Express and EJS

NodeJS Design Frameworks

```
npm install -g express
npm install -g express-generator

npm install -g express@3.4.x
```

express

web application framework for node

npm install -g express
npm install -g express-generator
npm install -g express@3.4.x

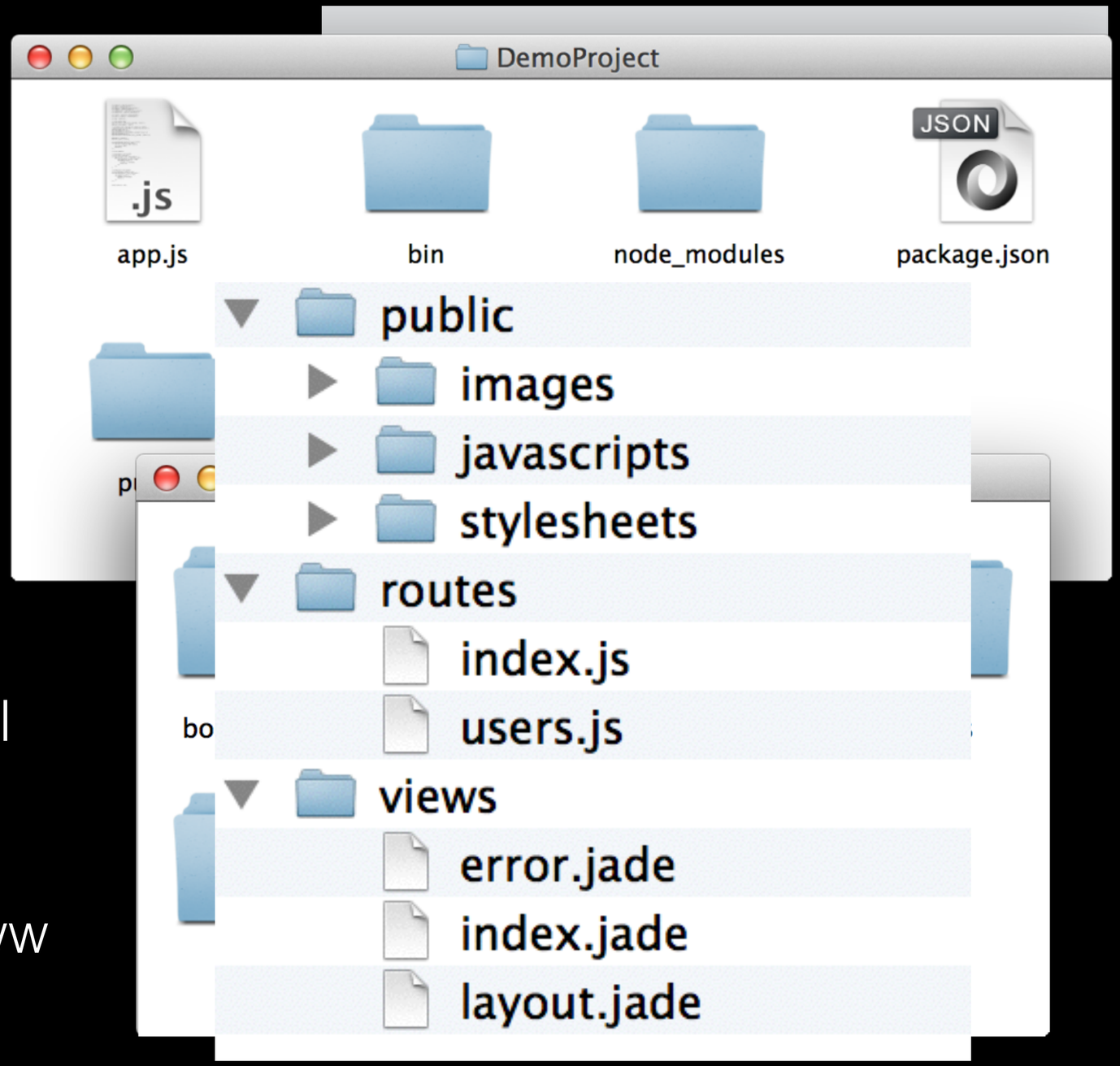
To create express project:

> express ProjectName

install dependencies:
> cd ProjectName && npm install

run the app:
>DEBUG= ProjectName ./bin/www

>npm start

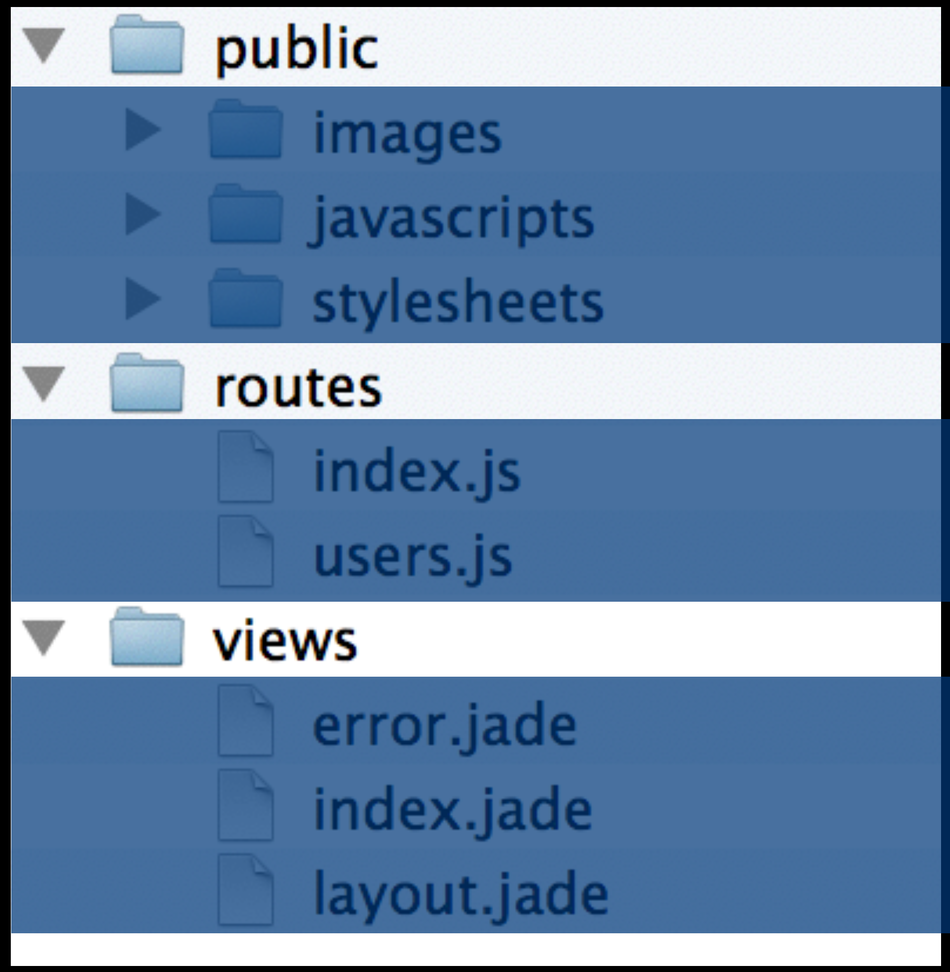


package.json

```
1  {
2    "name": "DemoProject",
3    "version": "0.0.0",
4    "private": true,
5    "scripts": {
6      "start": "node ./bin/www"
7    },
8    "dependencies": {
9      "express": "~4.9.0",
10     "body-parser": "~1.8.1",
11     "cookie-parser": "~1.3.3",
12     "morgan": "~1.3.0",
13     "serve-favicon": "~2.1.3",
14     "debug": "~2.0.0",
15     "jade": "~1.6.0"
16   }
17 }
```

npm init

Express Project Contents



Public Folder
All files needed for service

Routes Folder - localhost:3000/users

Views Folder
Templates for page layout

Another Way to Create Simple Projects Manually

```
mkdir MyProject
cd MyProject
npm init (Follow Setup)
npm install express --save

npm install MODULENAME --save
```

package.json

```
1  {
2    "name": "SimpleProject",
3    "version": "1.0.0",
4    "description": "Simple Demo",
5    "main": "app.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "keywords": [
10     "Hello",
11     "There"
12   ],
13   "author": "Leon Baird",
14   "license": "ISC",
15   "dependencies": {
16     "express": "^4.9.0"
17   }
18 }
19
```

Create Simple App

```
var express = require(‘express’);
var app = express();

app.get(“/“, function(req, res){
  // deal with request
  res.send(“Hello There”);
});
```

```
app.get( pattern , callback);
```

“/“ “/path/name” REGEX - /^\/service.+/i

/pathname/:valueA/:valueB

req.params.valeA

/pathname/name?valueA=100&valueB=200&valueC=300

Routers and Middleware

var router = express.Router([options]);

```
var express = require('express');
var router = express.Router();

/* GET users listing. */
router.get('/', function(req, res) {
  res.send(‘Hello from Router');
});

module.exports = router;
```

```
// invoked for any requests passed to this router
router.use(function(req, res, next) {
  // .. some logic here .. like any other middleware
  next();
});

// will handle any request that ends in /events
router.get('/events', function(req, res, next) {
  // ..
});
```

// only requests to /calendar/* will be sent to our "router"
app.use('/calendar', router);

```
var express = require('express');
var app = express();
var router = express.Router();
```

Setup App and Router

```
// simple logger for this router's requests
// all requests to this router will first hit this middleware
router.use(function(req, res, next) {
  console.log('%s %s %s', req.method, req.url, req.path);
  next();
});
```

Responds to ALL routes in context /foo/*

```
// this will only be invoked if the path ends in /bar
router.use('/bar', function(req, res, next) {
  // ... maybe some additional /bar logging ...
  next();
});
```

Looks for /bar in context: /foo/bar

```
// always invoked
router.use(function(req, res, next) {
  res.send('Hello World');
});
```

Responds to ALL routes as the LAST item

```
app.use('/foo', router);

app.listen(3000);
```

Set up Server and user router for all URLS starting with /foo

routes/index.js

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res) {
      res.send("Hello from the index.js file");
});

module.exports = router;
```

routes/users.js

```
var express = require('express');
var router = express.Router();

/* GET users listing. */
router.get('/', function(req, res) {
      res.send("Hello from users.js file.");
});

module.exports = router;
```

/app.js

```
var index = require('./routes/index');
var users = require('./routes/users');
app.use('/', index);
app.use('/users', users);
```

```
// catch 404 and forward to error handler
app.use(function(req, res, next) {
    var err = new Error('Not Found');
    err.status = 404;
    next(err);
});
```

Template Engine

JADE

EJS

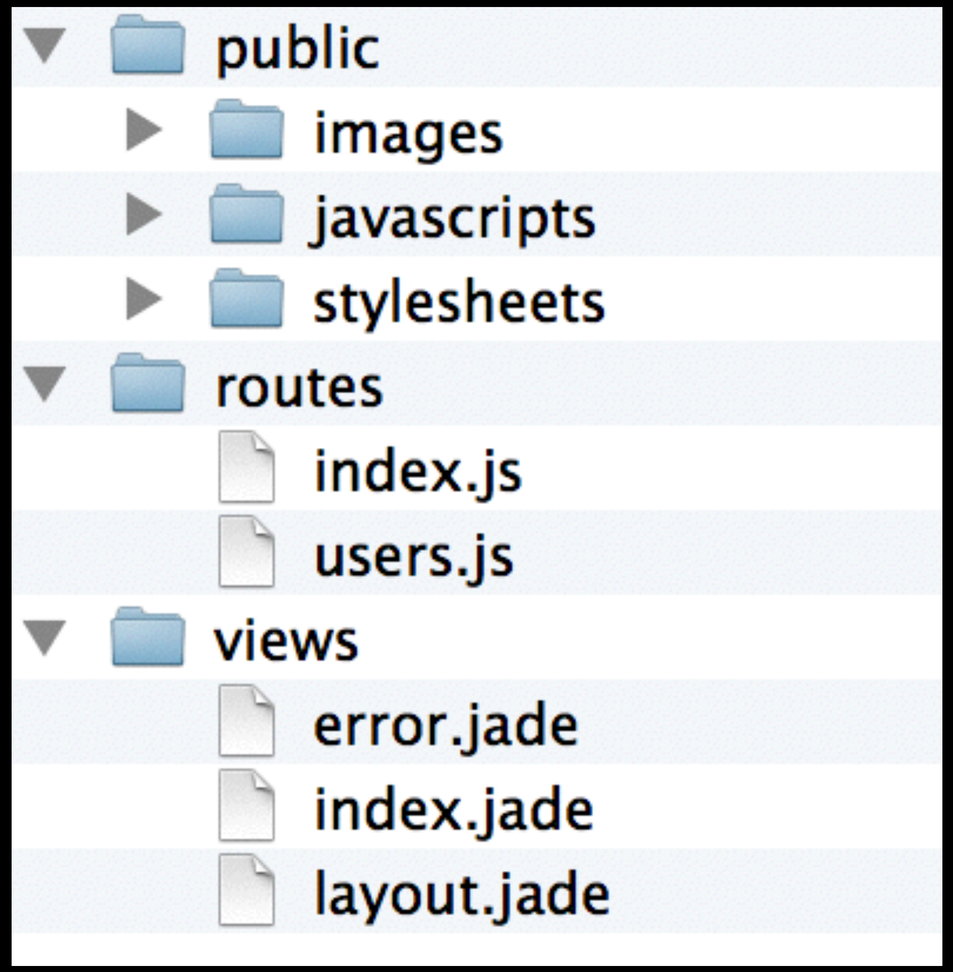
```
npm install ejs --save
app.set(‘view engine’, ‘ejs’);
res.render(“templatename”, {DATA});
app.use(express.static(__dirname + '/public'));
```

template.ejs

```
<% JAVASCRIPT %>

<h1> <%= pageTitle %> </h1>
```

Express Project Contents



app.js

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

app.get("/", function(req, res){
   // data for view
   var data = {
      pageTitle: "User List",
      people:["Mike", "Mary", "Martha", "Sue"]
   }
   res.render('main_template', data);
});
```

app.js

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

app.get("/", function(req, res){
      // data for view
      var data = {
            pageTitle: "User List",
            people:["Mike", "Mary", "Martha", "Sue"]
      }
      res.render('main_template', data);
});
```

main_template.ejs

```
<!DOCTYPE html>
<html>
    <head>
        <title><%= pageTitle %></title>
        <link rel='css' href='css/styles.css' />
    </head>
    <body>
        <h1><%= pageTitle %></h2>
        <% if (people.length > 0) { %>
        <p>List of people</p>
        <ul>
            <% for(var i in people) { %>
                <li><%= people[i] %></li>
            <% } %>
        </ul>
        <% } else { %>
        <p>There are currently no people!</p>
        <% } %>
    </body>
</html>
```

Partials

You can break your templates into modular sections,
similar to using include/require in PHP.

```
<% include PATH %>
```

---

views/main.ejs
views/partials/head.ejs
views/partials/page_header.ejs
views/partials/page_footer.ejs

```
<head><% include partials/header.ejs %></header>
<body>
    <% include partials/page_header.ejs %>

    <!— Page Content Here —>

    <% include partials/page_footer.ejs %>
</body>
```

NoSQL Databases
MongoDB

MongoDB

www.mongodb.org

node.js OSCAR LINK video Colecovision MJ API Key iOS7 Icons BUILDER CBIMail TV Android REF Admin – Local Admin – Online RJFG

mongoDB

Docs Try It Out Downloads Community Blog Search

Agile and Scalable

MongoDB (from "hu**mongo**us") is an open-source document database, and the leading NoSQL database. Written in C++, MongoDB features:

- **Document-Oriented Storage »**
  JSON-style documents with dynamic schemas offer simplicity and power.
- **Full Index Support »**
  Index on any attribute, just like you're used to.
- **Replication & High Availability »**
  Mirror across LANs and WANs for scale and peace of mind.
- **Auto-Sharding »**
  Scale horizontally without compromising functionality.
- **Querying »**
  Rich, document-based queries.
- **Fast In-Place Updates »**
  Atomic modifiers for contention-free performance.
- **Map/Reduce »**
  Flexible aggregation and data processing.

Newsletter Signup

* Email Address

Submit

MongoDB 2.6 is now available

Download MongoDB 2.6

Upcoming Events

Sep 17 Partner Webinar: MongoDB and Pentaho for Financial Services

Sep 18 Webinar: How to Achieve Scale with MongoDB

Sep 23 Webinar: How Financial Services Organizations Use MongoDB

Sep 24 Webinar: Big Data VMMare Analytics with ESXSTOP, MongoDB and JSON Studio

The MongoDB 2.6 Manual — MongoDB Manual 2.6.4

docs.mongodb.org/manual/

node.js OSCAR LINK video Colecovision MJ API Key iOS7 Icons BUILDER CBIMail TV Android REF Admin – Local Admin – Online RJFG

mongoDB

Try it Out Drivers Downloads Community Blog Search

MONGODB MANUAL 2.6 (current)

- Installation
- MongoDB CRUD Operations
- Data Models
- Administration
- Security
- Aggregation
- Indexes
- Replication
- Sharding
- Frequently Asked Questions
- Reference
- Release Notes

About | Contents | Index

OPTIONS

The MongoDB 2.6 Manual

Welcome to the MongoDB Manual! MongoDB is an open-source, document-oriented database designed for ease of development and scaling.

The Manual introduces MongoDB and continues to describe the query language, operational considerations and procedures, administration, and application development patterns, and other aspects of MongoDB use and administration. See the Introduction to MongoDB for an overview of MongoDB's key features. The Manual also has a thorough reference section of the MongoDB interface and tools.

This manual is under constant development. See the About MongoDB Documentation for more information on the MongoDB Documentation project.

MONGODB 2.6 RELEASED:

See Release Notes for MongoDB 2.6 for new features in MongoDB 2.6.

Getting Started	Developers	Administrators	Reference
Introduction to MongoDB	Database Operations	Production Notes	Shell Methods
Installation Guides	Aggregation	Replica Sets	Query Operators
First Steps	SQL to MongoDB Mapping	Sharded Clusters	Complete System Reference
Frequently Asked Questions	Indexes	MongoDB Security	Glossary

Community

Connect to MongoDB Module
DB can be local OR hosted (hosted is easier)

Connect to MongoDB Module
DB can be local OR hosted (hosted is easier)

mongoose

npm install --save mongoose

Connection String:
mongodb://<dbuser>:<dbpassword>@ds000000.mongolab.com:00000/db_name

Make a database module:
database.js

```
var mongoose = require(‘mongoose’);
mongoose.connect(DB_STRING);

module.exports = mongoose.connection;
```

Import the database into your app:
app.js

```
var db = require('./database');

db.commands();
```

Create a schema for database:
model.js

```
var mongoose = require(‘mongoose’);

module.exports = mongoose.model(‘ModelName’, {
  field1: String,
  field2: Number
});
```

Supported Datatypes

- String
- Number
- Date
- Buffer
- Boolean
- Mixed
- Objectid
- Array

```
var schema = new Schema({
  name:    String,
  binary:  Buffer,
  living:  Boolean,
  updated: { type: Date, default: Date.now }
  age:     { type: Number, min: 18, max: 65 }
  mixed:   Schema.Types.Mixed,
  _someId: Schema.Types.ObjectId,
  array:      [],
  ofString:   [String],
  ofNumber:   [Number],
  ofDates:    [Date],
  ofBuffer:   [Buffer],
  ofBoolean:  [Boolean],
  ofMixed:    [Schema.Types.Mixed],
  ofObjectId: [Schema.Types.ObjectId],
  nested: {
    stuff: { type: String, lowercase: true, trim: true }
  }
})
```

Create a schema for database:
model.js

```
var mongoose = require(‘mongoose’);

module.exports = mongoose.model(‘ModelName’, {
  field1: String,
  field2: Number
});
```

Import the database into your app:
app.js

```
var db = require('./database');
var ModelSchema = require('./schemas/model');

var newRecord = new ModelSchema({
  field1: "My String",
  field2: 123456
});

newRecord.save(function(err) { //handle error });
```

```
var db = require('./database');
var ModelSchema = require('./schemas/model');

ModelSchema.find({search:'value'})    .find()
  .setOptions({ sort: 'field1' })
  .exec(function(err, results) {
    // handle error
    // parse results (array of models)
    res.render('modelView', { dbResutls: results});
  });
```