STUDY OF JSON API STRUCTURE FOR PAGINATION OF WEB-PAGES WITH A LARGE SET OF DATA ON THE EXAMPLE OF CURSOR PAGINATION

Knyazev I.V.[1], Kopteva A.V.[2] (Russian Federation)
Email: Knyazev582@scientifictext.ru

[1]Knyazev Ilia Vadimovich – Senior Software Developer,
JUNE HOMES, BELGOROD;
[2]Kopteva Anna Vitalievna – Senior Software Developer,
YANDEX, MOSCOW

Abstract: *the article analyzes methods and approaches to building a pagination architecture for web pages using the JSON API, offers specific code examples, discusses the pros and cons of the method, as well as recommendations for building an effective backend-frontend integration.*
Keywords: *javascript, json, json api, json formatter, data fetching, pagination, web-pagination, cursor pagination.*

ИССЛЕДОВАНИЕ СТРУКТУРЫ JSON API ДЛЯ ПАГИНАЦИИ ВЕБ-СТРАНИЦ С БОЛЬШИМ НАБОРОМ ДАННЫХ НА ПРИМЕРЕ КУРСОРНОЙ ПАГИНАЦИИ

Князев И.В.[1], Коптева А.В.[2] (Российская Федерация)

[1]Князев Илья Вадимович – старший разработчик программного обеспечения,
June Homes, г. Белгород;
[2]Коптева Анна Витальевна – старший разработчик программного обеспечения,
Яндекс, г. Москва

Аннотация: *в статье анализируются методы и подходы к построению архитектуры пагинации веб-страниц с использованием JSON API, предлагаются конкретные примеры кода, рассматриваются плюсы и минусы метода, а также рекомендации для построения эффективной интеграции бекенд-фронтенд.*
Ключевые слова: *javascript, json, json api, json formatter, data fetching, pagination, web-pagination, cursor pagination.*

Пагинация на основе курсора (также известная как разбивка на страницы с помощью набора ключей) - это распространенная стратегия разбивки на страницы, которая позволяет избежать многих ошибок разбивки на страницы «смещение – ограничение».

Например, при разбивке на страницы с ограничением смещения, если элемент с предыдущей страницы удаляется, пока клиент выполняет разбиение на страницы, все последующие результаты будут сдвинуты вперед на один. Следовательно, когда клиент запрашивает следующую страницу, есть один результат, который он пропустит и никогда не увидит. И наоборот, если результат добавляется к списку результатов, когда клиент выполняет разбиение на страницы, клиент может видеть один и тот же результат несколько раз на разных страницах. Пагинация на основе курсора может предотвратить обе эти возможности.

Пагинация на основе курсора также лучше работает для больших наборов данных в большинстве реализаций.

Для поддержки разбивки на страницы на основе курсора в этой спецификации определены три параметра запроса - страница [размер], страница [до] и страница [после], а также метод предоставления клиентам ссылок для разбивки на страницы и курсоров в теле ответа.

Например, этот запрос получат следующие 100 человек после курсора qwerty:

GET /people?page[size]=100&page[after]=qwerty

Замена страницы [после] на страницу [до] позволит клиенту перемещаться по страницам в обратном направлении.

В качестве альтернативы, чтобы найти всех людей между курсорами qwerty и asdf (эксклюзивно), клиент может запросить:

GET /people?page[after]=qwerty&page[before]=asdf

Возможны другие комбинации, и эти параметры описаны более подробно ниже.

Технические характеристики и концепции.

Пагинация применяется только к упорядоченному списку результатов. Этот порядок не должен меняться между запросами, если только не изменяются базовые данные, чтобы результаты не перемещались между страницами произвольно.

Если запрос клиента с разбивкой на страницы включает параметр запроса (?sort), который только частично упорядочивает результаты, сервер должен применить дополнительные ограничения сортировки - в соответствии с запрошенными клиентом - для создания уникального упорядочивания, если он желает поддерживать разбиение на страницы этих данных.

Например, предположим, что клиент запрашивает:

GET /people?sort=age&page[size]=10

Если несколько человек имеют одинаковый возраст, их относительный порядок не определен (и может варьироваться в зависимости от запроса), что делает невозможным разбиение на страницы. Следовательно, для выполнения запроса сервер должен обрабатывать все запросы с разбивкой на страницы с помощью ?sort=age, как если бы клиент вместо этого попросил отсортировать по возрасту, за которым следует какое-то уникальное поле или комбинация полей (например, ?sort=age, id).

Точно так же, когда разбиваемая на страницы коллекция не имеет естественного или запрошенного клиентом порядка (например, набор объектов идентификаторов ресурсов в отношении), сервер должен назначить порядок, если он желает поддерживать разбиение на страницы.

Сервер может отклонять запросы на разбиение на страницы, если клиент запросил сортировку результатов таким образом, чтобы сервер не мог эффективно разбивать их на страницы. В этом случае сервер должен отклонить запрос в соответствии с правилами для неподдерживаемой ошибки сортировки.

Курсоры.

«Курсор» - это строка, созданная сервером с использованием любого метода, который делит список результатов на те, которые попадают перед курсором.

Например, представьте, что список результатов, разбиваемых на страницы, выглядит следующим образом:

```
[
  { "type": "someType", "id": "1" },
  { "type": "someType", "id": "5" },
  { "type": "someType", "id": "7" },
  { "type": "someType", "id": "8" },
  { "type": "someType", "id": "9" }
]
```

Для этого списка сервер может создать строку курсора qwerty как способ кодирования «id = 5». Таким образом, с этим курсором первый результат попадет перед курсором, второй результат - на курсор, а другие результаты - после него.

Список результатов может меняться между запросами клиента на пагинацию. Например, результат с «id»: «5» может быть удален из набора результатов, если ресурс, на который он ссылается, будет удален. В этом случае курсор qwerty больше не будет попадать на какой-либо один результат, но те же результаты все равно будут перед ним и после него.

В редких случаях сервер может счесть неприемлемым изменение списка результатов между запросами клиента на разбиение на страницы. В таких случаях сервер может закодировать в курсоре информацию, однозначно идентифицирующую клиента или его сеанс, и использовать этот идентификатор для возврата согласованных результатов из одного «моментального снимка» во времени.

Параметры запроса.

Параметр page [size] указывает количество результатов, которые клиент хотел бы видеть в ответе.

Если указана page [size], она должна быть положительным целым числом. Если это требование не выполняется - например, если page [size] отрицательный - сервер должен ответить в соответствии с правилами на ошибку недопустимого параметра запроса.

Для каждой конечной точки, на которой он поддерживает разбиение на страницы, сервер может определить максимальное количество результатов, которые он отправит в ответ на запрос с разбивкой на страницы к этой конечной точке. Это называется «максимальный размер страницы». Если сервер не выбирает максимальный размер страницы для данной конечной точки, он неявно бесконечен.

Если page [size] превышает определяемый сервером максимальный размер страницы, сервер должен отвечать в соответствии с правилами для ошибки превышения максимального размера страницы.

Если page [size] опущен, сервер должен выбрать «размер страницы по умолчанию». Этот размер по умолчанию должен быть целым числом от 1 до максимального размера страницы включительно

Значение page [size] или размер страницы по умолчанию, если page [size] опущен, называется «используемым размером страницы».

В любом допустимом запросе с разбивкой на страницы количество возвращаемых элементов разбиения на страницы должно равняться используемому размеру страницы - при условии, что в списке результатов есть как минимум такое количество элементов и которые удовлетворяют ограничениям страницы [after] и / или страницы [before] параметры (если есть).

Параметры page [after] и page [before] являются необязательными, и оба, если они указаны, принимают курсор в качестве своего значения. Если их значение не является допустимым курсором, сервер должен ответить в соответствии с правилами для ошибки недопустимого параметра запроса.

Параметр page [after] обычно отправляется клиентом для получения следующей страницы, в то время как page [before] используется для получения предыдущей страницы.

Более формально, когда предоставляется page [after], возвращаемые данные с разбивкой на страницы должны иметь в качестве своего первого элемента тот, который находится сразу после курсора в списке результатов. (Исключением является то, что, если в списке результатов нет элементов, следующих за курсором, возвращаемые данные с разбивкой на страницы должны быть пустым массивом.)

Когда предоставляется page [before], последний элемент, возвращаемый в данных с разбивкой на страницы, должен быть элементом, который ближе всего, но все еще находится перед курсором в списке результатов без разбивки на страницы (Аналогично предыдущему, если в списке результатов нет элементов, которые попадают перед курсором, возвращаемые данные с разбивкой на страницы должны быть пустым массивом).

Например, представьте, что список результатов, разбиваемых на страницы, снова следующий:

```
[
  { "type": "someType", "id": "1" },
  { "type": "someType", "id": "5" },
  { "type": "someType", "id": "7" },
  { "type": "someType", "id": "8" },
  { "type": "someType", "id": "9" }
]
```

Далее, представьте, что курсор xxx попадает на запись с «id»: «9», в то время как курсор qwerty по-прежнему падает на запись с «id»: «5».

Тогда, например, если запрос был:

```
GET /some-data?page[after]=qwerty&page[size]=2
```

Ответ будет содержать:

```
{
  "links": {
    "prev": "/some-data?page[before]=yyy&page[size]=2",
    "next": "/some-data?page[after]=zzz&page[size]=2"
  },
  "data": [
    { "type": "someType", "id": "7", "meta": { "page": { "cursor": "yyy" } } },
    { "type": "someType", "id": "8", "meta": { "page": { "cursor": "zzz" } }  }
  ]
}
```

В качестве альтернативы, если запрос был:

```
GET /some-data?page[before]=xxx&page[size]=3
```

Ответ будет содержать:

```
{
  "links": {
    "prev": "/some-data?page[before]=qwerty&page[size]=3",
    "next": "/some-data?page[after]=zzz&page[size]=3"
  },
  "data": [
    { "type": "someType", "id": "5" },
    { "type": "someType", "id": "7" },
    { "type": "someType", "id": "8" }
  ]
}
```

Обратите внимание, что page [before] = xxx приводит к тому, что последний элемент в разбитых на страницы данных ответа становится записью с "id": "8", в то время как количество элементов в разбитых на страницы данных над этим элементом контролируется используемой страницей.

Если запрос клиента с разбивкой на страницы не включает ни параметры page [after], ни page [before], возвращаемые данные с разбивкой на страницы должны начинаться с первого элемента из списка результатов (Если список результатов пуст, данные с разбивкой на страницы должны быть пустым массивом).

Клиенты могут использовать параметры page [after] и page [before] вместе в одном запросе. Они называются «запросами разбивки на страницы по диапазону», поскольку клиент запрашивает все результаты, начиная с момента сразу после курсора page [after] и продолжая до курсора page [before].

Серверы не обязаны поддерживать такие запросы. Если сервер решает не поддерживать эти запросы, он должен отвечать в соответствии с правилами для ошибки разбивки на страницы, не поддерживаемой.

При запросах разбивки на страницы по диапазону сервер должен использовать свой максимальный размер страницы для этой конечной точки в качестве размера страницы по умолчанию. Другими словами, используемый размер страницы будет зависеть от значения параметра page [size] или максимального размера страницы.

Если количество результатов, удовлетворяющих ограничениям page [after] и page [before], превышает используемый размер страницы, сервер должен ответить теми же разбитыми на страницы данными, которые он имел бы, если бы параметр страницы [до] не был предоставлен. Однако в этом случае сервер должен также добавить «rangeTruncated» = true к метаданным разбиения на страницы, чтобы указать клиенту, что разбитые на страницы данные не содержат всех запрошенных им результатов.

Например, с учетом приведенных выше примеров данных и курсоров представьте клиентские запросы:

GET /some-data?page[after]=qwerty&page[before]=xxx

Затем, предполагая, что максимальный размер страницы нашего сервера больше 1, ответ будет содержать:

```
{
  "links": {
    "prev": "/some-data?page[before]=yyy",
    "next": "/some-data?page[after]=zzz"
  },
  "data": [
    { "type": "someType", "id": "7" },
    { "type": "someType", "id": "8" }
  ]
}
```

Однако, если максимальный размер страницы нашего сервера равен 1 или клиент включил page [size] = 1 в свой запрос, ответ будет содержать:

```
{
  "meta": {
    "page": { "rangeTruncated": true }
  },
  "links": {
    "prev": "/some-data?page[before]=yyy&page[size]=1",
    "next": "/some-data?page[after]=yyy&page[size]=1"
  },
  "data": [
    { "type": "someType", "id": "7" }
  ]
}
```

Структура документа.

В этом профиле используются следующие термины для обозначения различных элементов документа:

- Данные с разбивкой на страницы: массив в документе ответа JSON API, который содержит результаты, извлеченные из полного списка результатов, разбиваемых на страницы. Это всегда значение ключа данных. Когда первичные данные разбиваются на страницы, значение набора данных верхнего уровня документа - это данные с разбивкой на страницы. Когда объекты идентификатора ресурса в связи разбиваются на страницы, значение ключа данных в объекте связи - это данные с разбивкой на страницы.
- Ссылки на страницы: объект ссылок, который является родственником данных, разбитых на страницы.
- Метаданные разбивки на страницы: член страницы метаобъекта, который является родственником данных, разбитых на страницы (и ссылок на страницы).
- Элемент разбивки на страницы: запись в массиве данных с разбивкой на страницы.
- Метаданные элемента разбиения на страницы: член страницы метаобъекта, который находится на верхнем уровне элемента данных с разбивкой на страницы.

Чтобы продемонстрировать эти термины, в следующем примере обозначены различные элементы:

GET /people?page[size]=1

```
{
 "links": { },
 "meta": {
  "page": {}
 },
 "data": [
  {
   "type": "people",
   "id": "1",
   "meta": { "page": {} },
   "attributes": {},
   "relationships": {
    "friends": {
     "links": {},
     "data": [
      { "type": "people", "id": "3", "meta": { "page": { } } }
     ]
    }
   }
  }
 ]
}
```

Этот профиль резервирует члена страницы в каждом мета объекте JSON API. (Каждый из этих членов страницы составляет элемент, определенный этим профилем, поэтому они могут иметь псевдонимы.)

Член страницы, если он присутствует, должен содержать объект в качестве своего значения, а любые другие значения не распознаются.

Курсоры элементов.

Сервер может выбрать отправку некоторых или всех элементов пагинации обратно клиенту с элементом курсора в метаданных элемента пагинации. Если он присутствует, этот член должен удерживать курсор, который (во время ответа) «падает» на элемент, с которым он возвращается. Клиенты могут использовать этот курсор для перехода с этого элемента на страницы.

Например, ответ может содержать:

```
{
 "data": [{
  "type": "people",
  "id": "3",
  "meta": {
   "page": { "cursor": "someOpaqueString" }
  }
  //...
 }]
}
```

С этим ответом клиенты могут использовать page[before]=someOpaqueString или page[after]=someOpaqueString для разбивки на страницы от человека с id = 3 в любом направлении.

Типы ссылок JSON API.

JSON API допускает четыре типа ссылок на страницы: предыдущая, следующая, первая и последняя. Рекомендуется, чтобы серверы включали первую и последнюю ссылки, если их вычисление не требует больших затрат. Однако серверы должны включать в ответ предыдущую и следующую ссылку для каждого экземпляра данных с разбивкой на страницы.

Если запрос не содержит параметра page [before], сервер должен определить, существует ли следующая страница, и вернуть null в качестве следующей ссылки, если нет.

Если запрос не содержит параметра page [after], сервер должен определить, существует ли предыдущая страница, и в противном случае вернуть null в качестве ссылки prev.

Во всех остальных случаях серверу следует установить для этих ссылок значение null, если он может без больших затрат определить, что текущий ответ относится к первой или последней странице соответственно.

Однако, если сервер не может легко определить, есть ли предыдущие результаты (при вычислении предыдущей ссылки) или последующие результаты (при вычислении следующей ссылки), он может использовать в этих ссылках URI, который возвращает пустой массив в качестве данных с разбивкой на страницы.

Например, представьте себе запрос:

GET /some-data?page[before]=xyz

Для выполнения этого запроса сервер, скорее всего, выдаст запрос, который фильтрует полный список результатов примеров данных, чтобы найти только те записи, которые находятся перед курсором. По результатам этих запросов сервер не узнает, есть ли дополнительные результаты после курсора, и, возможно, у него нет “дешевого” способа узнать.

Таким образом, в этом случае сервер может просто вернуть следующий URI, в котором параметр page [after] установлен на курсор элемента для последнего элемента в разбитых на страницы данных ответа.

Если клиент отправляется за этой ссылкой, он либо получит пустой массив в качестве данных с разбивкой на страницы, и в этом случае он знает, что достиг конца, либо получит последующие результаты.

Размеры коллекции.

Метаданные разбивки на страницы могут содержать член total, содержащий целое число, указывающее общее количество элементов в списке результатов, которые разбиваются на страницы.

Например, ответ на GET /people?page[size]=2 может включать:

```
{
  "meta": {
    "page": { "total": 200 }
  },
  "data": [
    {
      "type": "people",
      // ...
    },
    {
      "type": "people",
      // ...
    }
  ]
}
```

Метаданные разбиения на страницы могут также содержать член EstimatedTotal. Если присутствует, то значение этого члена должно быть объектом. У этого объекта может быть один ключ, bestGuess. Если он присутствует, bestGuess должен содержать целое число, указывающее наилучшую оценку сервером размера полного списка результатов.

Сервер может выбрать использование EstimatedTotal вместо total, если вычисление точной суммы является “дорогостоящим”.

Обработка ошибок.

- Неподдерживаемая ошибка сортировки

Сервер должен ответить на эту ошибку, отправив 400 Bad Request. Документ ответа должен содержать объект ошибки, который определяет параметр сортировки как источник ошибки и имеет ссылку типа:

https://somesite.org/profiles/cursor-pagination/

- Ошибка превышения максимального размера страницы

Сервер ДОЛЖЕН ответить на эту ошибку, отправив 400 Bad Request. Документ ответа ДОЛЖЕН содержать объект ошибки, который:

- определяет page [size] как источник ошибки;
- предоставляет максимальный размер страницы в виде целого числа в члене maxSize элемента страницы в мета объекте параметра ошибки; а также
- включает ссылку типа https://somesite.org/profiles/cursor-pagination/max-size-exceeded

Если элемент страницы этого профиля не имеет псевдонима, объект ошибки может выглядеть так:

```
{
  "status": "400",
  "meta": {
    "page": { "maxSize": 100 }
  },
  "title": "Запрошенный размер страницы слишком большой",
  "detail": "Вы запросили размер страницы 200, но размер 100 является максимумом",
  "source": {
    "parameter": "page[size]"
  },
```

```
  "links": {
    "type": ["https://somesite.org/profiles/cursor-pagination/max-size-exceeded"]
  }
 }
```

- Ошибка неверного значения параметра

Сервер должен отреагировать на эту ошибку, отправив 400 Bad Request с объектом ошибки в ответном документе, который идентифицирует проблемный параметр в исходном элементе объекта ошибки.

Например, сервер может отправить:

```
{
  "errors": [{
    "title": "...",
    "detail": "...",
    "source": { "parameter": "page[size]" },
    "status": "400"
  }]
}
```

- Ошибка разбивки на страницы не поддерживается

Сервер должен ответить на эту ошибку, отправив 400 Bad Request. Документ ответа должен содержать объект ошибки, имеющий ссылку типа:

https://somesite.org/profiles/cursor-pagination/range-pagination-not-supported

Список литературы / References

1. Документация JSON API / [Электронный ресурс], 2021. Режим доступа: https://jsonapi.org/examples/#pagination/ (дата обращения: 06.09.2021).
2. Документация Google Cloud Storage JSON API / [Электронный ресурс], 2021. Режим доступа: https://cloud.google.com/storage/docs/json_api/ (дата обращения: 04.09.2021).
3. Репозиторий GitHub JSON API / [Электронный ресурс], 2021. Режим доступа: https://github.com/json-api/json-api/ (дата обращения: 01.09.2021).