

Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 22.Б15-мм

Рефакторинг экранов результата задачи в Desbordante

ЕРМОЛОВИЧ Анна Сергеевна

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
Ассистент кафедры ИАС, Г.А. Чернышев

Санкт-Петербург
2023

Оглавление

Введение	3
1. Постановка задачи	4
2. Обзор	5
2.1. Существующие решения	5
2.2. Используемые термины и технологии	6
3. Реализация	8
3.1. Новая структура файлов	8
3.2. Переход на динамический раутинг	8
3.3. Переход на CSR	9
3.4. Результаты	9
Заключение	10
Список литературы	11

Введение

Профилирование данных — это процесс извлечения метаданных из данных. Очевидно, что в зависимости от выбранного критерия, в качестве свойств может выступать огромное множество различных данных, в связи с чем возникает проблема выявления именно тех и только тех свойств, которые интересны изучающему объект субъекту. В то же время огромный объем аккумулируемой в различных базах информации повлек необходимость создания алгоритмов, методов профилирования. Здесь следует, прежде всего, упомянуть открытую платформу Metanome на языке Java — совместный проект компаний Hasso Plattner Institute и Qatar Computing Research Institute, который был начат в 2015 году. Metanome включает в себя разнообразные алгоритмы профилирования, включая алгоритмы поиска функциональных зависимостей. В качестве ее более современной альтернативы возник Desbordante [1] — проект, который с лета 2019 года реализуется компанией Юнидата. По сравнению с предшественником, Desbordante работает быстрее и эффективнее по памяти в среднем в два раза [2], потому что написан на C++. Ключевая особенность профилирования данных Desbordante — его способность обнаруживать сложные математические закономерности — так называемые примитивы, под которыми понимается описание правила, действующего над данными или их частью, описанное математическими методами.

В зависимости от примитива результат может иметь разные представления. Но в текущей веб-версии для всех примитивов используются одинаковые компоненты, что заставляет прибегать к различным обходным решениям. Например, для одного из примитивов создается отдельная страница экрана результата, так как не удалось подстроить её под уже существующий шаблон. В связи с чем появилась необходимость рефакторинга архитектуры экранов просмотра результата задачи. Об этом и пойдет речь в настоящей работе.

1. Постановка задачи

Целью работы является рефакторинг экранов результата задачи в Desbordante. Для её выполнения были поставлены следующие задачи:

1. перейти на динамический раутинг;
2. перейти с SSR на CSR;
3. переработать текущую структуру файлов;

2. Обзор

2.1. Существующие решения

2.1.1. Текущая реализация

Для каждого примитива сейчас существует свой набор доступных вкладок. Однако, для вкладок с одинаковым `pathName` используется один и тот же компонент, даже если примитивы разные. Из-за этого приходится подстраивать вкладки всех существующих примитивов под один конкретный компонент. Затем, внутри этого компонента, менять поведение в зависимости от выбранного примитива. В случае с `Metric Dependency Verification` страница результата не подходила ни под один из существовавших шаблонов. Это привело к созданию отдельной страницы только для этого примитива.

Также, сейчас вкладки результата реализованы через SSR (`Server-Side Rendering`), но изначально результаты задумывались так, что пользователь будет часто между ними переключаться. Следовательно, хотелось бы осуществлять это переключение максимально быстро. В случае SSR страница генерируется два раза — на сервере и на клиенте. SSR приводит также к дополнительным сложностям при работе с авторизованным контентом: надо передавать токен авторизации в запрос на бэкенд внутри `getServerSideProps`, и там же обновлять его.

2.1.2. Вывод

Было решено объединить создание вкладок результата в один функциональный компонент, который будет действовать в зависимости от выбранного примитива. Для этого было решено перейти на динамический роутинг и изменить структуру файлов. Кроме того, следует перейти с SSR на CSR (`Client-Side Rendering`).

2.2. Используемые термины и технологии

2.2.1. SSR

SSR (Server-Side Rendering) — техника генерации HTML-контента на сервере перед его отправкой клиенту, позволяющая предоставить полностью отрисованный интерфейс при первой загрузке страницы.

2.2.2. CSR

CSR (Client-Side Rendering) — подход к рендерингу веб-страниц, при котором содержимое страницы генерируется клиентским JavaScript, обеспечивая динамическое взаимодействие без полной перезагрузки страницы.

2.2.3. Динамический раутинг

Динамический раутинг — механизм в веб-приложениях, позволяющий определять и обрабатывать маршруты в реальном времени, основываясь на параметрах URL, для генерации специфичного контента или страниц. [3]

2.2.4. React

React [5] — открытая JavaScript-библиотека для построения пользовательских интерфейсов. Он позволяет создавать эффективные и масштабируемые веб-приложения, используя компонентный подход. React.js предоставляет простой и декларативный способ описания взаимодействия пользовательского интерфейса с данными, что значительно упрощает разработку и поддержку приложений.

2.2.5. Next.js

Next.js [4] — фреймворк для серверного и статического рендеринга, построенный на React, который предлагает возможности построения универсальных приложений, оптимизацию маршрутизации и улучшенное SEO.

2.2.6. TypeScript

TypeScript [6] — это статически типизированный язык программирования, который расширяет синтаксис JavaScript. Он предоставляет возможность определения типов переменных, параметров функций и возвращаемых значений, что позволяет обнаруживать и предотвращать ошибки на этапе разработки.

3. Реализация

Реализация была поделена на несколько этапов:

3.1. Новая структура файлов

Предлагается создать новую структуру файлов, где каждый report будет иметь свой собственный подкаталог с именем, соответствующим taskID. Необходимо сохранять taskID в URL, потому что абсолютное большинство компонентов получает ID именно из URL.

Было решено разбить все примитивы на директории, в которых для каждого примитива хранятся компоненты, отвечающие за его вкладки. Все компоненты, отвечающие за вкладки примитива, импортировались как массив вкладок. Так, например, для примитива FD, у которого три вкладки: Statistics, Instance list, Dataset snippet, импортируется массив из компонентов, которые должны рисоваться при выборе той или иной вкладки FDtabs.

3.2. Переход на динамический раутинг

Для перехода на динамический раутинг следует создать новый файл с именем [tab].tsx внутри каталога [taskID]. В этом файле можно определить компонент, который будет отображать содержимое конкретной вкладки report. Компонент может использовать taskID из URL.

Для удобной организации и управления вкладками отчета был создан компонент reportsConfig. В этом компоненте прописано соответствие типа примитива массиву его вкладок. Такая структура позволяет легко добавлять и изменять вкладки для каждого типа примитива, а также обеспечивает централизованное хранение конфигурации вкладок. Каждая вкладка в reportsConfig имеет свои свойства, такие как путь, название, иконка и компонент, которые будут отображаться при выборе данной вкладки.

Используя reportsConfig, можно динамически определить выбранную вкладку на основе текущей вкладки из URL. Для этого можно

применить метод `find`, чтобы найти в массиве вкладок соответствующую вкладку на основе пути текущей вкладки. Если найденная вкладка существует, ее компонент отобразится на странице.

Создание файла `[tab].tsx` внутри каталога `[taskID]` и использование `reportsConfig` позволяют реализовать динамический роутинг и удобное управление вкладками результата. Это обеспечивает гибкость и расширяемость системы, позволяя легко добавлять новые типы примитивов и вкладок.

3.3. Переход на CSR

Для отключения SSR было решено использовать `dynamic import` из `Next.js`. Ранее в проекте уже использовался этот подход, поэтому он был выбран в качестве решения.

Для реализации `dynamic import` в проекте, было решено удалить все функции `getServerSideProps`. Ранее эти функции использовались для получения данных на сервере перед рендерингом страницы. Теперь функциональность, ранее определенная в `getServerSideProps`, была перенесена внутрь соответствующих компонентов, к которым эти функции относились.

Каждый компонент теперь отвечает за свои собственные данные и может использовать `dynamic import` для асинхронной загрузки необходимых компонентов только по мере необходимости.

3.4. Результаты

Все пять примитивов были переписаны на динамический роутинг и CSR. Предоставлена новая структура данных. Также было проведено ручное тестирование, в ходе которого не было выявлено проблем.

Результаты данной работы могут быть использованы для создания унифицированного хранилища данных в будущем.

Заключение

По итогам данной работы был произведен рефакторинг экранов результата задачи в Desbordante, а именно:

- совершен переход на динамический раутинг;
- совершен переход с SSR на CSR;
- переработана текущая структура файлов;

Ссылка на GitHub-репозиторий¹, ссылка на pull request².

¹<https://github.com/Anna-er/desbordante-web/tree/refactoring-work>

²<https://github.com/Anna-er/desbordante-web/pull/1>

Список литературы

- [1] Desbordante. — URL: <https://desbordante.unidata-platform.ru/> (дата обращения: 22 декабря 2023 г.).
- [2] Desbordante: a Framework for Exploring Limits of Dependency Discovery Algorithms. — URL: <https://ieeexplore.ieee.org/document/9435469/> (дата обращения: 22 декабря 2023 г.).
- [3] Dynamic Routes. — URL: <https://nextjs.org/docs/pages/building-your-application/routing/dynamic-routes> (дата обращения: 22 декабря 2023 г.).
- [4] Next.js by Vercel - The React Framework. — URL: <https://nextjs.org/docs> (дата обращения: 22 декабря 2023 г.).
- [5] React. — URL: <https://react.dev/> (дата обращения: 22 декабря 2023 г.).
- [6] TypeScript Documentation. — URL: <https://www.typescriptlang.org/docs/> (дата обращения: 22 декабря 2023 г.).