Разработка backend-фреймворка для Node.js на TypeScript

И. М. Шелепугин

Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)

shelepuginivanm@gmail.com

Аннотация. **Проведен сравнительный анализ backend-фреймворков для платформы Node.js. Разработан и протестирован backend-фреймворк, а также его вспомогательные компоненты.**

Ключевые слова: информатика; программирование; серверная разработка; Node.js; TypeScript; высоконагруженные системы

I. Введение

На сегодняшний день создание сайтов, приложений и сервисов зачастую предполагает не только клиентскую разработку, но и серверную. Несмотря на появление различных сред выполнения и подходов, таких как SSG, SSR, ISR [1], позволяющих писать меньше серверного кода, для высоконагруженных систем наличие полноценного серверного приложения является необходимостью. Node.js – среда исполнения языка JavaScript – одна из самых распространённых платформ для написания серверных приложений, обладающая рядом преимуществ перед другими языками программирования:

- высокая скорость обработки HTTP-запросов [2];
- простота написания кода;
- обширная стандартная библиотека;
- наличие драйверов и SDK для различных инфраструктурных решений.

Современная разработка на Node.js также требует наличие backend-фреймворка – программной платформы, предоставляющей набор классов, методов и утилит для разработки ПО. Фреймворк напрямую взаимодействует с низкоуровневым API платформы, предоставляет более широкий функционал по сравнению с её встроенными модулями. Использование фреймворков экономит время на разработку продукта и снижает риск возникновения ошибок. К прочим преимуществам использования фреймворков также относят:

- чистота, адаптируемость кода, избежание дублирования;
- повышенная безопасность ПО;
- возможность масштабирования приложения;
- возможность сосредоточится на написании бизнес-логики, специфичной для данного проекта.

В настоящей статье описан процесс проектирования, разработки и тестирования backend-фреймворка Lunatic.

II. Обзор существующих backend-фреймворков для Node.js

На сегодняшний день существует большое количество backend-фреймворков, написанных на Node.js, однако в подавляющем большинстве проектов используются лишь самые популярные из них. Сравнение наиболее популярных [3] backend-фреймворков для платформы Node.js представлено в табл. 1. Данные о количестве загрузок в неделю получены из npm – реестра пакетов Node.js [4].

ТАБЛИЦА I.

Название	Загрузок в неделю	Преимущества	Недостатки
Express	36.9 млн.	Гибкость	Отсутствие встроенной типизации. Отсутствие базового функционала по умолчанию.
Nest.js	4.2 млн.	Встроенная типизация Широкий функционал Единообразие модулей	Менее производительный Большой размер пакета
Koa	3.7 млн.	Гибкость	Нет встроенной типизации. Небольшой функционал. Необходимо устанавливать доп. зависимости.
Fastify	1.2 млн.	Гибкость Встроенная типизация Высокая скорость работы	Очень большой размер. Необходимо устанавливать доп. зависимости.

Три из четырёх наиболее популярных backend фреймворков обладают небольшим функционалом и требуют установки дополнительных зависимостей для работы часто используемого функционала. Кроме того, Express и Koa не типизированы по умолчанию – для работы с TypeScript требуется установка пакетов с объявлениями типов. Последний фреймворк, Nest.js, обладает широким функционалом, который можно расширять при помощи дополнительных модулей, однако он менее производительный как во время тестирования и сборки проекта, так и во время его работы, также конечное приложение обладает большим размером. Кроме того, его функциональность, как правило, избыточна для большинства проектов.

III. Проектирование фреймворка

Backend-фреймворк должен обладать определенным, наиболее часто используемым функционалом: обработка запросов разных форматов и различных HTTP-методов, маршрутизация, промежуточные обработчики и проч. Весь функционал можно логически разделить на две зоны ответственности, где каждая зона отвечает за определённый набор функций:

- ядро фреймворка: маршрутизация, получение запросов, регистрация обработчиков;
- Middleware-функции: обработка и валидация различных форматов входных данных (формы, JSON, cookie-файлы и проч.), расширение базового функционала фреймворка.

Существуют особенности, специфичные для платформы Node.js и других сред исполнения языка JavaScript:

- встроенная поддержка языка TypeScript (типизации);
- поддержка разных типов модулей (CommonJS, ES Modules и проч.);
- совместимость с различными версиями сред исполнения JavaScript.

Таким образом, определим основной функционал фреймворка, который необходимо реализовать:

- ядро фреймворка;
- набор middleware-функций;
- поддержка различных сред исполнения и их версий;
- поддержка различных типов модулей;
- поддержка типизации.

IV. Разработка ядра фреймворка

Ядро фреймворка – это набор основных классов, функций и методов, а также все его основополагающие и центральные компоненты. Оно обеспечивает функционирование фреймворка, а также предоставляет инструменты и модули для разработки программных приложений. Как правило, ядро включает в себя различные компоненты, такие как внутренняя логика, слои абстракции, общие вспомогательные модули и т. д. Кроме того, ядро зачастую отвечает за интеграцию с другими технологиями, фреймворками и библиотеками, что позволяет использовать множество решений в рамках одного проекта.

Ниже приведены некоторые из реализованных функций ядра фреймворка:

- маршрутизация, обработка параметров пути запроса, в том числе динамическая маршрутизация;
- система обработки запросов, регистрация функций-обработчиков для специфичных HTTP-методов и путей запроса;
- слои абстракции для входящего запроса и ответа сервера в виде классов Request и Response соответственно.
- совместимость с другими фреймворками и библиотеками за счет внедрения зависимостей;
- конфигурация некоторых настроек фреймворка, изменяющих его поведение.

V. Создание middleware-функций

Middleware-функции – это программный слой между различными компонентами приложения, который предоставляет набор функциональных возможностей и сервисов, облегчающих интеграцию и взаимодействие различных программных систем, а также расширение её существующего функционала.

Middleware-функции используются для широкого спектра задач, к числу которых относят аутентификацию и авторизацию пользователей, обработку различных типов входных данных, кэширование данных, мониторинг и логирование, безопасность системы и проч. На данный момент в фреймворке реализованы middleware-функции, решающие следующие задачи:

- поддержка различных форматов тел HTTP-запросов, таких как application/json, text/plain, multipart/form-data;
- обработка CORS-запросов;
- возможность использования файлов cookie;
- отправка статических файлов, таких как, изображения, шрифтов и проч.).

VI. Тестирование кодовой базы фреймворка

Тестирование – это процесс выявления ошибок и уязвимостей в работе программы. Это необходимый этап разработки, который позволяет убедиться в правильности работы отдельных модулей фреймворка, оценить их производительность, предотвратить наличие уязвимостей, а также проверить работоспособность системы в целом.

Кодовая база проекта протестирована более чем на 99 %, написаны unit-тесты на отдельные компоненты и модули фреймворка, а также интеграционные тесты, проверяющие корректность работы фреймворка как единой системы. Для написания тестов использовалась библиотека Jest, процесс тестирования автоматизирован при помощи сервиса GitHub Actions [5]: при изменении главной ветки репозитория, а также при отправке Pull Request’ов, производится запуск тестов на всех стабильных версиях Node.js, начиная с 14.x.

```
File                | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
--------------------|---------|----------|---------|---------|-------------------
All files           |   99.37 |    98.63 |   96.38 |   99.36 |
 src                |   98.66 |    97.29 |   93.87 |   98.64 |
  index.ts          |     100 |      100 |       0 |     100 |
  request.ts        |     100 |    66.66 |     100 |     100 | 23-36
  response.ts       |    97.5 |      100 |    90.9 |   97.46 | 35-36
  router.ts         |     100 |      100 |     100 |     100 |
  server.ts         |   97.56 |     92.3 |     100 |    97.5 | 78
 src/middlewares    |     100 |      100 |     100 |     100 |
  body-parser.ts    |     100 |      100 |     100 |     100 |
  cookie-parser.ts  |     100 |      100 |     100 |     100 |
  cors.ts           |     100 |      100 |     100 |     100 |
  form-parser.ts    |     100 |      100 |     100 |     100 |
  serve-static.ts   |     100 |      100 |     100 |     100 |
 src/utils          |     100 |      100 |     100 |     100 |
  constants.ts      |     100 |      100 |     100 |     100 |
  mime.ts           |     100 |      100 |     100 |     100 |
  normalize-path.ts |     100 |      100 |     100 |     100 |
  trim-path-start.ts |     100 |      100 |     100 |     100 |
 tests/mocks        |     100 |      100 |     100 |     100 |
  req.body.mock.ts  |     100 |      100 |     100 |     100 |
  req.files.mock.ts |     100 |      100 |     100 |     100 |
--------------------|---------|----------|---------|---------|-------------------

Test Suites: 21 passed, 21 total
Tests:       109 passed, 109 total
Snapshots:   0 total
Time:        2.581 s, estimated 4 s
```

Рис. 1. Тестирование функционала фреймворка

VII. Документирование фреймворка

Важным этапом разработки проекта является документирование. Документация включает в себя такие разделы, как инструкция по установке, гайд по использованию различных модулей фреймворка, справочник методов, классов и функций.

Документация к проекту написана на русском и английском языках, доступна для ознакомления на сайте фреймворка [6].

VIII. Заключение

Проведен сравнительный анализ существующих backend-фреймворков для платформы Node.js. Спроектирован, разработан и протестирован backend-фреймворк Lunatic.

Список литературы

[1] Fukuzaki, Arisa Understanding Next.js's ISR against the SSR & SSG / Arisa Fukuzaki. — Текст: электронный // Storyblok : [Электронный ресурс]. — URL: https://www.storyblok.com/mp/nextjs-incremental-static-regeneration (дата обращения: 06.03.2025).

[2] Peabody, Brad Server-side I/O Performance: Node vs. PHP vs. Java vs. Go / Brad Peabody. — Текст: электронный // Toptal: [Электронный ресурс]. — URL: https://www.toptal.com/back-end/server-side-io-performance-node-php-java-go (дата обращения: 06.03.2025).

[3] Vailshery L.S. Most used web frameworks among developers worldwide, as of 2024. — Текст: электронный // Statista: [Электронный ресурс]. — URL: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/ (дата обращения: 06.03.2025).

[4] Node Package Manager. — Текст: электронный // npm: [Электронный ресурс]. — URL: https://www.npmjs.com/ (дата обращения: 06.03.2025).

[5] Automate your workflow from idea to production. — Текст: электронный // GitHub Actions: [Электронный ресурс]. — URL: https://github.com/features/actions (дата обращения: 06.03.2025).

[6] Шелепугин И. М. Lunatic Backend-фреймворк для Node.js — Текст: электронный // Lunatic: [Электронный ресурс]. — URL: https://lunatic.shelepugin.ru/ru/ (дата обращения: 06.03.2025