Metadata Filters

The **Metadata Filters** in Replyke allow developers to filter entities based on their `metadata` property, which is flexible and can store unstructured data. This filter is highly versatile, enabling fine-grained control over the entity list by matching key-value pairs, verifying key existence, and excluding unwanted metadata.

Overview of Metadata Filters

The `metadataFilters` property can be passed into the `EntityListProvider` as an object with the following structure:

```
export interface MetadataFilters {
  includes?: Record<string, unknown>;
  doesNotInclude?: Record<string, unknown>;
  exists?: string[];
  doesNotExist?: string[];
}
```

- **`includes`**: Specifies key-value pairs that must exist in the entity's `metadata`. Entities must include all specified key-value pairs.
- **`doesNotInclude`**: Specifies key-value pairs that must not exist in the entity's `metadata`. If any specified key-value pair exists, the entity is excluded.
- **`exists`**: An array of keys that must exist in the entity's `metadata`, regardless of their value.
- **`doesNotExist`**: An array of keys that must not exist in the entity's `metadata`.

Dynamically Updating Metadata Filters

Developers can dynamically update the metadata filters using the `setMetadataFilters` function provided by the `useEntityList` hook. This function expects a new `metadataFilters` object or `null` to clear the filters.

```
const { setMetadataFilters } = useEntityList();

// Set a filter to include entities with specific metadata
setMetadataFilters({ includes: { category: "tech", featured: true } });

// Reset metadata filters
```

```
setMetadataFilters(null);
```

How Metadata Filters Work

Metadata Filters allow for complex queries that include:

1. **Key-Value Matching**:
   - `includes`: Filters entities to include those whose `metadata` contains all specified key-value pairs.
   - `doesNotInclude`: Filters entities to exclude those whose `metadata` contains any specified key-value pairs.
2. **Key Existence**:
   - `exists`: Ensures entities include the specified keys in their `metadata`.
   - `doesNotExist`: Ensures entities do not include the specified keys in their `metadata`.
3. **Combined Conditions**: Developers can combine these conditions to create sophisticated filtering logic.

Example Use Cases

1. Passing Static Filters to the EntityListProvider

To set static metadata filters at the entity list's initialization, pass them directly to the `EntityListProvider`:

```
<EntityListProvider
  metadataFilters={{
    includes: { category: "tech", featured: true },
    doesNotInclude: { deprecated: true },
    exists: ["author"],
    doesNotExist: ["archived"],
  }}
>
  <MyFeedComponent />
</EntityListProvider>
```

This setup ensures that the entity list includes:

- Entities where `metadata.category` is "tech" and `metadata.featured` is `true`.
- Entities where `metadata.deprecated` is not `true`.

- Entities with a `metadata.author` key.
- Entities without a `metadata.archived` key.

2. Dynamically Updating Filters Based on User Interaction

Developers can allow users to refine the entity list dynamically using the `setMetadataFilters` function:

```
const { setMetadataFilters, entities, loadMore } = useEntityList();

const applyFilters = () => {
  setMetadataFilters({
    includes: { category: "tech" },
    exists: ["author"],
  });
};

return (
  <div>
    <h1>Filtered Entity List</h1>
    <button onClick={applyFilters}>Apply Tech Filters</button>
    <button onClick={() => setMetadataFilters(null)}>Clear Filters</button>
    <ul>
      {entities.map((entity) => (
        <li key={entity.id}>
          <h2>{entity.title}</h2>
          <p>Metadata: {JSON.stringify(entity.metadata)}</p>
        </li>
      ))}
    </ul>
    <button onClick={loadMore}>Load More</button>
  </div>
);
```

In this example, clicking the “Apply Tech Filters” button updates the entity list to include entities categorized as “tech” and with an `author` key in their metadata.

Important Notes

- Metadata Filters are powerful for working with unstructured data, allowing highly customized entity lists.
- Filters are applied immediately, and the entity list resets to reflect the updated conditions.
- Passing `null` to `setMetadataFilters` clears all metadata-based filtering.

Conclusion

Metadata Filters provide a robust mechanism to filter entities based on the `metadata` property. With support for key-value matching, key existence validation, and exclusion conditions, these filters allow developers to implement advanced and highly specific entity list customization. Whether setting static filters at initialization or dynamically adjusting them based on user input, Metadata Filters make it easy to tailor the entity list for diverse application needs.

Last updated on May 7, 2025