

12 фреймворков NodeJS для ускорения разработки Web и API

19.04.2024

NodeJS существует в индустрии уже довольно давно. Благодаря своей асинхронной природе и поддержке движка Chrome V8 он стал широко популярен. NodeJS предлагает ряд фреймворков, которые поставляются с различными библиотеками, инструментами и шаблонами, чтобы помочь разработчикам обойти препятствия при создании приложений. Nodejs – это, пожалуй, один из лучших JavaScript-фреймворков для разработки полнофункционального приложения. Если вы решили использовать Nodejs, **следующие фреймворки** и плагины пригодятся вам при разработке бэкенда и API-сервисов.

Express

Express – один из самых популярных фреймворков для разработки веб-интерфейсов и API для NodeJS. Он получил настолько широкое распространение, что почти каждый проект веб-разработки начинается с интеграции Express JS.



Существует множество причин, по которым мы выбрали ExpressJS в качестве первого плагина.

- Большой набор функций для поддержки всего, что вам нужно в ваших задачах разработки
- Удобная маршрутизация для направления веб-запросов к функциям
- Предоставляет организованную платформу для кодирования API
- Поддерживается большинством других вспомогательных библиотек и плагинов
- Охрана и постоянное обслуживание, чтобы соответствовать стандартам
- Большая поддержка сообщества

В дополнение к этим преимуществам разработчики плагина также создали простой в использовании генератор проектов. Этот генератор может создать шаблонный проект, чтобы вы быстрее приступили к работе.

Настройка проекта

Давайте создадим проект, чтобы изучить основы экспресса. Убедитесь, что вы установили ноду в своей системе.

1. Создайте папку проекта.
2. Откройте его в терминале.
3. Установите `express`, `body-parser`, `cookie-parser`, `cors` и `nodemon`.
4. Добавьте файл `index.js` в проект.
5. Добавьте скрипт запуска приложения в файл `packages.json` с помощью команды `nodemon index.js`.

Пакеты

- **express**: это основной пакет нашего приложения, который помогает нам создавать API.
- **body-parser**: это промежуточное программное обеспечение, которое анализирует входящие данные из API и добавляет

их в req.body.

- **cookie-parser**: это промежуточное программное обеспечение, которое анализирует заголовок Cookie и добавляет его в req.cookie.
- **cors**: это промежуточное программное обеспечение, которое используется для включения CORS.
- **nodemon**: используется для запуска нашего приложения, которое будет перезапускать сервер при изменении любого файла.

Это основные пакеты, необходимые для экспресс-приложения, чтобы сделать нашу жизнь проще. Вам может понадобиться больше пакетов, исходя из особенностей вашего проекта. Не беспокойтесь об этом сейчас, добавление пакетов осуществляется одной командой.

Экспресс-приложение

Давайте посмотрим на базовое приложение с различными API.

```
const express = require("express");const bodyParser =
require("body-parser");const cookieParser = require("cookie-
parser");const cors = require("cors");// инициализация
expressconst app = express();const port = 3000;// добавляем
промежуточные           модули           в
приложениеapp.use(bodyParser.json());app.use(cookieParser());a
pp.use(cors());// req: мы будем использовать этот параметр для
получения деталей API-запроса// res: мы будем
использоватьapp.get("/", (req, res)=> { return
res.send("Hello, World!");});app.get("/json", (req, res)=> {
return res.json({ greetings: "Hello, World!" }); // также
можно использовать res.send({})});app.get("/path-
params/:name", (req, res)=> { // все параметры пути будут
присутствовать в объекте req.params const { name } =
req.params; return res.json({ greetings: `Hello, ${name}!`
});});app.get("/query-params", (req, res)=> { // все параметры
запроса будут присутствовать в объекте req.query const { name
} = req.query; return res.json({ greetings: `Hello, ${name ?
name : "Geekflare"}!` });});app.post("/post", (req, res)=> {
```

```
// данные будут присутствовать в req.body const { name } = req.body; console.log(req.body); return res.json({ greetings: `Hello, ${name ? name : 'Geekflare'}!` });}); app.listen(port, () => { console.log(`App listening on port ${port}`); });
```

Запустите приложение с помощью `npm start` и попробуйте все API, которые мы написали. Читайте документацию, чтобы узнать больше о каждой концепции.

Sails

Sails – это полноценный фреймворк с архитектурой MVC. В его основе используются ExpressJS и SocketIO. Sails.js стал популярен благодаря своей архитектуре корпоративного уровня, которая позволяла быстрее интегрироваться с базой данных с помощью объектов модели.



Вот некоторые из преимуществ:

- Sails.JS поставляется с проектом для немедленного создания шаблона проекта
- Структура папок в Sails.JS очень хорошо организована
- Разработка объектных моделей и их отображение с помощью фронтенда происходит быстро
- Позволяет легко интегрировать промежуточное программное обеспечение для авторизации, аутентификации и предварительной обработки.
- Встроенная поддержка AWS S3 и GridFS

Настройка проекта

Чтобы создать проект `sailsjs`, нам понадобится пакет `npm` под названием `sail`. Давайте установим его глобально с помощью следующей команды.

```
npm install sails -g
```

Перейдите в каталог, где вы хотите создать свой проект. Теперь выполните следующую команду, чтобы создать приложение `sailsjs`.

паруса новый базовый апп

Появится запрос на выбор шаблона. Выберите вариант "**Пустой**". Дождитесь окончания создания проекта. Откройте проект в своем любимом редакторе кода. Выполните команду `sails lift`, чтобы запустить приложение. Откройте URL-адрес `http://localhost:1337/` в браузере, чтобы увидеть приложение. Если вы посмотрите на приложение в редакторе кода, то обнаружите множество папок и файлов. Подробное объяснение каждой папки и файла вы можете найти на странице документации `Sails`. В этом уроке мы рассмотрим одну папку `api/controllers` и `config/routes.js`.

Приложение SailsJS

Давайте посмотрим, как создавать API в приложении `sailsjs`. Чтобы создать API в приложении `sailsjs`, выполните следующие действия.

1. Добавьте конечную точку API в файл `config/routes.js`.
2. Создайте действие для конечной точки API с помощью команды `sails generate action sample --no-actions2`.
3. Добавьте код API в файл действий, созданный в шаге 2.

Маршруты

После добавления конечных точек API файл `routes.js` будет выглядеть примерно так, как показано ниже.

```
module.exports.routes = {" GET /": { action: "home" }, " GET /json": { action: "json" }, " GET /path-params/:name": { action: "path-params" }, " GET /query-params": { action: "query-params" }, " POST /post": { action: "post" },};
```

Каждая конечная точка API указывает на одно действие. Мы должны сгенерировать эти файлы действий с помощью команды, упомянутой в предыдущем разделе. Давайте сгенерируем все файлы действий для указанных выше конечных точек.

Действия

Мы добавили 5 конечных точек. Давайте проверим соответствующий код для каждой конечной точки.

home.js

```
module.exports = async function home(req, res) { return res.send("Hello, World!");}
```

json.js

```
module.exports = async function json(req, res) { return res.json({ greetings: "Hello, World!" });}
```

path-params.js

```
module.exports = async function pathParams(req, res) { const { name } = req.params; return res.json({ greetings: `Hello, ${name}!` });}
```

post.js

```
module.exports = async function post(req, res) { const { name } = req.body; console.log(req.body); return res.json({ greetings: `Hello, ${name ? name : 'Geekflare'}!` }); };
```

query-params.js

```
module.exports = async function queryParams(req, res) { const { name } = req.query; return res.json({ greetins: `Здравствуйте, ${name ? name : "Geekflare"}!` });}
```

Существует и другой способ написания действий. Прочитайте

документацию, чтобы узнать больше о фреймворке.

Нарі

Фреймворк Нарі изначально был создан для устранения недостатков фреймворка ExpressJS. Walmart заметила эти недостатки во время подготовки к мероприятию с большим трафиком.



Нарі.JS – это надежный фреймворк для создания сервисов и API. Он известен своей стабильностью и надежностью.

Настройка проекта

Давайте создадим проект, чтобы прикоснуться к основам **Нарі.JS**. Мы можем настроить проект Нарі аналогично обычному проекту NodeJS. Выполните приведенные ниже команды для настройки проекта.

```
cd your_project_folder_path npm init -y## для инициализации проекта
node npm install @hapi/hapi## установка основного пакета для работы с Нарі.JS
npm install nodemon ## для запуска нашего приложения
```

Создайте файл `index.js` в проекте. Добавьте стартовый скрипт в файл `package.json` с помощью команды `nodemon index.js`.

Приложение HapiJS

Ознакомьтесь с основными API в Hapi ниже.

```
const hapi = require("@hapi/hapi");const app = async ()=> { //
инициализация сервера hapi const server = hapi.server({port:
3000,host: "localhost",});server.route({method: "GET",path:
"/",handler: (request, h)=> { return "Hello,
World!";}},);server.route({method: "GET",path:
"/json",обработчик: (request, h)=> { return { greetings:
"Hello, World!" }};});server.route({method: "GET",path:
"/path-params/{name}",handler: (request, h)=> { const name =
request.params.name; return { greetings: `Здравствуйте,
${name}!` }};});server.route({method: "GET",path: "/query-
params",handler: (request, h)=> { const name =
request.query.name; return { greetings: `Здравствуйте, ${name
? name : "Geekflare"}!` }};});server.route({method:
"POST",path: "/post",handler: (request, h)=> { const data =
request.payload;console.log(data); return { greetings:
`Здравствуйте, ${data.name ? data.name : "Geekflare"}!`
}};}); // запуск сервера await
server.start();console.log(`App is running on
${server.info.uri}`);};app();
```

Мы добавили различные API для изучения основ Hapi. Вы можете перенести все маршруты в отдельный файл, чтобы сделать их чистыми.

Total

Total – это серверная платформа, которая предоставляет готовую к использованию платформу для создания приложений реального времени, чатботов, IoT, eCommerce, REST. Она также позволяет премиум-пользователям публиковать свои приложения на платформе для использования другими пользователями.



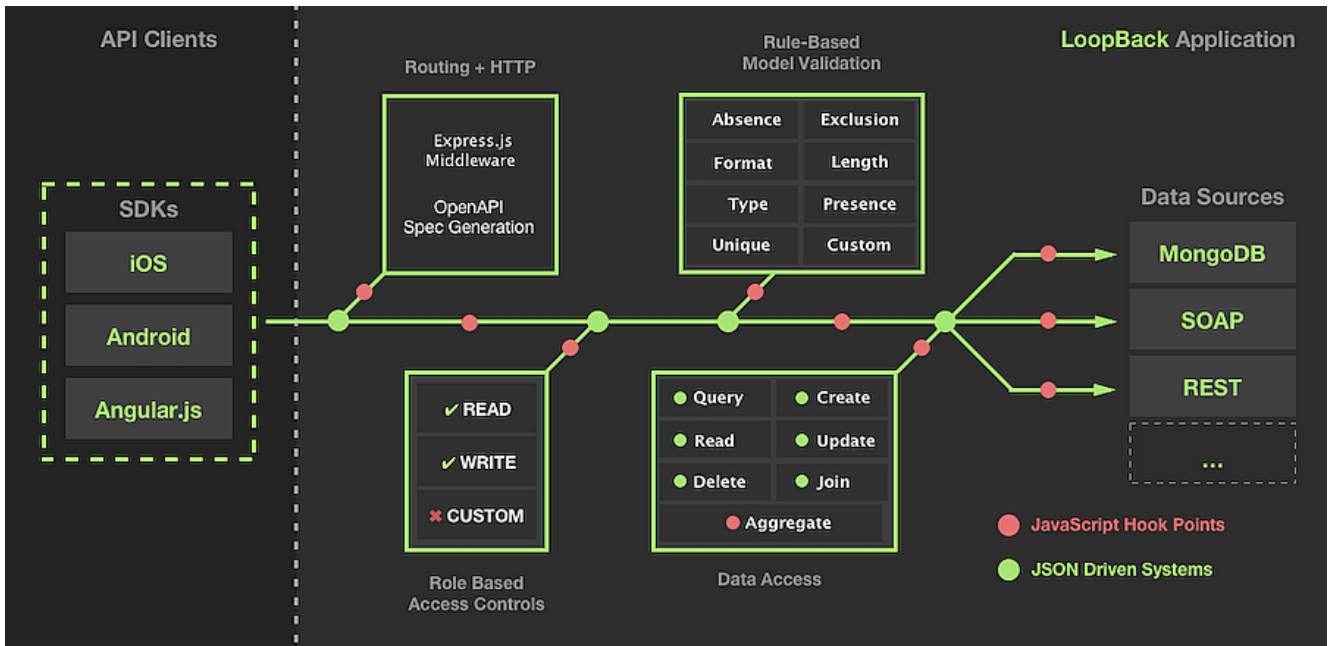
total.js

Преимущества использования Total.JS в качестве основы для разработки заключаются в следующем:

- Возможности быстрого создания прототипов
- В комплект входит множество готовых компонентов, что позволяет ускорить разработку
- Хранит библиотеку приложений, которые можно легко извлечь и интегрировать в ваше приложение
- Модульная основа, позволяющая упростить распределение работ в большом проекте
- Чат сообщества
- Постоянное поддержание хранилища приложений, готовых к использованию

LoopBack

LoopBack – это фреймворк для разработки API, который интегрирован с проводником API. Проводник API можно легко подключить к клиентским приложениям с помощью легкодоступных LoopbackJS SDK. SDK доступны для Android, AngularJS, Angular 2+, а также iOS-приложений.



LoopBack доверяют GoDaddy, Symantec, Bank of America и многие другие. На их сайте вы найдете множество примеров по созданию внутренних API, безопасных REST API, сохранению данных и т.д. И да, здесь есть встроенный проводник API.

Настройка проекта

Давайте создадим проект loopback и посмотрим, как с его помощью создавать базовые API. Выполните следующую команду, чтобы установить loopback CLI.

```
npm install -g@loopback/cli
```

Выполните приведенную ниже команду, чтобы начать настройку проекта.

```
приложение lb4
```

Ответьте на все вопросы в терминале. Вы можете ответить на вопросы в соответствии с вашими предпочтениями. Откройте приложение в вашем любимом редакторе кода и запустите его командой `npm start`. Перейдите на сайт <http://localhost/>, чтобы проверить приложение loopback.

Приложение LoopBack

Если вы заглянете в папку `src`, там будет папка `controllers`. Это место, куда мы добавим контроллеры, которые будут содержать наши API. Для создания контроллеров в `loopback` нам нужно использовать следующую команду.

```
контроллер lb4
```

Ознакомьтесь с различными API ниже. Мы добавили несколько API в контроллер.

```
import {get, param, post, requestBody} from
 '@loopback/rest';interface Response {greetings: string;}export
 class GeekflareController {@get('/hello') home(): string {
 return 'Hello, World!'; }@get('/json') json(): Response {
 return {greetings: 'Hello, World!'}; } // доступ к параметрам
 пути с помощью декоратора @param@get('/path-params/{name}')
 pathParams(@param.path.string('name') name: string): Response
 { return {greetings: `Здравствуйтесь, ${name}!`}; }@get('/query-
 params') queryParams(@param.query.string('name') name:
 string): Response { return {greetings: `Здравствуйтесь, ${name ?
 name : "Geekflare"}!`}; } // доступ к параметру пути с помощью
 декоратора @requestBody@post('/post')
 postMethod(@requestBody() data: any): Response
 {console.log(data); const {name} = data; return {greetings:
 `Здравствуйтесь, ${name ? name : 'Geekflare'}!`};}}
```

Мы рассмотрели, как создавать API и получать доступ к различным вещам, необходимым для основ REST API. Во фреймворке `LoopBack` есть еще много чего интересного. Их документация – подходящее место для глубокого погружения во фреймворк.

Meteor

`Meteor` – это комплексное решение для веб-разработки и создания API с невероятным дизайном в основе. `Meteor` – это фреймворк, который используется для быстрого создания приложений. Архитектура `Meteor` позволяет выполнять код как на фронтенде, так и на бэкенде без необходимости переписывать код.



Это значительно повышает скорость разработки. Существенными преимуществами использования Meteor являются:

- Система разработки гибридных приложений
- С помощью единой кодовой базы вы можете создать приложение для настольных компьютеров, веб-приложение, а также мобильное приложение.
- Он поставляется с тесно связанным фронтендом, который помогает сократить объем кода.
- Высокая расширяемость за счет множества плагинов
- Поддерживает различные шаблонизаторы фронтенда
- Поддерживает функцию “горячего проталкивания” кода, что позволяет избавиться от необходимости обновлять мобильные приложения

Restify

Создайте готовый к производству семантически корректный REST-полный веб-сервис с помощью Restify.



Он использует только необходимые модули Express JS, что делает

кодovou базу более легкой по сравнению с другими фреймворками. Ему доверяют Netflix, Pinterest, Joyent и др. – вы не ошибетесь, выбрав его.

Настройка проекта

Давайте создадим проект `restify` и посмотрим, как писать базовые API. Выполните следующие команды, чтобы создать новый проект `restify`.

Прежде чем продолжить, убедитесь, что у вас есть node версии 11. Restify не поддерживает последние версии node.

```
cd your_project_folder npm init -y## инициализация проекта
node npm i restify ## установка основного пакета
npm i restify-plugins## для добавления некоторых промежуточных инструментов парсинга в наше приложение
npm i nodemon ## для запуска нашего приложения
```

После установки всех пакетов добавьте стартовый скрипт в файл `package.json` с помощью команды `nodemon index.js`. Не забудьте добавить файл `index.js` в проект.

Приложение Restify

Давайте создадим несколько API, чтобы изучить основы.

```
const restify = require("restify");
const restifyPlugins = require("restify-plugins");
function home(req, res, next) {
  res.send("Hello, World!");
  next();
}
function json(req, res, next) {
  res.json({ greetings: "Hello, World!" });
  // вы также можете использовать req.send(JSONData)
  next();
}
function pathParams(req, res, next) {
  const { name } = req.params;
  res.json({ greetings: `Hello, ${name}!` });
  next();
}
function queryParams(req, res, next) {
  const { name } = req.query;
  res.json({ greetings: `Hello, ${name ? name : "Geekflare"}!` });
  next();
}
function post(req, res, next) {
  const data = req.body;
  console.log(data);
  res.json({ greetings: `Hello, ${data.name ? data.name : "Geekflare"}!` });
  next();
}
// создание сервера restify
const server =
```

```
restify.createServer();// добавление промежуточных элементов
парсингасerver.use(restifyPlugins.jsonBodyParser({ mapParams:
true }));server.use(restifyPlugins.queryParser({ mapParams:
true }));// добавление маршрутовserver.get("/",
home);server.get("/json", json);server.get("/path-
params/:name", pathParams);server.get("/query-params",
queryParams);server.post("/post", post);// запуск
приложенияserver.listen(3000, function () {console.log(`App is
running at ${server.url}`)});
```

Ознакомьтесь с документацией по restify, чтобы узнать больше о фреймворке.

Коа

Коа в первую очередь использует генераторы кода, позволяющие разработчикам ускорить процесс разработки. В комплект поставки входят различные промежуточные модули и плагины, помогающие управлять сессиями, запросами, куками, а также транзакциями данных.



next generation web framework for node.js

Разработкой koa занимается та же команда, что и Express. Он работает с Nodejs 7.6+ и содержит множество примеров для начала работы.

Настройка проекта

Давайте настроим проект koa с помощью следующих команд

```
cd your_project_folder npm init -y## инициализация проекта
node npm i koa ## основной пакетkoa npm i koa-route## пакет
route для работы с маршрутами API npm i koa-bodyparser##
```

пакет `parser` для разбора тела запроса `npm i nodemon ##` для запуска

Приложение Коа

Создание API с помощью коа очень простое, как и в других фреймворках, которые мы рассматривали ранее. Давайте посмотрим на код.

```
const koa = require("koa");const route = require("koa-route");const bodyParser = require("koa-bodyparser");const app = new koa();const home = async (ctx)=> {ctx.body = "Hello, World!";};const json = async (ctx)=> {ctx.body = { greetings: "Hello, World!" };;};// все параметры пути будут переданы в функцию с тем же именем, которое указано в маршрутеconst pathParams = async (ctx, name)=> {ctx.body = { greetings: `Привет, ${name}!` };;};const queryParams = async (ctx)=> {const name = ctx.query.name;ctx.body = { greetings: `Здравствуйтесь, ${name ? name : "Geekflare"}!` };;};const post = async (ctx)=> {const {body: { name }, } = ctx.request;ctx.body = { greetings: `Здравствуйтесь, ${name ? name : "Geekflare"}!` };;};app.use(bodyParser());app.use(route.get("/", home));app.use(route.get("/json", json));app.use(route.get("/path-params/:name", pathParams));app.use(route.get("/query-params", queryParams));app.use(route.post("/post", post));app.listen(3000);
```

Nest

Hello, nest!

A progressive Node.js framework for building efficient, reliable and scalable server-side applications.

Documentation

Source code

Nest – это фреймворк для создания серверных приложений для узлов. Он использует express под капотом для HTTP-сервера. Мы также можем настроить его с помощью Fastify. Он поддерживает TypeScript и построен с использованием концепций ООП, функционального программирования и функционально-реактивного программирования. Давайте создадим в нем проект и посмотрим, как писать базовые API.

Настройка проекта

Выполните следующие команды, чтобы создать проект.

```
npm i -g@nestjs/cli## глобальная установка nest CLI nest new your_project_name ## создание проекта с помощью nest CLI
```

Приложение Nest

В папке src приложения есть несколько файлов. app.controller.ts – это файл, в котором мы будем включать API. Помните, что мы только собираемся посмотреть, как писать базовые приложения. Давайте рассмотрим различные базовые API, которые покажут различные концепции API.

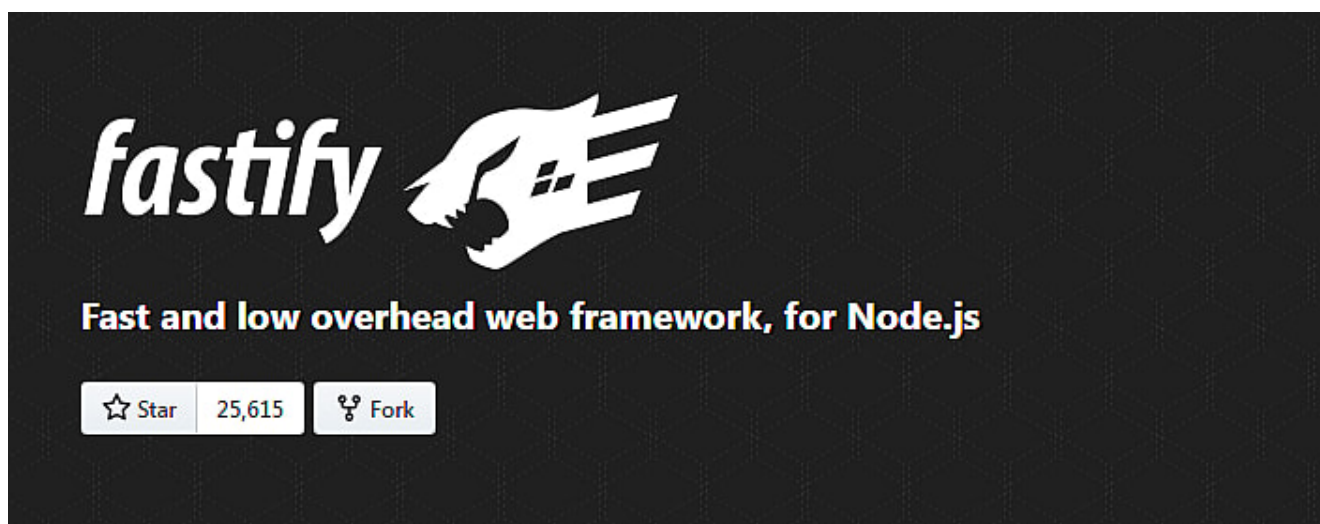
```
import { Body, Controller, Get, Param, Post, Query } from '@nestjs/common';import { AppService } from './app.service';export interface IResponse {greetings:string;}@Controller()export class AppController {constructor(private readonly appService: AppService){}@Get('/') home(): string { return 'Hello, World!';
```



```
}@Get('/json') json(): IResponse { return { greetings: 'Hello, World!' }; }@Get('/path-params/:name') pathParams(@Param() params): IResponse { const { name } = params; return { greetings: `Здравствуйте, ${name}!` }; }@Get('/query-params') queryParams(@Query() params): IResponse { const { name } = params; return { greetings: `Здравствуйте, ${name ? name : 'Geekflare'}!` }; }@Post('/post') post(@Body() body): IResponse { const { name } = body; return { greetings: `Здравствуйте, ${name ? name : 'Geekflare'}!` };}}
```

Запустите приложение и проверьте все API. Вы также можете написать эти API в сервисном слое и получить к ним доступ в контроллере. Обратитесь к документации nest, чтобы узнать больше и понять фреймворк.

Fastify



Fastify – еще один фреймворк в семействе фреймворков NodeJS. Как следует из названия, он претендует на звание одного из самых быстрых NodeJS-фреймворков. Давайте посмотрим на некоторые основные возможности фреймворка.

- Быстрый и высокопроизводительный. Утверждается, что он может обрабатывать до 30 тысяч запросов в секунду в зависимости от сложности кода.
- Дружественный TypeScript.
- Удобство для разработчиков с выразительным API для быстрой разработки.

- Он поставляется со встроенным логгером. Он использует Pino (логгер NodeJS).

Настройка проекта

Давайте создадим проект `fastify`, чтобы изучить основы разработки API. Выполнение следующих команд создаст для вас новый проект `fastify`.

```
npm install --global fastify-cli## установка fastify CLI
fastify generate project_name ## создание проекта с помощью fastify CLI
```

Приложение Fastify

Fastify CLI сгенерировал для нас проект. В настоящее время нас волнует только написание API. Чтобы написать новые API, откройте файл `routes/root.js`. Добавьте в файл следующие API.

```
module.exports = async function (fastify, opts) {
  fastify.get("/", async function (request, reply) { return "Hello, World!"; });
  fastify.get("/json", async function (request, reply) { return { greetings: "Hello, World!" }; });
  fastify.get("/path-params/:name", async function (request, reply) { const { name } = request.params; return { greetings: `Здравствуйте, ${name}!` }; });
  fastify.get("/query-params", async function (request, reply) { const { name } = request.query; return { greetings: `Здравствуйте, ${name ? name : "Geekflare"}!` }; });
  fastify.post("/post", async function (request, reply) { const { name } = request.body; return { greetings: `Здравствуйте, ${name ? name : "Geekflare"}!` }; });
};
```

Запустите приложение и протестируйте все API. Это основы написания API.

tinyhttp

`tinyhttp` – это легкий и экспресс-подобный JS-фреймворк. Он

поставляется со встроенным логгером, JWT и CORS. Давайте создадим проект `tiny` и изучим его основы.



Настройка проекта

Выполните следующие команды, чтобы настроить проект `tinyhttp`.

```
cd you_project_folder npm init -ynpm i @tinyhttp/app##  
основной пакет tinyhttp npm i milliparsec ## для разбора данных  
запроса и добавления их в объект запроса к ключу body npm i  
nodemon ## для запуска нашего приложения, которое будет  
перезапускаться автоматически при каждом изменении файлов
```

Нам нужно сделать еще одну вещь. Добавьте модуль `type key with value` в файл `package.json`. Мы добавляем его, потому что `tinyhttp` не поддерживает `require`, вместо него нужно использовать утверждения `import`.

`tinyhttp` App

API `tinyhttp` выглядит почти так же, как и у `express` app. Давайте проверим различные API в нем.

```
import { App } from "@tinyhttp/app";import { json } from  
"milliparsec";const port = 3000;const app = new  
App();app.use(json());// req: мы будем использовать этот  
параметр для получения деталей API-запроса// res: мы будем  
использоватьapp.get("/", (req, res)=> {res.send("Hello,  
World!");});app.get("/json", (req, res)=> {res.json({  
greetings: "Hello, World!" }); // также можно использовать  
res.send({})});app.get("/path-params/:name", (req, res)=> { //
```

```
все параметры пути будут присутствовать в объекте req.params
const { name } = req.params;res.json({ greetings: `Hello,
${name}!` });});app.get("/query-params", (req, res)=> { // все
параметры запроса будут присутствовать в объекте req.query
const { name } = req.query;res.json({ greetings: `Hello,
${name ? name : "Geekflare"}!` });});app.post("/post", (req,
res)=> { // данные будут присутствовать в req.body const {
name } = req.body;res.json({ greetings: `Hello, ${name ? name
: "Geekflare"}!` });});app.listen(port, ()=> {console.log(`App
running on port ${port}`)});});
```

SocketIO



Socket.IO

Bidirectional and low-latency communication for every platform

[Get started](#)

[Documentation](#)

SocketIO – это фреймворк для веб-сокетов, который доступен для нескольких языков программирования. В NodeJS SocketIO позволяет создавать приложения с веб-сокетами, такие как чат-боты, бегущие строки, API для приборных панелей и другие. SocketIO имеет значительные преимущества перед обычной библиотекой веб-сокетов NodeJS.

- Поддержка пользовательской маршрутизации URL для веб-сокетов
- Автоматически генерируемые идентификаторы для каждого сокета
- Простое управление комнатами розеток для передачи данных
- Более простая интеграция с Express JS
- Поддержка кластеризации с Redis
- Поддержка аутентификации сокетов с помощью дополнительного плагина – socketio-auth

- Встроенный запасной протокол HTTP, основанный на обработке сервера, который не поддерживает HTTP 1.1

Заключение

Люди широко используют NodeJS в своих проектах. В результате мы имеем различные фреймворки на выбор. Если посмотреть на фреймворки, то большинство из них имеют схожие API. Поэтому, если у вас есть хорошие знания о NodeJS и любом фреймворке, этого более чем достаточно, чтобы начать работу с любым другим фреймворком NodeJS. Разработка приложений не так проста из-за этих фреймворков. Используйте те фреймворки, которые вам нравятся, и продолжайте изучать другие.