FREE eBook

LEARNING express

Free unaffiliated eBook created from
Stack Overflow contributors.

#express

Table of Contents

About ... 1

Chapter 1: Getting started with express ... 2

Remarks ... 2

Versions ... 2

Versions from here ... 2

Examples ... 13

Installation ... 13

To create and run a new express server ... 13

Hello World App, using ExpressJS 4 and Node >= 4 ... 13

Preface ... 13

Installation ... 14

Directory Contents ... 14

Code ... 14

Execution ... 14

Starting an application with the Express generator ... 15

Creating an EJS app ... 16

Chapter 2: Connect ... 17

Examples ... 17

Connect and Express ... 17

Middleware ... 17

Errors and error middleware ... 18

Chapter 3: Error handling ... 19

Syntax ... 19

Parameters ... 19

Examples ... 19

Basic sample ... 19

Chapter 4: Explain Routing in Express ... 20

Examples ... 20

Express Router ... 20

Chainable route handlers for a route path by using app.route ... 20

Chapter 5: Express Database Integration 21
Examples 21
Connect to MongoDB with Node & Express 21
Chapter 6: express-generator 23
Parameters 23
Remarks 23
Examples 23
Installing Express Generator 23
Creating an App 23
Start App 23
Chapter 7: Handling static files 25
Syntax 25
Remarks 25
Examples 25
Basic Example 25
Multiple Directories Example 25
Virtual Path Prefix Example 26
Absolute Path to Static Files Directory Example 26
Absolute Path to Directory & Virtual Path Prefix Example 26
Basic static files and favicon serve example 26
Chapter 8: How does ExpressJs work 28
Examples 28
Handling request/response 28
The syntactic sugar 28
The Express App 28
Middlewares Stack 28
Chapter 9: Logging 30
Remarks 30
Examples 30
Installation 30
Simple Express logging of all request to STDOUT 30

Write Express logs to a single file ... 30
Write Express logs to a rotating log file ... 31
Chapter 10: Routing ... 32
Examples ... 32
Routing Hello World ... 32
Routing middleware ... 32
Multiple Routes ... 33
Chapter 11: using https with express ... 35
Examples ... 35
Using https with express ... 35
Chapter 12: View engine setup ... 36
Introduction ... 36
Remarks ... 36
Examples ... 36
1:setting up the views ... 36
2.EJS file example(refer 1.setting up... before this) ... 36
3.rendering view with express(refer 2.EJS file... before this) ... 36
4.after rendering final HTML is created(refer 3.rendering... before this) ... 37
Chapter 13: Writing Express Middleware ... 38
Syntax ... 38
Parameters ... 38
Remarks ... 38
Examples ... 38
Logger Middleware ... 38
requestTime Middleware ... 39
CORS Middleware ... 40
Credits ... 42

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: express

It is an unofficial and free express ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official express.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with express

Remarks

Express.js is, in the words of the developers, a "fast, unopinionated, minimalist web framework for Node.js."

Designed to be minimal and flexible, Express offers a set of features for creating web and mobile applications. From HTTP methods to built-in middleware, Express is designed to provide you with the features you need to build a web or mobile app on Node.js.

If you want to build an app on Node.js Express is a great choice, whether you use vanilla Express or one of the many frameworks either based on Express or built on top of Express. A few of these frameworks can be found here.

Versions

Versions from **here**.

Version	Notes	Release Date
4.15.3		2017-05-16
4.15.2		2017-03-06
4.15.1		2017-03-05
4.15.0		2017-03-01
4.14.1		2017-01-28
4.14.0		2016-06-16
4.13.4		2016-01-21
4.13.3		2015-08-02
4.13.2		2015-07-31
4.13.1		2015-07-05
4.13.0		2015-06-20
4.12.4		2015-05-17
4.12.3		2015-03-17

Version	Notes	Release Date
4.12.2		2015-03-02
4.12.1		2015-03-01
4.12.0		2015-02-23
4.11.2		2015-01-20
4.11.1		2015-01-20
4.11.0		2015-01-13
4.10.8		2015-01-13
4.10.7		2015-01-04
4.10.6		2014-12-12
4.10.5		2014-12-10
4.10.4		2014-11-24
4.10.3		2014-11-23
4.10.2		2014-11-09
4.10.1		2014-10-28
4.10.0		2014-10-23
4.9.8		2014-10-17
4.9.7		2014-10-10
4.9.6		2014-10-08
4.9.5		2014-09-24
4.9.4		2014-09-19
4.9.3		2014-09-18
4.9.2		2014-09-17
4.9.1		2014-09-16
4.9.0		2014-09-08
4.8.8		2014-09-04

Version	Notes	Release Date
4.8.7		2014-08-29
4.8.6		2014-08-27
4.8.5		2014-08-18
4.8.4		2014-08-14
4.8.3		2014-08-10
4.8.2		2014-08-07
4.8.1		2014-08-06
4.8.0		2014-08-05
4.7.4		2014-08-04
4.7.3		2014-08-04
4.7.2		2014-07-27
4.7.1		2014-07-26
4.7.0		2014-07-25
4.6.1		2014-07-12
4.6.0		2014-07-11
4.5.1		2014-07-06
4.5.0		2014-07-04
4.4.5		2014-06-26
4.4.4		2014-06-20
4.4.3		2014-06-11
4.4.2		2014-06-09
4.4.1		2014-06-02
4.4.0		2014-05-30
4.3.2		2014-05-28
4.3.1		2014-05-23

Version	Notes	Release Date
4.3.0		2014-05-21
4.2.0		2014-05-11
4.1.2		2014-05-08
4.1.1		2014-04-27
4.1.0		2014-04-24
4.0.0		2014-04-09
3.21.2	From here to	2015-07-31
3.21.1	3.18.6 dates	2015-07-05
3.21.0	seem wrong	2015-06-18
3.20.3		2015-05-17
3.20.2		2015-03-16
3.20.1		2015-02-28
3.20.0		2015-02-18
3.19.2		2015-02-01
3.19.1		2015-01-20
3.19.0		2015-01-09
3.18.6		2014-12-12
3.18.5		2014-12-11
3.18.4		2014-11-23
3.18.3		2014-11-09
3.18.2		2014-10-28
3.18.1		2014-10-22
3.18.0		2014-10-17
3.17.8		2014-10-15
3.17.7		2014-10-08

Version	Notes	Release Date
3.17.6		2014-10-02
3.17.5		2014-09-24
3.17.4		2014-09-19
3.17.3		2014-09-18
3.17.2		2014-09-15
3.17.1		2014-09-08
3.17.0		2014-09-08
3.16.10		2014-09-04
3.16.9		2014-08-29
3.16.8		2014-08-27
3.16.7		2014-08-18
3.16.6		2014-08-14
3.16.5		2014-08-11
3.16.4		2014-08-10
3.16.3		2014-08-07
3.16.2		2014-08-07
3.16.1		2014-08-06
3.16.0		2014-08-05
3.15.3		2014-08-04
3.15.2		2014-07-27
3.15.1		2014-07-26
3.15.0		2014-07-22
3.14.0		2014-07-11
3.13.0		2014-07-03
3.12.1		2014-06-26

Version	Notes	Release Date
3.12.0		2014-06-21
3.11.0		2014-06-19
3.10.5		2014-06-11
3.10.4		2014-06-09
3.10.3		2014-06-05
3.10.2		2014-06-03
3.10.1		2014-06-03
3.10.0		2014-06-02
3.9.0		2014-05-30
3.8.1		2014-05-27
3.8.0		2014-05-21
3.7.0		2014-05-18
3.6.0		2014-05-09
3.5.3		2014-05-08
3.5.2		2014-04-24
3.5.1		2014-03-25
3.5.0		2014-03-06
3.4.8		2014-01-13
3.4.7		2013-12-10
3.4.6		2013-12-01
3.4.5		2013-11-27
3.4.4		2013-10-29
3.4.3		2013-10-23
3.4.2		2013-10-18
3.4.1		2013-10-15

Version	Notes	Release Date
3.4.0		2013-09-07
3.3.8		2013-09-02
3.3.7		2013-08-28
3.3.6		2013-08-27
3.3.4		2013-07-08
3.3.3		2013-07-04
3.3.2		2013-07-03
3.3.1		2013-06-27
3.3.0		2013-06-27
3.2.6		2013-06-02
3.2.5		2013-05-21
3.2.4		2013-05-09
3.2.3		2013-05-07
3.2.2		2013-05-03
3.2.1		2013-04-29
3.2.0		2013-04-15
3.1.2		2013-04-12
3.1.1		2013-04-01
3.1.0		2013-01-25
3.0.6		2013-01-04
3.0.5		2012-12-19
3.0.4		2012-12-05
3.0.3		2012-11-13
3.0.2		2012-11-08
3.0.1		2012-11-01

Version	Notes	Release Date
3.0.0		2012-10-23
3.0.0rc5		2012-09-18
3.0.0rc4		2012-08-30
3.0.0rc3		2012-08-13
3.0.0rc2		2012-08-03
3.0.0rc1		2012-07-24
3.0.0beta7		2012-07-16
3.0.0beta6		2012-07-13
3.0.0beta5		2012-07-03
3.0.0beta4		2012-06-25
3.0.0beta3		2012-06-15
3.0.0beta2		2012-06-06
3.0.0beta1		2012-06-01
3.0.0alpha5		2012-05-30
3.0.0alpha4		2012-05-09
3.0.0alpha3		2012-05-04
3.0.0alpha2		2012-04-26
3.0.0alpha1		2012-04-15
2.5.9		2012-04-02
2.5.8		2012-02-08
2.5.7		2012-02-06
2.5.6		2012-01-13
2.5.5		2012-01-08
2.5.4		2012-01-02
2.5.3		2011-12-30

Version	Notes	Release Date
2.5.2		2011-12-10
2.5.1		2011-11-17
2.5.0		2011-10-24
2.4.7		2011-10-05
2.4.6		2011-08-22
2.4.5		2011-08-19
2.4.4		2011-08-05
2.4.3		2011-07-14
2.4.2		2011-07-06
2.4.1		2011-07-06
2.4.0		2011-06-28
2.3.12		2011-06-22
2.3.11		2011-06-04
2.3.10		2011-05-27
2.3.9		2011-05-25
2.3.8		2011-05-24
2.3.7		2011-05-23
2.3.6		2011-05-20
2.3.5		2011-05-20
2.3.4		2011-05-08
2.3.3		2011-05-03
2.3.2		2011-04-27
2.3.1		2011-04-26
2.3.0		2011-04-25
2.2.2		2011-04-12

Version	Notes	Release Date
2.2.1		2011-04-04
2.2.0		2011-03-30
2.1.1		2011-03-29
2.1.0		2011-03-24
2.0.0		2011-03-17
2.0.0rc3		2011-03-17
2.0.0rc2		2011-03-17
2.0.0rc		2011-03-14
2.0.0beta3		2011-03-09
2.0.0beta2		2011-03-07
2.0.0beta		2011-03-03
1.0.8		2011-03-01
1.0.7		2011-02-07
1.0.6		2011-02-07
1.0.5		2011-02-05
1.0.4		2011-02-05
1.0.3		2011-01-13
1.0.2		2011-01-10
1.0.1		2010-12-29
1.0.0		2010-11-16
1.0.0rc4		2010-10-14
1.0.0rc3		2010-09-20
1.0.0rc2		2010-08-17
1.0.0rc		2010-07-28
1.0.0beta2		2010-07-23

Version	Notes	Release Date
1.0.0beta		2010-07-15
0.14.0		2010-06-15
0.13.0		2010-06-01
0.12.0		2010-05-22
0.11.0		2010-05-06
0.10.1		2010-05-03
0.10.0		2010-04-30
0.9.0		2010-04-14
0.8.0		2010-03-19
0.7.6		2010-03-19
0.7.5		2010-03-16
0.7.4		2010-03-16
0.7.3		2010-03-16
0.7.2		2010-03-16
0.7.1		2010-03-16
0.7.0		2010-03-15
0.6.0		2010-03-11
0.5.0		2010-03-10
0.4.0		2010-02-11
0.3.0		2010-02-11
0.2.1		2010-02-05
0.2.0		2010-02-03
0.1.0		2010-02-03
0.0.2		2010-01-10
0.0.1	Intial Release	2010-01-03

Examples

Installation

Express JS is the goto framework for developing `Web Applications`, `APIs` and almost any kind of `Backend` using Node.

To install `express`, all you have to do is run the **npm** command

```
npm install express --save
```

And you're done.

To create and run a new express server

create a file `app.js` and add this code

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

// when "/" is opened in url, this function will be called.
app.get('/', function (req, res) {
  res.json({ code: 200, message: 'success' });
})

app.listen( 3000, function () {
  console.log('Express server running at http://localhost:3000');
});
```

- In your terminal, run `node app.js` and
- Open the url `http://localhost:3000` in web browser to see your newly created express server.

It's also a good idea to install `body-parser` and `express-session` along with `express` as most of the time you will want to read the data sent in `POST` request and manage user sessions.

- body-parser on github
- express-session on github

Hello World App, using ExpressJS 4 and Node >= 4

Preface

You'll need `node >= 4` and `express 4` for this project. You can get the latest `node` distribution from their download page.

Before this tutorial, you should initialize your node project by running

```
$ npm init
```

from the command line and filling in the information you want. Note that you can change the info at any time by editing the `package.json` file.

Installation

Install `express` with `npm`:

```
$ npm install --save express
```

After installing Express as a node module, we can create our entry point. This should be in the same directory as our `package.json`

```
$ touch app.js
```

Directory Contents

The folder should have the following directory structure:

```
<project_root>
 |-> app.js
 |-> node_modules/
 '-> package.json
```

Code

Open `app.js` in your preferred editor and follow these four steps to create your first Express app:

```
// 1. Import the express library.
import express from 'express';

// 2. Create an Express instance.
const app = express();

// 3. Map a route. Let's map it to "/", so we can visit "[server]/".
app.get('/', function(req, res) {
   res.send('Hello World');
});

// 4. Listen on port 8080
app.listen(8080, function() {
   console.log('Server is running on port 8080...');
});
```

Execution

From the project directory, we can run our server using the command

```
$ node app.js
```

You should see the text

```
$ Our Express App Server is listening on 8080...
```

Now, visit `http://localhost:8080/` and you'll see the text "Hello World!"

Congratulations, you've created your first Express app!

Starting an application with the Express generator

To get started quickly with Express, you can use the Express generator which will create an application skeleton for you.

First, install it globally with npm:

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

You may need to put `sudo` before this command if you get a "permission denied" error.

Once the generator is installed, you can start a new project like this:

```
express my_app
```

The above command will create a folder called `my_app` with a `package.json` file, an `app.js` file, and a few subfolders like `bin`, `public`, `routes`, `views`.

Now navigate to the folder and install the dependencies:

```
cd first_app
npm install
```

If you're on Linux or macOS, you can start the app like this:

```
DEBUG=myapp:* npm start
```

Or, if you're on Windows:

```
set DEBUG=myapp:* & npm start
```

Now, load http://localhost:3000/ in your web browser and you should see the words "Welcome to

Express".

Creating an EJS app

```
a@coolbox:~/workspace$ express --ejs my-app
a@coolbox:~/workspace$ cd my-app
a@coolbox:~/workspace/my-app$ npm install
a@coolbox:~/workspace/my-app$ npm start
```

Read Getting started with express online: https://riptutorial.com/express/topic/1616/getting-started-with-express

Chapter 2: Connect

Examples

Connect and Express

Express is based on Connect, which is what provides the middleware functionality of Express. To understand what connect is, you can see that it provides the basic app structure that you use when you use express

```
const connect = require('connect')

const app = connect()
app.listen(3000)
```

This will open a "empty" http server that will respond 404 to all requests.

Middleware

Middleware are attached to the app object, usually before listen is called. Example of a simple logging middleware:

```
app.use(function (req, res, next) {
    console.log(`${req.method}: ${req.url}`)
    next()
})
```

All this will do is log `GET: /example` if you where to GET `localhost:3000/example`. All requests will still return 404 since you are not responding with any data.

The next middleware in the chain will be run as soon as the previous one calls `next()`, so we can go ahead and respond to the requests by adding yet another middleware like this:

```
app.use(function (req, res, next) {
    res.end(`You requested ${req.url}`)
})
```

Now when you request ´localhost:3000/example`you will be greeted with "You requested /example". There is no need to call`next` this time since this middleware is the last in the chain (but nothing bad will happen if you did),

Complete program this far:

```
const connect = require('connect')

const app = connect()

app.use(function (req, res, next) {
```

```
    console.log(`${req.method}: ${req.url}`)
    next()
})

app.use(function (req, res, next) {
    res.end(`You requested ${req.url}`)
    next()
})

app.listen(3000)
```

Errors and error middleware

If we would like to limit the access to our app, we could write a middleware for that too! This example only grants you access on thrusdays, but a real world example could, for example, be *user authentication*. A good place to put this would be after the logging middleware but before any content is sent.

```
app.use(function (req, res, next) {
    if (new Date().getDay() !== 4) {
        next('Access is only granted on thursdays')
    } else {
        next()
    }
})
```

As you can see in this example, sending an error is as easy as providing a prameter to the `next()` function.

Now, if we visit the website on any day different than a thursday we would be greeted with a 500 error and the string `'Access is only granted on thursdays'`.

Now, this isn't good enough for our site. We would rather send the user a HTML message in another middleware:

```
app.use(function (err, req, res, next) {
    res.end(`<h1>Error</h1><p>${err}</p>`)
})
```

This works kind of like a catch block: any error in the middleware prior to the error middleware will be sent to the former. An error middleware is identified by its 4 parameters.

You could also use the error middleware to recover from the error by calling the next method again:

```
app.use(function (err, req, res, next) {
    // Just joking, everybody is allowed access to the website!
    next()
})
```

Read Connect online: https://riptutorial.com/express/topic/4031/connect

Chapter 3: Error handling

Syntax

- `app.use(function(err, req, res, next) {})` // Basic middleware

Parameters

Name	Description
`err`	Object with error information
`req`	HTTP request object
`res`	HTTP response object
`next`	function used to start next middleware execution

Examples

Basic sample

Unlike other middleware functions error-handling middleware functions have four arguments instead of three: `(err, req, res, next)`.

Sample:

```
app.use(function(err, req, res, next) {
  console.error(err.stack);
  res.status(500).send('Error found!');
});
```

Read Error handling online: https://riptutorial.com/express/topic/2739/error-handling

Chapter 4: Explain Routing in Express

Examples

Express Router

Express router allows you to create multiple "mini apps" so you can namespace your api, public, auth and other routes into separate routing systems.

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

router.get('/', function(req, res){
   res.send('Get request received');
});

router.post('/', function(req, res){
   res.send('Post requestreceived');
});

app.use('/', router);

app.listen(8080);
```

Chainable route handlers for a route path by using app.route

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

app.route('/user')
  .get(function (req, res) {
    res.send('Get a random user')
  })
  .post(function (req, res) {
    res.send('Add a user')
  })
  .put(function (req, res) {
    res.send('Update the user details')
  })
  .delete(function (req, res) {
    res.send('Delete a user')
  });
```

Read Explain Routing in Express online: https://riptutorial.com/express/topic/6536/explain-routing-in-express

Chapter 5: Express Database Integration

Examples

Connect to MongoDB with Node & Express

First, ensure you have installed *mongodb* and *express* via npm. Then, in a file conventionally titled *db.js*, use the following code:

```
var MongoClient = require('mongodb').MongoClient

var state = {
    db: null,
}

exports.connect = function(url, done) {
    if (state.db) return done()

    MongoClient.connect(url, function(err, db) {
        if(err) return done(err)
        state.db = db
        done()
    })
}

exports.get = function() {
    return state.db
}

exports.close = function(done) {
    if (state.db) {
        state.db.close(function(err, result) {
            state.db = null;
            state.mode = null;
            done(err);
        })
    }
}
```

This file will connect to the database and then you can just use the **db** object returned by the **get** method.

Now you need to include the db file by requiring it in you app.js file. Assuming your *db.js* file is in the same directory as *app.js* you can insert the line:

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

This, however, does not actually connect you to your MongoDB instance. To do that insert the following code before your app.listen method is called. In our example we integrate error handling and the app.listen method into the database connection. Please note this code only works if you are running your mongo instance on the same machine you Express app is located on.

```
db.connect('mongodb://localhost:27017/databasename', function(err) {
    if (err) {
        console.log('Unable to connect to Mongo.');
        process.exit(1);
    } else {
        app.listen(3000, function() {
            console.log('Listening on port 3000...');
        });
    }
});
```

There you go, your Express app should now be connected to your Mongo DB. Congrats!

Read Express Database Integration online: https://riptutorial.com/express/topic/7002/express-database-integration

Chapter 6: express-generator

Parameters

Parameter	Definition
-h, --help	output usage information
-V, --version	output the version number
-e, --ejs	add pjs (Embedded JavaScript) templating engine support (defaults to jade, which has been renamed to Pug)
--hbs	add handlebars templating engine support
-H, --hogan	add hogan.js engine support
--git	add .gitignore
-f, --force	force on non-empty directory
-c <engine>, --css <engine>	add stylesheet <engine> support (less, stylus ,compass, sass) (default is css)

Remarks

Express generator is a great tool for getting a project up and rolling quickly. Once you understand the organization it implements, it's a real time saver.

Examples

Installing Express Generator

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

Creating an App

```
express my-app
```

Start App

Using start option

```
npm start
```

Using Nodemon

```
nodemon
```

Using forever

```
forever start 'js file name'
```

To stop in forever

```
forever stop ''js file name'
```

To restart in forever

```
forever restart 'js filename'
```

List the server ruuning using forever

```
forever list
```

Read express-generator online: https://riptutorial.com/express/topic/4512/express-generator

Chapter 7: Handling static files

Syntax

1. To serve static files (Images, CSS, JS files, etc.) use the **express.static** middleware function.
2. Pass the name of the directory that contains the assets to **express.static** to serve the files directly. (Look to the *Basic Example*)
3. You can use multiple directories, simply call the **express.static** multiple times. Remember, Express looks up files in the order you set the directories with **express.static**. (Look to the *Multiple Directories Example*)
4. You can create a virtual path prefix (i.e. one where the path does not actually exist in the file system) with **express.static**, just specify a mount path. (Look to the *Virtual Path Prefix Example*)
5. All of the preceding paths have been relative to the directory from where you launch the *node* process. So, it is generally safer to use the absolute path of the directory you want to serve. (Look to the *Absolute Path to Static Files Directory Example*)
6. You can mix and match the options of this method, as seen in the *Absolute Path to Directory & Virtual Path Prefix Example*

Remarks

All of the examples can be run in node. Simply copy and paste into a node project with Express installed and run them with **node filename**. For an example of how to install express click here and ensure you have npm installed then follow the instructions on installing packages to install "express."

Examples

Basic Example

```
// Basic code for Express Instance
var express = require('express');
var app = express();

// Serve static files from directory 'public'
app.use(express.static('public'));

// Start Express server
app.listen(3030);
```

Multiple Directories Example

```
// Set up Express
var express = require('express');
```

```
var app = express();

// Serve static assets from both 'public' and 'files' directory
app.use(express.static('public');
app.use(express.static('files');

// Start Express server
app.listen(3030);
```

Virtual Path Prefix Example

```
// Set up Express
var express = require('express');
var app = express();

// Specify mount path, '/static', for the static directory
app.use('/static', express.static('public'));

// Start Express server
app.listen(3030);
```

Absolute Path to Static Files Directory Example

```
// Set up Express
var express = require('express');
var app = express();

// Serve files from the absolute path of the directory
app.use(express.static(__dirname + '/public'));

// Start Express server
app.listen(3030);
```

Absolute Path to Directory & Virtual Path Prefix Example

```
// Set up Express
var express = require('express');
var app = express();

/*  Serve from the absolute path of the directory that you want to serve with a
 */ virtual path prefix
app.use('/static', express.static(__dirname + '/public'));

// Start Express server
app.listen(3030);
```

Basic static files and favicon serve example

```
var express = require('express');
var path = require('path');
var favicon = require('serve-favicon');

var app = express();
```

```
app.use(favicon(__dirname + '/public/img/favicon.ico'));
app.use(express.static(path.join(__dirname, 'public')));

app.listen(3000, function() {
  console.log("Express App listening on port 3000");
})
```

Read Handling static files online: https://riptutorial.com/express/topic/6954/handling-static-files

Chapter 8: How does ExpressJs work

Examples

Handling request/response

The syntactic sugar

Most of the getting started examples of ExpressJs include this piece of code

```
var express = require('express');
var app = express();
...
app.listen(1337);
```

Well, `app.listen` is just a shortcut for:

```
var express = require('express');
var app = express();
var http = require('http');
http.createServer(app).listen(1337);
```

The Express App

The famous `http.createServer` accept a function which is known as the handler. The handler takes 2 parameters **request** and **response** as inputs, then manipulating them inside it's scope to do various things.

So basically `app = express()` is a function, taking place as the handler and dealing with request, response through a set of special components referred as middlewares.

Middlewares Stack

A basic middleware is a function that takes 3 arguments **request**, **response** and **next**.

Then by `app.use`, a middleware is mounted to the Express App Middlewares Stack. Request and response are manipulated in each middleware then piped to the next one through the call of `next()` .

For example, the below code:

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

app.use((request, response, next) => {
```

```
    request.propA = "blah blah";
    next();
});

app.use('/special-path', (request, response, next) => {
    request.propB = request.propA + " blah";
    if (request.propB === "blah blah blah")
        next();
    else
        response.end('invalid');
});

app.use((request, response, next) => {
    response.end(request.propB);
});

app.listen(1337);
```

Can roughly be translated to:

```
var http = require('http');
http.createServer((request, response) => {

    //Middleware 1
    if (isMatch(request.url, '*')) {
        request.propA = "blah blah";
    }

    //Middleware 2
    if (isMatch(request.url, "/special-path")) {
        request.propB = request.propA + " blah";
        if (request.propB !== "blah blah blah")
            return response.end('invalid');
    }

    //Middleware 3
    if (isMatch(request.url, "*")) {
        return response.end(request.propB);
    }
});

server.listen(1337);
```

Read How does ExpressJs work online: https://riptutorial.com/express/topic/7815/how-does-expressjs-work

Chapter 9: Logging

Remarks

`morgan` is an HTTP request logger middleware for node.js

Examples

Installation

First, install the `morgan` Middleware in your project

```
npm install --save morgan
```

Simple Express logging of all request to STDOUT

Add the following code to your `app.js` file:

```
var express = require('express')
var morgan = require('morgan')

var app = express()

app.use(morgan('combined'))

app.get('/', function (req, res) {
  res.send('hello, world!')
})
```

Now when you access your website you will see in the console you used to start the server that the requests are logged

Write Express logs to a single file

First, install `fs` and `path` in your project

```
npm install --save fs path
```

Add the following code to your `app.js` file:

```
var express = require('express')
var fs = require('fs')
var morgan = require('morgan')
var path = require('path')

var app = express()

// create a write stream (in append mode)
```

```
var accessLogStream = fs.createWriteStream(path.join(__dirname, 'access.log'), {flags: 'a'})

// setup the logger
app.use(morgan('combined', {stream: accessLogStream}))

app.get('/', function (req, res) {
  res.send('hello, world!')
})
```

Now when you access your website you will see a `access.log` file was created in your project directory

Write Express logs to a rotating log file

First, install `fs`, `file-stream-rotator` and `path` in your project

```
npm install --save fs file-stream-rotator path
```

Add the following code to your `app.js` file:

```
var FileStreamRotator = require('file-stream-rotator')
var express = require('express')
var fs = require('fs')
var morgan = require('morgan')
var path = require('path')

var app = express()
var logDirectory = path.join(__dirname, 'log')

// ensure log directory exists
fs.existsSync(logDirectory) || fs.mkdirSync(logDirectory)

// create a rotating write stream
var accessLogStream = FileStreamRotator.getStream({
  date_format: 'YYYYMMDD',
  filename: path.join(logDirectory, 'access-%DATE%.log'),
  frequency: 'daily',
  verbose: false
})

// setup the logger
app.use(morgan('combined', {stream: accessLogStream}))

app.get('/', function (req, res) {
  res.send('hello, world!')
})
```

Now when you access your website you will see a `log` directory was created and a log file with a name format of `access-%DATE%.log` was created in your log directory

Read Logging online: https://riptutorial.com/express/topic/7191/logging

Chapter 10: Routing

Examples

Routing Hello World

The main app file loads the routes file where routes are defined.

app.js

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

app.use('/', require('./routes'));

app.listen('3000');
```

routes.js

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

router.get('/', function(req, res) {
    res.send('Hello World!');
});

module.exports = router;
```

Routing middleware

Middleware is executed prior to the route execution and can decide whether to execute the router according to the URL.

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

router.use(function (req, res, next) {
    var weekDay = new Date().getDay();
    if (weekDay === 0) {
        res.send('Web is closed on Sundays!');
    } else {
        next();
    }
})

router.get('/', function(req, res) {
    res.send('Sunday is closed!');
});

module.exports = router;
```

Specific middleware can also be sent to each router handler.

```
var closedOnSundays = function (req, res, next) {
    var weekDay = new Date().getDay();
    if (weekDay === 0) {
        res.send('Web is closed on Sundays!');
    } else {
        next();
    }
}

router.get('/', closedOnSundays, function(req, res) {
    res.send('Web is open');
});

router.get('/open', function(req, res) {
    res.send('Open all days of the week!');
});
```

Multiple Routes

The main app file loads any routes files in which you would like to define routes. To do so we need the following directory structure: app.js routes/index.js routes/users.js

app.js

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

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

app.listen('3000');
```

routes/index.js

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

router.get('/', function(req, res) {
    res.send('Index Page');
});

router.get('/about', function(req, res) {
    res.send('About Page');
});

module.exports = router;
```

routes/users.js

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

router.get('/', function(req, res) {
    res.send('Users Index Page');
});

router.get('/list', function(req, res) {
    res.send('Users List Page');
```

```
});

module.exports = router;
```

Running `$ node app.js` there should now be pages at the following urls:

- localhost:3000/ - Displays "Index Page"
- localhost:3000/about - Displays "About Page"
- localhost:3000/users - Displays "Users Index Page"
- localhost:3000/users/list - Displays "Users List Page"

Read Routing online: https://riptutorial.com/express/topic/2589/routing

Chapter 11: using https with express

Examples

Using https with express

First you have to generate public and private keys using OpenSSL(tutorial).

```
var express = require("express");
var http =require ("http");
var https=require ("https");
var fs=require("fs");
var app=express();
var httpsKeys={
key:fs.readFileSync("<key.pem>");
crtifcte:fs.readFileSync("<certificate.pem>");
};
http.createserver(app).listen(3000);
https.createserver(httpsKeys,app).listen(3030);
```

Read using https with express online: https://riptutorial.com/express/topic/7844/using-https-with-express

Chapter 12: View engine setup

Introduction

Often the server needs to serve pages dynamically.For an example an user Mr.X visits the page and sees some thing like "Welcome Mr. X to my homepage".In this case views can be helpful.Even to populate a table view can be handy. Variables can be injected into HTML dynamically using view engine.View engine is something that renders the views.One can keep views to be served in a folder called view and serve upon request .The path of the folder can be shown to Express using path.resolve method .

Remarks

install ejs using the following(I know it's obvious)

```
sudo npm install ejs --save
```

Examples

1:setting up the views

```
var express=require("express");    //express is included
var path=require("path");    //path is included

var app=express();    //app is an Express type of application

app.set("views",path.resolve(__dirname,"views"));    //tells express about the location of the
views in views folder
app.set("view engine","ejs");    //tells express that ejs template engine is used
```

2.EJS file example(refer 1.setting up... before this)

the following is an ejs file.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Hello, world!</title>
    </head>
    <body>
        <%= message %>
    </body>
</html>
```

3.rendering view with express(refer 2.EJS file... before this)

```
app.get("/",function(req,res){
response.render("index",{                    //render the index when root(/) is requested
    message:"rendered view with ejs"
        });
});
```

4.after rendering final HTML is created(refer 3.rendering... before this)

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello, world!</title>
</head>
<body>
    message:"rendered view with ejs"
</body>
</html>
```

Read View engine setup online: https://riptutorial.com/express/topic/8104/view-engine-setup

Chapter 13: Writing Express Middleware

Syntax

1. Specify the instance of express you want to use. This is commonly *app*.
2. Define the HTTP method for which the function applies. In the example, this is *get*.
3. Define the path to which the function applies. In the example, this is *'/'*.
4. Define as a function with the *function* keyword.
5. Add the required parameters: req, res, next. (See note in remarks section)
6. Put some code in the function to do whatever you want

Parameters

Parameter	Details
req	The request object.
res	The response object.
next	The next() middleware call.

Remarks

A middleware function is a function with access to the request object (*req*), the response object (*res*), and the *next()* middleware function in the application's request-response cycle. The *next()* middleware function is commonly denoted by a variable named next.

Middleware functions are designed to perform the following tasks:

- Execute any code.
- Make changes to the request and response objects. (See the requestTime example)
- End the request-response cycle.
- Call the next middleware in the stack. (By calling the *next()* middleware)

Note: It doesn't have to be named next. But if you use something else no one will know what you mean and you will be fired. And your code won't work. So, just name it next. This rule applies to the request and response object. Some people will use request and response instead of req and res, respectively. That's fine. It wastes keystrokes, but it's fine.

Examples

Logger Middleware

If you are new to middleware in Express check out the Overview in the Remarks section.

First, we are going to setup a simple Hello World app that will be referenced and added to during the examples.

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

app.get('/', function(req, res) {
    res.send('Hello World!');
});

app.listen(3000);
```

Here is a simple middleware function that will log "LOGGED" when it is called.

```
var myLogger = function (req,res,next) {
    console.log('LOGGED');
    next();
};
```

*Calling **next()** invokes the next middleware function in the app.*

To load the function call app.use() and specify the function you wish to call. This is done in the following code block that is an extension of the Hello World block.

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

var myLogger = function (req, res, next) {
    console.log('LOGGED');
    next();
};

app.use(myLogger);

app.get('/', function(req, res) {
    res.send('Hello World!');
});

app.listen(3000);
```

Now every time the app receives a request it prints the message "LOGGED" to the terminal. So, how do we add more specific conditions to when middleware is called? Look at the next example and see.

requestTime Middleware

Let's create middleware that adds a property called requestTime to the request object.

```
var requestTime = function (req, res, next) {
    req.requestTime = Date.now();
    next();
};
```

Now let's modify the logging function from the previous example to utilize the requestTime middleware.

```
myLogger = function (req, res, next, requestTime) {
    console.log('LOGGED at ' + requestTime);
    next();
};
```

Let's add the middleware to our app:

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

myLogger = function (req, res, next) {
    console.log('LOGGED at ' + req.requestTime);
    next();
};

var requestTime = function(req, res, next) {
    req.requestTime = Date.now();
    next();
};

app.use(requestTime);

app.use(myLogger);

app.get('/', function(req, res) {
    res.send('Hello World!');
});

app.listen(3000);
```

Now the app will log the time at which the request was made. This covers the basics of writing and using Express middleware. For more information see Using Express Middleware.

!!!TODO: Create Using Express Middleware Section!!!

CORS Middleware

This example demonstrates how a cross origin http request can be handled using a middleware.

CORS Background

CORS is an access control method adopted by all major browsers to avert Cross Scripting Vulnerabilities inherent by them. In general browser security, scripts should maintain that all XHR requests has to be made only to the source the same scripts are served from. If an XHR request is made outside the domain the scripts are belonging to, the response will be rejected.

However if the browser supports CORS, it would make an exception to this rule if appropriate headers in the response indicate that the domain which the request is originated from is allowed. The following header indicates that any domain is allowed:

```
Access-Control-Allow-Origin: *
```

Example

Following example shows how Express middleware can include these headers in it's response.

```
app.use(function(request, response, next){

    response.header('Access-Control-Allow-Origin', '*');
    response.header('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE,OPTIONS');
    response.header('Access-Control-Allow-Headers', 'Content-Type, Authorization, Content-
Length, X-Requested-With');

   //Handle Preflight
   if (reqest.method === 'OPTIONS') {
      response.status(200).send();
   }
   else {
      next();
   }

});
```

Handling Preflight

The latter part of the above example handles Preflight. Preflight is a special OPTIONS request the browser send to test CORS if the request contain custom headers.

Useful References

MDN - CORS Http Tutorial

Read Writing Express Middleware online: https://riptutorial.com/express/topic/6993/writing-express-middleware

Credits

S. No	Chapters	Contributors
1	Getting started with express	Akshay Khale, Community, David Vogel, Dima Grossman, dkimot, Everettss, Gregory Worrall, Guillaume Lrv, Jared Hooper, jawadhoot, Kilmazing, Random User, Sumner Evans, user6939352
2	Connect	Henrik Karlsson, Overflowh
3	Error handling	gevorg, jawadhoot, Kilmazing
4	Explain Routing in Express	Dima Grossman, Sujithrao
5	Express Database Integration	dkimot
6	express-generator	rickrizzo, Rupali Pemare
7	Handling static files	dkimot, Sujithrao
8	How does ExpressJs work	rocketspacer
9	Logging	Mor Paz
10	Routing	jawadhoot, Kelvin, phobos, S.L. Barth, zurfyx
11	using https with express	nilakantha singh deo
12	View engine setup	Daniele Giussani, nilakantha singh deo
13	Writing Express Middleware	Charlie H, dkimot