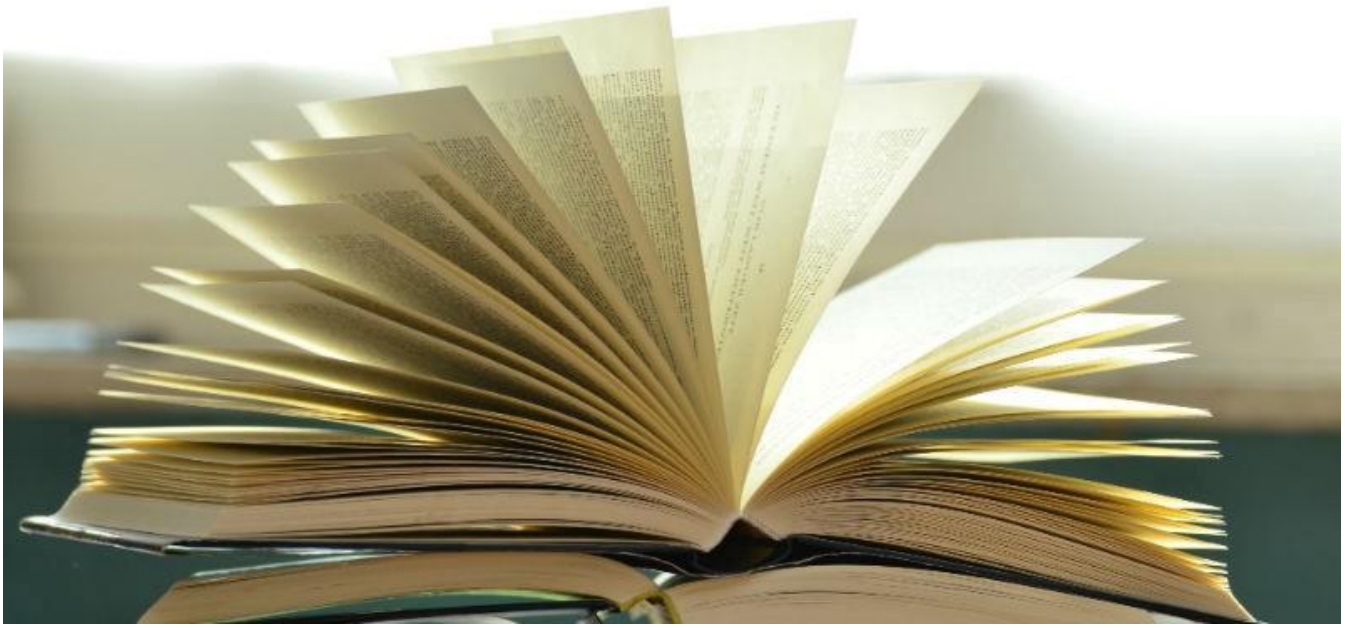
GraphQL Pagination By Example: Part 3

John Tucker
Aug 23, 2018 · 3 min read

Cursor paging; server and client implementation.

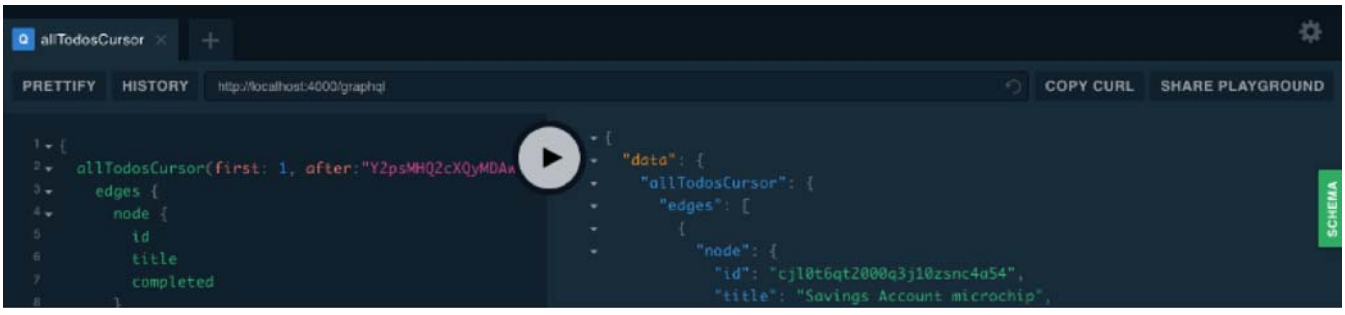


This article is part of a series starting with *GraphQL Pagination By Example: Part 1*.

The final example for this series is available for download.

Cursor Paging

Now we implement the more robust cursor paging in our example.



Observations:

- The *allTodosCursor* query accepts two optional parameters; *first* and *after*
- The *first* parameter limits the number of todos to return, e.g., limit to 1. If unspecified, the query will return all of them.
- The *after* parameter (cursor) indicates which todo to return first, e.g., begin with todo with after the specified one. If unspecified, the query will begin with the first todo.
- The result consists of *edges*; each edge consists of a *node* (the todo) and the *cursor*; cursors are essentially a todo identifier that is only relevant to paging
- The result also consists of *pageInfo*; consists of *endCursor* (the cursor of the last todo in result) and *hasNextPage* (boolean value indicating if there is another page of data available)
- Lastly, the result consists of *totalCount* (total number of todos)

The Code

The *GraphQL* type definitions need to be updated to support cursor paging.

src / server.ts

```
...
const typeDefs = gql`
  ...
  type Edge {
    cursor: String!
    node: Todo!
  }
  type PageInfo {
    endCursor: String
```

```
    hasNextPage: Boolean!
  }
  type TodosResultCursor {
    edges: [Edge]!
    pageInfo: PageInfo!
    totalCount: Int!
  }
  type Query {
    allTodos(
      first: Int,
      offset: Int
    ): TodosResult
    allTodosCursor(
      after: String
      first: Int,
    ): TodosResultCursor
  }
`;
const resolvers = {
  Query: {
    allTodos,
    allTodosCursor,
  },
};
...
```

then we add a resolver to support it.

src / todos.ts

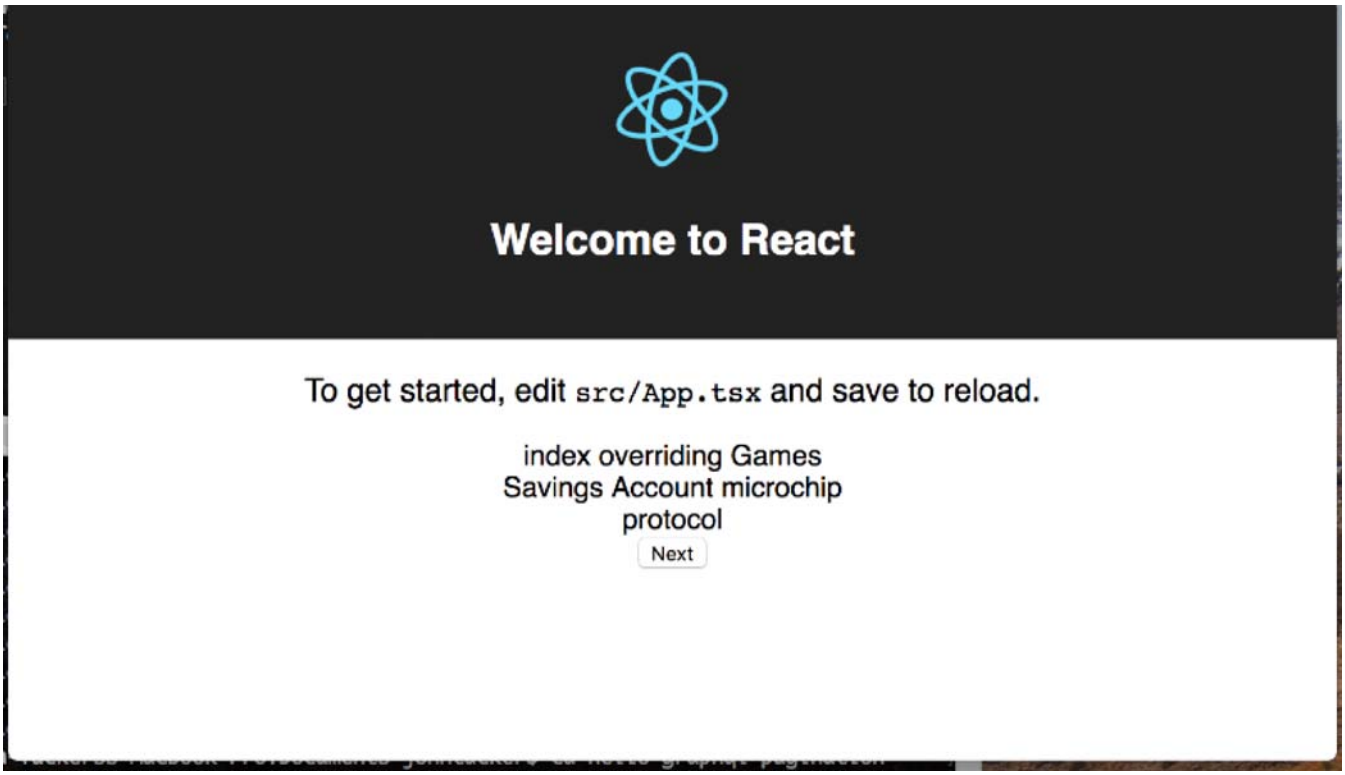
```
...
interface Edge {
  cursor: string;
  node: Todo;
}
interface PageInfo {
  endCursor?: string;
  hasNextPage: boolean;
}
interface TodosResultCursor {
  edges: Edge[];
  pageInfo: PageInfo;
  totalCount: number;
}
...
export const allTodosCursor = (_: any, { after, first }: {after: string, first: number }): Todo
```

```
  if (first < 0) {
    throw new UserInputError('First must be positive');
  }
  const totalCount = data.length;
  let todos = [] as Todo[];
  let start = 0;
  if (after !== undefined) {
    const buff = new Buffer(after, 'base64');
    const id = buff.toString('ascii');
    const index = data.findIndex((todo) => todo.id === id);
    if (index === -1) {
      throw new UserInputError('After does not exist');
    }
    start = index + 1;
  }
  todos = first === undefined ?
    data :
    data.slice(start, start + first);
  let endCursor: string;
  const edges = todos.map((todo) => {
    const buffer = new Buffer(todo.id);
    endCursor = buffer.toString('base64');
    return ({
      cursor: endCursor,
      node: todo,
    });
  });
  const hasNextPage = start + first < totalCount;
  const pageInfo = endCursor !== undefined ?
    {
      endCursor,
      hasNextPage,
    } :
    {
      hasNextPage,
    };
  const result = {
    edges,
    pageInfo,
    totalCount,
  };
  return result;
```

As with offset paging, we will write a *React + Apollo Client* application to consume this cursor paging API.

note: You can skip this section if you are neither interested in *Apollo Client* or familiar with it. This implementation is based on the final article, *GraphQL and Apollo Client by Example: Part 8*, in a series I wrote on *Apollo Client*.

The complete version of the offset client example is available for download.



Observations:

- This implementation does not have a previous button. This is because it is not obvious from the API if there is a previous page; *hasNextPage* indicates if there is a next page.
- One obvious solution to a previous button is to keep an ordered list of pages in the client and use it to determine if the current page is the first; didn't implement this in the example.

Code Highlights

- We first implement a *Pagination* component that maintains the state of the *after, endCursor, and hasNextPage* (the *FIRST* value is fixed to 3). While this example uses

React state, one could use a global state management solution, e.g., *Redux* or *Apollo Client* local state

- Other then the different state and API result shapes, this code is virtually identical to the earlier offset paging example

Wrap Up

It turns out that cursor based paging, despite the complicated sounding name, is not much different than the more familiar offset paging and likely the best solution for most situations.

Hope you found this useful.

GraphQL Typescript Web Development Apollo Nodejs