Express

Why Express.js?

```
const http = require('http');
const fs = require('fs');

const server = http.createServer((req, res) => {
    // console.log(req.url, req.method, req.headers);
    const url = req.url;
    const method = req.method;

    if (url === '/') {
       // do something…
    }

    if (url === '/messsage' && method === 'POST’) {
          // do something…
    }
     // do something…

  });

server.listen(3000);
```

Server Logic is Complex!

You want to focus on your Business Logic,
Not on the nitty-gritty Details

Use a Framework for the Heavy Lifting!

Framework: Helper functions, tools & rules that help you build your application!

Alternatives to Express.js

- Vanilla Node.js
- Adonis.js
- Koa
- Sails.js
- …

Express

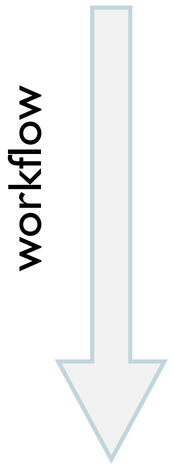
- Express.js is a web framework based on the core Node.js http module. Those components are called middleware.

- What Does Express.js Help You With?

Parsing Requests & Sending Responses	Routing	Managing Data
Extract Data	Execute different Code for different Requests	Manage Data across Requests (Sessions)
Render HTML Pages	Filter / Validate incoming Requests	Work with Files
Return Data / HTML Responses		Work with Databases

Express Application Structure

- The typical structure of an Express.js app (which is usually app.js file) roughly consists of these parts, in the order shown:
  1. Dependencies
  2. Instantiations
  3. Configurations
  4. Middleware
  5. Routes
  6. Error Handling
  7. Bootup



Configuration: port number,
view cache,

Your First Express App

- Create a new package.json file
  - `npm init`

1. Dependencies: Install Express
   - `npm install express -save`
2. Instantiations: Instantiate Express

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

const app = express();

app.listen(3000, () => {
    console.log('Your Server is running on 3000');
})
```

Configurations

- There are two ways to configure our application:

1. **set**
   - `app.set('port', process.env.PORT || 3000);`
   - `const port = app.get('port');`

2. **enable/disable**
   - `app.enable('case sensitive routing') === app.set('case sensitive routing', true)`
   - `app.disable('case sensitive routing') === app.set('case sensitive routing', false)`

- The app.set() exposes variables to templates application-wide
  - `app.set('appName', 'Online Shopping');`

Configurations - `'case sensitive routing'`

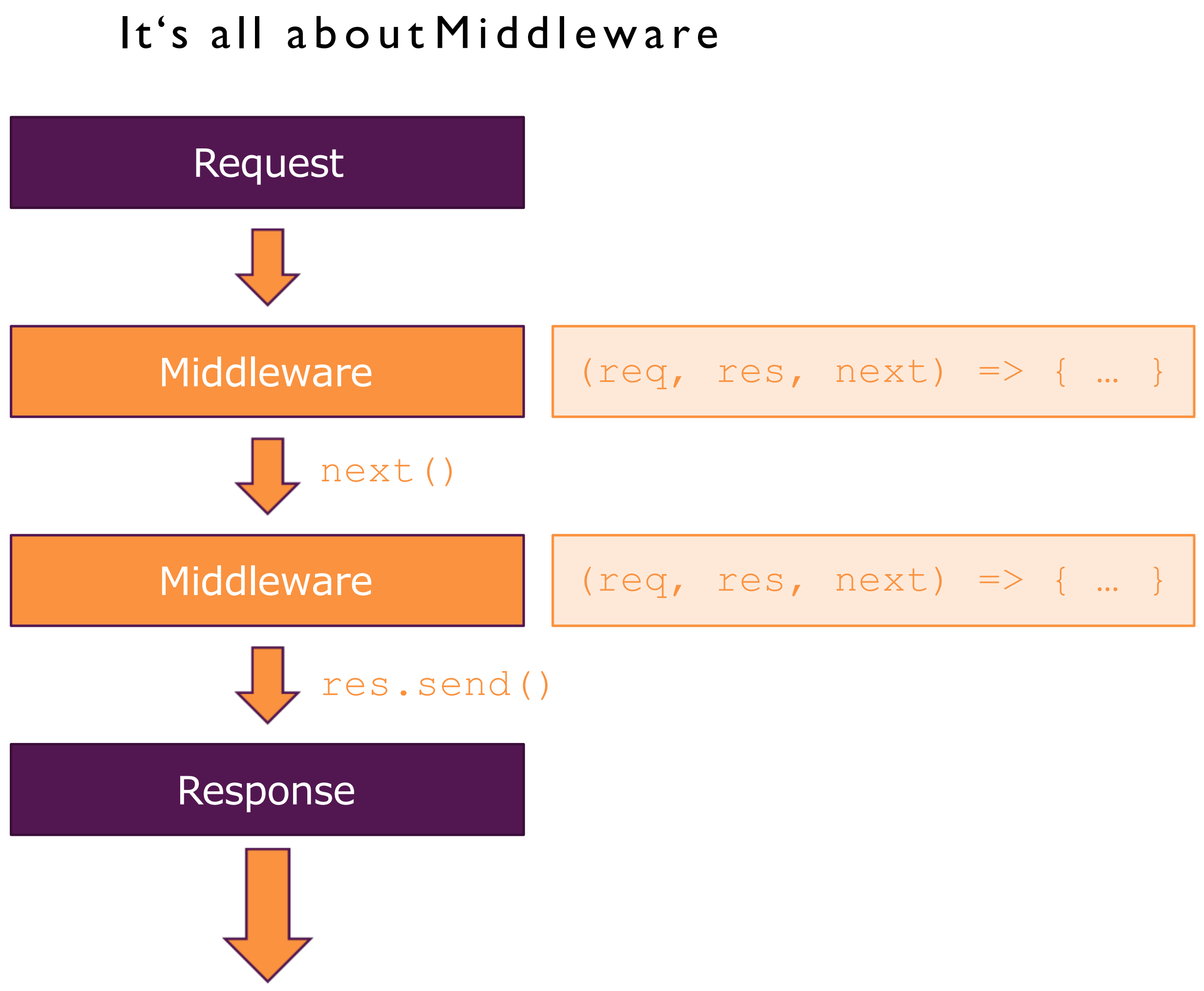
- To disregard the case of the URL paths set this setting to `false`, which is the default value, and do otherwise when the value is set to true.

```
app.enable('case sensitive routing');
```

- For example, when it's enabled, then **/users** and **/Users** won't be the same. It's best to leave this option disabled by default for the sake of avoiding confusion.

Middleware

- Middleware is a useful pattern that allows developers to reuse code within their applications and even share it with others in the form of NPM modules.
- The definition of middleware is a function with three arguments:
  - request
  - response
  - next


It's all about Middleware
Request
Middleware
(req, res, next) => { … }
next()
Middleware
(req, res, next) => { … }
res.send()
Response

Using Middleware

- To use a middleware, we call the `app.use()` method which accepts:
  - One optional string path
  - One mandatory callback function

```
app.use((req, res, next) => {
    console.log('This always run');
    next();
});

app.use('/add-product', (req, res, next) => {
    console.log('In the middleware!');
    res.send('<h1>The "Add Product" Page</h1>');
});

app.use('/', (req, res, next) => {
    console.log('In another middleware!');
    res.send('<h1>Hello from Express</h1>');
});
```

Middleware `body-parser`

- Node.js body parsing middleware to handle HTTP POST request.
- Parse incoming request bodies in a middleware before your handlers, available under the `req.body` property.

- The `body-parser` module has 4 distinct middlewares:
  - **`json()`** Processes JSON data
  - **`urlencoded()`** Processes URL-encoded data: name=value&name2=value2
  - **`raw()`** Returns body as a buffer type
  - **`text()`** Returns body as string type

- The result will be put in the `request` object with `req.body` property and passed to the next middleware and routes.

Middleware `body-parser`

```
const bodyParser = require('body-parser’);
app.use(bodyParser.urlencoded());

app.use('/add-product', (req, res, next) => {
    console.log('In the middleware!');
    res.send('<form action="/product" method="post"><input name="title"><button type="submit">Sub
mit</button></form>');
});


app.use('/product', (req, res, next) => {
    console.log(req.body); // { title: 'book' }

    res.redirect('/');
});
```

- **Note:** `body-parser` does not support multipart( ). instead, use busboy, formidable, or multiparty.

Using body-parser Only for certain route

```
const express = require('express')
const bodyParser = require('body-parser')
const app = express()

const jsonParser = bodyParser.json()
const urlencodedParser = bodyParser.urlencoded({ extended: false })

app.post('/login', urlencodedParser, function (req, res) {
      res.send('welcome, ' + req.body.username)
})
app.post('/api/users', jsonParser, function (req, res) {
      // create user in req.body
})
```

The extended option allows to choose between parsing the URL-encoded data with the querystring library (when false) or the qs library (when true).

Built-in MiddleWare express parser

- The express.json() and express.urlencoded() middleware have been added to provide request body parsing support out-of-the-box. This uses the expressjs/body-parser module module underneath.

```
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
```

- This option allows to choose between parsing the URL-encoded data with the querystringlibrary (when false) or the qs library (when true).
- This middleware is available in Express v4.16.0 onwards.

`next()`

- `next()` : Go to next request handler function(middleware, route), could be in the same URL route.
- `next('route')` : Skip current route and go to next one.
- `next(somethingElse)` : Go to Error Handler

Routing app.VERB()

- Routes an HTTP request, where METHOD is the HTTP method of the request, such as GET, PUT, POST, and so on, in lowercase.
- Each route is defined by a method call on an application object with a URL pattern as the first parameter (regex are supported)
- `app.METHOD(path, [callback...], callback);`

The callbacks that we pass to get() or post() methods are called **request handlers** because they take requests (req), process them, and write to the response (res) objects.

```
app.use('/product', (req, res, next) => {
    console.log(req.body);
    res.redirect('/');
});
```

```
app.post('/product', (req, res, next) => {
    console.log(req.body);
    res.redirect('/');
});
```

Routing app.all()

- This method is like the standard app.METHOD() methods, except it matches all HTTP verbs.

```
app.all('*', userAuth);
app.all('/api/*', apiAuth);

var userAuth = function (req, res, next) {
      return next();
});

var apiAuth = function (req, res, next) {
      return next();
});
```

The Router Class

- The Router class is a mini Express.js application that has only middleware and routes. This is useful for **abstracting modules** based on the business logic that they perform.

```
const express = require('express');
const options = {
    "caseSensitive": false,
    "strict": false
};
const router = express.Router(options);

router.get('/add-product', (req, res, next) => {
    console.log('In the middleware!');
    res.send('<form action="/product" method="post"><input name="title"><button type="submit">Submit</button></form>');
});

router.post('/product', (req, res, next) => {
    console.log(req.body);
    res.redirect('/');
});

module.exports = router;
```

Where options is an object that can have following properties:

- **caseSensitive**: Boolean
- **strict**: Boolean

Filtering Paths

- routes/admin.js

```
router.get('/admin/add-
product', (req, res, next) => {
    res.send('<form action="/admin/product" me
thod="post">…</form>');
});

router.post('/admin/product', (req, res, next)
 => {
    res.redirect('/');
});
```

- app.js

```
app.use(adminRoutes);
```

- routes/admin.js

```
router.get('/add-
product', (req, res, next) => {
        res.send('<form action="/admin/product
" method="post>…</form>');
});

router.post('/product', (req, res, next) => {
   res.redirect('/');
});
```

- app.js

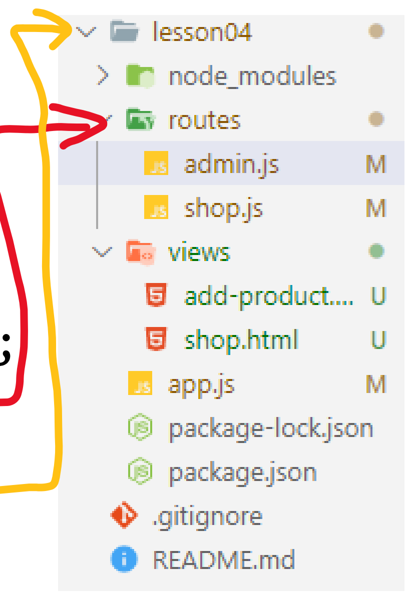
```
app.use('/admin', adminRoutes);
```

Serving HTML Pages

- `path.join([...paths])`: The `path.join()` method joins all given path segments together using the platform-specific separator as a delimiter, then normalizes the resulting path.
- `__dirname` tells you the absolute path of the directory containing the currently executing file.

- routes/admin.js

```
router.get('/add-product', (req, res, next) => {
    res.sendFile(path.join(__dirname, '../', 'views', 'add-product.html'));
});
```



```
lesson04
  node_modules
  routes
    admin.js           M
    shop.js            M
  views
    add-product....    U
    shop.html          U
  app.js               M
  package-lock.json
  package.json
  .gitignore
  README.md
```

Path Hepler Function

- Create your own module to help find root directory

util/path.js

```
const path = require('path');

module.exports = path.dirname(process.mainModule.filename);
```

routes/admin.js

```
router.get('/add-product', (req, res, next) => {
    res.sendFile(path.join(__dirname, '../', 'views', 'add-
product.html'));
});
```

routes/admin.js

```
const rootDir = require('../util/path');

router.get('/add-product', (req, res, next) => {
    res.sendFile(path.join(rootDir, 'views', 'add-product.html'));
});
```

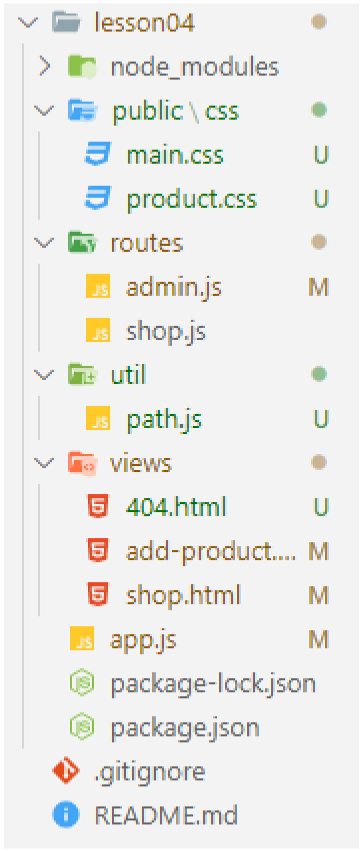
Serving Static Resources

- **static** is the **only** middleware that comes with Express.js before version 4.15.x. It enables pass-through requests for static assets.

```
app.use(express.static(path.join(__dirname, 'public')));
<link rel="stylesheet" href="/css/main.css">

app.use('/mycss', express.static(path.join(__dirname, 'public', 'css')));
app.use('/img', express.static(path.join(__dirname, 'public', 'images')));
app.use('/js', express.static(path.join(__dirname, 'public', 'js')));
<link rel="stylesheet" href="/mycss/main.css">
```

Once Express sees a request to the following paths /mycss or /img or /js it will stream those resources immediately without looking at the rest of the Routes or other Middleware.



Error Handling in Express

- Define error-handling middleware functions in the same way as other middleware functions, except error-handling functions have **four** arguments instead of three: `(err, req, res, next)`

```
app.use(function (err, req, res, next) {
    res.status(500).send('Something broke!');
});
```

Responses from within a middleware function can be in any format that you prefer, such as an HTML error page, a simple message, or a JSON string.

- **IMPORTANT:** You define error-handling middleware last, after other app.use() and routes calls.

Error Handling in Express

- For organizational (and higher-level framework) purposes, you can define several error-handling middleware functions, much as you would with regular middleware functions.

```
function logErrors (err, req, res, next) { console.error(err.stack); next(err); }
```

```
function clientErrorHandler (err, req, res, next) {
         if (req.xhr) { res.status(500).send({ error: 'Something failed!' })
} else { next(err) } }
```

```
function errorHandler (err, req, res, next) {
         res.status(500) res.render('error', { error: err })
}
```

```
app.use(logErrors)
app.use(clientErrorHandler)
app.use(errorHandler)
```

Notice that when ***not*** calling "next" in an error-handling function, you are responsible for writing (and ending) the response. Otherwise those requests will "hang" and will not be eligible for garbage collection.

Returning a 404 page

- The **HTTP 404**, **404 Not Found**, **404**, **Page Not Found**, or **Server Not Found** error message is a Hypertext Transfer Protocol (HTTP) standard response code, in computer network communications, to indicate that the browser was able to communicate with a given server, but the server could not find what was requested.

```
app.use((req, res, next) => {
    res.status(404).sendFile(path.join(__dirname, 'views', '404.html'));
});
```

- **IMPORTANT:** You define 404 page not found middleware last, after other app.use() and routes calls.

Middleware Order Matters

- The order of middleware loading is important: middleware functions that are loaded first are also executed first.

```
app.use((req, res, next) => {
    res.status(404).sendFile(path.join(__dirname, 'views', '404.html'));
});

//below is not executed
app.get('/add-product', (req, res, next) => {
    res.sendFile(path.join(__dirname, 'views', 'add-product.html'));
});
```

Resources

- Express Resources
  - ExpressJS
  - Connect
  - Express Wiki
  - morgan
  - body-parser
- Other Resources
  - Understanding Express.js
  - A short guide to Connect Middleware

Homework - Exercise

1. Create a npm project and install Express.js (Nodemon if you want)
2. Change your Express.js app which serves HTML files (of your choice with your content) for "/", "/users" and "/products".
3. For "/users" and "/products", provides GET and POST requests handling (of your choice with your content) in different routers.
4. Add some static (.js or .css) files to your project that should be required by at least one of your HTML files.
5. Customize your 404 page
6. Provide your own error handling