**Создание и Архитектура REST API в Node.js + Express.js**

🔍 Здесь ты узнаешь: как проектируется API, как работает фреймворк Express, как обрабатываются маршруты, как валидируются данные и что такое CRUD-операции.

**📚 1. Что такое REST API?**

**REST API** (Representational State Transfer) — это архитектурный стиль взаимодействия между клиентом и сервером по протоколу HTTP.

REST API обычно работает с **CRUD-операциями**:

| **Операция** | **HTTP метод** | **Описание** |
| --- | --- | --- |
| Create | POST | Создать ресурс |
| Read | GET | Получить ресурс |
| Update | PUT / PATCH | Обновить ресурс |
| Delete | DELETE | Удалить ресурс |

**⚙️ 2. Установка Express.js и настройка проекта**

**📦 Установка:**

bash

КопироватьРедактировать

mkdir my-api

cd my-api

npm init -y

npm install express

Создаём файл index.js:

js

КопироватьРедактировать

const express = require('express');

const app = express();

const PORT = 3000;

app.use(express.json()); // middleware для JSON

app.listen(PORT, () => {

console.log(`Сервер запущен на http://localhost:${PORT}`);

});

**🔁 3. CRUD-маршруты (на примере "задач")**

Создадим массив как "базу данных":

js

КопироватьРедактировать

let tasks = [

{ id: 1, title: "Изучить Node.js" },

{ id: 2, title: "Написать API" }

];

**✅ GET — Получение всех задач**

js

КопироватьРедактировать

app.get('/tasks', (req, res) => {

res.json(tasks);

});

🧵 **Разбор:**

* Метод app.get() — для GET-запросов.
* '/tasks' — маршрут (endpoint).
* res.json() — возвращает данные в формате JSON.

**🔍 GET по ID**

js

КопироватьРедактировать

app.get('/tasks/:id', (req, res) => {

const id = parseInt(req.params.id);

const task = tasks.find(t => t.id === id);

if (!task) {

return res.status(404).json({ error: "Задача не найдена" });

}

res.json(task);

});

**✏️ POST — Создание новой задачи**

js

КопироватьРедактировать

app.post('/tasks', (req, res) => {

const { title } = req.body;

const newTask = { id: Date.now(), title };

tasks.push(newTask);

res.status(201).json(newTask);

});

🧵 **Разбор:**

* req.body — тело POST-запроса.
* Date.now() — простой генератор уникального ID.
* Статус 201 — "Создано".

**🛠 PUT — Обновление задачи**

js

КопироватьРедактировать

app.put('/tasks/:id', (req, res) => {

const id = parseInt(req.params.id);

const task = tasks.find(t => t.id === id);

if (!task) return res.status(404).json({ error: "Не найдено" });

task.title = req.body.title || task.title;

res.json(task);

});

**❌ DELETE — Удаление задачи**

js

КопироватьРедактировать

app.delete('/tasks/:id', (req, res) => {

const id = parseInt(req.params.id);

tasks = tasks.filter(t => t.id !== id);

res.json({ message: "Удалено" });

});

**🧰 4. Архитектура проекта (MVC-подход)**

bash

КопироватьРедактировать

src/

├── controllers/

│ └── taskController.js

├── routes/

│ └── taskRoutes.js

├── app.js

├── server.js

Разделение логики (controller), маршрутов (routes), модели (если есть) — делает код масштабируемым.

**🔐 5. Валидация и защита API**

Установим express-validator:

bash

КопироватьРедактировать

npm install express-validator

js

КопироватьРедактировать

const { body, validationResult } = require('express-validator');

app.post('/tasks',

body('title').isLength({ min: 3 }),

(req, res) => {

const errors = validationResult(req);

if (!errors.isEmpty()) return res.status(400).json({ errors: errors.array() });

const newTask = { id: Date.now(), title: req.body.title };

tasks.push(newTask);

res.status(201).json(newTask);

}

);

**🧪 10 ПРАКТИЧЕСКИХ ЗАДАНИЙ (REST API)**

| **№** | **Задание** |
| --- | --- |
| 1 | Создай REST API с /users маршрутом |
| 2 | Реализуй GET, POST, PUT, DELETE для пользователей |
| 3 | Добавь валидацию полей через express-validator |
| 4 | Возвращай 404 и 400 при ошибках |
| 5 | Напиши middleware логгера запросов |
| 6 | Реализуй хранение данных в JSON-файле |
| 7 | Добавь сортировку пользователей по имени |
| 8 | Добавь пагинацию по ?page= и ?limit= |
| 9 | Настрой nodemon и .env для порта |
| 10 | Раздели код по папкам (controllers, routes, models) |