Лекция 19
Промежуточное ПО в Express.js

Разберитесь в функциях промежуточного ПО и в том, как они улучшают обработку запросов в Express.js.

Содержание:

1. Что такое промежуточное программное обеспечение (Middleware)?
2. Использование встроенного промежуточного программного обеспечения
3. Создание пользовательского промежуточного программного обеспечения

1. Промежуточное ПО в Express.js

1.1 Что такое промежуточное программное обеспечение (Middleware)?

Промежуточное ПО (middleware) — это **ключевое понятие** в Express.js, обозначающее функции, имеющие доступ к объектам запроса и ответа, а также к nextфункциям в цикле запрос-ответ приложения. **Функции промежуточного ПО** могут выполнять ряд задач, таких как выполнение кода, изменение объектов запроса или ответа, завершение цикла запрос-ответ или передача управления следующей функции промежуточного ПО. **Поток выполнения** промежуточного ПО является последовательным: при получении запроса Express.js пропускает его через каждую функцию промежуточного ПО в порядке их определения до тех пор, пока не будет отправлен ответ или запрос не будет передан следующему обработчику.

В Express.js существует несколько типов промежуточного ПО:

- Промежуточное ПО уровня приложения: подключается к экземпляру приложения expressи применяется ко всем или к определенным маршрутам;
- Промежуточное ПО уровня маршрутизатора: подключается к экземпляру express.Router()и применяется к маршрутам, обрабатываемым этим маршрутизатором;
- Промежуточное ПО для обработки ошибок: функции с четырьмя аргументами, используемые для обработки ошибок, возникающих в приложении.

```
function logRequest(req, res, next) {
  console.log(req.method, req.url);
  next();
}

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

app.use(logRequest);

app.get('/', (req, res) => {
  res.send('Home page');
});
```

В этом примере logRequestфункция промежуточного ПО получает запрос (**req**), ответ (**res**) и **next** функцию. Она выводит в лог метод запроса и URL, а затем вызывает функцию **next()** для перехода к следующему обработчику промежуточного ПО или маршрута. Используя **app.use(logRequest)**, вы гарантируете, что это промежуточное ПО будет выполняться для каждого входящего запроса, демонстрируя последовательный и многоуровневый поток промежуточного ПО в Express.js.

1.2 Использование встроенного промежуточного программного обеспечения

Express.js предоставляет несколько встроенных функций промежуточного ПО, которые помогают обрабатывать распространенные задачи веб-приложений с минимальной настройкой. Две из наиболее часто используемых встроенных функций промежуточного ПО — это **express.json() и express.static().** Эти функции упрощают обработку входящих данных запроса и предоставление статических ресурсов, таких как изображения, CSS и JavaScript-файлы.

Чтобы увидеть, как эти промежуточные программы работают на практике, рассмотрим этот пример простого приложения Express, которое использует **express.json()** и **express.static():**

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

// Use express.json() to parse incoming JSON requests
app.use(express.json());

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

app.post('/api/data', (req, res) => {
  // Access parsed JSON data from the request body
  const data = req.body;
  res.json({ received: data });
});

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

В этом примере express.json()в приложение добавляется промежуточное ПО с помощью app.use(). Это промежуточное ПО анализирует входящие тела запросов в формате JSON и делает полученный объект доступным по адресу **req.body**. Когда клиент отправляет POST-запрос с полезной нагрузкой JSON на **/api/data** конечную точку, вы можете напрямую получить доступ к проанализированным данным как к объекту JavaScript.

Промежуточное **express.static()** ПО также используется с **app.use()** параметром, указывающим на каталог "public". Это позволяет приложению обслуживать статические файлы — такие как HTML, CSS, изображения или JavaScript —

непосредственно из этого каталога. Например, если вы поместите файл с именем **logo.png** в папку "public", он станет доступен по адресу http://localhost:3000/logo.png без необходимости определять отдельный маршрут.

Вкратце, **express.json()** он анализирует тело входящих запросов с помощью **Content-Type application/json**, что позволяет легко обрабатывать данные JSON, отправляемые клиентами. **express.static()** – обслуживает статические файлы из указанного каталога, позволяя эффективно доставлять ресурсы пользователям без написания пользовательских обработчиков маршрутов для каждого файла.

1.3 Создание пользовательского промежуточного программного обеспечения

Если вы хотите, чтобы ваше приложение Express.js соблюдало определенные правила или выполняло проверки входящих запросов, вы можете написать собственные функции промежуточного ПО. Например, вы можете захотеть убедиться, что каждый запрос включает определенный заголовок, прежде чем он достигнет обработчиков маршрутов. Ниже приведен пример пользовательского промежуточного ПО, которое проверяет наличие заголовка с именем `.` "x-api-key". Если заголовок отсутствует, промежуточное ПО отправляет ответ со 400 Bad Requestстатусом и сообщением; в противном случае оно разрешает продолжение запроса.

```
function checkApiKey(req, res, next) {
  const apiKey = req.headers['x-api-key'];
  if (!apiKey) {
    return res.status(400).json({ error: 'Missing x-api-key header' });
  }
  next();
}
```

Это промежуточное ПО проверяет заголовки входящего запроса на наличие соответствующего параметра "**x-api-key**". Если заголовок отсутствует, оно немедленно останавливает обработку запроса и отправляет JSON-ответ с ошибкой. Если заголовок присутствует, оно вызывает **next()** функцию для передачи управления следующему промежуточному ПО или обработчику маршрута.

Чтобы понять, как работает это промежуточное программное обеспечение и как его использовать, выполните следующие шаги:

1. Определите функцию промежуточного программного обеспечения, например **checkApiKey**, с параметрами **req, res,** и **next**;
2. Внутри функции получите доступ к заголовкам запроса, используя **req.headers['x-api-key']**;
3. Если заголовок отсутствует, отправьте ответ **res.status(400).json({ error: 'Missing x-api-key header' })** и верните значение, чтобы остановить дальнейшую обработку;
4. Если заголовок присутствует, вызовите функцию next()для перехода к следующему промежуточному ПО или маршруту;
5. Интегрируйте собственное промежуточное ПО в ваше приложение Express, передавая его в app.use()качестве параметра определенным маршрутам:

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

function checkApiKey(req, res, next) {
  const apiKey = req.headers['x-api-key'];
  if (!apiKey) {
    return res.status(400).json({ error: 'Missing x-api-key header' });
  }
  next();
}

// Apply middleware globally
app.use(checkApiKey);

app.get('/protected', (req, res) => {
  res.send('Access granted!');
});

app.listen(3000);
```

Выполнив эти шаги, ваше пользовательское промежуточное ПО будет перехватывать запросы и обеспечивать наличие необходимого заголовка до запуска какой-либо защищенной логики маршрута.

Ссылка
https://codefinity.com/courses/v2/fc82fa76-7239-478a-bd4e-99ee5e48c1c9/8d828d2b-047e-436e-8288-c2674c2fb6d9/2c581567-2219-4100-ac2f-32ea9c4a04c7