

# JSON Server

**JSON Server** - это библиотека, позволяющая "получить полный фейковый REST API без предварительной настройки менее чем за 30 секунд". Также имеется возможность создания полноценного сервера. Данная библиотека реализована с помощью **lowdb** и **express**. Наиболее известным примером ее использования является **JSON Placeholder**.

## Установка

Для запуска сервера необходимо:

```
# Глобально установить json-server
yarn global add json-server
# или
npm i -g json-server

# Создать базу данных в формате JSON, например, `db.json`

# Выполнить команду
json-server -w db.json
```

После этого сервер будет доступен по адресу `http://localhost:3000`. Данные из БД можно получить по соответствующему ключу, например: `http://localhost:3000/todos`. Поддерживаются HTTP-запросы `GET`, `POST`, `PUT`, `PATCH` и `DELETE`.

## CLI

- `--port, -p` - выбор порта
- `--watch, -w` - отслеживание файла
- `--routes, -r` - путь к файлу с маршрутами
- `--middlewares, -m` - путь к файлу с посредниками
- `--static, -s` - путь к директории со статическими файлами
- `--delay, -d` - добавление задержки к ответу в мс

## Использование

Создаем такую БД:

```
// db.json
{
  "todos": [
    {
      "id": "1",
      "text": "Eat",
      "completed": true,
      "meta": {
```

```

    "author": "John",
    "createdAt": "today"
  },
  {
    "id": "2",
    "text": "Code",
    "completed": true,
    "meta": {
      "author": "Jane",
      "createdAt": "yesterday"
    }
  },
  {
    "id": "3",
    "text": "Sleep",
    "completed": false
  },
  {
    "id": "4",
    "text": "Repeat",
    "completed": false
  }
]
}

```

Запускаем сервер с помощью `json-server -w db.json`.

Возможности сервера:

```

// Адрес нашего сервера
const SERVER_URL = 'http://localhost:3000/todos'

// Основная функция для получения всех задач.
// Она принимает строку запроса и адрес сервера
async function getTodos(query, endpoint = SERVER_URL) {
  try {
    // Определяем наличие строки запроса
    query ? (query = `?${query}`) : (query = '')

    const response = await fetch(`${endpoint}${query}`)

    if (!response.ok) throw new Error(response.statusText)

    const json = await response.json()
    console.log(json)
    return json
  } catch (err) {
    console.error(err.message || err)
  }
}

getTodos()

// Получение задачи по ключу
const getTodoByKey = (key, val) => getTodos(`${key}=${val}`)

```

```

// По `id`
const getTodoById = (id) => getTodoByKey('id', id)
getTodoById('2')

// По имени автора
const getTodoByAuthorName = (name) => getTodoByKey('meta.author', name)
getTodoByAuthorName('John')

// Получение активных задач
const getActiveTodos = () => getTodoByKey('complete', false)
getActiveTodos()

// Более сложный пример -
// получение задач по массиву `id`
const getTodosByIds = (ids) => {
  let query = `id=${ids.splice(0, 1)}`
  ids.forEach((id) => {
    query += `&id=${id}`
  })
  getTodos(query)
}
getTodosByIds([2, 4])

// Получение нечетных задач
const getOddTodos = async () => {
  const todos = await getTodos()
  const oddIds = todos.reduce((arr, { id }) => {
    id % 2 !== 0 && arr.push(id)
    return arr
  }, [])
  getTodosByIds(oddIds)
}
getOddTodos()

/* Пагинация */
// _page &| _limit - у нас нет страниц, так что...
const getTodosByCount = (count) => getTodos(`_limit=${count}`)
getTodosByCount(2)

/* Сортировка */
// _sort & _order - asc по умолчанию
const getSortedTodos = (field, order) =>
  getTodos(`_sort=${field}&_order=${order}`)
getSortedTodos('id', 'desc')

/* Часть */
// _start & _end | _limit
// _start не включается
const getTodosSlice = (min, max) => getTodos(`_start=${min}&_end=${max}`)
getTodosSlice(1, 3)

/* Операторы */
// Диапазон
// _gte | _lte
// _gte включается
const getTodosByRange = (key, min, max) =>
  getTodos(`_${key}_gte=${min}&_${key}_lte=${max}`)

```

```

getTodosByRange('id', 1, 2)

// Исключение
// _ne
const getTodosWithout = (key, val) => getTodos(`${key}_ne=${val}`)
getTodosWithout('id', '3')

// Фильтрация
// _like
const getTodosByFilter = (key, filter) => getTodos(`${key}_like=${filter}`)
getTodosByFilter('text', '^[cr]') // Code & Repeat - начинаются с 'c' & 'r', соответственно

/* Полнотекстовый поиск */
// q
const getTodosBySearch = (str) => getTodos(`q=${str}`)
getTodosBySearch('eat') // Eat & Repeat - включают 'eat'

```

Дополнительно:

```

/* Отношения */
// _embed - дочерние ресурсы
GET /posts?_embed=comments
GET /posts/1?_embed=comments

// _expand - родительские ресурсы
GET /comments?_expand=post
GET /comments/1?_expand = post

/* База данных */
GET /db

/* Домашняя страница */
GET /

```

## Дополнительный функционал

### Динамическая генерация БД

Создаем файл, например, `users.js`:

```

module.exports = () => {
  const data = { users: [] }
  // Создаем 10 пользователей
  for (let i = 0; i < 10; i++) {
    data.users.push({ id: i, name: `user-${i}` })
  }
  return data
}

```

Запускаем сервер: `json-server users.js`.

## Кастомная маршрутизация

Создаем файл, например, `routes.posts.json`:

```
{
  "/api/*": "/$1",
  "/:key/:id/show": "/:key/:id",
  "/posts/:val": "/posts?category=:val",
  "/articles\\?id=:id": "/posts/:id"
}
```

Запускаем сервер: `json-server db.json -r routes.posts.json`.

## Добавление посредников

Создаем файл, например, `hello.js`:

```
// Добавляем в ответ кастомный заголовок
module.exports = (_, res, next) => {
  res.header('X-Hello', 'World')
  next()
}
```

Запускаем сервер: `json-server db.json -m hello.js`.

## Использование `JSON Server` в качестве модуля

Пример добавления кастомных маршрутов:

```
// server.js
const jsonServer = require('json-server')
const server = jsonServer.create()
const router = jsonServer.router('db.json')
const middlewares = jsonServer.defaults()

// Добавляем дефолтных посредников (logger, static, cors и no-cache)
server.use(middlewares)

// Добавляем кастомные маршруты перед роутером `JSON Server`
server.get('/echo', (req, res) => {
  res.jsonp(req.query)
})

// Для обработки POST, PUT и PATCH необходимо использовать body-parser
server.use(jsonServer.bodyParser)
server.use((req, _, next) => {
  if (req.method === 'POST') {
    req.body.createdAt = Date.now()
  }
})
// Передаем управление роутеру `JSON Server`
next()
```

```
})

// Используем дефолтный роутер
server.use(router)
server.listen(3000, () => {
  console.log('Server ready')
})
```

Запускаем сервер: `node server.js`.

## Пример

### Создание проекта и установка зависимостей

```
# Создание директории
mkdir json-server-todo
cd json-server-todo
# Инициализация проекта
yarn init -yp
# Установка зависимостей
yarn add json-server open-cli concurrently
```

- `open-cli` - утилита для открытия вкладки браузера по указанному адресу
- `concurrently` - утилита для одновременного выполнения двух и более команд

Добавляем команды в `package.json`:

```
{
  "scripts": {
    "dev": "concurrently \"json-server -w db.json -s /\" \"open-cli http://localhost:3000\"",
    "start": "json-server db.json"
  }
}
```

Запускаем сервер: `yarn dev`.

### Структура проекта

```
| --modules
| --client.js
| --constants.js
| --helpers.js
|--db.json
|--index.html
|--package.json
|--script.js
```

### Разработка проекта

db.json - база данных:

```
{
  "todos": [
    {
      "id": "1",
      "text": "Eat",
      "completed": true
    },
    {
      "id": "2",
      "text": "Code",
      "completed": true
    },
    {
      "id": "3",
      "text": "Sleep",
      "completed": false
    },
    {
      "id": "4",
      "text": "Repeat",
      "completed": false
    }
  ]
}
```

index.html - разметка:

```
<body>
  <!-- Заголовок -->
  <h1>JSON Server Todo</h1>
  <!-- Форма для добавления задачи в список -->
  <form>
    <!-- Поле для текста новой задачи -->
    <input type="text" />
    <!-- Кнопка для отправки формы -->
    <button>Add</button>
  </form>
  <!-- Список задач -->
  <ul></ul>
  <!-- Наличие атрибута "type" со значением "module" является обязательным -->
  <script src="script.js" type="module"></script>
</body>
```

modules/constants.js - константы:

```
// Адрес сервера
// http://localhost:3000 можно опустить, поскольку мы определили
// директорию со статическими файлами в `package.json` как "/"
export const SERVER_URL = '/todos'

// HTTP-методы
```

```
export const POST = 'POST'
export const DELETE = 'DELETE'
export const PUT = 'PUT'

// Операции
export const UPDATE = 'UPDATE'
export const REMOVE = 'REMOVE'
```

modules/helpers.js - глобальные/общие утилиты:

```
// Вспомогательная функция для получения элементов по селектору
export const getEl = (selector, all = false, el = document) =>
  all ? [...el.querySelectorAll(selector)] : el.querySelector(selector)

// Вспомогательная функция для генерации уникального `id`
export const uuid = (n) =>
  ([1e7] + [-1e3 + -4e3 + -8e3 + -1e11])
    .replace(/[018]/g, (c) =>
      (
        c ^
        (crypto.getRandomValues(new Uint8Array(1))[0] & (15 >> (c / 4)))
      ).toString(16)
    )
    .slice(0, n)
```

modules/api.js - API:

```
import { SERVER_URL, POST, DELETE, PUT } from './constants.js'

export const api = async (method, payload, endpoint = SERVER_URL) => {
  let config = {}

  if (method) {
    config = {
      method,
      headers: {
        'Content-Type': 'application/json'
      }
    }
  }

  if (method === POST || method === PUT) {
    config.body = JSON.stringify(payload.body)
  }

  if (method === DELETE || method === PUT) {
    endpoint = `${endpoint}/${payload.id}`
  }
}

try {
  const response = await fetch(endpoint, config)
  if (response.ok) {
    let message
```



```

switch (method) {
  case POST: {
    message = 'Data has been added'
    break
  }
  case DELETE: {
    message = 'Data has been removed'
    break
  }
  case PUT: {
    message = 'Data has been updated'
    break
  }
  default: {
    message = 'Data has been received'
  }
}
console.log(message)

const result = await response.json()
return result
}
throw new Error(response.statusText)
} catch (err) {
  console.error(err.message || err)
}
}

```

`modules/script.js` - основной скрипт:

```

// api
import { api } from './modules/api.js'
// Константы - методы & операции
import { POST, DELETE, PUT, UPDATE, REMOVE } from './modules/constants.js'
// "Глобальные" утилиты
import { getEl, uuid } from './modules/helpers.js'

// Состояние
let todos = []

// Локальная утилита
const getTodoById = (id) => todos.find((todo) => todo.id === id)

// DOM-элементы
const formEl = getEl('form')
const textInput = getEl('input')
const listEl = getEl('ul')

// Функции
// Добавляем задачу в DOM
const addTodo = (todo, replace = false) => {
  // Шаблон задачи
  const itemTemplate = `
    <li
      data-id="${todo.id}"
      style="margin: 0.5rem 0"

```



```

    await api(PUT, { id, body: todo })

    // Замена вместо добавления
    itemEl.innerHTML = addTodo(todo, true)

    // Кнопки обновленной задачи необходимо снова инициализировать
    initButtons()
    break
  }
  case REMOVE: {
    await api(DELETE, { id })

    itemEl.remove()
  }
  default:
    return
}
}

// Добавляем задачу
formEl.onSubmit = async (e) => {
  e.preventDefault()

  const trimmed = textInput.value.trim()

  if (trimmed) {
    const newTodo = {
      id: uuid(5),
      text: trimmed,
      completed: false
    }

    await api(POST, { body: newTodo })


    // Необходимо обновить массив с задачами
    getTodos()

    textInput.value = ''
  }
}

```

Tags: [javascript](#) [js](#) [node.js](#) [nodejs](#) [node](#) [json server](#) [npm](#) [package](#) [cheatsheet](#) [шпаргалка](#) [библиотека](#)

[пакет](#)

 [Edit this page](#)