УДК 004.051

Способы оптимизации загрузки данных из разнотипных источников в рамках реализации frontend-части web-приложения “BioSense”

Колупаев А.А.
Алтайский государственный университет, г. Барнаул
alexanderkolypaev84@gmail.com

Аннотация

Статья посвящена способам оптимизации работы с данными в клиентской части web-приложения в рамках проекта BioSense. Рассмотрены такие подходы, как использование новейшей системы сборки; уменьшение итогового размера исходного кода за счет использования более простых и легковесных библиотек; разбиение исходного кода на небольшие части; пагинация и виртуализация таблиц.

Ключевые слова: Оптимизация, frontend, React, web-приложение, JavaScript, большие данные.

В настоящее время ведётся разработка web-приложения “BioSense” для хранения, обработки и анализа данных, поступающих с климатических датчиков. Система предназначена для мониторинга различных параметров окружающей среды и их влияния на биологические объекты. В связи необходимостью обработки больших объемов данных на стороне клиента, перед разработчиками стояла задача оптимизации ресурсоемких операций с целью повышения комфорта работы пользователей в системе.

Подробно рассмотрим каждый из подходов к оптимизации работы с данными:

1. Переход с системы сборки Webpack на RSBuild

Любое web-приложение, созданное с использованием библиотеки React [1], в процессе разработки компилируется с помощью системы сборки, которая формирует набор файлов, передаваемых пользователю при загрузке страницы. На практике используются разные системы сборки, каждая из которых обладает своими особенностями.

На начальном этапе разработки проекта для этих целей был выбран Webpack, который на тот момент являлся наиболее популярным решением. Однако с развитием функционала проекта (рост объёма программного кода) значительно увеличилось время сборки и применения изменений, что негативно сказывалось на эффективности работы команды разработчиков. Для решения этой проблемы было принято решение о переходе на новую систему сборки – RSBuild.

RSBuild, написанная на языке Rust, обеспечивает высокую производительность и скорость сборки. В результате перехода на этот инструмент время выполнения сборки и применения изменений сократилось более чем в 10 раз (время полной сборки проекта на моем личном компьютере уменьшилось с 80 до 5 секунд), что позволило существенно повысить производительность команды и комфорт разработки.

2. Использование Lazy-загрузки модулей приложения

При загрузке главной страницы приложения пользователь, не обладающий доступом к административной панели, не нуждается в получении её исходного кода. Загрузка лишних модулей, не требующихся для текущей сессии, увеличивает время начальной загрузки и неоправданно расходует ресурсы.

Эту проблему эффективно решает механизм ленивой загрузки (Lazy Loading) модулей. Приложение разделяется на логические блоки, такие как отдельные страницы или функциональные модули, которые загружаются только в момент их непосредственного использования. Такой подход позволяет значительно уменьшить объём данных, передаваемых при первичной загрузке, что способствует ускорению старта приложения и улучшению пользовательского опыта.

```
const Login : LazyExoticComponent<function(): any> = lazy( load: () : Promise<{...}> => import("./pages/login/Login.jsx"))
const Map : LazyExoticComponent<function(): any> = lazy( load: () : Promise<{...}> => import("./pages/map/Map.jsx"))
const DataIn : LazyExoticComponent<function(): any> = lazy( load: () : Promise<{...}> => import("./pages/dataIn/DataIn.jsx"))
const Main : LazyExoticComponent<function(): any> = lazy( load: () : Promise<{...}> => import("./pages/main/Main.jsx"))
const DataOut : LazyExoticComponent<function(): any> = lazy( load: () : Promise<{...}> => import("./pages/dataOut/DataOut.jsx"))
const FileInput : LazyExoticComponent<function(): any> = lazy( load: () : Promise<{...}> => import("./pages/fileInput/FileInput.jsx"))
const AdminPanel : LazyExoticComponent<function(): any> = lazy( load: () : Promise<{...}> => import("./pages/adminPanel/AdminPanel.jsx"))
const UserProfile : LazyExoticComponent<function(): any> = lazy( load: () : Promise<{...}> => import('./pages/users/usersProfile/UsersProfile.jsx'))
const NotFound : LazyExoticComponent<function(): any> = lazy( load: () : Promise<{...}> => import('./pages/notFound/NotFound.jsx'))
const LostConnection : LazyExoticComponent<function(): any> = lazy( load: () : Promise<{...}> => import('./pages/lostConnection/LostConnection.jsx'))
```

Рисунок 1. Реализация Lazy-загрузки страниц

3. Использование более легковесных библиотек или отказ от них

Аудит используемых в проекте библиотек является сложным и трудоёмким процессом, требующим внимательного анализа. Применение инструментов для анализа скомпилированного проекта значительно упрощает эту задачу за счёт визуализации структуры бандла. Одним из таких инструментов является **webpack-bundle-analyzer** [2].

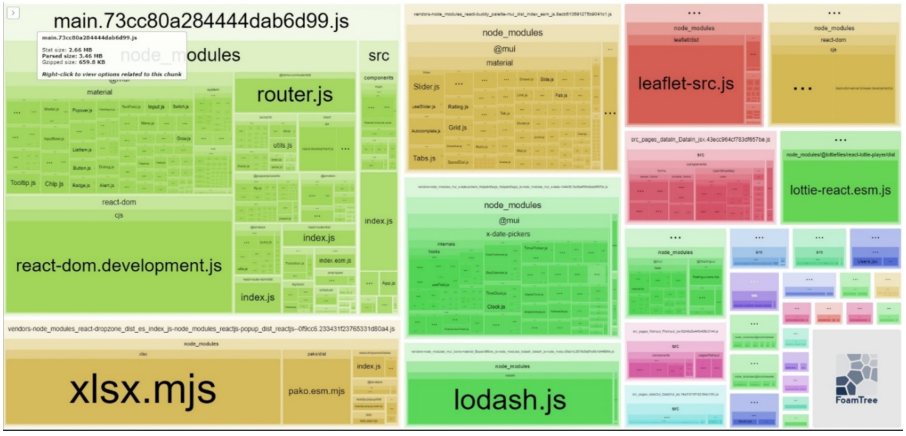

Рисунок 2. Распределение размера компилированного приложения по модулям

На примере, представленном на рисунке 2, видно, что модуль библиотеки **xlsx** занимает около 1,5 МБ, что существенно увеличивает объём данных, загружаемых пользователем. При этом данная библиотека использовалась единственный раз – для формирования шаблона заполнения данных в формате **xlsx**.

Использование этой библиотеки можно оптимизировать, формируя шаблон в формате **csv** с помощью стандартных методов JavaScript. Такой подход позволяет исключить из бандла модуль **xlsx**, сократив объём загружаемых данных на 1,5 МБ, что особенно критично для ускорения загрузки страниц.

Подобным образом можно заменить большие и тяжелые библиотеки на более узконаправленные и легковесные, если их функционала достаточно для решения поставленных задач.

4. Пагинация и виртуализация таблиц

Так как приложение предназначено для работы с большими объемами данных, часто представляемых в виде таблиц, содержащими большие объёмы данных, то оптимизация работы с ними является ключевой задачей для обеспечения производительности и удобства использования приложения. Загрузка всех строк таблиц сразу (в таблицах могут

быть десятки тысяч строк) может привести к значительному увеличению времени загрузки страницы, перегрузке памяти браузера и снижению отзывчивости интерфейса.

Для решения этой проблемы в проекте используется комбинация методов пагинации и виртуализации.

Пагинация – это разбиение таблицы на страницы с фиксированным числом строк на каждой из них, что позволяет загружать данные порциями, ограниченными текущей страницей и видимым пользователю диапазоном. Это сокращает количество передаваемых данных и уменьшает нагрузку на сервер и клиентское приложение. Например, при отображении таблицы данные запрашиваются небольшими порциями, соответствующими текущей странице, что предотвращает избыточную передачу информации.

Виртуализация – метод оптимизации отображения больших таблиц на странице, который заключается в том, чтобы рендерить только те строки таблицы, которые пользователь видит в конкретный момент времени, что позволяет значительно уменьшить количество элементов в виртуальном DOM-дереве. Независимо от общего числа записей, в DOM-дерево рендерится лишь ограниченный набор элементов. Это позволяет существенно снизить потребление памяти и ускорить работу интерфейса при работе с большими таблицами. Данных подход был реализован средствами библиотеки **TanStack Virtual** [3].

Совместное использование этих методов обеспечивает быструю загрузку и обработку таблиц, снижает нагрузку на клиентскую и серверную части, а также улучшает пользовательский опыт.

Заключение

Оптимизация загрузки данных в web-приложении – это комплексный процесс, включающий выбор эффективных инструментов, пересмотр используемых подходов и поиск точек для улучшения производительности. В рамках разработки проекта “BioSense” реализация таких решений, как переход на более быстрый сборщик RSBuild, использование ленивой загрузки модулей, оптимизация объёма бандла, а также внедрение пагинации и виртуализации таблиц, позволили не только улучшить производительность приложения, но и создать комфортные условия для разработчиков и пользователей.

Эти меры являются универсальными инструментами для повышения эффективности современных web-приложений и могут быть адаптированы под задачи любого проекта, где критически важны скорость загрузки, отзывчивость интерфейса и удобство взаимодействия.

Список литературы

1. React [Электронный ресурс]. — URL: https://react.dev/learn. Дата обращения 22.11.2024.

2. Webpack Bundle Analyzer [Электронный ресурс]. — URL: https://www.npmjs.com/package/webpack-bundle-analyzer. Дата обращения 22.11.2024.

3. TanStack Virtual [Электронный ресурс]. — URL: https://tanstack.com/virtual/latest. Дата обращения 22.11.2024.