Санкт-Петербургский государственный университет

Группа 20Б.09-мм

БАРУТКИН Илья Дмитриевич

Визуализация результатов поиска примитивов в веб-приложении Desbordante

Отчёт по учебной практике

Научный руководитель:
ассистент кафедры ИАС Г.А. Чернышев

Санкт-Петербург
2022

Оглавление

Введение

Профилирование данных [1] — это процесс извлечения метаданных из данных. Метаданные — это, к примеру, размер файла, время создания, авторство. Но это также и любые закономерности, сокрытые в данных. Закономерности, которые могут присутствовать в данных, описываются с помощью различных примитивов. Примитив — это некоторое правило, действующее над данными (или их частью), описанное математическими методами.

Поскольку инструменты профилирования данных в основном нацелены на пользователей, эффективная визуализация результатов имеет первостепенное значение. Только тогда пользователи смогут интерпретировать результаты и реагировать на них [1].

В предыдущей работе [3] была описана разработка новой версии веб-приложения для платформы Desbordante, высокопроизводительного профайлера данных. В приложении уже присутствовала визуализация результатов поиска примитивов в виде списка зависимостей с возможностью фильтрации по атрибутам и сортировки. Однако для удобства анализа результатов в приложении требовался иной подход, который бы позволил оценивать значимость различных атрибутов в множестве зависимостей. В предыдущей версии веб-приложения уже была разработана страница, реализующая такой подход. Она визуализирует результаты поиска примитивов “функциональная зависимость” и “условная функциональная зависимость” в виде диаграмм. Требовалось разработать аналогичный функционал в новой версии веб-приложения, учитывая изменения в дизайне.

1. Постановка задачи

1.1. Цель работы

Целью работы является реализация страницы, визуализирующей результаты поиска примитивов “функциональная зависимость” и “условная функциональная зависимость” в виде диаграмм, в рамках разрабатываемой версии веб-приложения Desbordante. Для её выполнения были поставлены следующие задачи:

1. сверстать макеты приложения
2. реализовать отдельные компоненты, необходимые для страницы
3. реализовать взаимодействие с серверной частью приложения и отображение получаемых данных
4. проверить работоспособность приложения

1.2. Требования к приложению

Были поставлены следующие требования к странице, визуализирующей результаты поиска примитивов. Оно должно позволять пользователю

1. просматривать две диаграммы: с атрибутами левой стороны зависимостей и с атрибутами правой стороны зависимостей
2. фильтровать атрибуты по названию
3. просматривать диаграммы по слоям, ограничивая количество показываемых секторов диаграммы в одном слое
4. выбирать атрибуты по нажатию на сектор на диаграмме для последующей фильтрации страницы со списком зависимостей
5. исследовать результаты выполнения задач на поиск двух видов примитивов

- функциональные зависимости
- условные функциональные зависимости

2. Обзор существующих решений

2.1. Metanome

Metanome — совместный проект Hasso Plattner Institute и Qatar Computing Research Institute. Это открытая платформа для профилирования данных, реализованная на языке Java. Она поддерживает различные методы визуализации результатов, такие как диаграмма солнечных лучей и префиксное дерево атрибутов [6].

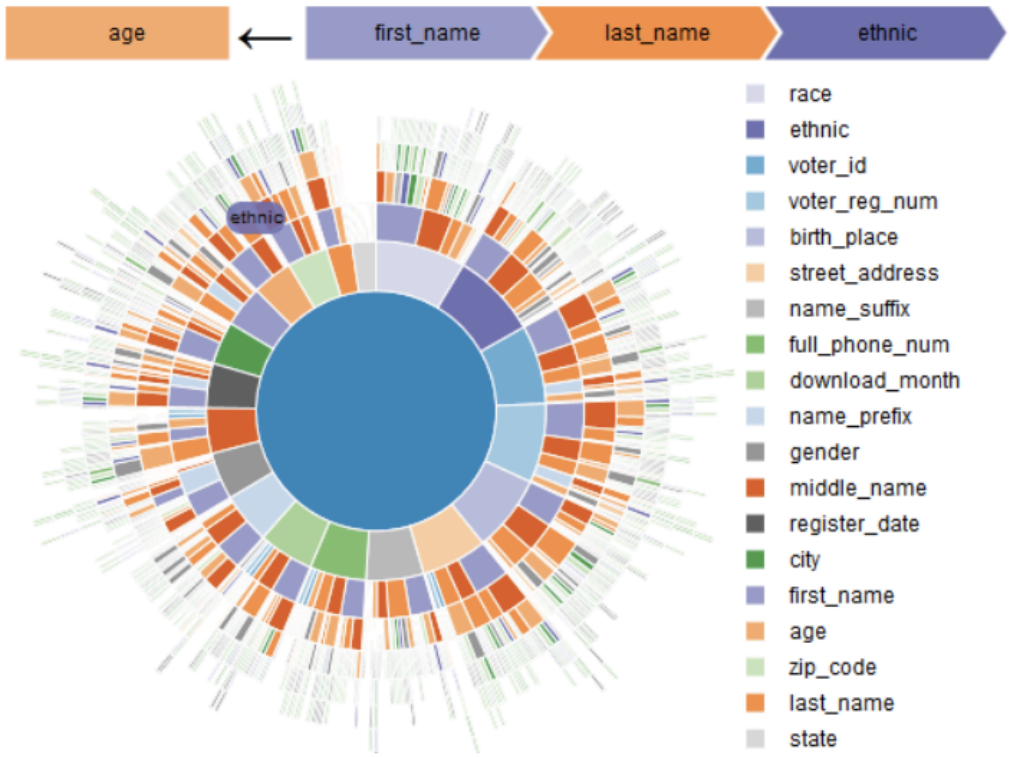

Рис. 1: Диаграмма солнечных лучей, реализованная в платформе Metanome

2.2. Desbordante

Desbordante реализовывалась как альтернатива платформе Metanome. По сравнению с ней Desbordante работает быстрее и требует меньше памяти, в среднем в два раза [5]. Веб-интерфейс инструмента позволяет визуализировать результаты поиска примитивов “функциональная за-

висимость” и “условная функциональная зависимость” в виде диаграмм. Каждый атрибут представляется в виде сектора размера, соответствующего общей ценности атрибута.

3. Реализация

3.1. Используемые технологии

Проект разрабатывается на языке TypeScript с помощью фреймворков React и Next.js. Также используются специальные библиотеки для взаимодействия с GraphQL API и линтер [3].

Для создания диаграмм в ранней версии проекта [4] был сделан выбор в пользу Chart.js — наиболее популярной javascript-библиотеки для создания графиков и диаграмм. Для отрисовки используется HTML-элемент canvas. Для того, чтобы интегрировать функционал данной библиотеки в React-приложение на языке TypeScript была дополнительно использована библиотека react-chartjs-2. Она позволяет использовать диаграммы как любые другие React-компоненты и имеет полностью типизированный интерфейс [2].

3.2. Формула веса атрибута

Для определения веса атрибута в приложении использовалась следующая формула.

$S(A) = \sum_f \frac{\sigma(f,A)}{|f|_l}$ где f — зависимость,

$$\sigma(f, A) = \begin{cases} 1 & \text{A присутствует в левой (правой) части } f \\ 0 & \text{иначе} \end{cases},$$

$|f|_l$ — длина левой (правой) части f

В ранней работе [4] был обоснован выбор этой формулы тем соображением, что чем длиннее у зависимости левая часть, тем меньше её практическая ценность.

3.3. Страница

Задача представляет собой написание новой страницы в виде React-компонента типа NextPage, которая будет доступна по URL-адресу /reports/charts. В разрабатываемом приложении используется специальный тип страниц NextPageWithLayout, предоставляющий возмож-

ность указать функцию разметки. В нашем случае эта функция оборачивает содержимое страницы в контекст TaskContext. Такой подход рекомендован в документации Next.js [7].

3.4. Контекст

При разработке страницы использовался контекст — TaskContext, содержащий информацию о текущей задаче и выбранные фильтры. Использование контекста помогло избежать дублирования кода и лишних запросов к graphQL API при переключении между страницами.

3.5. Жизненный цикл

В компоненте страницы используются два вызова функций-хуков: useTaskContext и useQuery

```
1 const { taskID, dependenciesFilter, setDependenciesFilter } =
2     useTaskContext();
3
4 const { loading, data, error } = useQuery<
5     getPieChartData,
6     getPieChartDataVariables
7     >(GET_PIE_CHART_DATA, { variables: { taskID } });
```

Листинг 1: Функции-хуки в компоненте ReportsCharts

Хук useTaskContext — это функция, которая получает состояние контекста TaskContext или показывает ошибку в случае отсутствия провайдера контекста в дереве предков текущего компонента. Хук useQuery — это функция из библиотеки Apollo, предназначенная для выполнения запроса к graphQL API. При первой отрисовке компонента объект data еще не определен, так как запрос не был совершен. В этом случае булев флаг loading равен true, и мы показываем сообщение о загрузке страницы.

3.6. Диаграммы

Для визуального представления значимости различных атрибутов в найденных зависимостях было решено использовать круговые диаграммы, которые доступны в библиотеке react-chartjs-2. Однако при большом количестве атрибутов диаграмма может стать перегружена так, что взаимодействовать с ней станет невозможно. Для решения этой проблемы был разработан компонент LayeredChart, который разбивает содержимое на слои, ограничивая количество отображаемых секторов в диаграмме и представляя непоместившиеся атрибуты сектором с названием «Other». Был учтен опыт ранней реализации, и выполнена декомпозиция основного компонента диаграммы на компоненты ChartControls и Chart так, что параметры компонента пробрасываются не более, чем на уровень.

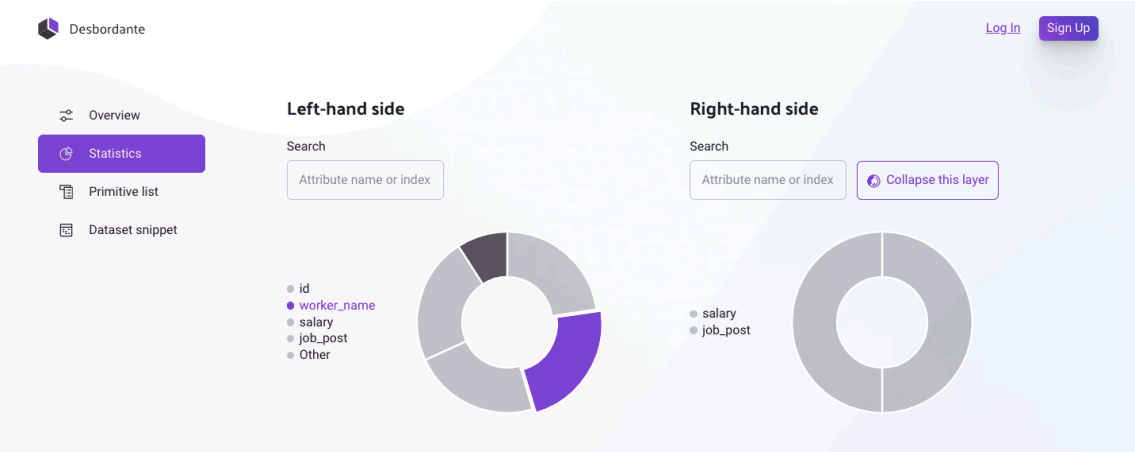

Рис. 2: Страница с визуализаций результатов поиска примитива “функциональная зависимость”

3.6.1. Параметры компонента

Компонент LayeredChart имеет следующий набор параметров.

- attributes — массив атрибутов с подсчитанным весом
- selectedAttributeIndices, setSelectedAttributeIndices — выбранные пользователем атрибуты и метод, вызываемый при изменении выбора

- maxItemsShown — максимальное количество отображаемых секторов на диаграмме
- title — заголовок

3.6.2. Состояние и жизненный цикл компонента

Состоние компонента определяется с помощью React-хуков useState и состоит из глубины просмотра диаграммы, поисковой строки, выбранного сектора и массива атрибутов в текущем слое.

Изменение глубины просмотра происходит по клику на сектор «Other» или кнопку «Collapse». Механизм фильтрации атрибутов реализован в хуке useEffect с помощью указания зависимостей, при изменении которых должна происходить повторная отрисовка компонента. Описания элементов управление вынесено в отдельный компонент ChartControls, а сама диаграмма из библиотеки react-chartjs-2 интегрирована в многоуровневую диаграмму LayeredChart с помощью библиотечного React-компонента Chart.

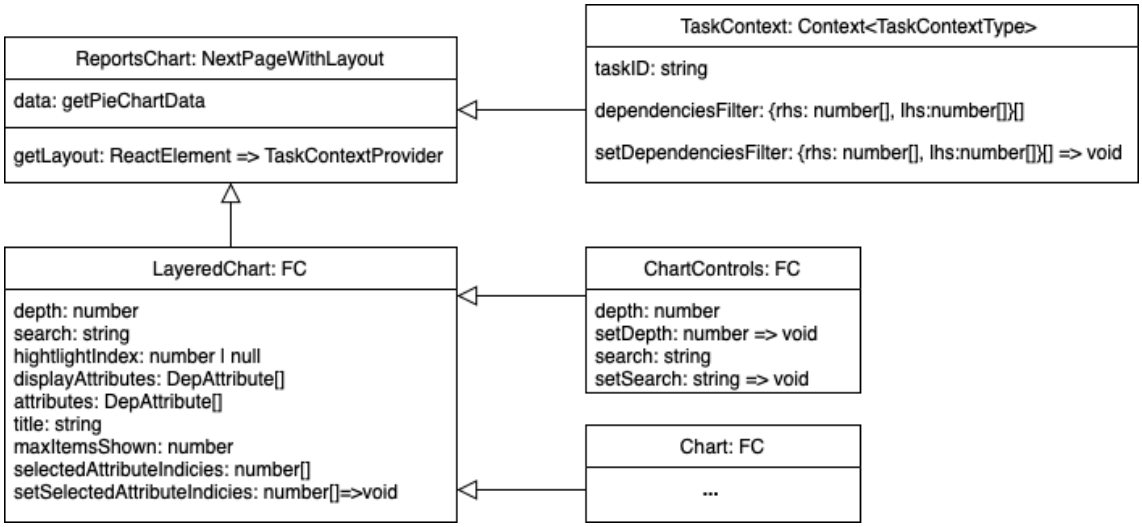

Рис. 3: Диаграмма компонентов, используемых в разработанной странице

4. Тестирование

Было произведено ручное тестирование приложения. Проверена работоспособность диаграмм атрибутов. Проверен функционал поиска атрибутов на диаграмме и выбора атрибутов для фильтрации на других страницах. Проверена отзывчивость верстки при изменении размера экрана.

Тестирование проходило на задачах следующей конфигурации.

- поиск функциональных зависимостей был произведен на наборе табличных данных с 1281732 строками и 10 колонками с помощью алгоритма Pyro, было найдено 54 зависимости, и левая, и правая диаграмма представляли собой два уровня с секторами размера, вычисленном на стороне сервера;

- поиск условных функциональных зависимостей был произведен на наборе табличных данных с 3 колонками и 10 строками с помощью алгоритма CTane, было найдено 60 зависимостей, причем диаграммы представляли собой 3 сектора равного размера, поскольку вычисленный на сервере вес атрибутов был одинаков.

Заключение

В ходе данной работы были выполнены следующие задачи:

- выполнена адаптивная верстка страницы
- реализован компонент многоуровневой диаграммы и элементов управления
- реализовано взаимодействие с серверной частью приложения
- проведена проверка работоспособности приложения

  Ссылка на GitHub-репозиторий https://github.com/vs9h/Desbordante/tree/webapp-redesign (имя пользователя — iliya-b).

Список литературы

[1] Abedjan Ziawasch, Golab Lukasz, Naumann Felix. Profiling Relational Data: A Survey // The VLDB Journal. — 2015. — aug. — Vol. 24, no. 4. — P. 557–581. — URL: https://doi.org/10.1007/s00778-015-0389-y.

[2] Ayerst Jeremy. react-chartjs-2 documentation. — 2022. — accessed: 2022-09-22. URL: https://react-chartjs-2.js.org.

[3] Ilia Barutkin. Разработка клиентской части веб-приложения Desbordante. — 2022. — accessed: 2022-09-22. URL: https://github.com/Mstrutov/Desbordante/blob/main/docs/papers/Frontend%20-%20Ilia%20Barutkin%20-%202021%20spring.pdf.

[4] Kirill Stupakov. Разработка клиентской части веб-приложения Desbordante. — 2022. — accessed: 2022-09-22. URL: https://github.com/Mstrutov/Desbordante/blob/main/docs/papers/Frontend%20-%20Kirill%20Stupakov%20-%202021%20autumn.pdf.

[5] M. Strutovskiy N. Bobrov K. Smirnov, Chernishev G. Desbordante: a Framework for Exploring Limits of Dependency Discovery Algorithms // 2021 29th Conference of Open Innovations Association (FRUCT). — 2021. — P. 344–354.

[6] Papenbrock Thorsten. Data profiling - efficient discovery of dependencies : Ph. D. thesis / Thorsten Papenbrock. — 2017. — 01. — P. 99–100.

[7] Vercel Inc. Next.js Layout Documentation. — 2022. — accessed: 2022-09-22. URL: https://nextjs.org/docs/basic-features/layouts#per-page-layouts.