Лабораторна робота №2.

Конвеєр обробки запиту та middleware. Відправлення відповіді. Маршрутизація. Переадресація.

Коли фреймворк Express отримує запит, цей запит передається в конвеєр обробки. Конвеєр складається з набору компонентів або **middleware**, які отримують дані запиту і вирішують, як його обробляти.

Так, в попередній роботі файл додатка виглядав наступним чином:

const express = require("express");

const app = express();

app.get("/", function(request, response){

    response.send("<h1>Главная страница</h1>");

});

app.get("/about", function(request, response){

    response.send("<h1>О сайте</h1>");

});

app.get("/contact", function(request, response){

    response.send("<h1>Контакты</h1>");

});

app.listen(3000);

Тут конвеєр обробки складався з викликів ***app.get()***, які порівнювали запитану адресу з маршрутом, і якщо між адресою та маршрутом є відповідність, то даний запит оброблявся методом ***app.get()***. При необхідності ми можемо вбудувати в конвеєр обробки запиту на будь-якому етапі будь-яку функцію **middleware**. Для цього застосовується метод ***app.use()***. Так, змінимо файл ***app.js*** наступним чином:

const express = require("express");

const app = express();

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

    console.log("Middleware 1");

    next();

});

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

    console.log("Middleware 2");

    next();

});

app.get("/", function(request, response){

    console.log("Route /");

    response.send("Hello");

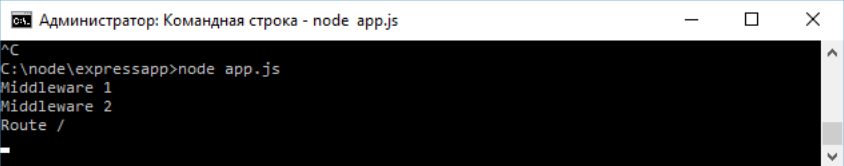
});

app.listen(3000);

Функція, яка передається в ***app.use()***, приймає три параметри:

* **request**: дані запиту
* **response**: об'єкт для управління відповіддю
* **next**: наступна в конвеєрі обробки функція

Кожна з функцій **middleware** просто виводить на консоль повідомлення і в кінці викликає наступну функцію за допомогою виклику ***next()***. При запуску програми після звернення на адресу http://localhost:3000/ послідовно відпрацюють всі три **middleware**:



Однак необов'язково викликати всі наступні **middleware**, ми можемо на якомусь етапі зупинити обробку:

const express = require("express");

const app = express();

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

    console.log("Middleware 1");

    next();

});

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

    console.log("Middleware 2");

    response.send("Middleware 2");

});

app.get("/", function(request, response){

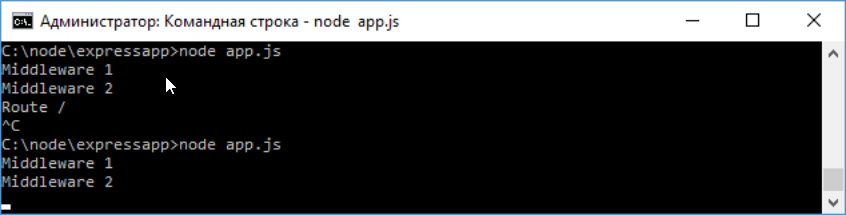
    console.log("Route /");

    response.send("Hello");

});

app.listen(3000);

Тепер обробка завершується на **Middleware 2**, так як в цьому методі відбувається відправка відповіді за допомогою ***response.send()***, а виклику наступної функції через ***next()***:



Функції **middleware** також можуть зіставлятися з певними маршрутами. наприклад:

const express = require("express");

const app = express();

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

    console.log("Middleware 1");

    next();

});

app.use("/about", function(request, response, next){

    console.log("About Middleware");

    response.send("About Middleware");

});

app.get("/", function(request, response){

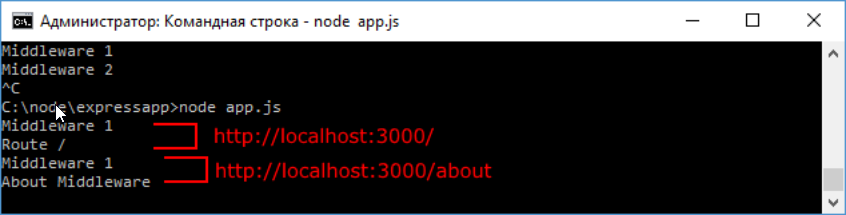
    console.log("Route /");

    response.send("Hello");

});

app.listen(3000);

В даному випадку друга функція **middleware** явно зіставляється з маршрутом /about, тому вона буде обробляти тільки запит http://localhost:3000/about. Перша функція **middleware** як і раніше обробляє всі запити:



**Приклад middleware**

**Middleware** допомагають виконувати деякі завдання, які повинні бути зроблені до відправлення відповіді. Стандартна задача - логування запитів.

Наприклад, змінимо файл ***app.js*** наступним чином:

const express = require("express");

const fs = require("fs");

const app = express();

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

    let now = new Date();

    let hour = now.getHours();

    let minutes = now.getMinutes();

    let seconds = now.getSeconds();

    let data = `${hour}:${minutes}:${seconds} ${request.method} ${request.url} ${request.get("user-agent")}`;

    console.log(data);

    fs.appendFile("server.log", data + "\n", function(){});

    next();

});

app.get("/", function(request, response){

    response.send("Hello");

});

app.listen(3000);

Тут за допомогою об'єкта **request** отримуємо різну інформацію про запит і додаємо її в файл ***server.log***, використовуючи модуль **fs**.

Відправлення відповіді

Для відправлення відповіді в **express** у об'єкта **response** можна використовувати ряд функцій. Найпоширеніший спосіб надсилання відповіді представляє функція ***send()***. Як параметр ця функція може приймати об'єкт Buffer, рядок, в тому числі з html-кодом, об'єкт javascript або масив.

const express = require("express");

const app = express();

app.use(function (request, response) {

    response.send("<h2>Hello</h2>");

});

app.listen(3000);

Відправлення об’єкту:

response.send({id:6, name: "Tom"});

Відправлення масиву:

response.send(["Tom", "Bob", "Sam"]);

Відправлення об’єкту **Buffer:**

response.send(Buffer.from("Hello Express"));

Об'єкт **Buffer** формально представляє деякі бінарні дані. Так, у вищевказаному випадку при виконанні коду за замовчуванням браузер завантажить файл, в якому буде рядок "*Hello Express*".

**sendFile**

Метод **send** зручний для відправки рядків, деякого коду html невеликої довжини, проте якщо відправляється досить великий код html, то відповідно код додатка теж стає громіздким. Наприклад, ми можемо написати так:

const express = require("express");

const app = express();

app.use(function (request, response) {

  response.send(`<!DOCTYPE html>

  <html>

  <head>

      <title>Главная</title>

      <meta charset="utf-8" />

  </head>

  <body>

      <h1>Главная страница</h1>

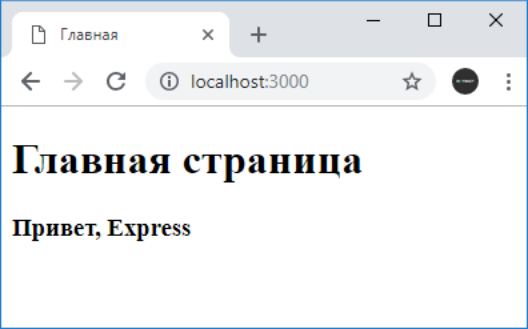
      <h3>Привет, Express</h3>

  </body>

  <html>`);

});

app.listen(3000);



Однак набагато краще визначати код html в окремих файлах і потім ці файли відправляти за допомогою функції ***sendFile()***.

Наприклад, визначимо в папці проекту новий файл **index.html**:

<!DOCTYPE html>

<html>

  <head>

      <title>Главная</title>

      <meta charset="utf-8" />

  </head>

  <body>

      <h1>Главная страница</h1>

      <h3>Привет, Express</h3>

  </body>

<html>

Відправимо цей файл за допомогою функції **sendFile**:

const express = require("express");

const app = express();

app.use(function (request, response) {

  response.sendFile(\_\_dirname + "/index.html");

});

app.listen(3000);

У підсумку ми отримаємо той же самий результат.

Слід враховувати, що в функцію **sendFile** необхідно передавати абсолютний шлях до файлу, саме для цього за допомогою **\_\_dirname** отримуємо абсолютний шлях до поточного проекту і потім додаємо до нього шлях до файлу в рамках поточного проекту.

**Відправка статусних кодів**

Функція **sendStatus()** відправляє користувачеві певний статусний код з деяким повідомленням за замовчуванням. Наприклад, відправимо статусний код ***404***, який говорить, що ресурс не знайдений:

const express = require("express");

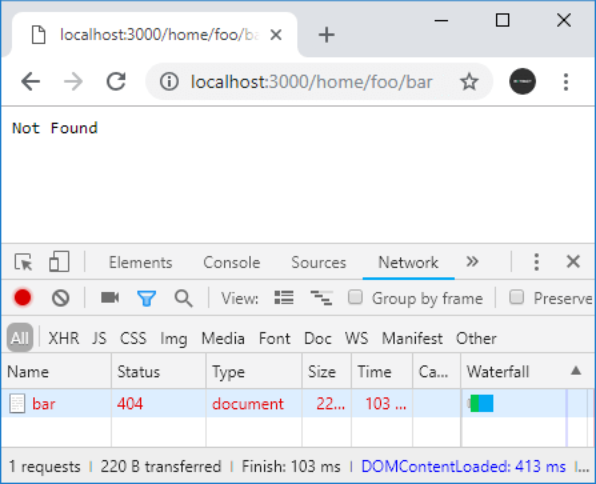
const app = express();

app.use("/home/foo/bar",function (request, response) {

  response.sendStatus(404)

});

app.listen(3000);



Як видно зі скріншоту, при відправці статусного коду ***404*** також відправляється повідомлення "*Not Found*". Але, можливо, ми захочемо відправляти якісь свої більш інформативні повідомлення. В цьому випадку можна використовувати комбінацію функції **status()**, яка також відправляє статусний код, і функції **send()**:

const express = require("express");

const app = express();

app.use("/home/foo/bar",function (request, response) {

  response.status(404).send(`Ресурс не найден`);

});

app.listen(3000);

Розглянемо, як в Node.js + Express відправляти користувачеві статичні файли html. Для роботи зі статичними файлами в Express визначений спеціальний компонент **express.static()**, який вказує на каталог з файлами.

Створимо для статичних файлів в проекті каталог **public**, в який додамо нову html-сторінку, яку назвемо **about.html**:

<!DOCTYPE html>

<html>

<head>

    <title>О сайте</title>

    <meta charset="utf-8" />

</head>

<body>

    <h1>О сайте</h1>

</body>

<html>

В результаті проект буде виглядати наступним чином:

app.js

node\_modules

package.json

public

about.html

Тепер змінимо файл **app.js**:

const express = require("express");

const app = express();

app.use(express.static(\_\_dirname + "/public"));

app.use("/", function(request, response){

    response.send("<h1>Главная страница</h1>");

});

app.listen(3000);

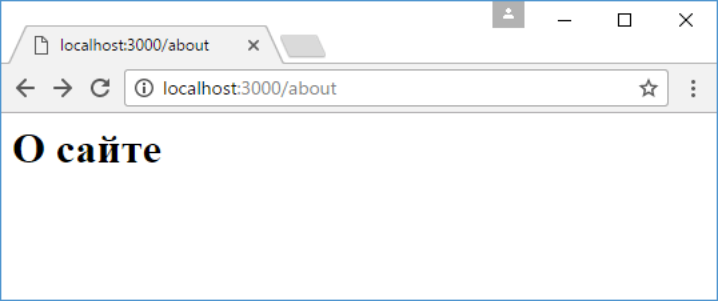
Щоб вбудувати компонент **express.static** в процес обробки запиту, викликається функція **app.use()**. Ця функція дозволяє додавати різні компоненти, які ще називаються **middleware**, в конвеєр обробки запиту:

app.use(express.static(\_\_dirname + "/public"));

Причому даний виклик поміщається до всіх інших викликів функції **app.get().**

У саму ж функцію **express.static()** передається шлях до папки зі статичними файлами. Спеціальне вираження **\_\_dirname** дозволяє отримати повний шлях до папки.

Запустимо додаток на виконання і звернемося в браузері шляхом <http://localhost:3000/about.html>:



Слід зазначити, що для звернення до файлу ми вказуємо тільки ім'я файлу без назви каталогу **public**. Також не слід плутати подібне звернення до файлів з дією функції **sendFile**: в даному випадку ми безпосередньо звертаємося до статичних файлів, а функція **sendFile** фактично бере вміст з файлу і відсилає його користувачеві.

Додатково ми можемо змінити шлях до каталогу статичних файлів:

const express = require("express");

const app = express();

app.use("/static", express.static(\_\_dirname + "/public"));

app.use("/", function(request, response){

   response.send("<h1>Главная страница</h1>");

});

app.listen(3000);

Тепер щоб звернутися до файлу **about.html**, необхідно відправити запит <http://localhost:3000/static/about.html>

Маршрутизація

При обробці запитів фреймворк Express спирається на систему маршрутизації. У додатку визначаються маршрути, а також обробники цих маршрутів. Якщо запит відповідає визначеним маршрутом, то викликається для обробки запиту відповідний обробник.

Для обробки даних за певним маршрутом можна використовувати ряд функцій, зокрема:

* use
* get
* post
* put
* delete

В якості першого параметра ці функції можуть приймати шаблон адреси, запит по якому буде оброблятися. Другий параметр функції представляє функцію, яка буде обробляти запит по адресі що співпала з шаблоном. Наприклад:

const express = require("express");

const app = express();

// обработка запроса по адресу /about

app.get("/about", function(request, response){

    response.send("<h1>О сайте</h1>");

});

// обработка запроса по адресу /contact

app.use("/contact", function(request, response){

    response.send("<h1>Контакты</h1>");

});

// обработка запроса к корню веб-сайта

app.get("/", function(request, response){

    response.send("<h1>Главная страница</h1>");

});

app.listen(3000);

Коли приходить запит Express зіставляє адресу, що запитується з кожним з маршрутів. Потім вибирається перший маршрут що співпав. При збігу маршруту викликається його функція обробника.

**Символи підстановок**

Використовувані шаблони адрес можуть містити регулярні вирази або спеціальні символи підстановок. Зокрема, ми можемо використовувати такі символи, як ?, +, \*, ().

Наприклад, символ ? вказує, що попередній символ може зустрічатися один раз або бути відсутнім. І ми можемо визначити наступну функцію:

app.get("/bo?k", function (request, response) {

  response.send(request.url)

});

Такий маршрут буде відповідати рядку запиту "/bk" або "/bok".

Символ + вказує, що попередній символ може зустрічатися один і більше разів:

app.get("/bo+k", function (request, response) {

  response.send(request.url)

});

Такий маршрут буде відповідати запитам "/bok", "/book", "/boook" і так далі.

Символ зірочка \* вказує, що на місці цього символу може бути будь-яка кількість символів:

app.get("/bo\*k", function (request, response) {

  response.send(request.url)

});

Такий маршрут буде відповідати запитам "/ bork", "/ bonk", "/bor.dak", "/ bor / ok" і так далі.

Дужки () дозволяють оформити групу символів, які можуть зустрічатися в запиті:

app.get("/book(.html)?", function (request, response) {

  response.send(request.url)

});

Вираз (.html)? вказує, що подстрока ".html" може зустрічатися або бути відсутнім в рядку запиту. І такий маршрут буде відповідати запитам "/book" та "/book.html".

Також замість визначення маршрутів ми можемо вказувати регулярні вирази. Наприклад, необхідно перехопити запит до всіх файлів html або всі шляхи, які в кінці мають ".html":

app.get(/.\*(\.)html$/, function (request, response) {

  response.send(request.url)

});

Переадресація

Для переадресації застосовується метод **redirect()**:

redirect([status,] path)

В якості параметру **path** передається шлях, на який буде перенаправлений користувач. Додатковий параметр **status** задає статусний код переадресації. Якщо цей параметр не заданий, тоді за замовчуванням відправляється статусний код **302**, який передбачає тимчасову переадресацію.

За допомогою даного методу можна переадресовувати як за відносними шляхами, так і за абсолютними, в тому числі на інші домени.

Переадресація по абсолютному шляху:

const express = require("express");

const app = express();

app.use("/index",function (request, response) {

  response.redirect("https://google.com")

});

app.listen(3000);

В даному випадку при зверненні по шляху "/index" буде йти переадресація на сайт https://google.com.

Переадресація по відносним шляхам також не дуже складна, але тут важливо враховувати, як саме визначається адреса для перенаправлення. Розглянемо редірект відносно поточного шляху, з якого відбувається редірект. Наприклад:

const express = require("express");

const app = express();

app.use("/home",function (request, response) {

  response.redirect("about")

});

app.use("/about", function (request, response) {

  response.send("<h1>About</h1>");

});

app.listen(3000);

В даному випадку буде йти перенаправлення з ресурсу "/home" на ресурс "/about", тобто, умовно кажучи, з http://localhost:3000/home на http://localhost:3000/about.

Тепер розглянемо схожий приклад:

const express = require("express");

const app = express();

app.use("/home/bar",function (request, response) {

  response.redirect("about")

});

app.use("/home/about", function (request, response) {

  response.send("<h1>About</h1>");

});

app.listen(3000);

Тут з ресурсу "/home/bar" також йде переадресація на ресурс "about", проте в реальності тепер це буде не http://localhost:3000/about, а http://localhost: 3000/home/about. Тобто ми ніби піднімаємося на один рівень вгору - з "home/bar" на "home/" і потім до нього додається "about".

Якщо нам необхідно виконати переадресацію не відносно поточного ресурсу, а відносно кореневого каталогу програми, то на початку адреси ставиться слеш:

const express = require("express");

const app = express();

app.use("/home/bar",function (request, response) {

  response.redirect("/about")

});

app.use("/about", function (request, response) {

  response.send("<h1>About</h1>");

});

app.listen(3000);

Ще кілька прикладів. Переадресація відносно поточної адреси на адресу на тому ж рівні:

app.use("/home/foo/bar",function (request, response) {

  response.redirect("./about")

});

Тут йде переадресація з http://localhost:3000/home/foo/bar на http://localhost:3000/home/foo/about

Переадресація на адресу, яка розташовується рівнем вище:

app.use("/home/foo/bar",function (request, response) {

  response.redirect("../about")

});

Тут йде переадресація з http://localhost:3000/home/foo/bar на http://localhost:3000/home/about

Переадресація на рівень вище:

app.use("/home/foo/bar",function (request, response) {

  response.redirect(".")

});

Тут йде переадресація з http://localhost:3000/home/foo/bar на http://localhost:3000/home/foo

Переадресація на два рівні вище:

app.use("/home/foo/bar",function (request, response) {

  response.redirect("..")

});

Тут йде переадресація з http://localhost:3000/home/foo/bar на http://localhost:3000/home

За замовчуванням при редіректі передається статусний код ***302***, який вказує, що ресурс тимчасово доступний за новою адресою. Але ми можемо вказати статусний код ***301***, щоб зробити переадресацію постійною:

response.redirect(301, "/about");

Завдання для самостійної роботи

* В корне проекта создать каталог «api». Переместить все файлы из лабораторной 1 в каталог «api».
* В корне проекта создать каталог «proxy».
* В каталоге «proxy» создать приложение, которое похоже на «api», но использует порт 3001 и для пути ‘/‘ отправляет запрос к «api» и возвращает полученный результат.
* Для «proxy» через .env файл и пакет convict указать url приложения «api» (localhost:port) и порт, используемый приложением «proxy».
* Разбить приложение «proxy» на слои - сервер, роутинг, сервис, клиент (использует axios). Запустить оба приложения.
* В браузере ввести адрес приложения «proxy» и увидеть Hello world от приложения «api».

Посилання

Express.js https://expressjs.com/body-parser

helmet https://www.npmjs.com/package/helmet

express-jwt https://www.npmjs.com/package/express-jwtcors

convict https://www.npmjs.com/package/convict

axios https://www.npmjs.com/package/axios

jest (mock, spy) https://jestjs.io/docs/en/mock-function-api.html

nock https://www.npmjs.com/package/nock