



Преподаватель:
Коляда
Никита Владимирович

Обратная связь:
• сообщения на inStudy

ОСНОВЫ NODE.JS И ФРЕЙМВОРКА EXPRESS

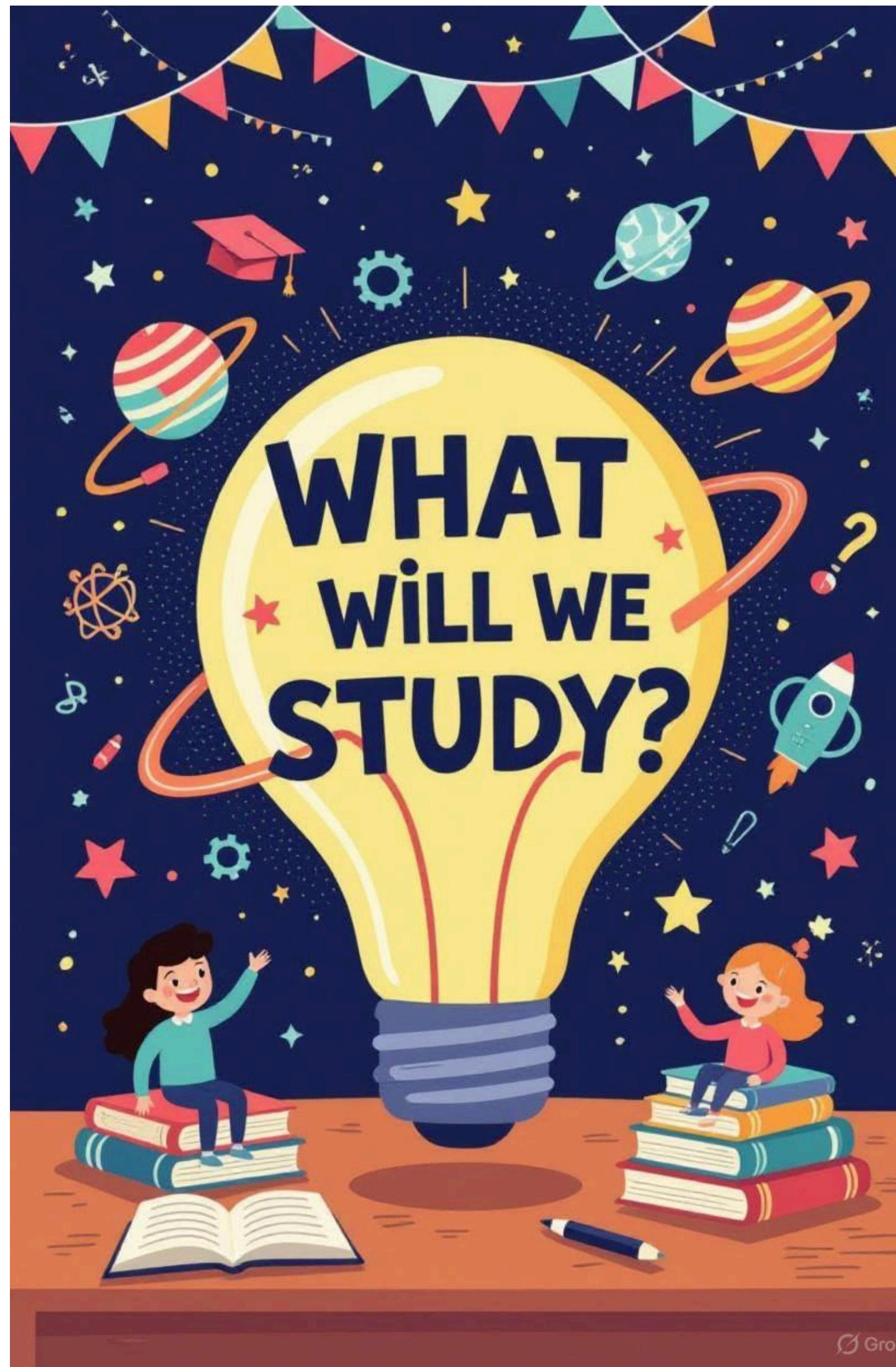
(BACKEND - СЕРВЕРНАЯ ЧАСТЬ)

2026 г.



ВОПРОСЫ?

- ПО ТЕМАМ ДИСЦИПЛИНЫ?
- ПО ПРАКТИЧЕСКИМ/ИТОГОВЫМ РАБОТАМ?
- ПО ОРГАНИЗАЦИОННЫМ МОМЕНТАМ?



КЛЮЧЕВЫЕ ТЕМЫ

1. Зачем нужен Node.js – JavaScript на сервере
2. Сервер как программа – Node.js работает постоянно
3. Установка Node.js – подготовка среды разработки
4. npm и зависимости – установка библиотек
5. Что такое Express – фреймворк для сервера
6. Создание проекта – структура backend-приложения
7. Первый сервер – запуск и прослушивание порта
8. Первый маршрут – как сервер отвечает на запрос
9. JSON-ответы – формат общения с frontend
10. Проблема ручного перезапуска – почему неудобно
11. Nodemon – автоматический перезапуск сервера
12. Скрипты запуска – удобная команда разработки

ЗАЧЕМ НУЖЕН NODE.JS

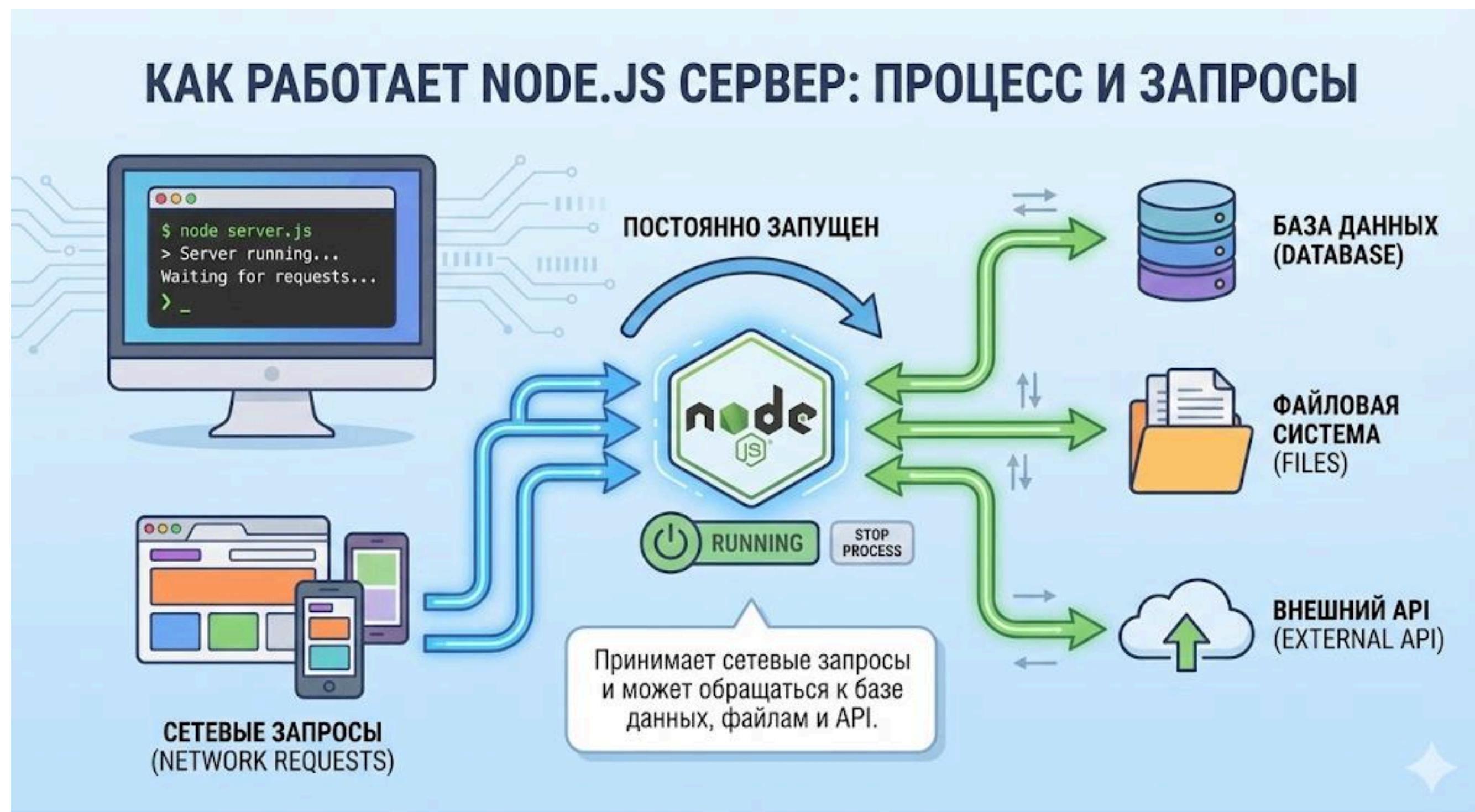
JAVASCRIPT НА СЕРВЕРЕ



1. Node.js позволяет запускать JavaScript вне браузера – на сервере.
2. Это значит, что:
 - один язык используется и на фронтенде, и на бэкенде
 - можно писать серверную логику на JavaScript
 - проще переходить от frontend к backend
3. Node.js – это среда выполнения, а не фреймворк.

ГДЕ РАБОТАЕТ NODE.JS

СЕРВЕР КАК ПРОГРАММА



1. Сервер – это обычная программа, которая постоянно запущена и ждёт запросы.
2. Node.js запускается в терминале и работает, пока процесс не остановлен.
3. Он принимает сетевые запросы и может обращаться к базе данных, файлам и API.

УСТАНОВКА NODE.JS

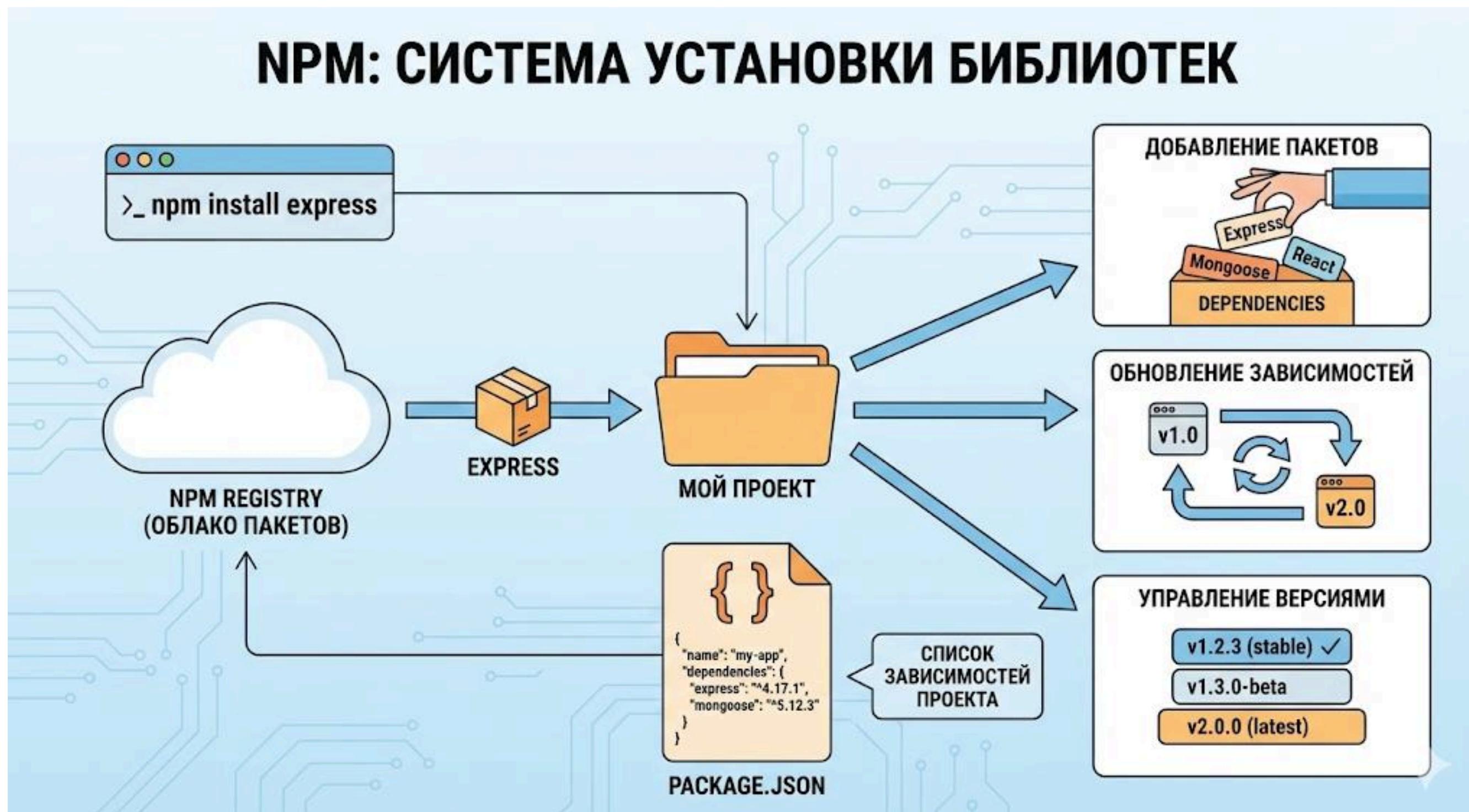
ПОДГОТОВКА СРЕДЫ РАЗРАБОТКИ



1. Для работы нужен установленный Node.js.
2. После установки становятся доступны:
 - команда node – запуск программ
 - команда npm – установка библиотек
3. Проверка установки:
node -v и npm -v в терминале

ЧТО ТАКОЕ NPM

МЕНЕДЖЕР ПАКЕТОВ



1. npm – это система установки библиотек.
2. С её помощью:
 - добавляют Express и другие пакеты
 - обновляют зависимости
 - управляют версиями
3. Проект содержит файл package.json, в котором хранится список зависимостей.

ЧТО ТАКОЕ EXPRESS

ФРЭЙМВОРК ДЛЯ СОЗДАНИЯ СЕРВЕРА

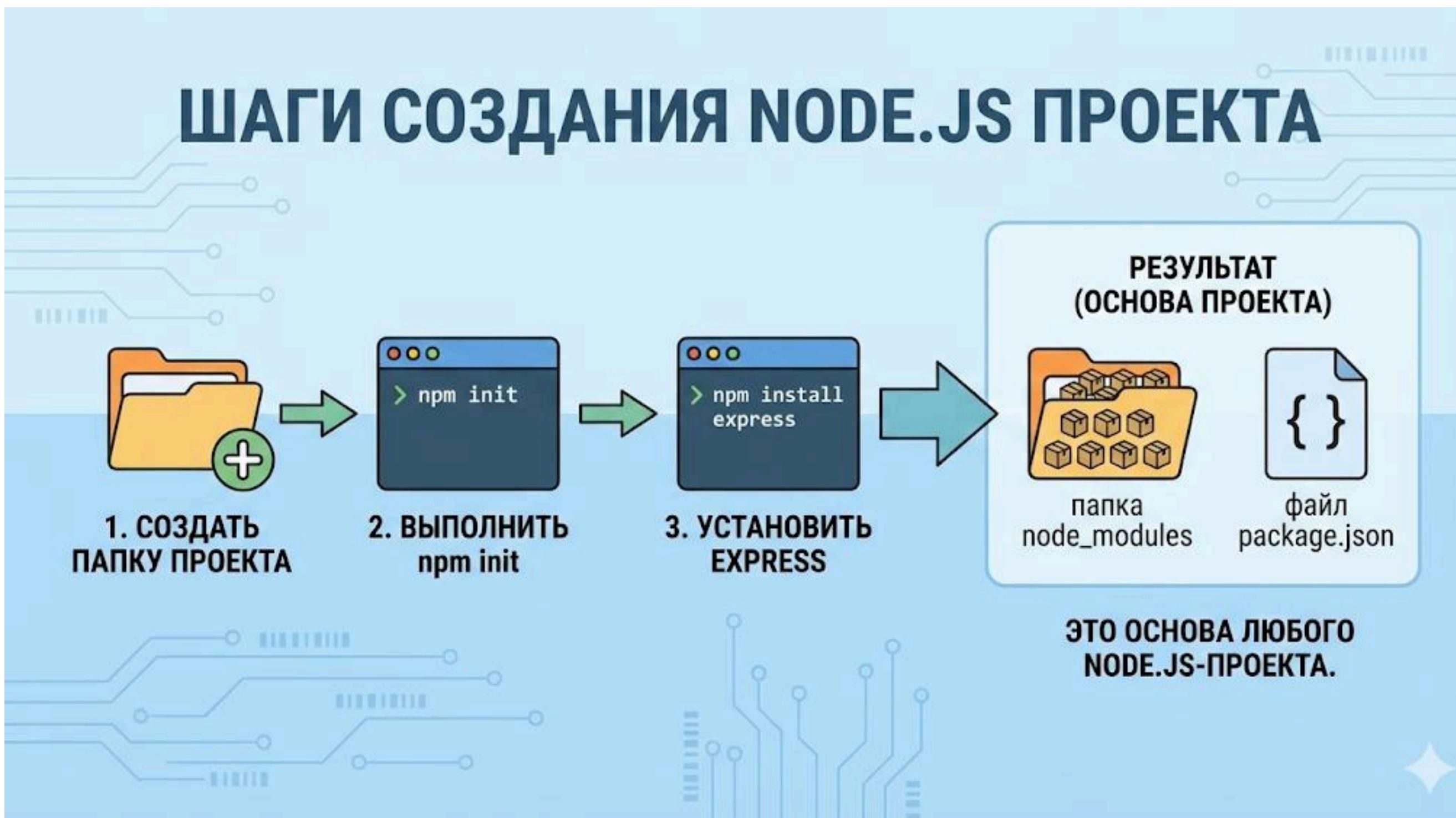
ЭФФЕКТИВНОЕ СОЗДАНИЕ ВЕБ-СЕРВЕРА С EXPRESS ДЛЯ NODE.JS



1. Express – это библиотека для Node.js, которая упрощает создание веб-серверов.
2. Он позволяет:
 - обрабатывать маршруты
 - принимать данные
 - отправлять ответы
3. Без Express сервер тоже возможен, но код будет значительно сложнее.

СОЗДАНИЕ ПРОЕКТА

СТРУКТУРА BACKEND-ПРИЛОЖЕНИЯ



1. Шаги создания проекта:

- Создать папку проекта
- Выполнить npm init
- Установить Express

2. После этого появляется:

- папка node_modules
- файл package.json

3. Это основа любого Node.js-проекта.

ПЕРВЫЙ СЕРВЕР

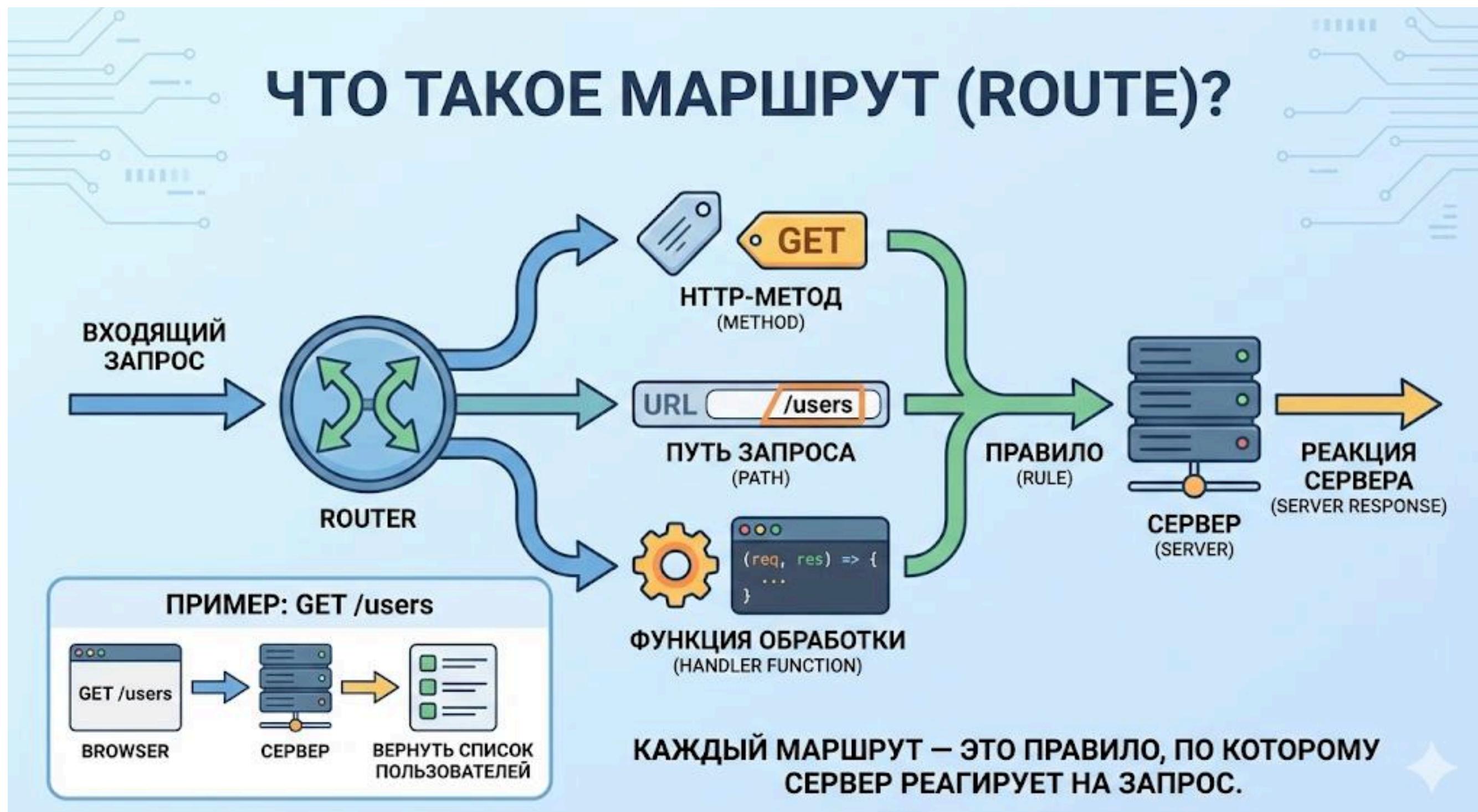
МИНИМАЛЬНЫЙ КОД EXPRESS



1. Простейший сервер делает три вещи:
2. – создаёт приложение Express
 - настраивает маршрут
 - запускает прослушивание порта
3. После запуска сервер доступен по адресу:
`http://localhost:3000`
4. Это уже полноценный backend.

ЧТО ТАКОЕ МАРШРУТ

КАК СЕРВЕР ПОНИМАЕТ ЗАПРОСЫ

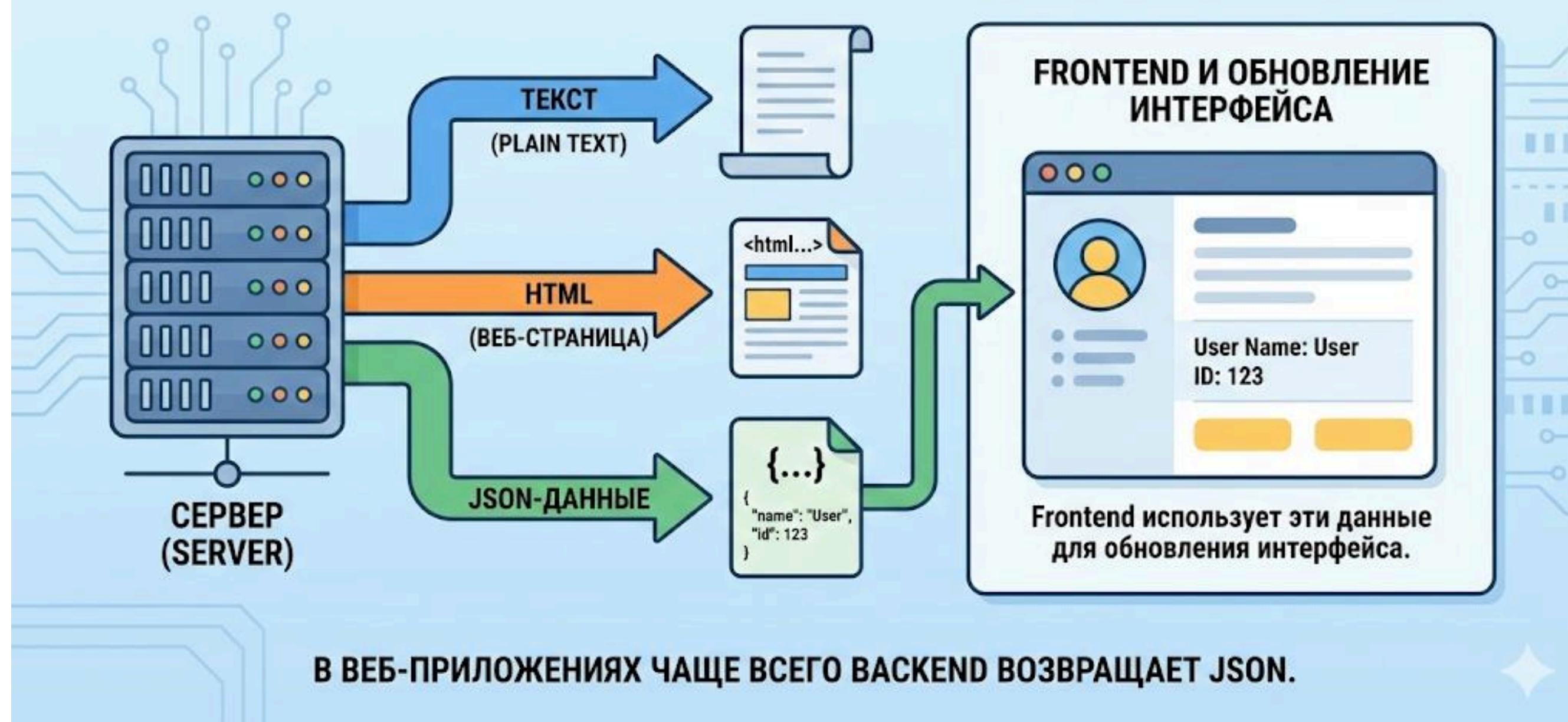


1. Маршрут определяет:
 - путь запроса
 - HTTP-метод
 - функцию обработки
2. Пример:
GET /users → вернуть список пользователей
3. Каждый маршрут – это правило, по которому сервер реагирует на запрос.

ОТВЕТ СЕРВЕРА

ЧТО ПОЛУЧАЕТ КЛИЕНТ

ТИПЫ ДАННЫХ ОТ СЕРВЕРА И ИСПОЛЬЗОВАНИЕ JSON



1. Сервер может отправлять:
2. – текст
 - HTML
 - JSON-данные
3. В веб-приложениях чаще всего backend возвращает JSON.
4. Frontend использует эти данные для обновления интерфейса.

ПРОБЛЕМА ПЕРЕЗАПУСКА

ПОЧЕМУ НЕУДОБНО РАБОТАТЬ БЕЗ NODEMON



1. Обычный сервер нужно перезапускать вручную
после каждого изменения кода.
2. Это замедляет разработку и постоянно отвлекает.
3. Для автоматического перезапуска используют утилиту nodemon.

ЧТО ДЕЛАЕТ NODEMON

АВТОМАТИЧЕСКИЙ ПЕРЕЗАПУСК СЕРВЕРА

ПРИНЦИП РАБОТЫ NODEMON: АВТОМАТИЗАЦИЯ ПЕРЕЗАПУСКА



1. nodemon отслеживает изменения файлов.
2. Когда код изменяется, он:
 - останавливает сервер
 - запускает его снова автоматически
3. Разработчику не нужно перезапускать сервер вручную.

НАСТРОЙКА NODEMON

УДОБНЫЙ РЕЖИМ РАЗРАБОТКИ



1. nodemon устанавливается через npm как dev-зависимость.
2. После этого в package.json добавляют команду запуска сервера.
3. Разработка становится быстрее и комфортнее.

ПЕРВЫЙ СЕРВЕР НА NODE.JS

ЧТО МЫ СЕГОДНЯ СОЗДАЁМ

```
server.listen(PORT, HOST, () => {
  console.log('');
  console.log('=====');
  console.log('  УРОК 1: БАЗОВЫЙ HTTP СЕРВЕР');
  console.log('=====');
  console.log(`    ✅ Сервер успешно запущен!`);
  console.log(`    ⚡ Адрес: http://\${HOST}:\${PORT}`);
  console.log('');
  console.log('  Доступные маршруты:');
  console.log(`    • http://\${HOST}:\${PORT}/ - Главная`);
  console.log(`    • http://\${HOST}:\${PORT}/about - О проекте`);
  console.log(`    • http://\${HOST}:\${PORT}/api/status - API (JSON)`);
  console.log('');
  console.log('  Для остановки: Ctrl + C');
  console.log('=====');
  console.log('');
});

});
```

1. В этом проекте мы создадим простой веб-сервер на Node.js, который:

- принимает HTTP-запросы
- обрабатывает разные URL
- возвращает HTML и JSON
- показывает ошибки 404

2. Это базовая модель работы любого backend-приложения.

3. Пример кода доступен по ссылке:

a. <https://github.com/mrsky1001/fullstack-web-development-course/tree/main/2.%20backend/webinars-lessons/03-nodejs-and-express-basics/>

ВСТРОЕННЫЙ МОДУЛЬ HTTP. HOST И PORT

ОСНОВА ДЛЯ ВЕБ-СЕРВЕРОВ

```
1 // ШАГ 1: Подключаем встроенный модуль 'http'  
2 // -----  
3 // Node.js имеет много встроенных модулей (http, fs, path и др.)  
4 // Модуль 'http' позволяет создавать веб-серверы  
5 // Функция require() - способ подключения модулей в Node.js  
6  
7  
8 const http = require('http');  
9  
10 // ШАГ 2: Определяем настройки сервера  
11 const HOST = 'localhost'; // localhost = ваш компьютер (127.0.0.1)  
12 const PORT = 3000; // Порт 3000 - стандартный для разработки  
13
```

1. Node.js содержит набор встроенных модулей для работы с сетью, файлами и системой. Для создания веб-сервера используется модуль **http**.
2. Подключение выполняется через функцию `require()`.
3. Серверу нужно указать параметры по которым он будет принимать запросы:
 - адрес (`host`)
 - порт (`port`)
4. `localhost` означает, что сервер доступен только на текущем компьютере.
5. Порт `3000` традиционно используется для разработки backend-приложений.
6. В продакшене адрес и порт обычно задаются через переменные окружения

СОЗДАНИЕ СЕРВЕРА

ОБРАБОТЧИК HTTP-ЗАПРОСОВ

```
14 // ШАГ 3: Создаём HTTP сервер
15 // -----
16 // http.createServer() принимает функцию-обработчик (callback)
17 // Эта функция вызывается при КАЖДОМ запросе к серверу
18 // Параметры:
19 // - req (request) - объект с информацией о запросе (URL, метод, за
20 // - res (response) - объект для формирования ответа клиенту
21
22 const server = http.createServer((req, res) => {
23     // Выводим информацию о запросе в консоль сервера
24     console.log(`[${new Date().toLocaleTimeString()}] ${req.method}`)
25 })
```

1. Сервер создаётся с помощью функции:

```
http.createServer((req, res) => { ... });
```

i. **(req, res) => { ... }** - Эта callback-функция вызывается при каждом новом HTTP-запросе. Внутри неё реализуется логика обработки маршрутов и формирования ответов.

ii. **req** – объект запроса (URL, метод, заголовки)

iii. **res** – объект ответа, через который отправляются данные клиенту

МАРШРУТИЗАЦИЯ ОБРАБОТКА РАЗНЫХ URL

```
26 // ШАГ 3.1: Простая маршрутизация по URL
27 // -----
28 // req.url содержит путь запроса (например, '/' или '/about')
29 // Мы проверяем URL и отвечаем по-разному
30
31 > if (req.url === '/') { ... } else if (req.url === '/about') {
49     // Страница "0 проекте"
50
51     res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
52     res.end('...');
57
58 } else if (req.url === '/api/status') {
59     // API endpoint - возвращает данные в формате JSON...
60
61     res.writeHead(200, { 'Content-Type': 'application/json; charset=utf-8' });
62
63     // JSON.stringify() превращает объект JavaScript в строку JSON
64     res.end(JSON.stringify({status: 'ok'}));
65
71
72 } else {
73     // Если URL не найден - возвращаем 404 (Not Found)
74
75     res.writeHead(404, { 'Content-Type': 'text/html; charset=utf-8' });
76     res.end('...');
81 }
```

1. Простейшая маршрутизация реализована через проверку `req.url`:

```
if (req.url === '/') { ... }
else if (req.url === '/about') { ... }
```

2. Таким образом сервер определяет, какой ресурс запрашивает клиент, и возвращает соответствующий контент.

3. Во фреймворках вроде **Express** эта логика реализована через удобную систему маршрутов, но базовый принцип остаётся тем же.

HTML-ОТВЕТЫ

ОТПРАВКА ВЕБ-СТРАНИЦ

```
30
31  if (req.url === '/') {
32      // Главная страница
33
34      // Устанавливаем HTTP статус и заголовок Content-Type
35      res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
36
37      // Отправляем тело ответа и завершаем его
38      res.end(`

39          <h1>🚀 Добро пожаловать!</h1>
40          <p>Это ваш первый Node.js сервер!</p>
41          <p>Попробуйте:</p>
42          <ul>
43              <li><a href="/about">О проекте</a></li>
44              <li><a href="/api/status">API статус (JSON)</a></li>
45          </ul>
46      `);
`
```

1. Для HTML-страниц устанавливается заголовок:

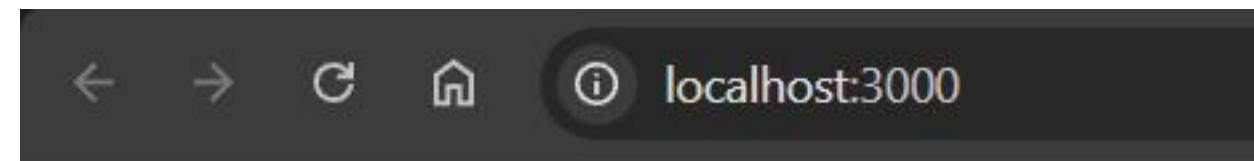
```
res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
```

2. После этого отправляется HTML-код:

```
res.end(`<h1>Добро пожаловать!</h1>`);
```

3. Браузер интерпретирует этот код и отображает страницу пользователю.

4. Так формируются обычные веб-страницы на серверной стороне.



Добро пожаловать!

Это ваш первый Node.js сервер!

Попробуйте:

- [О проекте](#)
- [API статус \(JSON\)](#)

API И JSON

ОТПРАВКА ВЕБ-СТРАНИЦ

```
30
31  if (req.url === '/') {
32      // Главная страница
33
34      // Устанавливаем HTTP статус и заголовок Content-Type
35      res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
36
37      // Отправляем тело ответа и завершаем его
38      res.end(`

39          <h1>🚀 Добро пожаловать!</h1>
40          <p>Это ваш первый Node.js сервер!</p>
41          <p>Попробуйте:</p>
42          <ul>
43              <li><a href="/about">0 проекте</a></li>
44              <li><a href="/api/status">API статус (JSON)</a></li>
45          </ul>
46      `);
`;
```

1. Сервер как источник данных
2. Для API-запросов сервер возвращает данные в формате JSON:

```
res.writeHead(200, { 'Content-Type': 'application/json' });
res.end(JSON.stringify({ status: 'ok' }));
```

3. Такой формат используется для передачи данных между backend и frontend, а также между различными сервисами.
4. Этот принцип лежит в основе **REST API** и микросервисной архитектуры

ОБРАБОТКА ОШИБОК

ОТВЕТЫ ПРИ ОТСУТСТВИИ МАРШРУТА

```
71
72 } else {
73     // Если URL не найден - возвращаем 404 (Not Found)
74
75     res.writeHead(404, { 'Content-Type': 'text/html; charset=utf-8' });
76     res.end(`

# ✗ Ошибка 404


77     <p>Страница "${req.url}" не найдена</p>
78     <p><a href="/">← На главную</a></p>
79   `);
80 }
81 );
82 );
83 |
```

1. Если маршрут не найден, сервер возвращает код 404:

```
res.writeHead(404, { 'Content-Type': 'text/html' });
```

2. Это сообщает браузеру и поисковым системам, что ресурс не существует. Корректная обработка ошибок – обязательная часть любого backend-приложения.

ЗАПУСК СЕРВЕРА

ПРОСЛУШИВАНИЕ ПОРТА

```
server.listen(PORT, HOST, () => {
    console.log('');
    console.log('=====');
    console.log('  ↗ УРОК 1: БАЗОВЫЙ HTTP СЕРВЕР');
    console.log('=====');
    console.log(`  ✓ Сервер успешно запущен!`);
    console.log(`  ⚡ Адрес: http://\${HOST}:\${PORT}`);
    console.log('');
    console.log('  Доступные маршруты:');
    console.log(`    • http://\${HOST}:\${PORT}/ - Главная`);
    console.log(`    • http://\${HOST}:\${PORT}/about - О проекте`);
    console.log(`    • http://\${HOST}:\${PORT}/api/status - API (JSON)`);
    console.log('');
    console.log('  Для остановки: Ctrl + C');
    console.log('=====');
    console.log('');
});
```

1. Сервер запускается методом:

```
server.listen(PORT, HOST, () => { ... });
```

2. Callback-функция выполняется один раз после успешного старта. В ней выводятся сообщения для разработчика:

- а. адрес сервера
- б. доступные маршруты
- с. статус запуска

3. Это помогает быстро проверить работоспособность приложения.

ОБРАБОТКА СИСТЕМНЫХ ОШИБОК

КОНТРОЛЬ СБОЕВ СЕРВЕРА

```
server.listen(PORT, HOST, () => {
    console.log('');
    console.log('=====');
    console.log('  УРОК 1: БАЗОВЫЙ HTTP СЕРВЕР');
    console.log('=====');
    console.log(`  ✅ Сервер успешно запущен!`);
    console.log(`  ⚡ Адрес: http://\${HOST}:\${PORT}`);
    console.log('');
    console.log('  Доступные маршруты:');
    console.log(`    • http://\${HOST}:\${PORT}/ - Главная`);
    console.log(`    • http://\${HOST}:\${PORT}/about - О проекте`);
    console.log(`    • http://\${HOST}:\${PORT}/api/status - API (JSON)`);
    console.log('');
    console.log('  Для остановки: Ctrl + C');
    console.log('=====');
    console.log('');
});
```

1. Сервер подписывается на событие **error**:

```
server.on('error', (err) => { ... });
```

2. Это позволяет:

- а. обнаружить занятый порт
- б. понятное сообщение
- с. корректно завершить процесс

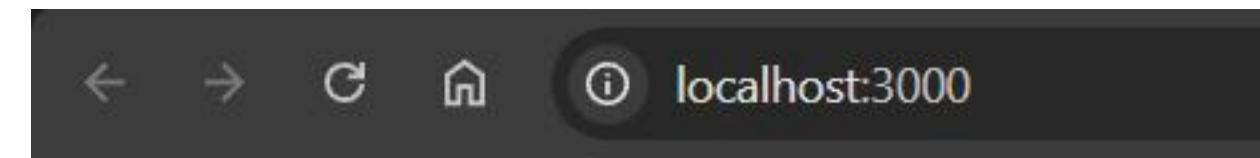
3. Такой подход необходим для стабильной работы приложений в продакшене

ЧТО МЫ ПОЛУЧИЛИ

РЕЗУЛЬТАТЫ ПРОЕКТА

```
=====
| УРОК 1: БАЗОВЫЙ HTTP СЕРВЕР
=====
| Сервер успешно запущен!
| Адрес: http://localhost:3000
|
Доступные маршруты:
• http://localhost:3000/ - Главная
• http://localhost:3000/about - О проекте
• http://localhost:3000/api/status - API (JSON)
|
Для остановки: Ctrl + C
=====
[15:48:35] GET /
[15:48:36] GET /favicon.ico
[16:02:45] GET /about
[16:02:47] GET /
[16:02:47] GET /api/status
```

1. В результате реализован полноценный минимальный backend:
2. HTTP-сервер
 - a. несколько маршрутов
 - b. HTML-страницы
 - c. API-endpoint с JSON
 - d. обработка ошибок





ВОПРОСЫ?

- ПО ТЕМАМ ДИСЦИПЛИНЫ?
- ПО ПРАКТИЧЕСКИМ/ИТОГОВЫМ РАБОТАМ?
- ПО ОРГАНИЗАЦИОННЫМ МОМЕНТАМ?