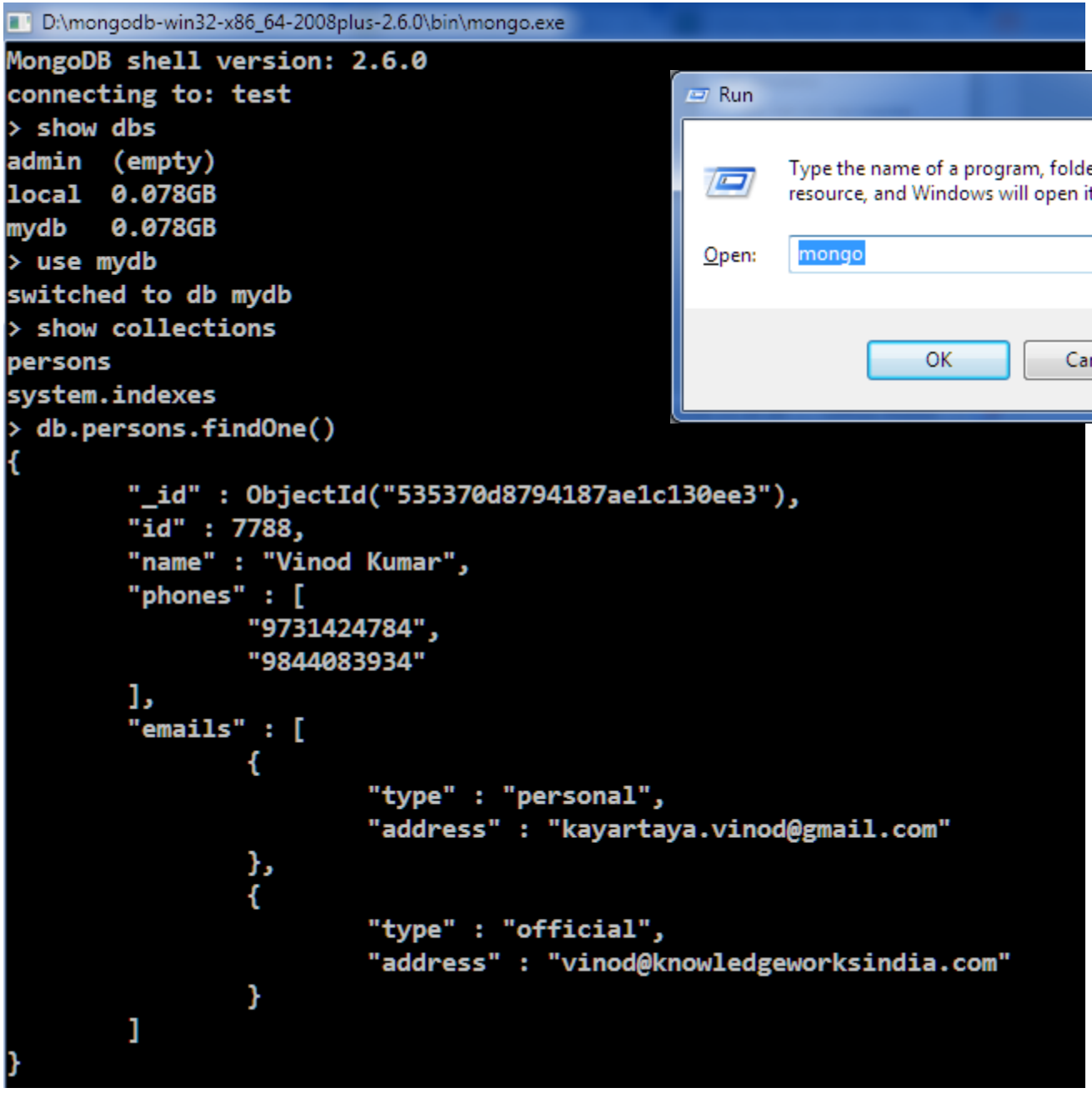



```
D:\mongodb-win32-x86_64-2008plus-2.6.0\bin\mongod.exe
2014-04-20T12:27:46.752+0530 [initandlisten] MongoDB starting : pid=1660 port=27017 dbpath=d:\mongodb-data 64-bit host=V
INOD-LENOVO
2014-04-20T12:27:46.753+0530 [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2014-04-20T12:27:46.753+0530 [initandlisten] db version v2.6.0
2014-04-20T12:27:46.753+0530 [initandlisten] git version: 1c1c76aeca21c5983dc178920f5052c298db616c
2014-04-20T12:27:46.753+0530 [initandlisten] build info: windows sys.getwindowsversion(major=6, minor=1, build=7601, pla
tform=2, service_pack='Service Pack 1') BOOST_LIB_VERSION=1_49
2014-04-20T12:27:46.754+0530 [initandlisten] allocator: system
2014-04-20T12:27:46.754+0530 [initandlisten] options: { storage: { dbPath: "d:\mongodb-data" } }
2014-04-20T12:27:46.788+0530 [initandlisten] journal dir=d:\mongodb-data\journal
2014-04-20T12:27:46.789+0530 [initandlisten] recover : no journal files present, no recovery needed
2014-04-20T12:27:46.822+0530 [FileAllocator] allocating new datafile d:\mongodb-data\local.ns, filling with zeroes...
2014-04-20T12:27:46.823+0530 [FileAllocator] creating directory d:\mongodb-data\_tmp
2014-04-20T12:27:46.914+0530 [FileAllocator] done allocating datafile d:\mongodb-data\local.ns, size: 16MB,  took 0.09 s
ecs
2014-04-20T12:27:46.917+0530 [FileAllocator] allocating new datafile d:\mongodb-data\local.0, filling with zeroes...
2014-04-20T12:27:47.131+0530 [FileAllocator] done allocating datafile d:\mongodb-data\local.0, size: 64MB,  took 0.213 s
ecs
2014-04-20T12:27:47.132+0530 [initandlisten] build index on: local.startup_log properties: { v: 1, key: { _id: 1 }, name
: "_id_", ns: "local.startup_log" }
2014-04-20T12:27:47.133+0530 [initandlisten]     added index to empty collection
2014-04-20T12:27:47.160+0530 [initandlisten] command local.$cmd command: create { create: "startup_log", size: 10485760,
 capped: true } ntoreturn:1 keyUpdates:0 numYields:0  reslen:37 312ms
2014-04-20T12:27:47.161+0530 [initandlisten] waiting for connections on port 27017
```

JavaScript Shell

- The executable **"mongo.exe"** provides an interface to issue direct commands on the db.
- By default the "mongo" command tries to connect to **"localhost"** and port **"27017"**
- You can connect to different ones using **--host** and **--port** options:

```
mongo --port 12345 --host vinod_homepc
```


D:\mongodb-win32-x86_64-2008plus-2.6.0\bin\mongo.exe
MongoDB shell version: 2.6.0
connecting to: test
> show dbs
admin (empty)
local 0.078GB
mydb   0.078GB
> use mydb
switched to db mydb
> show collections
persons
system.indexes
> db.persons.findOne()
{
"_id" : ObjectId("535370d8794187ae1c130ee3"),
"id" : 7788,
"name" : "Vinod Kumar",
"phones" : [
"9731424784",
"9844083934"
],
"emails" : [
{
"type" : "personal",
"address" : "kayartaya.vinod@gmail.com"
},
{
"type" : "official",
"address" : "vinod@knowledgeworksindia.com"
}
]
}

Run
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open:
mongo
OK
Cancel
Browse...

Some commands to start with..

- show dbs
  - displays the list of databases
- use mydb
  - switches to the database “mydb” if exists, or creates a new with the same name and switches to it
- db
  - displays the current database in use
- db.dropDatabase()
  - Deletes the current database

Importing external data

- Use the **mongoimport.exe** tool to import external data into a database

```
mongoimport
        --host localhost
        --port 27017
        --db mydb
        --collection orders
        --file d:\orders.json
        --jsonArray
```

Some commands to start with..

- db.<collection>.findOne()
  - Displays the first document in the collection

- db.<collection>.find().pretty()
  - displays the first 20 documents in an indented format

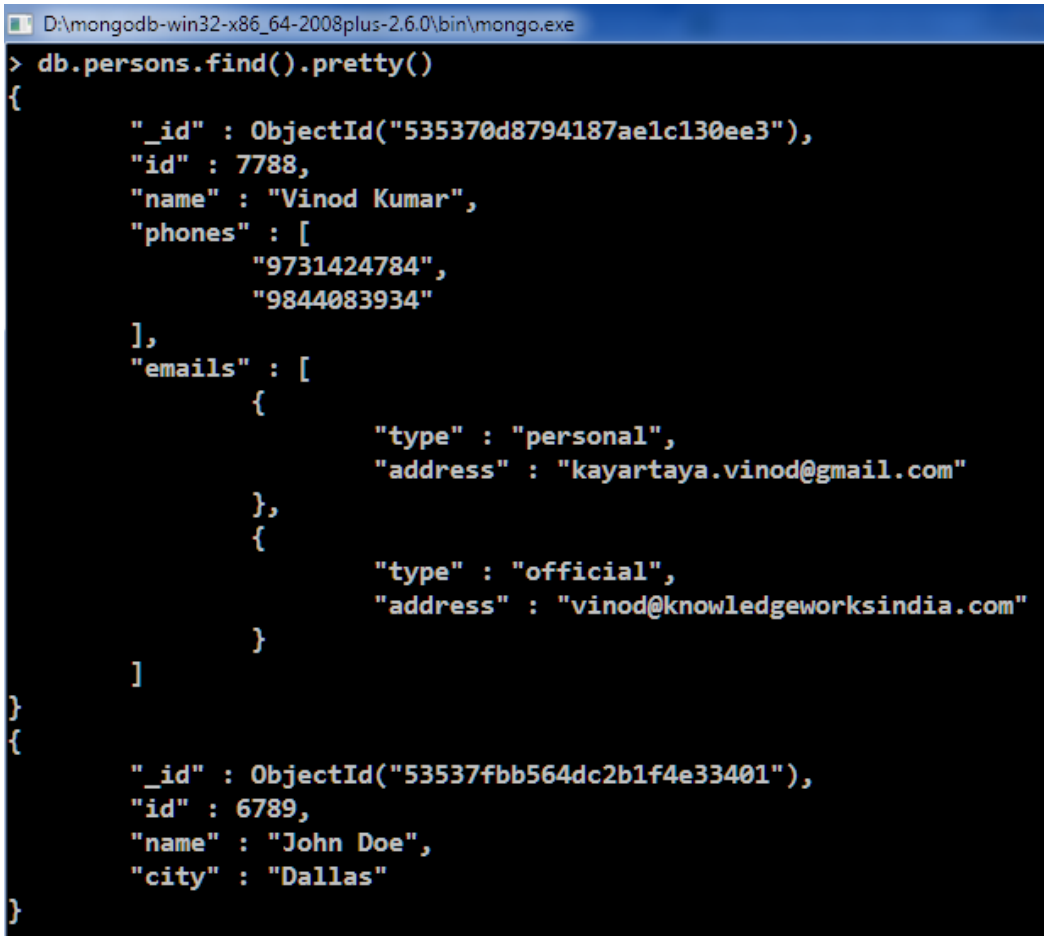
Some commands to start with..

- show collections
  - Displays the list of collections (tables in RDBMS)
- db.<collection>.insert(data)
  - If the collection exists, inserts the data, else creates a new collection with the same name and inserts the data

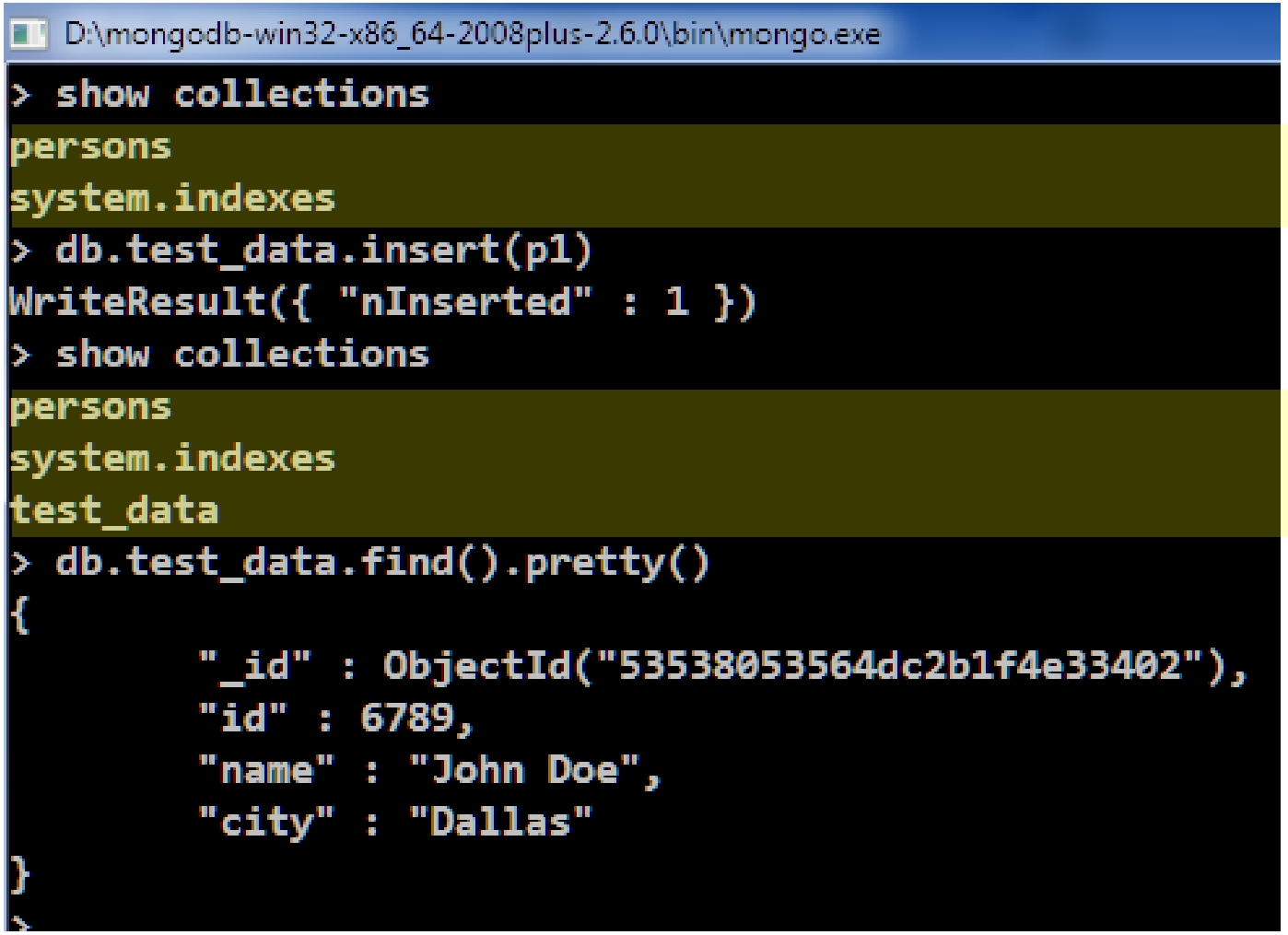
Some commands to start with..

```
p1 = {
        id      : 6789,
        name    : "John Doe",
        city    : "Dallas"
}

db.persons.insert(p1);
```



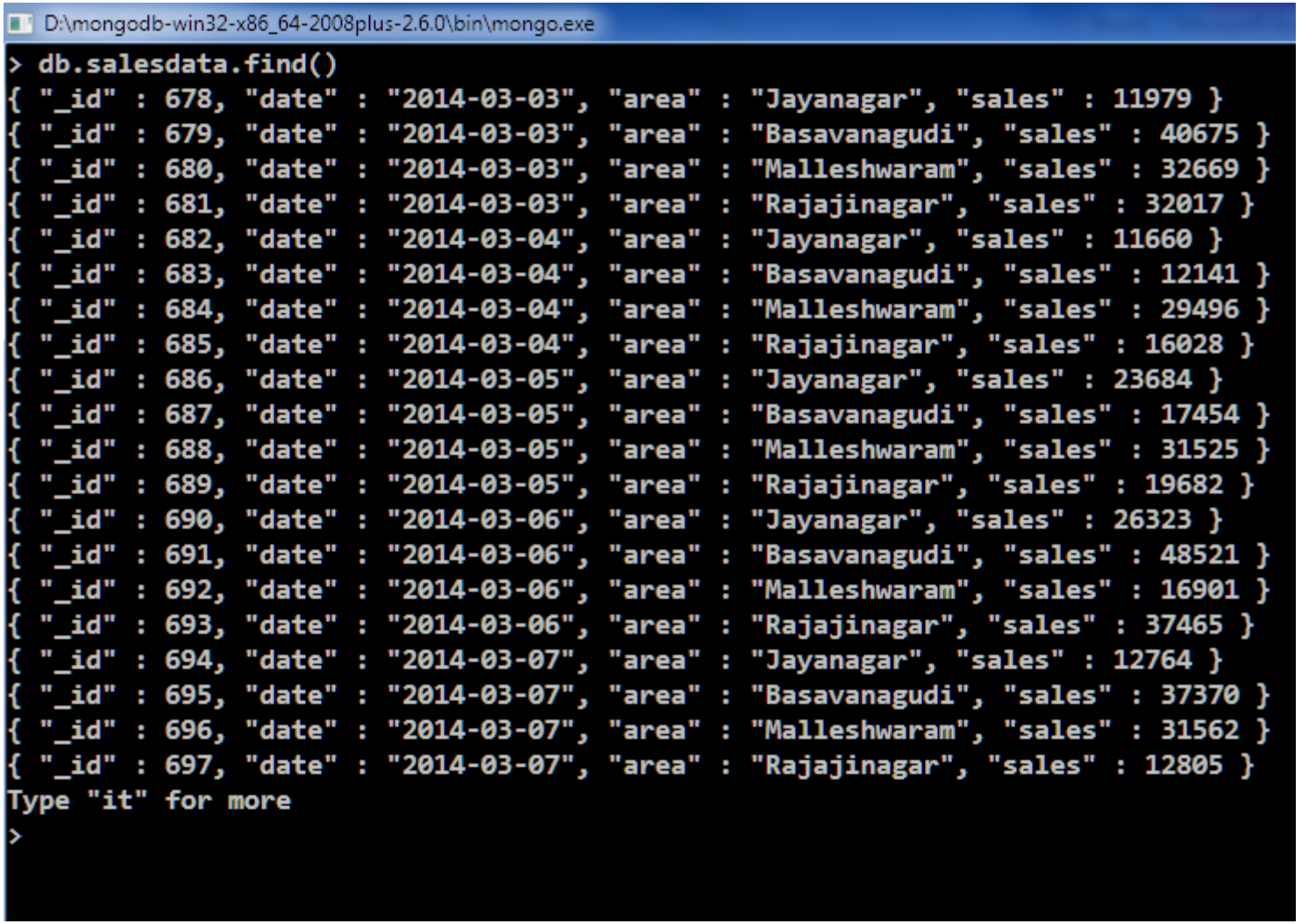
Some commands to start with..



Some commands to start with..

- db.<collection>.find()
  - Returns a cursor to the result
  - Displays the first 20 documents on the screen
  - type “it” to iterate again and get 20 more documents

Some commands to start with..

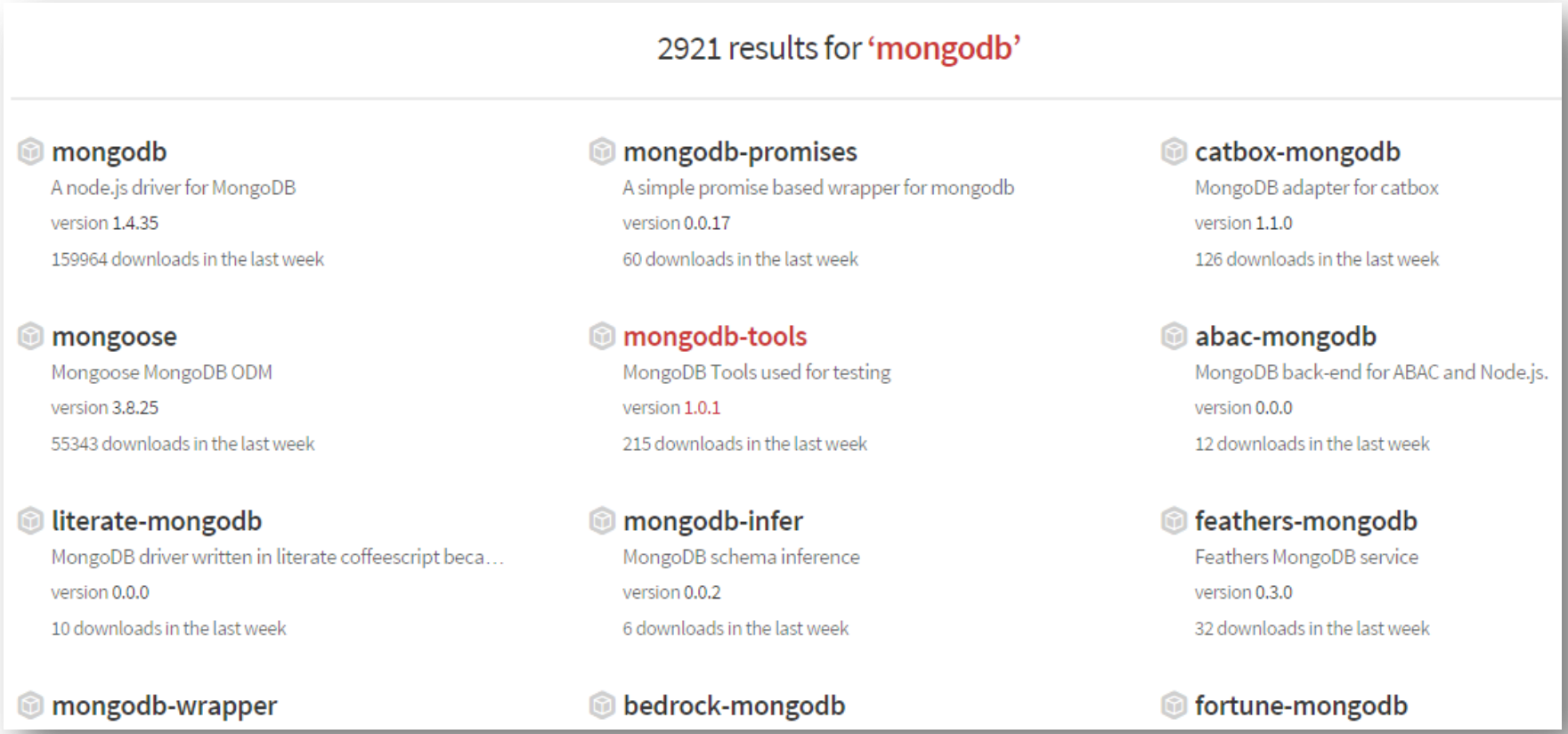


CONNECTING NODE WITH MONGO

JavaScript server app using data stored in JavaScript database

Choosing the driver

- For connecting to Mongodb, there are close to **3000** drivers provided by the **node community**

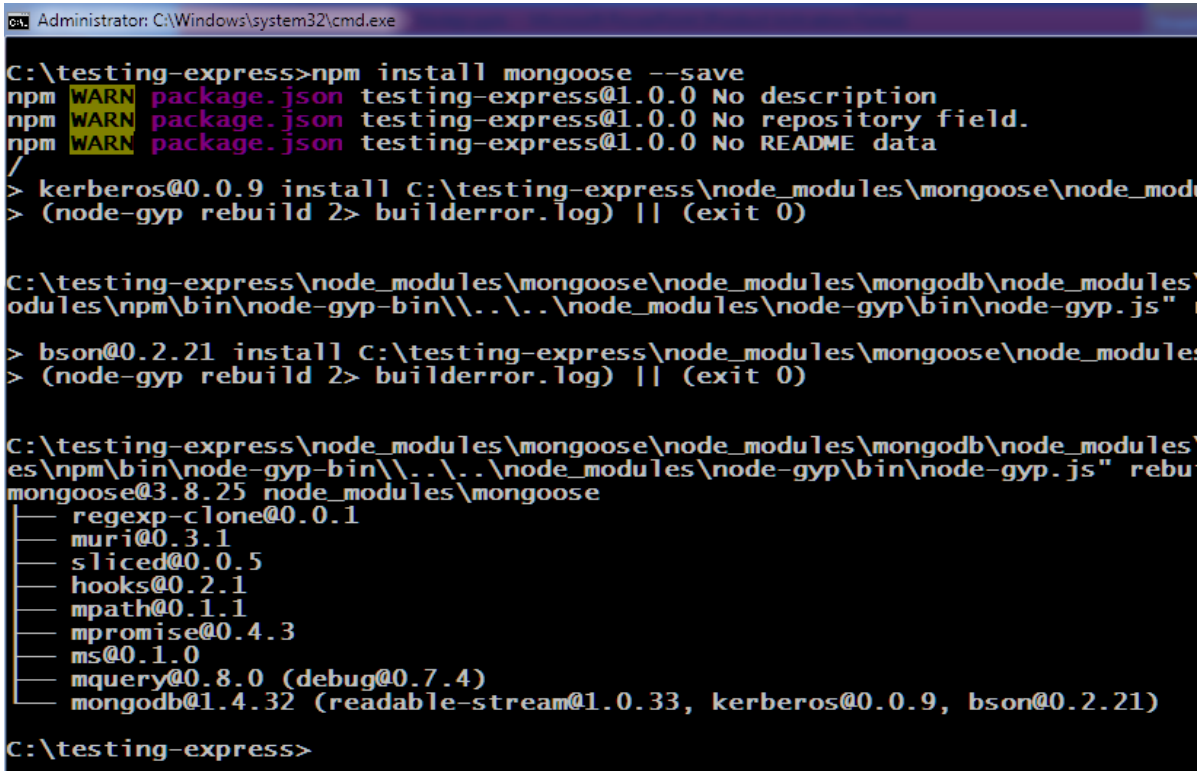


Using mongoose

- Run the command:

```
npm install mongoose --save
```

in the command prompt



Using mongoose - Connect to database

- Add **mongoose** to your application

```
var mongoose = require("mongoose");
```

- **Connect** to the database

```
mongoose.connect("mongodb://localhost/vindb");
```

- Create a **no-schema** model

```
var PersonModel = mongoose.model(
      "persons",
      new mongoose.Schema({_id: "Number"},
            {strict: false}));
```

Using mongoose - Insert

- Adding a **new document** to the collection

```
var person = new PersonModel({
      // person data to be added
});

person.save(function(err){
      if(!err){
            // respond with a success status
      }
      else{
            // respond with a failure status
      }
});
```

Using mongoose - Update

- Updating an **existing document** to the collection

```
PersonSchema.update({_id: 5},
      {// person data to be updated
       // make sure to exclude _id },
      { upsert: true },
      function(err, doc){
            if(!err){
                  // respond with success status
            }
            else{
                  // respond with failure status
            }
      });
```

Using mongoose - Query

- Finding **all** persons

```
PersonSchema.find(
        function(err, docs){
                if(!err){
                        // respond with docs
                }
        }
);
```

- Find **one** person

```
PersonSchema.find(
        { _id: 5 },
        function(err, docs){
                // process and respond
        }
);
```

LET'S BUILD REST API

Using Node, Express, Mongoose and Mongodb