DataView Migration Notes

Overview

Migration of `ServiceAccountsTable` component from PatternFly Table to PatternFly DataView.

Migration Steps

1. Replace Table with DataView Structure

- Replace `Table`, `Thead`, `Tbody`, `Tr`, `Th`, `Td` with `DataView`, `DataViewTable`, `DataViewToolbar`
- Import DataView components from `@patternfly/react-data-view/dist/dynamic/` paths
- Keep `ActionsColumn` from `@patternfly/react-table` for row actions

2. Implement URL-Integrated Pagination

- Use `useDataViewPagination` hook from DataView
- Hook automatically syncs pagination state with URL search params (`page`, `perPage`)
- Parent component reads these params and includes them in API calls

3. Add Loading and Empty States

- Use `DataViewState.loading` and `DataViewState.empty` for state management
- Implement skeleton loading with `SkeletonTableHead` and `SkeletonTableBody` from `@patternfly/react-component-groups`
- Create custom empty state component with conditional message based on active filters

4. Implement Server-Side Sorting

- Define column-to-API-field mapping for sortable columns
- Read `orderBy` and `sortOrder` from URL search params
- Pass sorting params to API call
- Update URL on sort change (resets to page 1)

5. Implement Server-Side Filtering

- Add `DataViewFilters` and `DataViewTextFilter` components to toolbar

- Implement debouncing (300ms) to avoid excessive API calls while typing
- Sync filter values with URL search params
- Pass filter params to API call

6. Update API Layer

- Extend `fetchServiceAccounts` function to accept `orderBy`, `sortOrder`, and `filters` parameters
- Build query string with all pagination, sorting, and filtering params

Storybook Test Updates

After migration, MSW handlers in Storybook stories required updates to support the new query parameters:

- **Sorting**: Handler now reads `orderBy` and `sortOrder` params and sorts mock data accordingly
- **Filtering**: Handler reads `name`, `clientId`, and `creator` params and filters mock data

The actual journey test steps (user interactions, assertions) remained unchanged since they test user behavior rather than implementation details.

Lessons Learned & Key Takeaways

1. Commit Early, Commit Often

Problem: The entire DataView migration, Storybook setup, and journey tests were developed in one large batch of changes. This made it difficult to:

- Isolate issues when something broke
- Roll back specific changes without losing other work
- Review changes incrementally

Recommendation: Break large migrations into smaller logical commits:

1. Storybook configuration and dependencies
2. Mock setup for Chrome components
3. Basic DataView migration (structure only)
4. Pagination implementation
5. Loading and empty states
6. Sorting implementation
7. Filtering implementation

8. Storybook journey tests

Each commit should be independently functional and testable.

2. API Readiness Issues

Problem: The backend API had several inconsistencies that complicated the frontend implementation:

- Filtering by `clientId` is supported, but sorting by `clientId` doesn't make practical sense
- Sorting by `owner` / `createdBy` is not supported, despite being a commonly requested feature
- Inconsistent naming: API uses `creator` parameter but returns `createdBy` field
- Non-standard pagination: API uses `first` / `max` instead of common `page` / `perPage` or `offset` / `limit`

Recommendation: Before starting frontend work that depends on API features (sorting, filtering), verify the API specification and test the endpoints directly. Document any API limitations early in the process.

3. PatternFly Dynamic Import Bug

Problem: Using dynamic imports for `DataViewFilters` caused a runtime error:

```
Cannot read properties of null (reading 'firstElementChild')
at ToolbarFilter
```

Broken code (dynamic import):

```
// ❌ Causes "firstElementChild" error during re-renders
import { DataViewFilters } from '@patternfly/react-data-view/dist/dynamic/DataViewFilt
import { DataViewTextFilter } from '@patternfly/react-data-view/dist/dynamic/DataViewT
```

Working code (cjs import):

```
// ✅ Works correctly - following rbac-ui pattern
import DataViewFilters from '@patternfly/react-data-view/dist/cjs/DataViewFilters';
import { DataViewTextFilter } from '@patternfly/react-data-view';
```

Root cause: The `dynamic` import variant appears to have timing/lifecycle issues with the `ToolbarFilter` component used internally by `DataViewTextFilter`. When the component re-renders (e.g., opening a modal), it tries to access a DOM element that doesn't exist yet.

Recommendation: When using PatternFly DataView filters, prefer `cjs` imports over `dynamic` imports. Always check reference implementations (like rbac-ui) for proven import patterns.