

Express

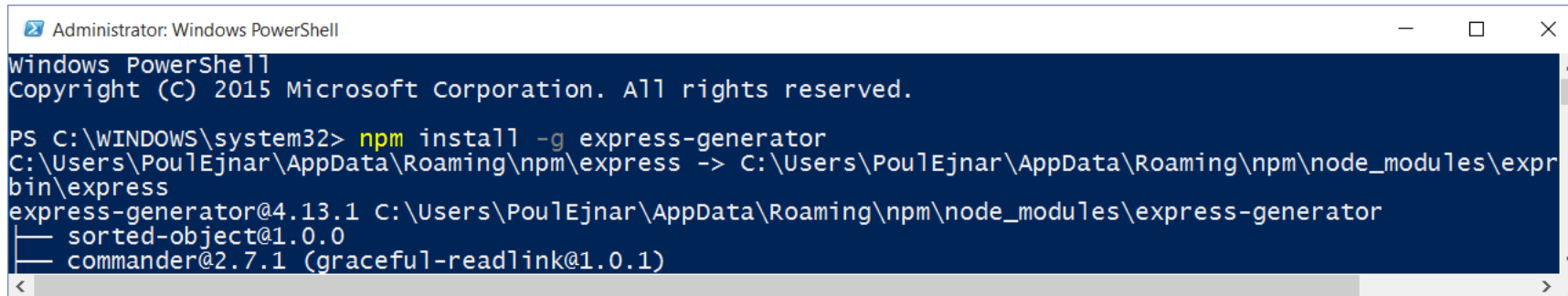


What is Express?

- Express is a Fast, unopinionated, minimalist web framework for **Node.js**
- Express is a **routing** and **middleware** web framework with minimal functionality of its own

Install Express globally

- To create new Express applications on the fly
 - Install Express generator globally:
npm install -g express-generator
 - you'll need to run this as an administrator (sudo on linux)



```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\WINDOWS\system32> npm install -g express-generator
C:\Users\PoulEjnar\AppData\Roaming\npm\express -> C:\Users\PoulEjnar\AppData\Roaming\npm\node_modules\express-generator
bin\express
express-generator@4.13.1 C:\Users\PoulEjnar\AppData\Roaming\npm\node_modules\express-generator
├─ sorted-object@1.0.0
├─ commander@2.7.1 (graceful-readlink@1.0.1)
```

Configuring an Express App

- Just enter express at the command line
 - You need to have express installed globally for this to work

To use pug and init a git repo:

```
express --view=pug --git app_name
```

- Default options is
 - No css preprocessor
 - Use of jade as HTML template engine
- If you don't want to use the default options
 - You can tweak the installation with parameteres
 - Use of css preprocessor: less, stylus, compass or sass
 - Use of other HTML template engine – like **pug**, ejs, hogan, twig, vash or hbs (handlebars)

Running Express

```
Windows PowerShell
C:\demo\Mean\express> express --view=pug --git app_name

create : app_name\
create : app_name\public\
create : app_name\public\javascripts\
create : app_name\public\images\
create : app_name\public\stylesheets\
create : app_name\public\stylesheets\style.css
create : app_name\routes\
create : app_name\routes\index.js
create : app_name\routes\users.js
create : app_name\views\
create : app_name\views\error.pug
create : app_name\views\index.pug
create : app_name\views\layout.pug
create : app_name\.gitignore
create : app_name\app.js
create : app_name\package.json
create : app_name\bin\
create : app_name\bin\www

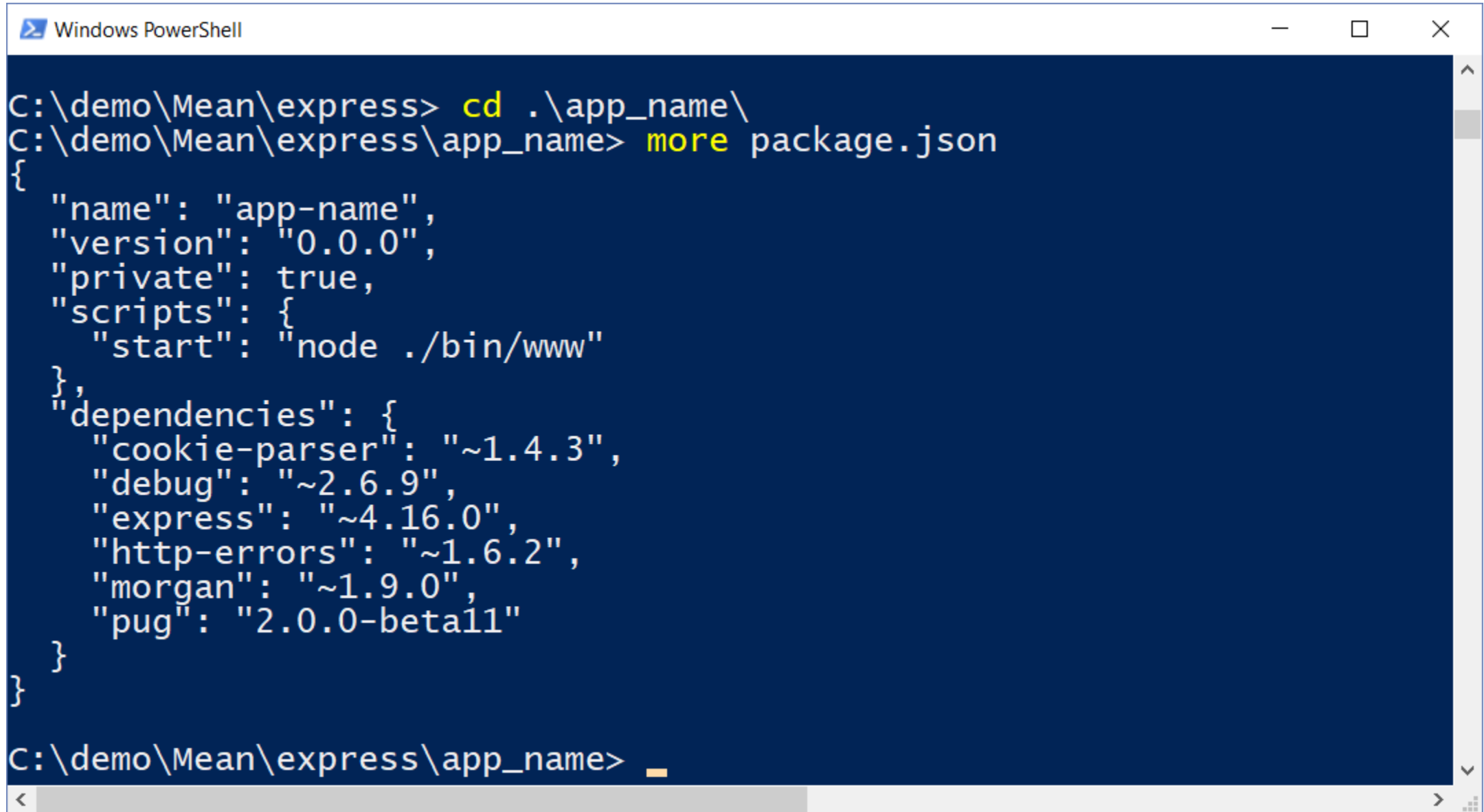
change directory:
> cd app_name

install dependencies:
> npm install

run the app:
> SET DEBUG=app-name:* & npm start

C:\demo\Mean\express>
```

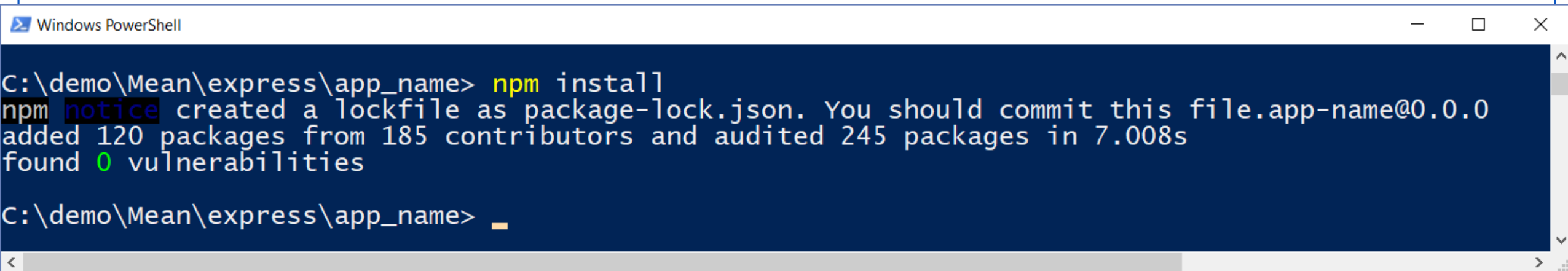
Default Express package.json



```
Windows PowerShell

C:\demo\Mean\express> cd .\app_name\
C:\demo\Mean\express\app_name> more package.json
{
  "name": "app-name",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "cookie-parser": "~1.4.3",
    "debug": "~2.6.9",
    "express": "~4.16.0",
    "http-errors": "~1.6.2",
    "morgan": "~1.9.0",
    "pug": "2.0.0-beta11"
  }
}
```

Run npm install



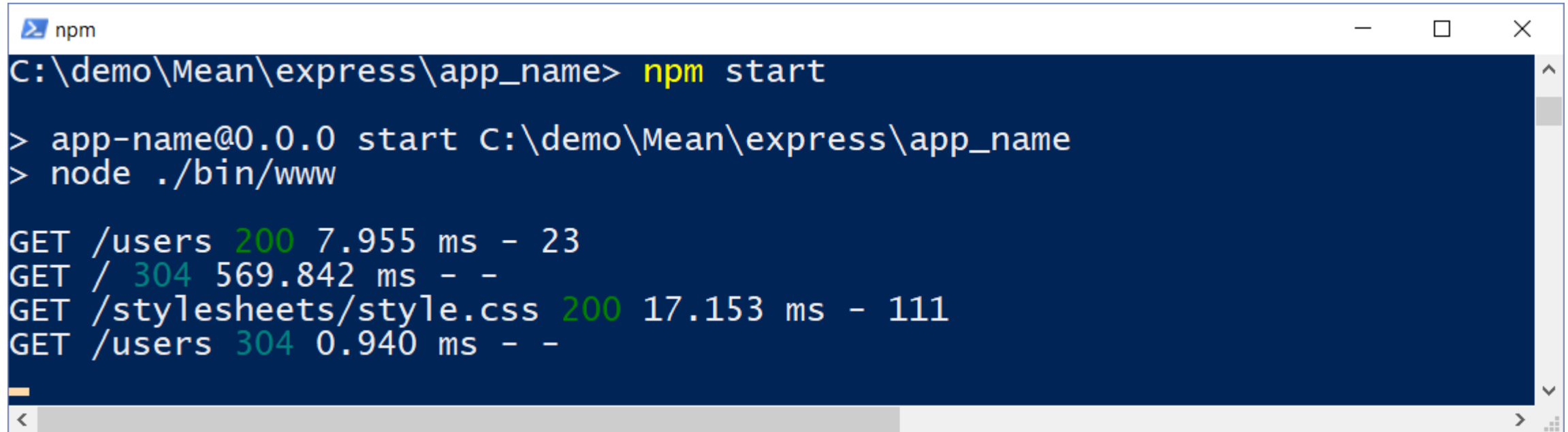
```
Windows PowerShell

C:\demo\Mean\express\app_name> npm install
npm notice created a lockfile as package-lock.json. You should commit this file.app-name@0.0.0
added 120 packages from 185 contributors and audited 245 packages in 7.008s
found 0 vulnerabilities

C:\demo\Mean\express\app_name> 
```

Start The Server

- Use **npm start** to start the server

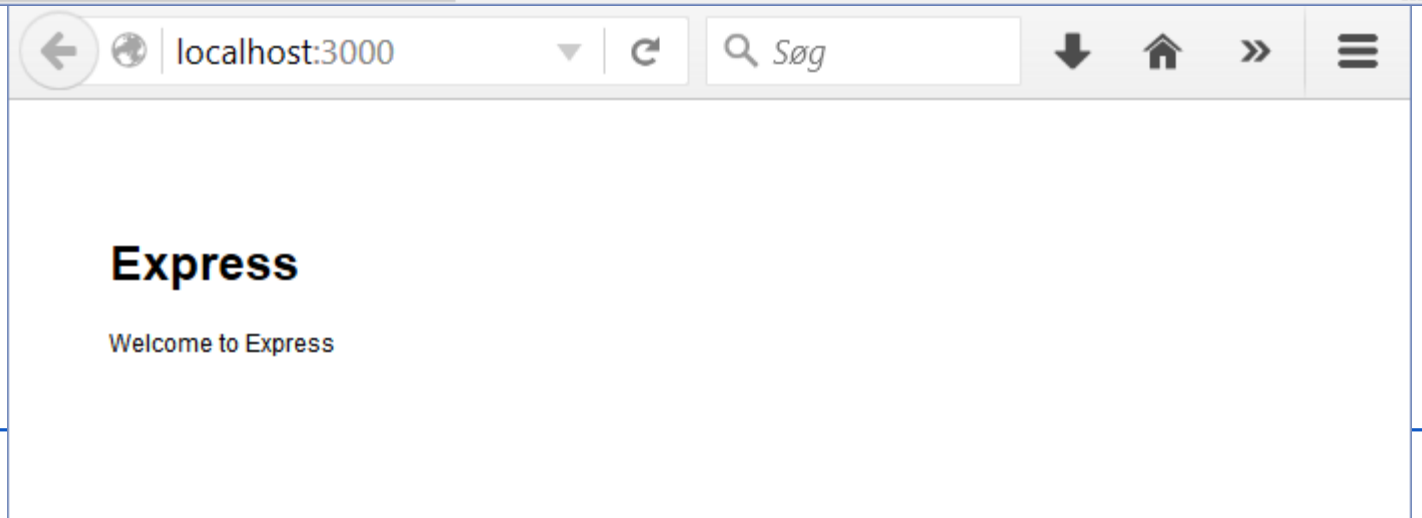


```
npm
C:\demo\Mean\express\app_name> npm start

> app-name@0.0.0 start C:\demo\Mean\express\app_name
> node ./bin/www

GET /users 200 7.955 ms - 23
GET / 304 569.842 ms - -
GET /stylesheets/style.css 200 17.153 ms - 111
GET /users 304 0.940 ms - -
```

- And enter Ctrl-c in the console to stop the server



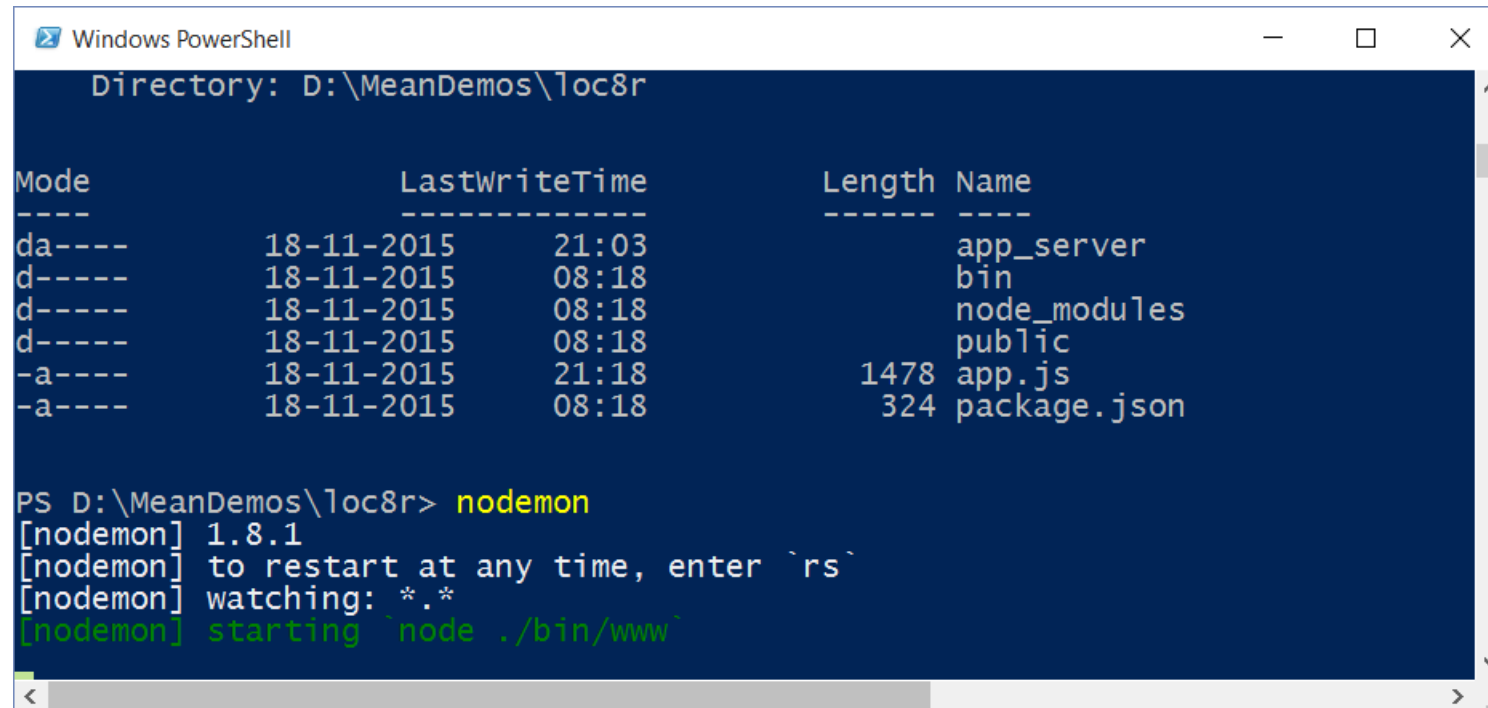
Auto Restart

- You can update static content without restarting the server
- But when you update application code (server side javascript) you must restart the server before it has any effect
- You can automate this with **nodemon**
- Nodemon monitor your code and will restart the server when it detects that changes have been made to your code
- To install nodemon:

```
ps > npm install -g nodemon
```
- To use nodemon to enable auto restart:

```
ps c:\MyExpressApp> nodemon
```

Using nodemon



```
Windows PowerShell
Directory: D:\MeanDemos\loc8r

Mode                LastWriteTime         Length Name
----                -
da----            18-11-2015    21:03             app_server
d-----            18-11-2015    08:18             bin
d-----            18-11-2015    08:18          node_modules
d-----            18-11-2015    08:18             public
-a----            18-11-2015    21:18          1478 app.js
-a----            18-11-2015    08:18          324 package.json

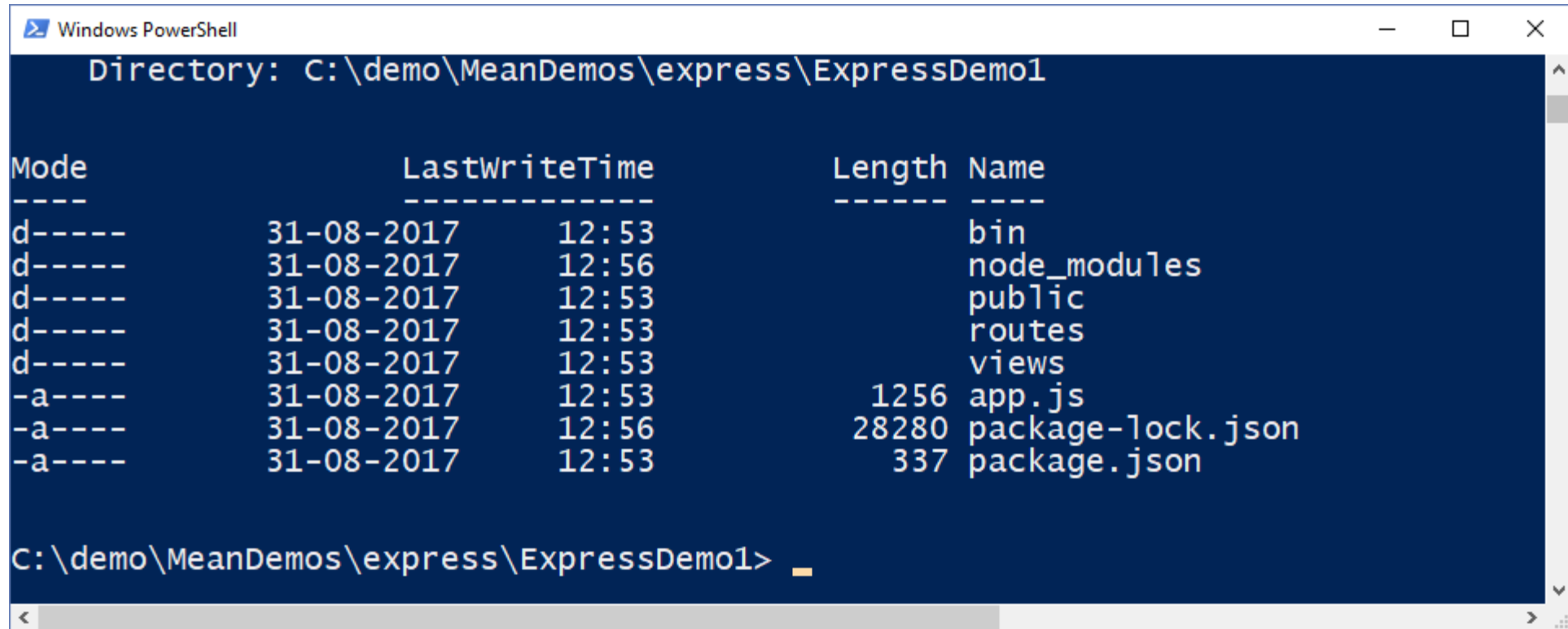
PS D:\MeanDemos\loc8r> nodemon
[nodemon] 1.8.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node ./bin/www`
```

What is Middleware?

- Middleware is a function with access to
 - the request object (req)
 - the response object (res)
 - and the next middleware in the application's request-response cycle (next)
- Middleware can:
 - Execute any code
 - Make changes to the request and the response objects
 - End the request-response cycle - or
 - Call the next middleware in the stack

Default Express Folder Structure

- app.js contains configuration
- public contains all the static files
- Routes contains the controllers
- Views contains the view templates



The screenshot shows a Windows PowerShell window with the title bar "Windows PowerShell". The command prompt shows the directory path "C:\demo\MeanDemos\express\ExpressDemo1". Below the path, a table lists the files and folders in the directory. The table has four columns: "Mode", "LastWriteTime", "Length", and "Name". The files listed are "bin", "node_modules", "public", "routes", "views", "app.js", "package-lock.json", and "package.json".

Mode	LastWriteTime	Length	Name
d-----	31-08-2017 12:53		bin
d-----	31-08-2017 12:56		node_modules
d-----	31-08-2017 12:53		public
d-----	31-08-2017 12:53		routes
d-----	31-08-2017 12:53		views
-a----	31-08-2017 12:53	1256	app.js
-a----	31-08-2017 12:56	28280	package-lock.json
-a----	31-08-2017 12:53	337	package.json

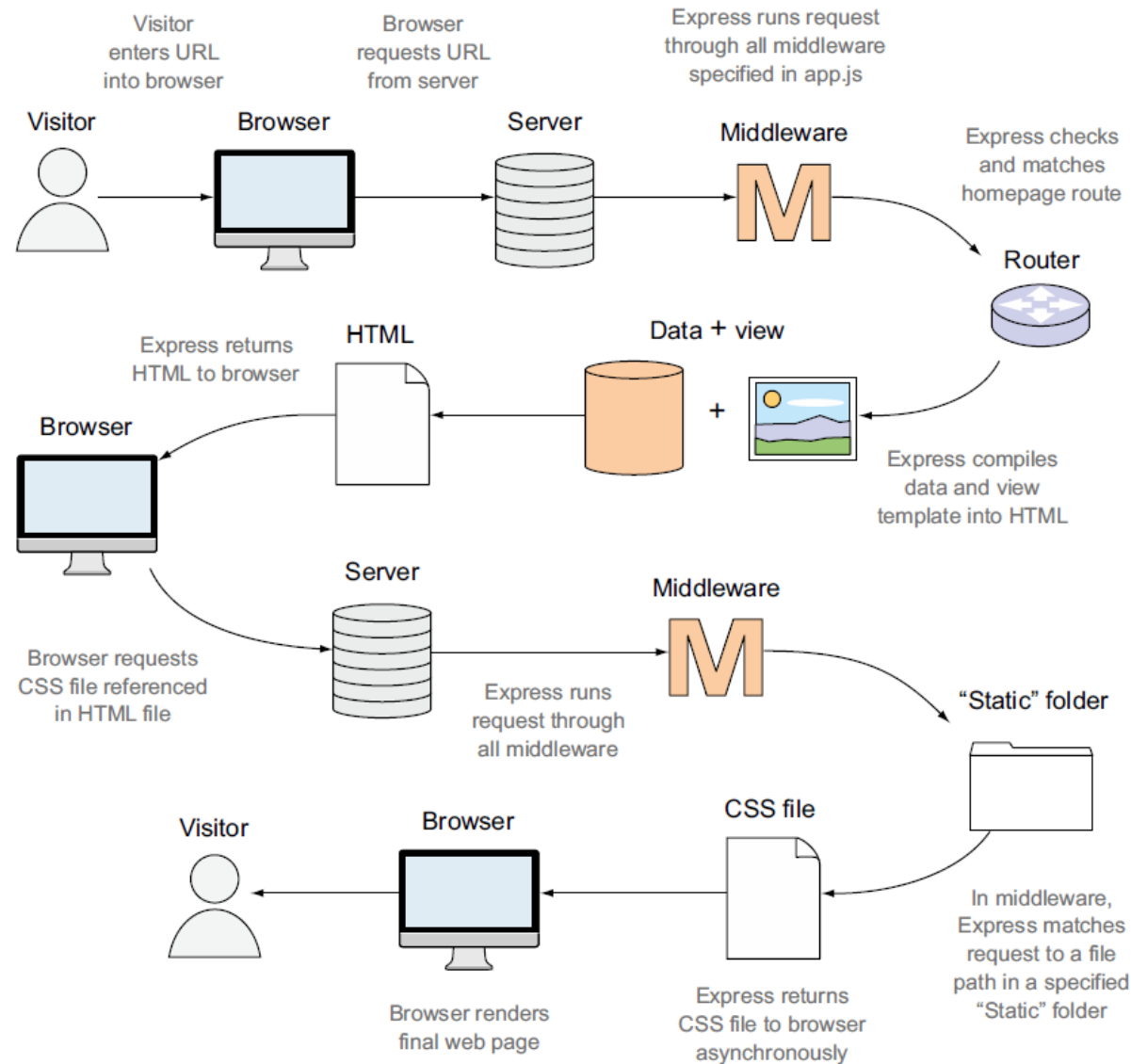
The command prompt shows the directory path "C:\demo\MeanDemos\express\ExpressDemo1" and the prompt "C:\demo\MeanDemos\express\ExpressDemo1>".

app.js

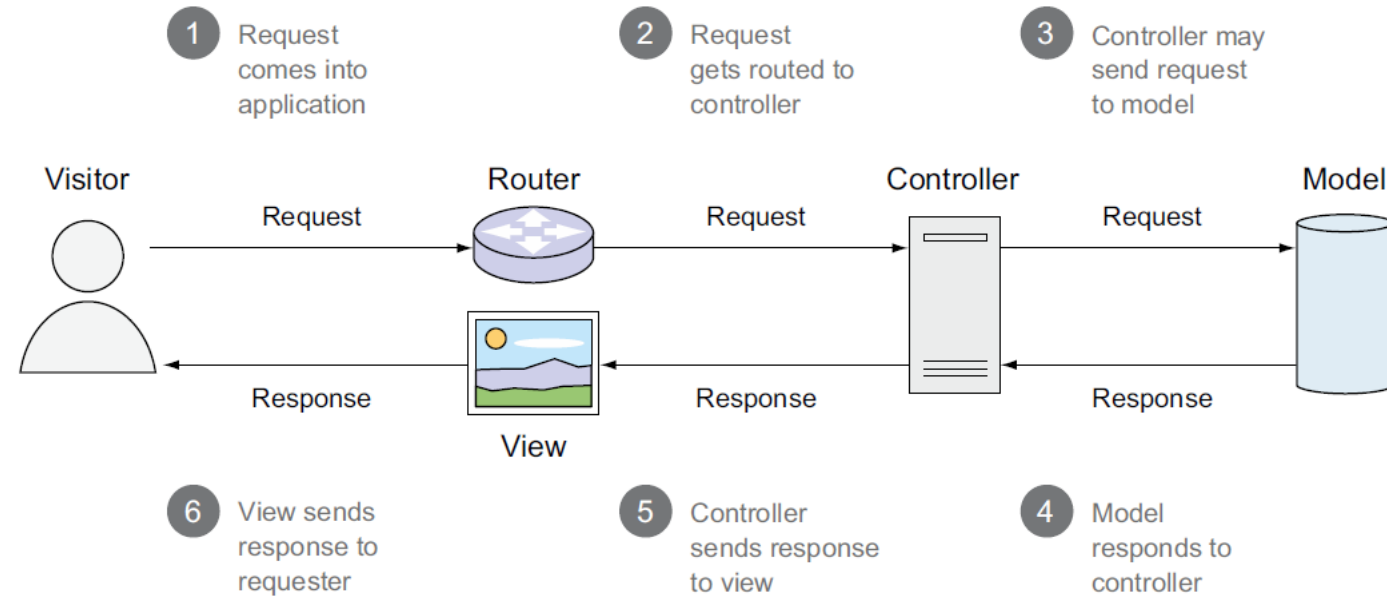
```
var createError = require('http-errors');
var express = require('express');
. . .
var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
var app = express();
// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');
// middleware setup
app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
. . .
// routes setup
app.use('/', indexRouter);
app.use('/users', usersRouter);
// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  . . .
  module.exports = app;
```

Responding to a Request



Modifying Express for MVC



New Folder Structure

- Tell Express that you have moved the views and routes folders

- In app.js change:

```
app.set('views', path.join(__dirname, 'views'));
```

To

```
app.set('views', path.join(__dirname, 'app_server', 'views'));
```

- And change:

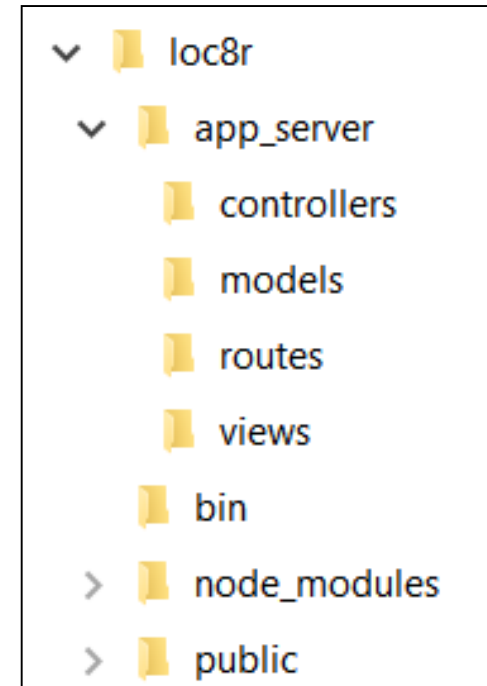
```
var routes = require('./routes/index');
```

```
var users = require('./routes/users');
```

To

```
const routes = require('./app_server/routes/index');
```

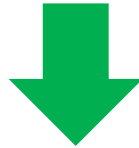
```
const users = require('./app_server/routes/users');
```



Splitting Controllers From Routes

- Controllers should manage the application logic, and routing should map URL requests to controllers

```
/* GET home page. */  
router.get('/', function (req, res, next) {  
    res.render('index', { title: 'Express' });  
});
```



```
var homepageController = function (req, res) {  
    res.render('index', { title: 'Express' });  
};  
/* GET home page. */  
router.get('/', homepageController);
```

Move the Controller to a New File

- Create a new file called **main.js** in `app_server/controllers`
- Create an export method called `index` and use it to house the `res.render` code

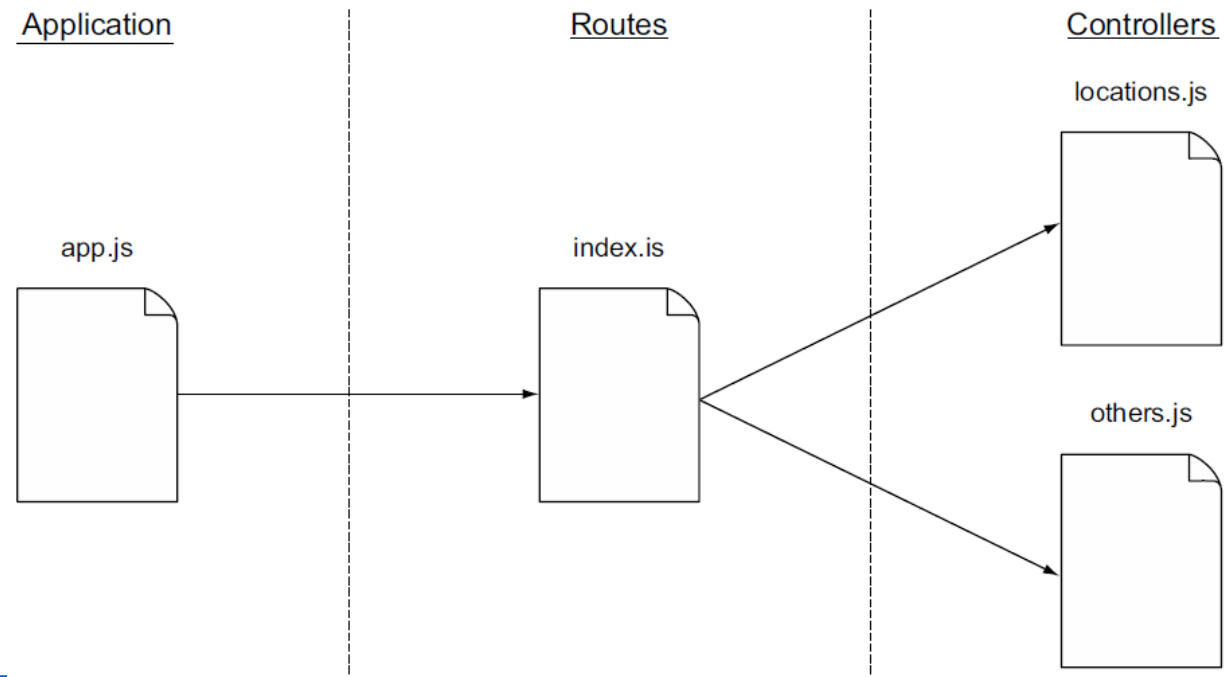
```
/* GET home page */  
module.exports.index = function (req, res) {  
    res.render('index', { title: 'Express' });  
};
```

- Require this controller module in the routes file so that we can use the exposed method in the route definition

```
const express = require('express');  
const router = express.Router();  
const ctrlMain = require('../controllers/main');  
/* GET home page. */  
router.get('/', ctrlMain.index);  
module.exports = router;
```

Defining a Routing Table

Page	URL path	Controller
List of locations (this will be the homepage)	/	Locations
Location detail	/location	Locations
Location review form	/location/review/new	Locations
About Loc8r	/about	Others



Implement Routing

- The routes/index.js file.

```
const express = require('express');
const router = express.Router();
const ctrlLocations = require('../controllers/locations');
const ctrlOthers = require('../controllers/others');

/* Locations pages */
router.get('/', ctrlLocations.homelist);
router.get('/location', ctrlLocations.locationInfo);
router.get('/location/review/new', ctrlLocations.addReview);

/* Other pages */
router.get('/about', ctrlOthers.about);
module.exports = router;
```

Routing Possibilities

- A route is a combination of a URI, a HTTP request, and one or more handlers for the endpoint
- The structure:
`app.METHOD(path, [callback...], callback)`
- Methods:
get, post, put, delete, patch , head, options, trace, copy, lock, mkcol, move, purge, propfind, proppatch, unlock, report, mkactivity, checkout, merge, m-search, notify, subscribe, unsubscribe, , search, and connect
- Route paths:
 - Strings e.g.: `app.get('/about', function (req, res) {...`
 - string patterns e.g.: `app.get('/ab*cd', function(req, res) {...`
 - regular expressions e.g.: `app.get(/.*fly$/, function(req, res) {...`

Error Handling in Express

Manual

You wrap all your call to db with try – catch:

```
app.get('/User', async function(req, res) {
  let users;
  try {
    users = await
      db.collection('User').find().toArray();
  } catch (error) {
    res.status(500).json({ error:
      error.toString() });
  }
  res.json({ users });
});
```

Smart

- Define error handling middleware

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

app.get('*', function(req, res, next) {
  // This middleware throws an error, so Express will go
  // straight to the next error handler
  throw new Error('woops');
});

app.get('*', function(req, res, next) {
  // This middleware is not an error handler (only 3
  // arguments), Express will skip it because there was an
  // error in the previous middleware
  console.log('this will not print');
});

app.use(function(error, req, res, next) {
  // Any request to this server will get here, and will
  // send an HTTP response with the error message 'woops'
  res.status(500).json({ message: error.message });
});

app.listen(3000);
```

Koa

What is koa?

- Koa is a newly popular web framework created by the team behind Express
- It aims to be a modern and more minimalist version of Express
- Koa uses new JavaScript features such as generators and async/await
- Some have even described it as Express 5.0 in spirit
- <https://koajs.com/>
- <https://scotch.io/tutorials/introduction-to-koa-the-future-of-express>

```
// app.js
const Koa = require('koa');
const app = new Koa();

app.use(async ctx => {
  ctx.body = 'Hello World';
});

app.listen(3000);
```

References & Links

- Express

<http://expressjs.com/>

- Routing

<http://expressjs.com/guide/routing.html>

<https://scotch.io/tutorials/learn-to-use-the-new-router-in-expressjs-4>

- Guide to Express Error Handling

[http://thecodebarbarian.com/80-20-guide-to-express-error-](http://thecodebarbarian.com/80-20-guide-to-express-error-handling.html?imm_mid=0f581a&cmp=em-web-na-na-newsltr)

[handling.html?imm_mid=0f581a&cmp=em-web-na-na-newsltr 20170816](http://thecodebarbarian.com/80-20-guide-to-express-error-handling.html?imm_mid=0f581a&cmp=em-web-na-na-newsltr)