EntityListProvider and useEntityList Hook

The `EntityListProvider` and `useEntityList` hook are central to implementing feeds in Replyke. Here, we'll explore how they work and the properties that can be passed to customize their behavior.

Using the EntityListProvider

To integrate feeds into your app, wrap any content requiring entity list context with the `EntityListProvider`. All child components within the provider gain access to the entity-list's data and functions via the `useEntityList` hook.

Multiple Entity List Providers

You are not limited to a single entity list in your app. You can implement multiple entity lists as needed, but avoid wrapping any content with more than one `EntityListProvider`, as this can cause unexpected behavior. For instance, you can use multiple `EntityListProvider` instances on the same screen. In a social app's home screen, you could display two feeds side by side - one sorted by "new" and the other by "popularity" - and allow users to switch between them with gestures, but they must be siblings rather than wrapped one withing the other.

EntityListProvider Properties

Below is the interface for `EntityListContextProps`, followed by an explanation of each property:

```
interface EntityListContextProps {
  sortBy?: EntityListSortByOptions;
  limit?: number;
  timeFrame?: TimeFrame | null;

  sourceId?: string | null;
  userId?: string | null;
  followedOnly?: boolean;

  keywordsFilters?: KeywordsFilters | null;
  titleFilters?: TitleFilters | null;
  contentFilters?: ContentFilters | null;
  attachmentsFilters?: AttachmentsFilters | null;
  locationFilters?: LocationFilters | null;
```

```
  metadataFilters?: MetadataFilters | null;

  idle?: boolean;
  onReset?: () => void;
  infuseData?: (foreignId: string) => Promise<Record<string, any> | null>;
}
```

Detailed Explanation of Properties

The `EntityListProvider` accepts the following properties to customize the feed:

`sortBy`: Determines the default sorting criteria for the entity list. Available options include:

- `"top"`: Sorts entities by net upvotes (upvotes minus downvotes).
- `"hot"`: Sorts by a "hotness" score, auto-generated by Replyke.
- `"new"`: Sorts entities by creation date, with the newest appearing first.
- `"controversial"`: Sorts by mixed voting score.

By default, this property is set to `"hot"`. Note that sorting by time frame (e.g., "hour," "day," "week," etc.) restricts results to entities created within the specified period.

`limit`: Specifies the number of entities fetched per "page" when loading more entities. A lower limit reduces costs by avoiding redundant fetching. If no value is provided, the default is `10`.

`timeFrame`: Restricts the list to entities created within a specific time frame. Supported options include:

- `"hour"`: Only includes entities created in the last hour.
- `"day"`: Includes entities created in the past 24 hours.
- `"week"`: Fetches entities from the past seven days.
- `"month"`: Returns entities from the past 30 days.
- `"year"`: Includes entities created in the past 12 months.

If no value is provided, this property defaults to `null`, meaning no time frame restriction is applied.

`userId`: If a user ID is provided, the list will include only entities created by that user. This property is particularly useful for building user profile pages. If no user ID is provided, the list will not b filtered by user. By default, this property is set to `null`.

`sourceId`: The sourceId property allows you to logically separate entities in Replyke based on their origin or use case within your application. This is useful when you're using Replyke to manage content across multiple contexts—such as a blog, forum, or core app features—and want to ensure that entities from one source don't get mixed in with others. By assigning a consistent sourceId to entities within the same context, you can easily filter and fetch only the relevant data for each part of your system.

`followedOnly`: A boolean flag that, when set to `true`, limits the list to entities created by accounts the currently logged-in-user follows. This property defaults to `false`.

`keywordsFilters`: Allows filtering by the `keywords` property of entities. This property expects an object with two optional fields:

- `includes`: An array of keywords. Entities must contain **all** specified keywords to be included.
- `doesNotInclude`: An array of keywords. Entities containing **any** of these keywords are excluded.

By default, no filtering is applied (`null`).

`titleFilters`: Filters entities based on their `title` property. This property accepts an object with the following optional fields:

- `hasTitle`: A boolean flag to include or exclude entities with a title.
- `includes`: A string or array of strings. Entities must include at least one of the specified strings in their title.
- `doesNotInclude`: A string or array of strings. Entities with any of these strings in their title are excluded.

By default, this property is set to `null`, meaning no filtering by title is applied.

`contentFilters`: Filters entities based on their `content` property. This property accepts an object with the following optional fields:

- `hasContent`: A boolean flag to include or exclude entities with content.
- `includes`: A string or array of strings. Entities must include at least one of the specified strings in their content.
- `doesNotInclude`: A string or array of strings. Entities with any of these strings in their content are excluded.

By default, this property is set to `null`, meaning no filtering by content is applied.

`attachmentsFilters`: Filters entities based on the presence of attachments. This property accepts an object with the `hasAttachments` flag, which can include or exclude entities based on whether they contain attachments. By default, no attachments filtering is applied (`null`).

`locationFilters`: Filters entities by geographic location. This property expects an object with the following fields:

- `latitude`: The latitude of the center point.
- `longitude`: The longitude of the center point.
- `radius`: The radius (in kilometers) around the center point within which entities should be included.

Entities without a location property or outside the specified radius are excluded. By default, this property is set to `null`.

`metadataFilters`: Enables filtering based on the `metadata` property of entities, which can store unstructured data. This property accepts an object with the following optional fields:

- `includes`: Key-value pairs that entities must contain.
- `doesNotInclude`: Key-value pairs that entities must not contain.
- `exists`: Keys that must be present in the metadata.
- `doesNotExist`: Keys that must not be present in the metadata.

By default, this property is set to `null`. The metadata filtering capabilities will be explored further in subsequent chapters.

`idle?: boolean`: The `idle` property allows developers to control when the `EntityListProvider` initiates its automatic fetch of entities. By default, `EntityListProvider` starts fetching data as soon as it is initialized, which is typically optimal for most use cases. However, there are scenarios where delaying the fetch might be necessary—for instance, when you need to fetch data from the user's device storage for filtering, request their location, or complete other preliminary actions. Setting the `idle` property to `true` cancels the automatic fetch, putting the provider in a paused state. This ensures that the first fetch is only triggered manually, as explained in the following chapter ('kickstart' function). Using `idle` helps prevent potential UI glitches and reduces unnecessary resource usage during the initial setup phase.

`onReset`: An optional callback function executed whenever filters change and the list is being reset. This function does not affect the entity list's behavior but can be used for custom actions such as displaying a toast notification or any other behavior, if desired, upon a list's data refresh.

`infuseData`: This optional callback is essential for applications integrating Replyke with external data sets. The function receives a `foreignId` and fetches additional data for that entity from an external database. The fetched data is merged with Replyke's entity data, providing a complete data object for the application. This feature will be covered in greater detail in the following chapters.

Note on Using Filters in the EntityListProvider

1. **Filters for Static or Initial Filtering**: Passing filters directly into the `EntityListProvider` is the correct approach both for cases where the list should always be filtered in a specific way (e.g., on a user's profile, where the list is filtered by that user ID) and for cases where we want the list to start with certain filters but allow dynamic changes later. This ensures the list is initialized with the desired filtering logic.
2. **Filters Passed to `EntityListProvider` Are Static After Initialization**: When filters are passed as props to the `EntityListProvider`, they are only used during the initialization of the list. If the variables used for these filters change after the list has been initialized, the list will **not** reset or reapply the updated filters. To apply dynamic changes to the filters after initialization, use the `useEntityList` hook to modify the list context dynamically.

For detailed guidance on dynamically updating filters, continue to the next chapters.

Last updated on May 7, 2025