Безопасный путь к файлу в fs и path

В этом выпуске мы рассмотрим, как при помощи Node.js создать веб-сервер, который будет возвращать файл юзеру из директории *public*. Может возникнуть вопрос: зачем здесь Node.js? почему бы не сделать это на другом сервере? Вопрос совершенно уместен. Да, для отдачи файлов, как правило, другие сервера будут более эффективны. С другой стороны, Node.js, во-первых, тоже работает весьма неплохо, а во-вторых, перед отдачей файла может совершить какие-то интеллектуальные действия, например, обратиться к базе данных, проверить, имеет ли юзер право на доступ к данному файлу, и только если имеет, тогда уже отдавать.

Итак, начинаем. *http.create* очень прост.

1. http.createServer(**function**(req, res) {
3. **if** (!checkAccess(req)) {
4. res.statusCode = 403;
5. res.end("Tell me the secret to access!");
6. **return**;
7. }

Он будет проверять, есть ли доступ к данному файлу, и если есть, то уже отдавать. Для проверки доступа мы будем использовать следующую, по сути заглушечную, функцию, которая будет парсить *url*, и если есть параметр *secret*, который будет равен 'o\_O', то считается, что доступ есть:

1. **function** checkAccess(req) {
2. **return** url.parse(req.url, **true**).query.secret == 'o\_O';
3. }

В реальной жизни такая проверка, конечно же, будет производиться при помощи куков, базы данных и т.д.

Основная функция, которая нас интересует, это *sendFileSafe*:

1. sendFileSafe(url.parse(req.url).pathname, res);

Именно эта функция должна, получив путь от юзера, отослать соответствующий файл с директории *public,* учитывая путь директории. Важнейший аспект, который в ней должен быть заложен, это безопасность. Какой бы путь не передал юзер, он ни в коем случае не должен получить файл вне этой директории (*public)*. Например, обращение *localhost:3000/index.html?secret=o\_O* должно обращать файл index.html и картинка здесь взята из директории *deep/node.js*. А если бы я не указал *secret,* то оно должно было бы выдать мне ошибку с кодом 403. А если бы я попробовал указать вот так: *localhost:3000/server.js,* то тоже была бы ошибка. И так для любых попыток выйти за пределы этой директории.

Итак, смотрим функцию *sendFileSafe*, чтобы получить пример безопасной работы с путем от посетителя. Эта функция состоит из нескольких шагов. На первом шаге я пропускаю путь через *decodeURIComponent*, ведь по стандарту *http* многие символы кодируются, в частности русская буква я будет выглядеть вот так: %D1%8F в *url.* Это корректно.

1. filePath = decodeURIComponent(filePath); // %D1%8F
2. **catch**(e) {
3. res.statusCode = 400;
4. res.end("Bad Request");
5. **return**;

Получив такой *url,* мы обязаны его декодировать обратно в русскую букву я при помощи вот такого вызова: *decodeURIComponent.* При этом, если *url* закодирован заведомо неверно, то возникнет ошибка, которую необходимо поймать и обработать. Код 400 как раз означает, что *url* некорректен и запрос неверен. Можно, конечно, вернуть и 404.

Когда мы раскодировали запрос, время его проверить. Есть специальный нулевой байт, который в строке *url* присутствовать не должен:

1. **if** (~filePath.indexOf('\0')) {
2. res.statusCode = 400;
3. res.end("Bad Request");
4. **return**;
5. }

Если он есть, то это означает, что кто-то его злонамеренно передал, потому что некоторые встроенные функции Node.js будут работать таким байтом немного некорректно. Соответственно, если такой байт есть, то мы тоже возвращаем «До свидания», запрос некорректен.

Теперь настало время получить полный путь к файлу на диске. Для этого мы будем использовать модуль *path*. Этот встроенный модуль содержит пачку самых разных функций для работы с путями. Например, *join* объединяет пути, *normalize* – удаляет из пути всякие странные вещи типа точки, две точки, два слеша и т.д., то есть, делает путь более корректным. Если *url,* который передал юзер, выглядел так: // /deep/nodejs.jpg, то после *join* с рутом, который представляет собой вот эту директорию:

1. **var** ROOT = \_\_dirname + "/public/";

он будет выглядеть уже по-другому:

1. //  /deep/nodejs.jpg ->  /Users/learn/node/path/public/deep/nodejs.jpg

Наша следующая задача – это убедиться, что этот путь действительно находится внутри директории *public.* Сейчас, когда у нас уже есть абсолютно точный, корректный путь, это сделать очень просто. Достаточно всего лишь проверить, что вначале находится вот такой префикс:

1. //  /deep/nodejs.jpg ->  /Users/learn/node/path/public/deep/nodejs.jpg

то есть, что путь начинается с рута. Проверяем. Если это не так, то «до свидания» файла нет:

1. //  /deep/nodejs.jpg ->  /Users/learn/node/path/public/deep/nodejs.jpg
2. filePath = path.normalize(path.join(ROOT, filePath));
4. **if** (filePath.indexOf(ROOT) != 0) {
5. res.statusCode = 404;
6. res.end("File not found");
7. **return**;
8. }

Если путь разрешен, то проверим, что по нему лежит. Если ничего нет, то *fs.stat* вернет ошибку:

1. fs.stat(filePath, **function**(err, stats) {
2. **if** (err || !stats.isFile()) {
3. res.statusCode = 404;
4. res.end("File not found");
5. **return**;
6. }

Если даже ошибки нет, то нужно проверить, файл ли это. В том случае, если это не файл, то это ошибка. Ну а если файл, то все проверено, и надо его отослать. Это делает вложенный вызов *sendFile*:

1. sendFile(filePath, res);
2. });
3. }

Его функция есть в этом же файле немного ниже:

1. **function** sendFile(filePath, res) {

Она для чтения файла использует вызов *fs.readFile*, и когда он будет прочитан, то выводит его через *res.end*:

1. fs.readFile(filePath, **function**(err, content) {
2. **if** (err) **throw** err;
4. **var** mime = require('mime').lookup(filePath); // npm install mime
5. res.setHeader('Content-Type', mime + "; charset=utf-8"); // text/html image/jpeg
6. res.end(content);
7. });
9. }

Обращаю ваше внимание вот на что: во-первых, ошибка в этом *callback* очень маловероятна, хотя бы потому, что мы уже проверили, файл есть, это действительно файл и его можно отдать. Но, тем не менее, мало ли что. Например, может возникнуть ошибка при чтении с диска. Так или иначе как-то обработать ошибку надо. Далее, мало просто считать содержимое файла и отправить его, ведь различные файлы должны снабжаться различными заголовками *Content-Tyр*e. Например, *html* файл должен иметь тип *text/html*, файл с картинкой jpeg – image/jpeg и т.д. Нужный тип файла определяется по расширению с использованием модуля 'mime'. Для того чтобы это работало, нужно его дополнительно поставить и затем вызвать.

И, наконец, последнее, этот выпуск был сосредоточен в первую очередь на том, чтобы корректно работать с путем от посетителя, чтобы сделать все необходимое: проверки, декодирование и т.д. Все это очень важно, но что касается отдачи файла, то этот код неверен

1. **function** sendFile(filePath, res) {
3. fs.readFile(filePath, **function**(err, content) {
4. **if** (err) **throw** err;
6. **var** mime = require('mime').lookup(filePath); // npm install mime
7. res.setHeader('Content-Type', mime + "; charset=utf-8"); // text/html image/jpeg
8. res.end(content);
9. });
11. }

потому что *readFile* полностью просчитывает файл, потом отсылает его в контент. А представьте, что будет, если файл очень большой. А если он превышает количество свободной памяти? Вообще все упадет! Поэтому, для того чтобы отсылать файл, нужно либо дать команду специализированному серверу, либо использовать потоки, которые мы рассмотрим в следующем выпуске скринкаста.