Эффективность фронтенд-разработки на основе анализа сборщиков

Б.С. Горячкин, В.М. Хижняков, И.Г. Стрихар, И.Г. Бондаренко

Московский государственный технический университет им. Н.Э. Баумана

Аннотация: Современные веб-приложения становятся всё более сложными и функционально насыщенными, что создаёт необходимость в эффективных инструментах для управления зависимостями, оптимизации и сборки проектов. Сборщики позволяют оптимизировать код, что напрямую влияет на скорость загрузки и выполнения приложений. Цель работы - провести сравнительный анализ сборщиков JavaScript: Webpack, Parcel и Rollup, чтобы выявить их преимущества и недостатки с точки зрения эргономики фронтенд-разработки. Это включает в себя оценку удобства настройки, эффективности работы с ресурсами, скорости сборки и других факторов, влияющих на производительность разработчика и конечное качество веб-приложений. Проведено практическое тестирование сборщиков на примере стандартного веб-проекта. Оценена эргономика работы с инструментами: выделены критерии и проведено сравнение на основе полученных данных. Разработаны рекомендации по выбору оптимального инструмента для различных типов проектов в фронтенд-разработке. Результаты исследования могут быть использованы как основа для обучения новых специалистов, а также для улучшения существующих практик в разработке веб-приложений при принятии обоснованных решений по выбору технологий для долгосрочных проектов.
Ключевые слова: веб-разработка, эффективность разработки, эргономика, фронтенд-разработка, тестирование, сборщик.

Введение

Сборка веб-приложения включает подготовку исходного кода и зависимостей для развертывания на сервере или в браузере, включая компиляцию, минификацию и оптимизацию ресурсов. Сборщик веб-приложения автоматизирует этот процесс, преобразуя код в оптимизированные файлы, что упрощает управление зависимостями и улучшает производительность.

Современные веб-приложения состоят из множества файлов и библиотек, и без сборщика управление ими становится сложным и трудоемким. Ранее разработчики вручную подключали все файлы в HTML, что увеличивало количество HTTP-запросов и замедляло загрузку страниц [1]. Сборщики делают процесс разработки более организованным, позволяя сосредоточиться на написании кода и автоматизируя пересборку при

изменениях. В результате они стали ключевыми инструментами в веб-разработке, способствуя созданию более производительных приложений с меньшими затратами времени и усилий [2].

Основные инструменты для сборки JavaScript

Webpack, Parcel и Rollup — основные инструменты для сборки JavaScript-проектов, каждый из которых оказал влияние на фронтенд-разработку.

Webpack, представленный в 2012 году, стал популярным благодаря гибкости и поддержке модульного подхода, а также внедрению загрузчиков (лоадеров) и плагинов для обработки файлов [3, 4].

Parcel, появившийся в 2017 году, привлек внимание "нулевой конфигурацией", что упрощает старт проектов. Он автоматически обрабатывает зависимости и использует многопроцессорную обработку для быстрой сборки [5].

Rollup, созданный в 2015 году, сосредоточен на сборке ES6-модулей и стал популярным среди разработчиков библиотек благодаря методу встряхивания дерева (tree shaking), позволяющему удалять неиспользуемый код и создавать легкие пакеты [6, 7].

Webpack, Parcel и Rollup — это сборщики с разными особенностями. Webpack требует обширной конфигурации для сложных проектов, что приводит к большим файлам конфигурации. Rollup имеет компактные конфигурации, особенно для библиотек, в отличие от Webpack, так как не требует настройки лоадеров. Parcel предлагает "ноль конфигурации", позволяя быстро начать работу без сложных настроек, что делает его удобным для новичков.

Ассеты – это любые не JavaScript-файлы, которые используются в веб-приложении. Таковыми являются стили, картинки, файлы с данными и так

далее. Webpack поддерживает обработку различных типов ассетов с использованием лоадеров, которые трансформируют эти ассеты в модули. Rollup также нуждается в дополнительных плагинах и конфигурации. Parcel автоматически обрабатывает ассеты без дополнительной настройки, включая изображения, стили.

Tree shaking — это оптимизация для удаления неиспользуемого кода из финального пакета (бандла), что уменьшает размер файла и улучшает производительность приложения.

Rollup эффективно выполняет tree shaking, удаляя неиспользуемый код, в то время как Webpack также справляется с этой задачей, но требует тонкой настройки и имеет ограничения по разделению CJS (Common Java Script) модулей. Parcel поддерживает tree shaking автоматически, но уступает Rollup.

Rollup ориентирован на ES6 модули и предлагает отличную поддержку современного стандарта. Webpack и Parcel также его поддерживают, но Webpack требует конфигурации babel для качественной транспиляции.

Разделение кода (code splitting) — это техника оптимизации, используемая в сборщиках веб-приложений, которая позволяет разделять код на более мелкие части или "чанки", улучшая производительность, загружая только необходимые части [8]. Webpack предлагает мощные возможности для этой техники, в то время как Rollup и Parcel имеют более ограниченные настройки.

Технология HMR (Hot Module Replacement) в Webpack ускоряет разработку, позволяя обновлять измененные модули без полной перезагрузки страницы, что позволяет разработчикам мгновенно видеть результаты изменений. Parcel и Webpack поддерживают HMR из коробки, упрощая разработку, в то время как Rollup требует плагинов для этой функции.

Оптимизация веб-приложений, включая минификацию, важна для повышения производительности.

Минификация удаляет ненужные символы из кода, уменьшая размер файлов и ускоряя загрузку. Webpack предлагает множество плагинов для оптимизации, но требует настройки, тогда как Parcel автоматически применяет оптимизации.

TypeScript, добавляющий статическую типизацию к JavaScript, улучшает разработку и надежность кода, позволяя выявлять ошибки на этапе компиляции. Сборщики веб-приложений компилируют TypeScript в JavaScript, что обеспечивает совместимость с браузерами. Webpack и Rollup требуют настройки для использования TypeScript, в то время как Parcel делает это автоматически.

Сервер разработки (DevServer, DEV-сервер) — это инструмент для разработки, который автоматически запускается на локальной машине и позволяет тестировать веб-приложение в реальном времени. Его основная функция — быстрое отображение изменений в коде без ручной пересборки проекта. DEV-сервер следит за файлами и при изменениях автоматически обновляет страницу в браузере, позволяя разработчикам мгновенно видеть результаты. Webpack и Parcel имеют встроенные DEV-серверы, тогда как Rollup требует дополнительных плагинов для этой функции.

Устройство сборщиков

Для принятия взвешенного решения при выборе инструмента сборки параметров стоит углубиться в устройство сравниваемых инструментов.

Сборщики преобразуют исходный код (JavaScript, TypeScript, JSX и другие современные синтаксисы) в обычный JavaScript для браузеров. Они также управляют внешними библиотеками, стилями (CSS, SASS, LESS), изображениями и другими ассетами для веб-приложения (рис.1).

Важно понимать, что сборщик – это тоже код, который может работать быстро или медленно, не только в зависимости от того как хорошо он написан, но и от того на каком языке он написан и как он запускается. Изначально все инструменты для работы с JavaScript писались также на JavaScript, и самые популярные сборщики, такие как Webpack и Rollup, написаны на JavaScript, а значит они запускаются с помощью Node.js. Использование этой платформы накладывает ограничения в производительности – Node.js однопоточен, использует сборщик мусора и динамический импорт исполняемых модулей, каждый из которых должен быть проинтерпретирован виртуальной машиной.

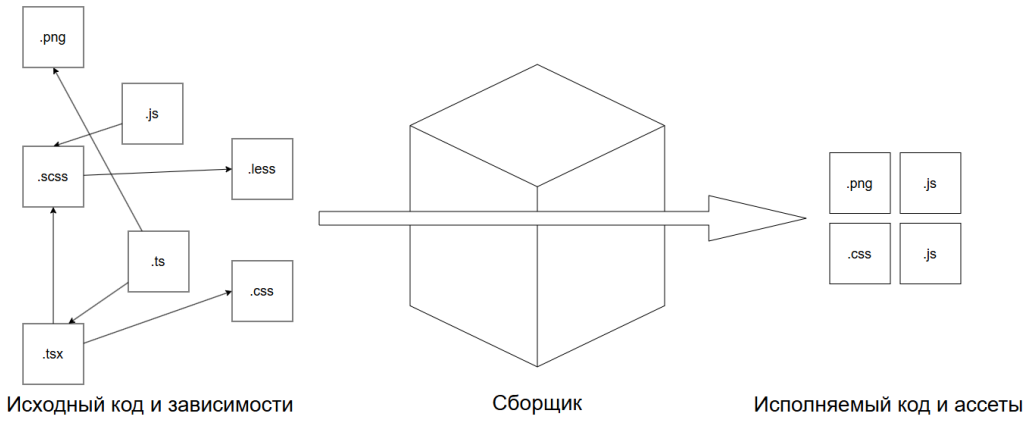

Рис. 1. – Схема работы сборщика

В последние несколько лет набирают популярность инструменты, написанные на Rust – это компилируемый язык, который по производительности сопоставим с C (рис.2).

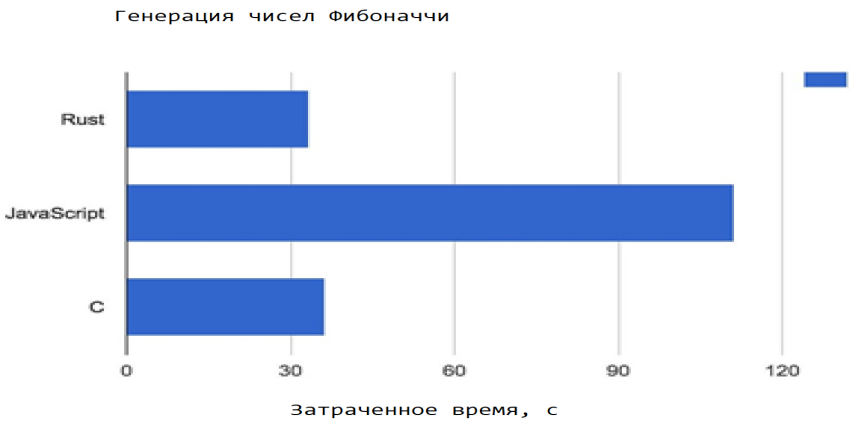

Рис. 2. – Сравнение производительности Node.js, Rust и C

Разработчики медленно переходят на новые инструменты, такие как Parcel, из-за надежности традиционных решений, которые за годы обросли множеством расширений и зарекомендовали себя. Новые инструменты, как правило, имеют недостатки, например, неоптимальный анализ AST-дерева (abstract syntax tree, абстрактное синтаксическое дерево) при tree shaking.

Сборщики также решают задачу рекурсивного опроса (резолвинга) импортов, анализируя зависимости между модулями и корректно связывая их. По сути – это базовая задача сборщика. Стоит пояснить, что JavaScript долгое время был скриптовым языком для браузеров, лишенным стандарта системы модулей. Такое положение толкало людей на создание своих систем, многие из которых стали весьма популярны и остаются таковыми по сей день. Однако после появления EcmaScript такое положение дел создало огромную проблему – существующие системы модулей несовместимы друг с другом, а написанного ранее кода крайне много. Наиболее популярными являются системы CommonJS и EsModule и они же являются принципиально несовместимыми [9]. Различия между CommonJS и EsModule приведены в таблице 1.

Таблица №1.

Различия между CommonJS и EsModule

Признак	**CommonJS**	**EsModule**
Синхронность	Синхронный	Асинхронный
Основное использование	Node.js	Браузеры и Node.js
Синтаксис	require/module.exports	import/export
Загрузка модулей	Во время выполнения	Статический анализ
Живые привязки	Нет	Да
Динамическая загрузка	Ограничено	Да (import())
Кэширование модулей	Да	Да

Сборщики помогают интегрировать эти системы, используя промежуточный слой хранения модулей, что влияет на финальный размер сборки. Webpack более универсален, но и тяжелее, чем Rollup и Parcel.

Tree shaking также выполняется сборщиком. Для этого проводится анализ AST-дерева (рис.3). Качество анализа зависит от алгоритма: Rollup и Webpack используют статический анализ, позволяющий точно определить неиспользуемые участки кода, но применим только к EsModule. Они также интерпретируют CommonJS, чтобы исключить лишний код. Parcel же работает только с EsModule, что может увеличить размер бандла.

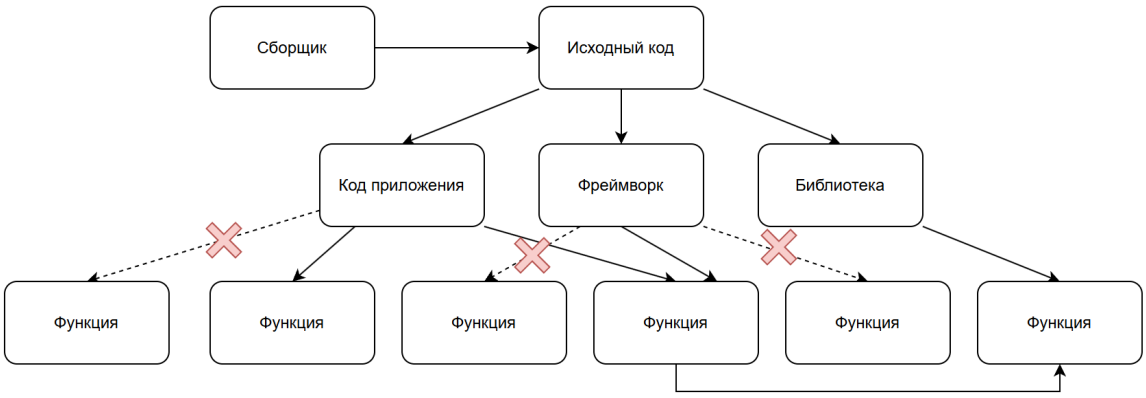

Рис. 3. – Схема работы TreeShaking

Сборщики обеспечивают транспиляцию языка в более старый стандарт для совместимости с устаревшими браузерами [10]. Это важно, поскольку разработчики могут использовать возможности нового стандарта языка, в то время как среда поддерживает только старый. Для транспиляции в старые версии ECMAScript используется Babel (рис.4).

Babel — это инструмент, интегрированный в процесс сборки через конфигурацию сборщика. Для достижения нужного результата разработчик также использует другие внешние пакеты, такие как лоадеры и плагины. Например, mini-css-extract-plugin необходим для создания отдельных файлов стилей при сборке через Webpack. Эти инструменты могут иметь различную производительность, и не все из них официально поддерживаются разработчиками сборщика. Webpack и Rollup имеют множество поддерживаемых плагинов, но Webpack обладает более широким сообществом. Parcel, благодаря концепции нулевой конфигурации и исходникам на Rust, изначально использует только официально поддерживаемые пакеты, что обеспечивает ему высокую скорость работы.

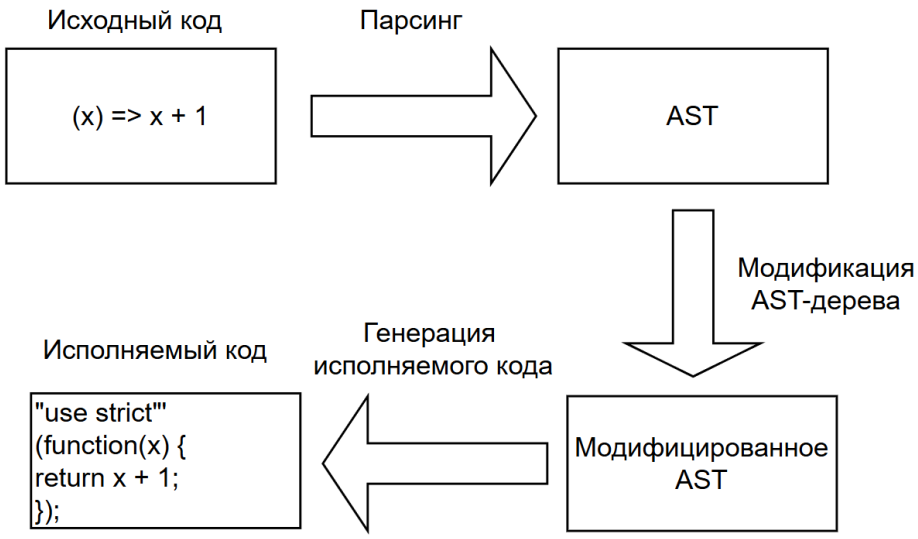

Рис. 4. – Схема работы Babel

Важной задачей сборщика также является минификация — удаление ненужных символов и сокращение имен переменных для уменьшения размера файла без изменения функциональности (рис.5).

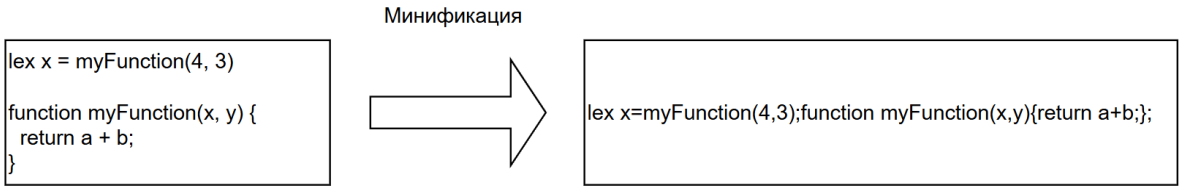

Рис. 5. – Пример минификации кода

Все рассматриваемые сборщики используют минификатор terser, что не влияет на скорость работы, хотя есть альтернативы с разной производительностью.

Также некоторые сборщики предоставляют Hot Module Replacement (рис.6) и DEV-сервер для упрощения локальной разработки. Скорость работы зависит от реализации сервера: Webpack имеет официальный webpack-DEV-server, Parcel — встроенный сервер, а Rollup использует сторонние решения, которые могут быть менее оптимизированными.

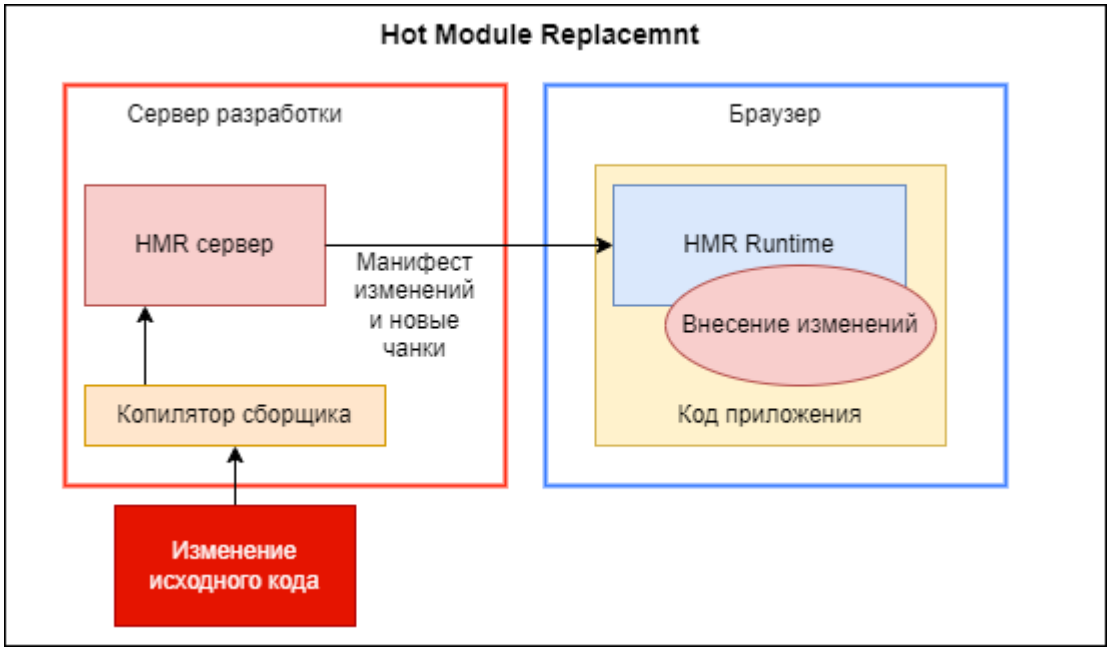

Рис. 6. – Схема работы Hot Module Replacement

Эргономические критерии для оценки сборщиков

В процессе выбора инструмента сборки особое внимание уделяется эргономическим параметрам, которые напрямую влияют на опыт и производительность разработчика. Эргономические признаки помогают оценить удобство и эффективность использования инструментов сборки, что является ключевым фактором для обеспечения высокой скорости и качества разработки.

Размер конфигурационного файла является первым значимым эргономическим параметром. Меньший размер файла указывает на более простую и понятную конфигурацию, что сокращает время настройки и облегчает внесение изменений. Это особенно важно для новых разработчиков в проекте, которым требуется быстро понять и начать работу с существующей конфигурацией.

$$W_{\text{конфиг}} = C_{\text{конфиг}} - N_{\text{пустые}}, \quad (1)$$

где $W_{\text{конфиг}}$ – размер конфигурационного файла (шт);

$C_{\text{конфиг}}$ - количество строк в конфигурационном файле (шт);

$N_{\text{пустые}}$ - количество пустых строк в конфигурационном файле (шт).

Количество дополнительных плагинов, необходимых для сборки, также играет важную роль. Чем меньше требуется дополнительных плагинов, тем проще управление зависимостями и меньше вероятность возникновения конфликтов между ними. Это напрямую влияет на стабильность и надежность процесса сборки, а также упрощает поддержку проекта в долгосрочной перспективе.

$$V_{\text{доп.плагинов}} = C_{\text{сборочные пакеты}} - 1, \quad (2)$$

где $V_{\text{доп.плагинов}}$ - количество дополнительных плагинов (шт);

$C_{\text{сборочные пакеты}}$ - общее количество пакетов для сборки (шт).

В формуле вычитается пакет самого сборщика, поскольку он присутствует всегда.

Скорость финальной (продакшн) сборки проекта критична для эффективности рабочего процесса. Быстрая сборка позволяет разработчикам чаще выпускать обновления и быстрее реагировать на изменения в проекте, что способствует более гибкой и адаптивной разработке.

$$V_{\text{сборки}} = \frac{W_{\text{сборки}}}{T_{\text{сборки}}}, \quad (3)$$

где $V_{\text{сборки}}$ - скорость сборки (Кб/с);

$T_{\text{сборки}}$ - время сборки (с);

$N_{\text{сборки}}$ - размер сборки (Кб).

Скорость запуска DEV-сервера напрямую влияет на начало рабочего процесса разработчика. Важно, чтобы DEV-сервер запускался быстро, позволяя разработчикам без задержек приступать к кодированию и тестированию изменений в реальном времени.

$$T_{DEV-\text{сервера}} = T_{\text{готовность } DEV-\text{сервера}} - T_{\text{запуск команды}}, \quad (4)$$

где $T_{DEV\text{-сервера}}$ - время запуска сервера (мс);

$T_{\text{готовность } DEV\text{-сервера}}$ - момент готовности сервера (мс);

$T_{\text{запуск команды}}$ - момент запуска команды (мс).

Скорость пересборки или HMR является ключевым фактором для повышения продуктивности разработки. Эта функция позволяет мгновенно видеть изменения в коде без необходимости полной перезагрузки страницы, что существенно ускоряет процесс разработки и отладки.

$$V_{\text{пересборки}} = \frac{N_{\text{пересобираемые}}}{T_{\text{пересборки}}}, \quad (5)$$

где $V_{\text{пересборки}}$ - скорость пересборки (модуль/мс);

$T_{\text{пересборки}}$ - время пересборки (мс);

$N_{\text{пересобираемые}}$ - количество пересобираемых модулей (шт).

Финальный размер сборки важен для производительности веб-приложения. Меньший размер бандла ускоряет загрузку приложения для конечного пользователя, особенно в условиях ограниченной пропускной способности сети. Оптимизация размера сборки может включать удаление неиспользуемого кода, минификацию и сжатие ресурсов, что напрямую влияет на удовлетворенность пользователей и общую производительность приложения.

$$W_{\text{сборки}} = \sum W_{js\ \text{чанков}} + \sum W_{css\ \text{чанков}} + \sum W_{\text{ассетов}}, \quad (6)$$

где $W_{\text{сборки}}$ - размер сборки (Кб);

$W_{js\ \text{чанков}}$ - размер *js* чанков (Кб);

$W_{\text{css чанков}}$ - размер *css* чанков (Кб);

$W_{\text{ассетов}}$ - размер ассетов (Кб).

Экспериментальные исследования и оценка полученных результатов

Каждый стенд был настроен с использованием одинакового исходного проекта, чтобы обеспечить сопоставимость результатов. Проект включал в себя стандартные компоненты, такие как JavaScript-файлы, стили и изображения, что позволяло максимально приблизить условия тестирования к реальным сценариям разработки.

Проект, который обрабатывается сборщиками – небольшое приложение для трекинга дел. Исходный код приложения написан с использованием JavaScript и Typescript, для написания стилей используется SCSS. Также проект требует работы в не самых новых браузерах. Для продакшн сборки требуется поддержка минификации бандла, а для разработки создание исходных карт.

Для получения достоверных данных каждый из сборщиков был многократно протестирован. Каждый сборщик запускался по 10 раз в трех

режимах – продакшн сборка, отслеживание изменений и разработка с локальным сервером. Запуск производился на одной машине, чтобы нивелировать различия в производительности устройств пользователей. Использование средних значений всех замеров позволило минимизировать влияние случайных факторов и получить более точные данные о параметрах, которые связаны с быстродействием сборщиков.

Технические характеристики машины, на которой проводились замеры:

- Наименование – Apple MacBook Pro, M1, 2020.
- Чип – Apple M1.
- ОЗУ – 16 Гб.
- ОС – MacOS 15.1.1.

Результаты измерений приведены на рис.7-10.

По результатам эксперимента видно, что по скорости работы, как в режиме разработки, так и в продакшн режиме, Parcel с большим отрывом обходит Webpack и Rollup. Однако по качеству минификации размера бандла традиционные инструменты опережают конкурента.

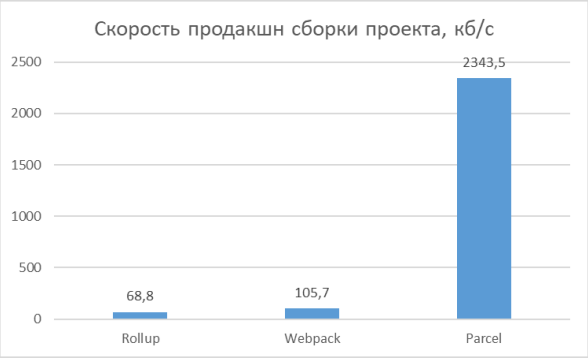

Рис. 7. – График замеров скорости продакшн сборки проекта

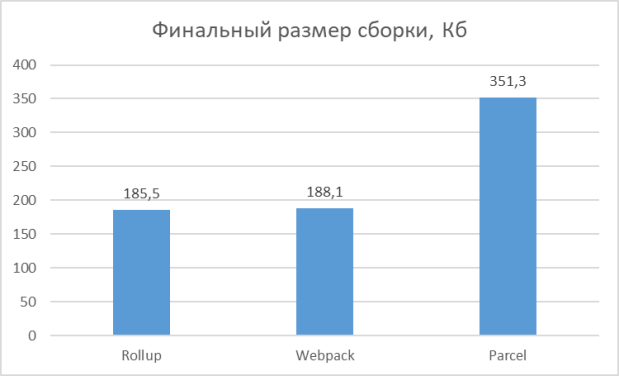

Рис. 8. – График замеров финального размера сборки

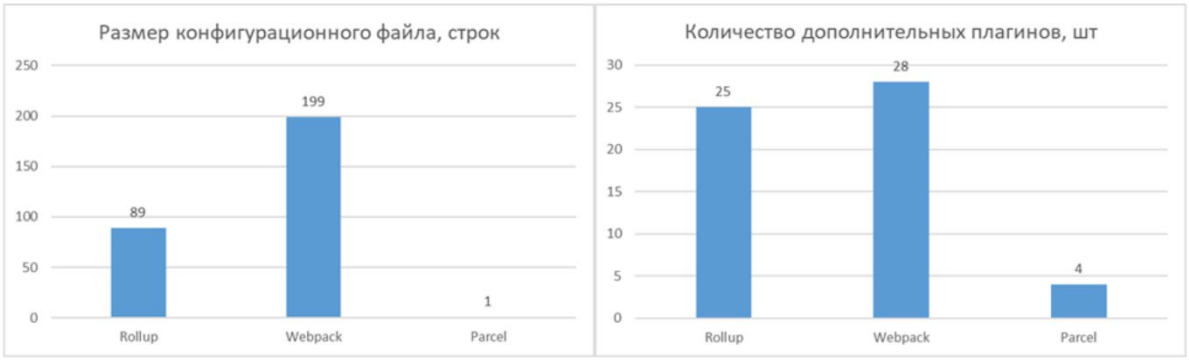

Рис. 9. – Графики размера конфигурационного файла и количества дополнительных плагинов

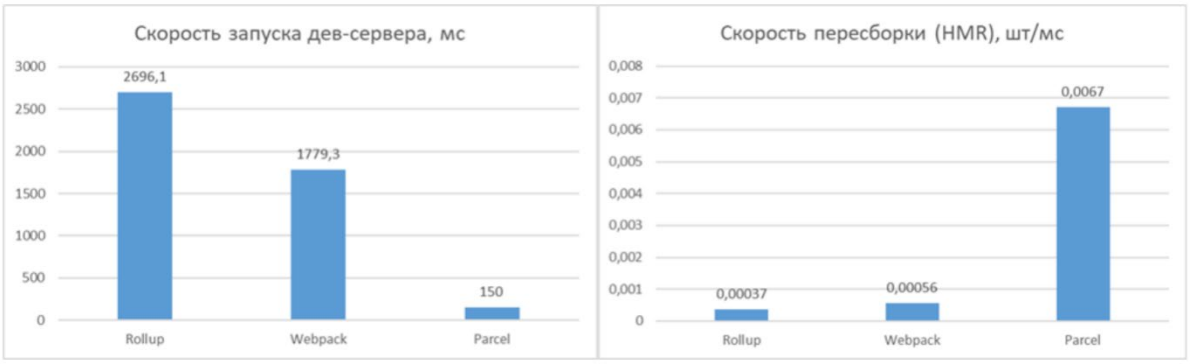

Рис. 10. – Графики замеров скорости пересборки и запуска дев-сервера

Заключение

Parcel выделяется своей простотой и интуитивно понятным подходом к конфигурации. Сборщик не требует сложных настроек и предоставляет возможность начать разработку практически мгновенно. Автоматическое управление зависимостями и встроенные функции, такие как поддержка HMR, делают его особенно привлекательным для разработчиков, стремящихся к быстрой и эффективной разработке. Благодаря этому, Parcel становится идеальным выбором для небольших проектов и стартапов, где скорость разработки и простота использования играют ключевую роль.

Тем не менее, нельзя недооценивать Webpack, который, хотя и требует более сложной конфигурации, предлагает выдающуюся гибкость и мощные возможности для настройки. Webpack позволяет создавать сложные и масштабируемые приложения, что делает его предпочтительным выбором для крупных проектов и команд, работающих с высоконагруженными системами. Его экосистема плагинов и расширений предоставляет разработчикам возможность адаптировать сборку под специфические требования проекта. Таким образом, несмотря на некоторые сложности в настройке, Webpack остается одним из наиболее популярных сборщиков благодаря своей функциональности и поддержке сообщества.

Rollup, в свою очередь, также имеет свои преимущества, особенно в контексте создания библиотек и компонентов. Его фокус на модульности и оптимизации выходного кода делает его отличным выбором для разработчиков, стремящихся к созданию легковесных и высокопроизводительных решений. Однако в сравнении с Parcel и Webpack, он может уступать в плане удобства работы над большими проектами с множеством зависимостей.

Литература

1. Горячкин Б. С., Ханмурзин Т. И. Повышение эффективности работы с веб-ресурсом за счет инструментария системного программиста // Динамика сложных систем - XXI век. 2022. Т. 16. № 3. С. 26-39.

2. Горячкин Б.С. Эргономический анализ систем обработки информации и управления // Вестник евразийской науки. 2017. Т. 9. №3. URL: naukovedenie.ru/PDF/79TVN317.pdf.

3. Clow M., Clow M. Introducing webpack //Angular 5 Projects: Learn to Build Single Page Web Applications Using 70+ Projects. 2018. pp. 133-137.

4. Попков И. В., Курзаева Л. В. Использование Webpack для сборки проекта // Международный студенческий научный вестник. 2018. №5. С. 164-164.

5. Parcel Documentation. URL: parceljs.org/docs

6. Laurila S. Comparison of javascript bundlers URL: theseus.fi/bitstream/handle/10024/345959/Laurila_Sonja.pdf?sequence=2/.

7. Мончeнко А. С. Анализ размеров бандлов, получаемых с помощью webpack и других сборщиков //Научно-технические инновации и веб-технологии. №1. С. 54-59.

8. Ekpobimi H. O., Kandekere R. C., Fasanmade A. A. Conceptual framework for enhancing front-end web performance: Strategies and best practices // Global Journal of Advanced Research and Reviews. 2024. V. 2. №1. pp. 099-107.

9. Агапов И. Понимание всех модульных форматов и инструментов JavaScript, 2020 URL: habr.com/ru/articles/501198/.

10. Satrom B. Building Polyfills: web platform APIs for the present and future. O'Reilly Media, Inc., 2014. 170 p.

References

1. Goryachkin B. S., Khanmurzin T. I. Dinamika slozhnykh sistem - XXI vek. 2022. V. 16. № 3. pp. 26-39.
2. Goryachkin B.S. Vestnik evraziyskoy nauki. 2017. V. 9. №3. URL: naukovedenie.ru/PDF/79TVN317.pdf.
3. Clow M., Clow M. Angular 5 Projects: Learn to Build Single Page Web Applications Using 70+ Projects. 2018. pp. 133-137.
4. Popkov I. V., Kurzaeva L. V. Mezhdunarodnyy studencheskiy nauchnyy vestnik. 2018. №5. pp. 164-164.
5. Laurila S. Comparison of javascript bundlers URL: theseus.fi/bitstream/handle/10024/345959/Laurila_Sonja.pdf?sequence=2/.

6. Monchenko A. S. Nauchno-tekhnicheskie innovatsii i veb-tekhnologii. №1. pp. 54-59.
7. Parcel Documentation. URL: parceljs.org/docs.
8. Satrom B. Building Polyfills: web platform APIs for the present and future. O'Reilly Media, Inc., 2014. 170 p.
9. Goryachkin B.S. Vestnik evraziyskoy nauki. 2017. V. 9. №3. URL: naukovedenie.ru/PDF/79TVN317.pdf.
10. Agapov I. Ponimanie vsekh modul'nykh formatov i instrumentov JavaScript, 2020 URL: habr.com/ru/articles/501198/.
11. Ekpobimi H. O., Kandekere R. C., Fasanmade A. A. Global Journal of Advanced Research and Reviews. 2024. V. 2. №1. pp. 099-107.

Дата поступления: 3.03.2025
Дата публикации: 25.04 2025