ДИСЦИПЛИНА	Фронтенд и бэкенд разработка
ИНСТИТУТ	ИПТИП
КАФЕДРА	Индустриального программирования
ВИД УЧЕБНОГО МАТЕРИАЛА	Методические указания к практическим занятиям по дисциплине
ПРЕПОДАВАТЕЛЬ	Загородних Николай Анатольевич Краснослободцева Дарья Борисовна
СЕМЕСТР	4 семестр, 2025/2026 уч. год

Практическое занятие №2

Сервер на Nodejs + Express

Рассмотрим использование Nodejs для разработки API с использованием Express.js. Решение практического задания осуществляется внутри соответствующей рабочей тетради, расположенной в СДО.

Что такое Express.js

Express.js (он же Express) - фреймворк для разработки веб-приложений на Node.js. Node.js в свою очередь - среда выполнения исходного кода, позволяющая запускать программы, написанные на языке JavaScript на серверах, а не в браузере.

Express.js создан с учетом потребностей разработчиков, желающих быстро создавать веб-приложения, используя синтаксис популярного JavaScript, и предоставляет набор готовых утилит для реализации API, таких как middleware, роутеры и обработчики.

Установка Express.js

Установить фреймворк можно через npm:
`npm install express`

> **Tip**
>
> Автоматически создастся файл `package.json` с настройками по умолчанию. Если этого не произошло - используйте `npm init -y`, после чего повторите процесс установки.

Начало работы с Express.js

Создадим файл `app.js` и добавим в него следующий код:

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
    res.send('Hello, world!');
});

app.listen(port, () => {
    console.log(`Сервер запущен на http://localhost:${port}`);
});
```

Итак, простейшее приложение на Express готово.
Запустим сервер командой `node app.js`, затем откроем браузер и перейдём по адресу:
http://localhost:3000

Откроется страница, на которой мы увидим приветственное сообщение "Hello, world!".

Разберём подробнее.

Сначала мы импортировали (подключили) фреймворк в наш проект

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

Теперь в переменной `express` хранится функция, вызов которой создаст объект приложения Express.

Далее мы создали переменную `app`, в которую и сохранили объект Express-приложения, вызвав функцию `express()`

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

Ниже была создана константа, отвечающая за номер порта, который будет использовать наше приложение, она понадобится нам позже

```
const port = 3000;
```

Перейдём к обработчикам.

Для того, чтобы сервер принимал запросы от клиентов, существуют функции-обработчики, которые "ловят" запросы, соответствующие их параметрам, и реализуют некоторую логику.

```
app.get('/', (req, res) => {
    res.send('Hello, world!');
});
```

В данном случае мы создали обработчик для GET-запроса в главный (корневой) маршрут приложения: `/`. Обработчиком является сама функция `get()`, она принимает в себя ряд параметров, таких как путь (маршрут) и функцию, которая и будет вызвана при получении запроса. Помимо функции `get` существуют одноимённые функции для других типов запросов: POST, PUT, PATCH и DELETE.

Последний шаг - запуск сервера:

```
app.listen(port, () => {
    console.log(`Сервер запущен на http://localhost:${port}`);
});
```

Функция `listen()` запустила сервер на указанном в константе порту и вывела в консоль сообщение о том, что сервер запущен.

Middleware

Для извлечения некоторых данных из запросов нужно использовать middleware. Middleware представляет собой некоторое промежуточное ПО, работа которого происходит в момент после принятия запроса программой и до его непосредственной обработки. Предназначено это для дополнительной обработки и фильтрации запросов, их изменения и сериализации.

В таблице рассмотрены основные элементы запроса и их зависимость от middleware:

Источник данных	Способ доступа	Требуемый middleware
URL-параметры	req.params	Не требуется
Query-параметры	req.query	Не требуется
Данные формы	req.body	express.urlencoded()
JSON-данные	req.body	express.json()
Заголовки запроса	req.headers	Не требуется
Cookies	req.cookies	cookie-parser

Кроме того, middleware открывает огромное множество других возможностей по предобработке запросов, таких как аутентификация, парсинг статических файлов и любой другой пользовательский функционал.

Чтобы подключить middleware в приложение, необходимо использовать функцию `app.use()`, в которую передаётся необходимая middleware-функция.
Рассмотрим пример приложения с такими функциями.

```
const express = require('express');
const app = express();
const port = 3000;

// Middleware для парсинга JSON
app.use(express.json());
// Middleware для парсинга данных форм
app.use(express.urlencoded({ extended: false }));
// Middleware для статических файлов
app.use(express.static('public'));

// Собственное middleware
app.use((req, res, next) => {
    // Выводим в консоль метод и путь запроса
    console.log(`${req.method} ${req.url}`);
    next();
});

// Маршруты
app.get('/', (req, res) => {
    res.send('Главная страница');
});

app.listen(port, () => {
    console.log(`Сервер запущен на http://localhost:${port}`);
});
```

Таким образом мы подключили middlware для парсинга JSON, парсинга данных формы (теперь в обработчиках можно использовать req.body, где и будет храниться JSON запроса), а также для получения статических файлов напрямую (теперь, любой файл из папки `public` можно получить по адресу http://localhost:3000/file.txt, где file.txt - имя целевого файла).

Кроме того, мы создали свою собственную middleware-функцию, которая выводит в консоль метод и URL каждого запроса.

Практический пример

Мы уже выяснили какие бывают составляющие в запросе и как к ним обращаться, а также узнали как обрабатывать запросы с разными методами. Давайте рассмотрим подробнее на практическом примере работу с маршрутизацией и параметрами запросов.

Напишем приложение, которое предоставляет функционал по управлению пользователями: предположим, есть некоторый объект пользователя, который имеет id, имя и возраст. Необходимо реализовать возможность просмотра всех пользователей, получение пользователя по его id, создание нового пользователя, редактирование и удаление пользователя по его id.

```
const express = require('express');
const app = express();
const port = 3000;

let users = [
    {id: 1, name: 'Петр', age: 16},
    {id: 2, name: 'Иван', age: 18},
    {id: 3, name: 'Дарья', age: 20},
]

// Middleware для парсинга JSON
app.use(express.json());

// Главная страница
app.get('/', (req, res) => {
    res.send('Главная страница');
});

// CRUD
app.post('/users', (req, res) => {
    const { name, age } = req.body;

    const newUser = {
        id: Date.now(),
        name,
        age
    };

    users.push(newUser);
    res.status(201).json(newUser);
});

app.get('/users', (req, res) => {
    res.send(JSON.stringify(users));
});

app.get('/users/:id', (req, res) => {
    let user = users.find(u => u.id == req.params.id);
    res.send(JSON.stringify(user));
});

app.patch('/users/:id', (req, res) => {
```

```
    const user = users.find(u => u.id == req.params.id);
    const { name, age } = req.body;

    if (name !== undefined) user.name = name;
    if (age !== undefined) user.age = age;

    res.json(user);
});

app.delete('/users/:id', (req, res) => {
    users = users.filter(u => u.id != req.params.id);
    res.send('Ok');
});

// Запуск сервера
app.listen(port, () => {
    console.log(`Сервер запущен на http://localhost:${port}`);
});
```

Таким образом в примере реализованы все CRUD (Create, Read, Update, Delete) операции.

Практическое задание

Необходимо реализовать API, которое предоставляет CRUD операции для списка товаров (просмотр всех товаров, просмотр товара по id, добавление товара, редактирование товара, удаление товара). Объект товара должен содержать следующие поля: id, название, стоимость.

Формат отчета

В качестве ответа на задание необходимо прикрепить ссылку на репозиторий с реализованной практикой. Ссылка подгружается в соответствующий раздел СДО: Задания для самостоятельной работы -> Практические занятия 1-2.