Лекция 20
Создание RESTful API

Разрабатывайте и внедряйте RESTful API с использованием Express.js, следуя передовым практикам.

Содержание:

1. Принципы RESTful API
2. Реализация CRUD-конечных точек

1. Промежуточное ПО в Express.js

1. Принципы RESTful API

RESTful API – это API, разработанный на основе ресурсов и стандартных методов HTTP. В приложениях Express.js принципы REST помогают создавать предсказуемые и организованные конечные точки, которые легко понять и поддерживать.

Ресурс представляет собой фрагмент данных в приложении, например, пользователей, товары или заказы.

REST API используют различные HTTP-методы для выполнения операций с ресурсами.

HTTP Method	Purpose	Example Endpoint
GET	Retrieve data	/users
POST	Create new data	/users
PUT	Update existing data	/users/1
DELETE	Remove data	/users/1

Примеры RESTful-маршрутов
Пример:

```
app.get('/users', (req, res) => {
  res.send('Get all users');
});

app.post('/users', (req, res) => {
  res.send('Create a user');
});
```

```
app.put('/users/:id', (req, res) => {
  res.send('Update a user');
});

app.delete('/users/:id', (req, res) => {
  res.send('Delete a user');
});
```

Каждый маршрут выполняет разные операции над одним и тем же ресурсом.

Рекомендации по использованию REST API

RESTful API обычно следуют нескольким общим принципам:

– Используйте существительные для обозначения ресурсов вместо глаголов;
– Обеспечьте согласованность и предсказуемость конечных точек;
– Для каждой операции используйте соответствующие методы HTTP;
– Возвращает содержательные коды состояния и ответы.

Например:

– /users лучше, чем /getUsers;
– /products/5 - явно нацелен на конкретный продукт.

Эти соглашения упрощают разработчикам использование и понимание API.

2. Реализация CRUD-конечных точек

CRUD расшифровывается как Create, Read, Update и Delete (создать, прочитать, обновить и удалить). Это четыре основные операции, которые большинство API выполняют при работе с данными.

В Express.js каждая операция CRUD обычно связана с определенным HTTP-методом.

CRUD Operation	HTTP Method	Example Endpoint	Purpose
Create	POST	/users	Add a new user
Read	GET	/users	Get users
Update	PUT	/users/:id	Update a user
Delete	DELETE	/users/:id	Remove a user

Простой пример CRUD (операции CRUD)

В следующем примере вместо базы данных используется простой массив данных в оперативной памяти. Это позволяет сосредоточиться на том, как работают конечные точки Express.js.

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

const app = express();

app.use(express.json());

let users = [
  { id: 1, name: 'Alice' },
  { id: 2, name: 'Bob' }
];

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

app.post('/users', (req, res) => {
  const newUser = {
    id: users.length + 1,
    name: req.body.name
  };

  users.push(newUser);

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

app.put('/users/:id', (req, res) => {
  const user = users.find(user => user.id === Number(req.params.id));

  if (!user) {
    return res.status(404).json({ message: 'User not found' });
  }

  user.name = req.body.name;

  res.json(user);
});

app.delete('/users/:id', (req, res) => {
  users = users.filter(user => user.id !== Number(req.params.id));

  res.json({ message: 'User deleted' });
});

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

Как это работает

- Конечная **GET /users** точка возвращает всех пользователей;
- Конечная **POST /users** точка считывает данные из **req.body**, создает нового пользователя и возвращает его с **201** кодом состояния;
- Данная **PUT /users/:id** конечная точка используется **req.params.id** для поиска пользователя и обновления его имени;
- Данная **DELETE /users/:id** конечная точка удаляет пользователя из массива и возвращает сообщение с подтверждением;
- Промежуточное **express.json()** ПО важно, поскольку оно позволяет Express.js считывать данные в формате JSON из тела запроса.