Express

Fast, unopinionated, minimalist web framework for Node.js

Express is used to create web applications, from serving up web pages to creating REST web services.

Express aims to provide only the necessary features, letting you extend them as necessary using what’s called Middleware.

Base Functionality

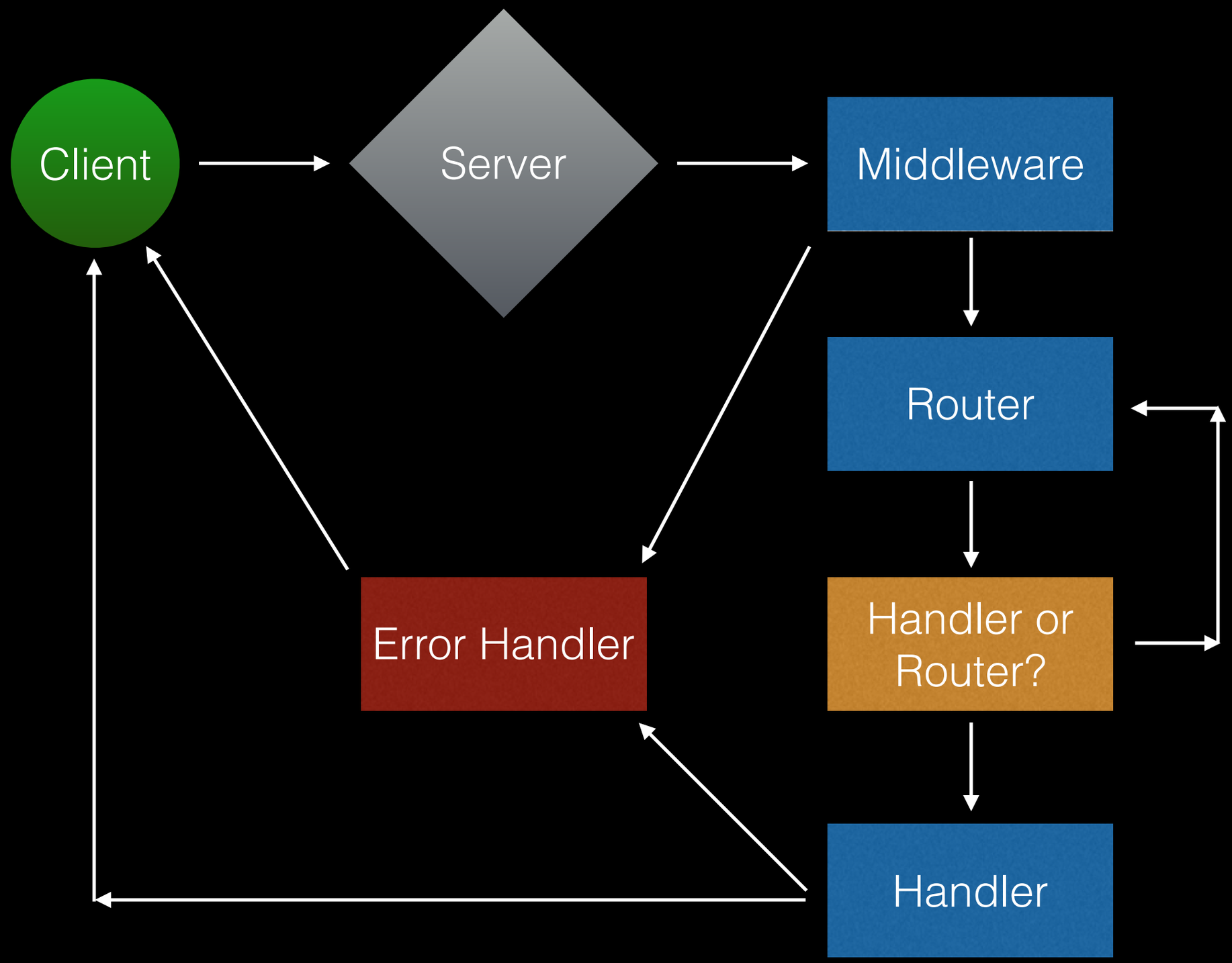
Middleware

Router

Template Engine

Error Handling

Request Flow



next()

- Available in all handlers added to an application as a parameter in the handler function.
- Calls the next handler in the chain.
- Middleware, Routers, Handlers, Error Handlers are all considered “handlers in the chain” and as such are executed in the order they are added.
- Failure to call next() results in the request timing out, unless res.send() is called.
- next(err) is called to raise an error.

Middleware

- Express supports application level plugins through the use of Middleware.
- Each Middleware is a function that gets executed before the request is routed, in the order that the middleware was added to the application.
- Middleware is given the same data that the final routed handler receives, and has the same level of access. Middleware can modify the request or response in and way, and can respond with data and close the request before the the request is routed.

Middleware

Middleware is added to the application via the “use()” method.

```
var aMiddleware = require(‘aMiddleware’);
app.use(aMiddleware);
```

or

```
app.use(function(req, res, next)
{
    // Do something with the request
    next();
});
```

Body Parser (Middleware)

Express provides a middleware for parsing a few different types of HTTP body content from incoming requests.

“body-parser” can be found on GitHub at:

https://github.com/expressjs/body-parser

Router

Supports about all HTTP methods, for example:
get, post, put, head, delete, options

```
app.get(‘/me’, noop);

app.put(‘/me’, noop);

app.post(‘/me’, noop);

app.head(‘/me’, noop);

app.delete(‘/me’, noop);
```

Router

Specify explicit paths at which a user can access functionality of your application

```
app.get('/', function(req, res, next)
{
    res.send('home page');
});
```

Here we specify “/“ as the explicit path to handle

Router

Can take static, parameterized, or regular expression matched paths

```
app.get('/about', noop);

app.get(‘/classes/:class’, noop);

app.get(/^\/classes\/([A-Za-z0-9_\-]+)$/,
noop);
```

Router

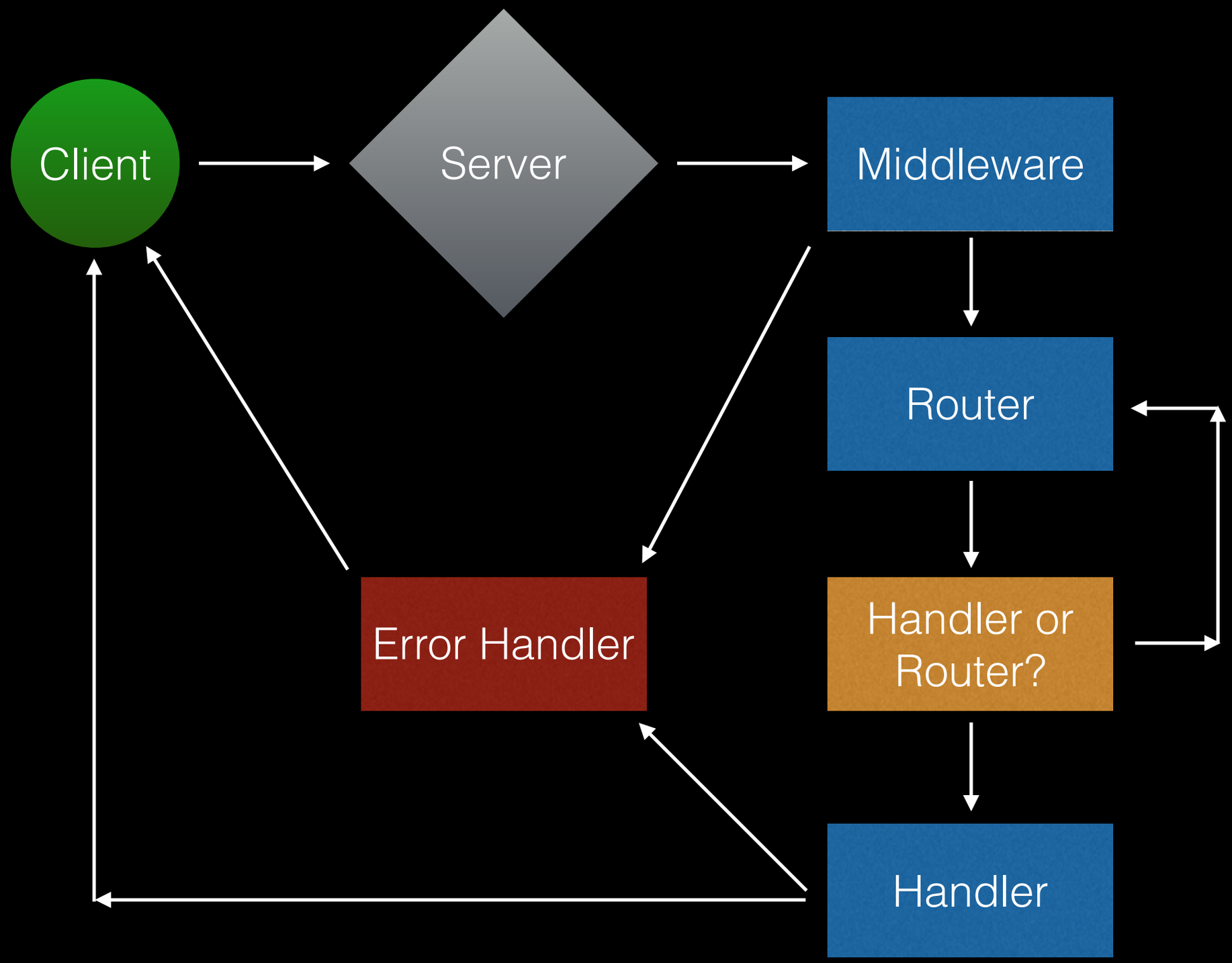
Router is a module – You can specify a router to handle a route

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

router.get('/', getAccount);
```

```
// index.js
var account = require('./account');
app.use('/account', account);
```

Request Flow



Template Engines

- Express provides a standard convention for template engines to support Express.
- Many already do, and for those that do not there is a library called Consolodate.js that adds support for other template engines.

```
app.set(‘views’, ‘./views’);
app.set(‘view engine’, ‘jade’);

// Render
res.render('index', { title: 'Welcome', body:
‘This is my homepage!’});
```

Error Handling

- Error handling in express is treated as Middleware.
- You may use any number of error handlers and they are executed in the order they are used.
- Always use() your error handlers LAST, just before app.listen().

```
function errorHandler(err, req, res, next)
{
    // On error, log to console and blow up
    console.log(err.stack);
    res.status(500);
    res.send('500 internal server error');
}
app.use(errorHandler);
```

What is REST?
What makes a service RESTful?

- Representational State Transfer
- Communication over HTTP using standard verbs (GET, POST, PUT, etc)
- Client-Server: Client concerned with interface, server concerned with data storage
- Stateless: Client context not stored in server between requests
- Cacheable: Responses are cacheable by client implicitly or explicitly; server would use standard HTTP cache headers
- Layered: Client has one access point and cannot determine how many, if any, intermediaries have been passed through along the way. I.e. load balancer.
- Uniform Interface: Service provides a standard interface and access point across all platforms. A standard, documented API.

Project: Library

Create a simple web service utilizing Middleware, Router, and Error Handler.

Create the Express Application

- Open Terminal, iTerm, or other bash shell
- “cd” to the directory you want your app to be in, such as “cd ~/Documents”
- $ mkdir library
- $ cd library
- $ npm init
- press “enter” ~10 times
- $ npm install express —save
- $ npm install body-parser —save

Requirements

	Guests	Members
Home	√	√
Inventory + Search	√	√
Checkout Books		√
Login	√	

Routes

- /
- /books
- /books/search/:term
- /users/me/books
- /users/me/books/:title
- /guest/login

Routes

- / - Anyone Router
  - GET: /
  - GET: /books
  - GET: /books/search/:term
- /guest - Guest Router
  - POST: /login
- /users/me - Member Router
  - GET+POST: /books
  - DELETE: /books/:title

index.js

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


// Middleware
var bodyParser = require('body-parser');
var resourceMiddleware = require('./resource-middleware');

// Use JSON body parser since we accept application/json
app.use(bodyParser.json());

// Injects resources such as the database into the request
app.use(resourceMiddleware);
```

```
// Set user if logged in
app.use(function(req, res, next)
{
    if(typeof req.query.access_token !== 'undefined' && req.query.access_token.length)
    {
        var matchedUser = req.resources.users.filter(function(ob)
        {
            return ob.access_token === req.query.access_token;
        });

        if(matchedUser.length == 1)
        {
            req.user = matchedUser[0];
        }
    }

    next();
});
```

index.js

```
// Routers
var mainRouter = require('./main-router');
var guestRouter = require('./guest-router');
var memberRouter = require('./member-router');

// Routes
app.use('/', mainRouter);
app.use('/guest', guestRouter);
app.use('/users/me', memberRouter);

// Error Handler
var errorHandler = require('./error-handler');

app.use(errorHandler);

app.listen(3000);
```

resource-middleware.js

```
var resources =
{
    users:
    [
        {
            name: 'A Tester',
            access_token: '1',
            username: 'tester1',
            password: 'tester',
            books: []
        }
    ],

    books:
    [
        {
            title: 'First Book',
            author: 'First Author'
        },
        {
            title: 'Second Book',
            author: 'Second Author'
        }
    ]
};

module.exports = function(req, res, next)
{
    req.resources = resources;
    next();
};
```

main-router.js

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

var getBooksFilteredByTerm = function(req, res, next)
{
    var term = req.params.term;

    if(typeof term !== 'undefined' && term.length)
    {
        var filteredBooks = req.resources.books.filter(function(ob)
        {
            return  ob.title.toLowerCase().indexOf(term.toLowerCase()) != -1 ||
                    ob.author.toLowerCase().indexOf(term.toLowerCase()) != -1;
        });
        res.json(filteredBooks);
    }else
    {
        res.json(req.resources.books);
    }
};

router.get('/', function(req, res, next)
{
    res.json({server: 'Book Library', version: '1.0'});
});

router.get('/books', getBooksFilteredByTerm);
router.get('/books/search/:term', getBooksFilteredByTerm);

module.exports = router;
```

guest-router.js

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

router.post('/login', function(req, res, next)
{
    var matchedUser = req.resources.users.filter(function(ob)
    {
        return ob.username === req.body.username && ob.password === req.body.password;
    });

    if(matchedUser.length == 1)
    {
        res.json(matchedUser[0]);
    }else
    {
        next(new Error('InvalidCredentials'));
    }
});

module.exports = router;
```

member-router.js

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

router.use(function(req, res, next)
{
    if(typeof req.user === 'undefined')
    {
        next(new Error('Unauthorized'));
    }else
    {
        next();
    }
});

router.get('/books', function(req, res, next)
{
    res.json(req.user.books);
});
```

member-router.js

```
router.post('/books', function(req, res, next)
{
    var newBook = req.body;

    var foundBook = req.resources.books.filter(function(ob)
    {
        return ob.title === newBook.title;
    });

    if(foundBook.length == 1)
    {
        foundBook = foundBook[0];

        var foundExistingBook = req.user.books.filter(function(ob)
        {
            return ob == foundBook;
        });

        if(foundExistingBook.length == 0)
        {
            req.user.books.push(foundBook);
        }

        res.json(req.user.books);
    }else
    {
        next(new Error('BookNotFound'));
    }
});
```

member-router.js

```
router.delete('/books/:title', function(req, res, next)
{
    var bookTitle = req.params.title;

    for(var i = 0; i < req.user.books.length; ++i)
    {
        var book = req.user.books[i];

        if(book.title === bookTitle)
        {
            req.user.books.splice(i, 1);

            res.json(req.user.books);
            return;
        }
    }

    next(new Error('BookNotFound'));
});

module.exports = router;
```

error-handler.js

```
var errors =
{
    'UnknownError': { code: 500, message: 'Unknown error.' },
    'InvalidCredentials': { code: 401, message: 'Invalid credentials.' },
    'BookNotFound': { code: 404, message: 'Book not found.' },
    'Unauthorized': { code: 401, message: 'Unauthorized.' }
};

module.exports = function(err, req, res, next)
{
    var error = false;

    if(err.message in errors)
    {
        var error = errors[err.message];
    }else
    {
        var error = errors['UnknownError'];
    }

    res.status(error.code).json(error);
};
```

Run Your App

```
$ NODE_ENV=production node index.js
```