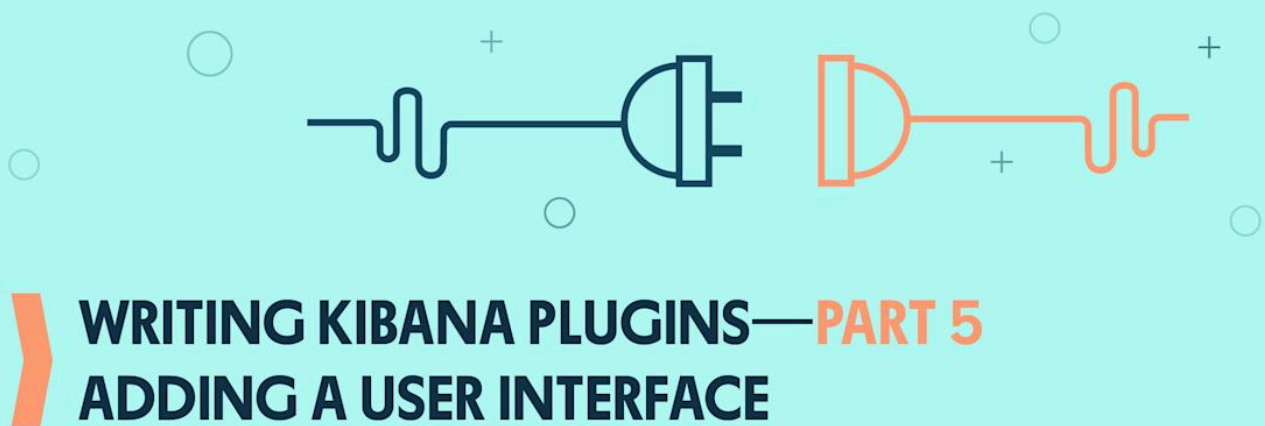


Writing Kibana Plugins , Part 5: Adding a User Interface

Jochen Kressin · 2026-08-31



Note for readers: Parts 1–4 were written earlier and may reference older Kibana versions. This final part targets **Kibana 9.0.1**. If you follow the full series, build your plugin against the same Kibana minor/patch you intend to run.

In this article we finish the plugin by wiring a small, reproducible UI on top of the server routes created earlier. The result is a searchable, paginated accounts table backed by Elasticsearch, with clone and delete actions and clear user feedback.

Use the same plugin id and index name used in Parts 1–4: `sgKibanaDemoPlugin` and `searchguard_demo_accounts`.

Manifest (`kibana.json`)

Update your plugin manifest so Kibana accepts the plugin at startup:

```
{
  "id": "sgKibanaDemoPlugin",
  "version": "1.0.0",
  "kibanaVersion": "9.0.1",
  "server": true,
  "ui": true,
  "requiredPlugins": ["navigation"],
  "optionalPlugins": []
}
```

Shared constants (`common/index.ts`)

```
export const PLUGIN_ID = 'sgKibanaDemoPlugin';
export const PLUGIN_NAME = 'SG Kibana Demo Plugin';
export const INDEX_NAME = 'searchguard_demo_accounts';
```

Server: wire ES client into routes

Parts 1–4 used the Kibana server to index data. For consistency, the routes below call Elasticsearch as the Kibana internal (server) user. Pass `core.elasticsearch.client` from `start()` into your routes.

`server/plugin.ts` (snippet)

```
// server/plugin.ts
import type { PluginInitializerContext, CoreSetup, CoreStart } from '@kbn/core/server';
import { defineRoutes } from './routes';

export class SgKibanaDemoPluginPlugin {
  private logger: any;

  constructor(initializerContext: PluginInitializerContext) {
    this.logger = initializerContext.logger.get();
  }

  public setup(core: CoreSetup) {
    // setup work (saved objects, features) if needed
  }

  public start(core: CoreStart) {
    const esClient = core.elasticsearch.client;
    const router = core.http.createRouter();
    const logger = this.logger ?? console;

    defineRoutes({ router, logger, esClient });
  }

  public stop() {
    // cleanup if necessary
  }
}
```

Server routes: `server/routes/index.ts`

Create or replace this file. It registers three routes (GET with search + pagination, POST to create, DELETE to remove). `@kbn/config-schema` validates inputs. The server uses `esClient.asInternalUser.*` for server-originated calls.

```
// server/routes/index.ts
import { schema } from '@kbn/config-schema';
import type { IRouter } from '@kbn/core/server';
import { INDEX_NAME } from '../../common';

type DefineRoutesDeps = {
  router: IRouter;
  logger: ReturnType<any>;
  esClient: any; // core.elasticsearch.client
};

export function defineRoutes({ router, logger, esClient }: DefineRoutesDeps) {
  // GET - list accounts with optional search and pagination
  router.get(
    {
      path: '/api/sg_kibana_demo_plugin/accounts',
      validate: {
        query: schema.object({
          q: schema.maybe(schema.string()),
          page: schema.maybe(schema.number({ min: 0 })),
          pageSize: schema.maybe(schema.number({ min: 1, max: 1000 })),
        }),
      },
    },
    async (context, request, response) => {
      try {
        const { q, page = 0, pageSize = 10 } = request.query as {
          q?: string;
          page?: number;
          pageSize?: number;
        };

        const from = page * pageSize;

        const esQuery = q
          ? {
              query: {
                multi_match: {
                  query: q,
                  fields: ['name^3', 'company', 'address', 'phone'],
                  type: 'phrase_prefix',
                },
              },
            }
          : { query: { match_all: {} } };

        const { body } = await esClient.asInternalUser.search({
          index: INDEX_NAME,
          body: {
            ...esQuery,
            from,
            size: pageSize,
            sort: [{ name: { order: 'asc' } }],
          },
        });

        return response.ok({ body });
      }
    }
  );
}
```

```

    } catch (error) {
      logger.error(`Get accounts error. ${error?.stack ?? error}`);
      return response.internalError({ body: error?.message ?? String(error) });
    }
  }
);

// POST - create an account
router.post(
  {
    path: '/api/sg_kibana_demo_plugin/accounts',
    validate: {
      body: schema.object({
        name: schema.string(),
        balance: schema.maybe(schema.string()),
        gender: schema.maybe(schema.string()),
        company: schema.maybe(schema.string()),
        phone: schema.maybe(schema.string()),
        address: schema.maybe(schema.string()),
        registered: schema.maybe(schema.string()),
        friends: schema.maybe(schema.arrayOf(schema.any())),
      }),
    },
  },
  async (context, request, response) => {
    try {
      const resp = await esClient.asInternalUser.index({
        index: INDEX_NAME,
        body: request.body,
        refresh: 'wait_for',
      });

      return response.ok({ body: resp.body || resp });
    } catch (error) {
      logger.error(`Add account error. ${error?.stack ?? error}`);
      return response.internalError({ body: error?.message ?? String(error) });
    }
  }
);

// DELETE - delete an account by id
router.delete(
  {
    path: '/api/sg_kibana_demo_plugin/accounts/{id}',
    validate: {
      params: schema.object({
        id: schema.string(),
      }),
    },
  },
  async (context, request, response) => {
    try {
      const {
        params: { id },
      } = request;

      const resp = await esClient.asInternalUser.delete({

```

```

        index: INDEX_NAME,
        id,
        refresh: 'wait_for',
    });

    return response.ok({ body: resp.body || resp });
  } catch (error) {
    logger.error(`Delete account error. ${error?.stack ?? error}`);
    return response.internalError({ body: error?.message ?? String(error) });
  }
}
);
}

```

Notes

- `from / size` implement server-side pagination.
- `multi_match` with `phrase_prefix` gives a friendly search across common fields.
- `refresh: 'wait_for'` makes writes visible quickly in dev/testing.

Types (`public/types.ts`)

```

export interface Friend {
  id: number;
  name: string;
}

export interface Account {
  _id: string;
  name: string;
  balance?: string;
  gender?: string;
  company?: string;
  phone?: string;
  address?: string;
  registered?: string;
  friends?: Friend[];
}

```

Client entry: `public/application.tsx`

Kibana 9 uses modern React; to be future-proof use React 18's root API. This mounts your app and hands core services into it.

```
// public/application.tsx
import React from 'react';
import { createRoot } from 'react-dom/client';
import type { AppMountParameters, CoreStart } from '@kbn/core/public';
import type { AppPluginStartDependencies } from './types';
import { SgKibanaDemoPluginApp } from './components/app';

export const renderApp = (
  coreStart: CoreStart,
  depsStart: AppPluginStartDependencies,
  { element }: AppMountParameters
) => {
  const root = createRoot(element);
  root.render(
    <SgKibanaDemoPluginApp
      basename="/app/sgKibanaDemoPlugin"
      notifications={coreStart.notifications}
      http={coreStart.http}
    />
  );

  return () => root.unmount();
};
```

Main React app: `public/components/app.tsx`

This is the complete UI component. It uses EUI for layout and `EuiBasicTable` for the paginated table.

If you already have a version of this component from earlier parts, the logic and props are the same — this copy is ready to drop into your `public/components` folder.

```
// public/components/app.tsx
import React, { useState, useEffect, useCallback } from 'react';
import { i18n } from '@kbn/i18n';
import {
  EuiPage,
  EuiPageBody,
  EuiPageContent,
  EuiPageContentBody,
  EuiPageContentHeader,
  EuiPageContentHeaderSection,
  EuiPageHeader,
  EuiTitle,
  EuiBasicTable,
  EuiHealth,
  EuiButton,
  EuiFieldSearch,
  EuiSpacer,
  EuiFlexGroup,
  EuiFlexItem,
  EuiConfirmModal,
  EuiText,
} from '@elastic/eui';
import type { CoreStart } from '@kbn/core/public';
import { PLUGIN_NAME } from '../../common';
import type { Account } from '../../types';

interface Props {
  basename: string;
  notifications: CoreStart['notifications'];
  http: CoreStart['http'];
}

export const SgKibanaDemoPluginApp: React.FC<Props> = ({ notifications, http }) => {
  const [accounts, setAccounts] = useState<Account[]>([]);
  const [loading, setLoading] = useState(false);
  const [searchValue, setSearchValue] = useState('');
  const [pageIndex, setPageIndex] = useState(0);
  const [pageSize, setPageSize] = useState(10);
  const [totalItems, setTotalItems] = useState(0);
  const [deleteModalAccount, setDeleteModalAccount] = useState<Account | null>(null);

  const fetchAccounts = useCallback(
    async (opts?: { q?: string; page?: number; pageSize?: number }) => {
      setLoading(true);
      try {
        const page = typeof opts?.page === 'number' ? opts.page : pageIndex;
        const size = typeof opts?.pageSize === 'number' ? opts.pageSize : pageSize;
        const q = opts?.q ?? searchValue;

        const params = new URLSearchParams();
        if (q) params.set('q', q);
        params.set('page', String(page));
        params.set('pageSize', String(size));

        const body: any = await http.get(`/api/sg_kibana_demo_plugin/accounts?${params.toString()}`);
      } catch {
        // Error handling
      }
    },
    [pageIndex, pageSize, searchValue, http]
  );
};
```

```

const hits: any[] = body?.hits?.hits || [];
const total: number = (body?.hits?.total?.value ?? body?.hits?.total) || hits.length;

const accountList: Account[] = hits.map((h) => ({ _id: h._id, ...h._source }));
setAccounts(accountList);
setTotalItems(total);
} catch (error: any) {
  notifications.toasts.addDanger({
    title: i18n.translate('sgKibanaDemoPlugin.fetchError', {
      defaultMessage: 'Failed to fetch accounts',
    }),
    text: error?.message ?? String(error),
  });
} finally {
  setLoading(false);
}
},
[http, notifications, searchValue, pageIndex, pageSize]
);

useEffect(() => {
  fetchAccounts({ page: 0 });
  setPageIndex(0);
}, [fetchAccounts]);

const deleteAccount = async (id: string) => {
  try {
    await http.delete(`/api/sg_kibana_demo_plugin/accounts/${id}`);
    notifications.toasts.addSuccess({
      title: i18n.translate('sgKibanaDemoPlugin.deleteSuccess', {
        defaultMessage: 'Account deleted successfully',
      }),
    });
  }

  const newPageIndex = Math.max(0, pageIndex);
  fetchAccounts({ page: newPageIndex, pageSize });
} catch (error: any) {
  notifications.toasts.addDanger({
    title: i18n.translate('sgKibanaDemoPlugin.deleteError', {
      defaultMessage: 'Failed to delete account',
    }),
    text: error?.message ?? String(error),
  });
} finally {
  setDeleteModalAccount(null);
}
};

const cloneAccount = async (account: Account) => {
  const { _id, ...payload } = account;
  const cloned = { ...payload, name: `${account.name} (Copy)` };
  try {
    await http.post('/api/sg_kibana_demo_plugin/accounts', {
      body: JSON.stringify(cloned),
    });
    notifications.toasts.addSuccess({
      title: i18n.translate('sgKibanaDemoPlugin.cloneSuccess', {

```

```
        defaultMessage: 'Account cloned successfully',
      )),
    });
    fetchAccounts({ page: 0, pageSize });
    setPageIndex(0);
  } catch (error: any) {
    notifications.toasts.addDanger({
      title: i18n.translate('sgKibanaDemoPlugin.cloneError', {
        defaultMessage: 'Failed to clone account',
      }),
      text: error?.message ?? String(error),
    });
  }
};

const columns = [
  {
    field: 'name',
    name: 'Name',
    sortable: true,
    truncateText: true,
  },
  {
    field: 'company',
    name: 'Company',
    truncateText: true,
  },
  {
    field: 'balance',
    name: 'Balance',
    sortable: true,
  },
  {
    field: 'gender',
    name: 'Gender',
    render: (gender: string) => (
      <EuiHealth color={gender === 'female' ? 'success' : 'primary'}>{gender}</EuiHealth>
    ),
  },
  {
    field: 'phone',
    name: 'Phone',
    truncateText: true,
  },
  {
    name: 'Actions',
    actions: [
      {
        name: 'Clone',
        description: 'Clone this account',
        icon: 'copy',
        type: 'icon',
        onClick: (account: Account) => cloneAccount(account),
      },
      {
        name: 'Delete',
        description: 'Delete this account',
      },
    ],
  },
];
```

```

        icon: 'trash',
        type: 'icon',
        color: 'danger',
        onClick: (account: Account) => setDeleteModalAccount(account),
      },
    ],
  },
];

const onTableChange = ({ page: tablePage }: { page?: any }) => {
  if (!tablePage) return;
  const { index: newIndex, size: newSize } = tablePage;
  setPageIndex(newIndex);
  setPageSize(newSize);
  fetchAccounts({ page: newIndex, pageSize: newSize, q: searchValue });
};

return (
  <>
    <EuiPage restrictWidth="1000px">
      <EuiPageBody component="div">
        <EuiPageHeader>
          <EuiTitle size="l">
            <h1>{PLUGIN_NAME}</h1>
          </EuiTitle>
        </EuiPageHeader>
        <EuiPageContent>
          <EuiPageContentHeader>
            <EuiPageContentHeaderSection>
              <EuiTitle>
                <h2>Account Management</h2>
              </EuiTitle>
            </EuiPageContentHeaderSection>
          </EuiPageContentHeader>
          <EuiPageContentBody>
            <EuiFlexGroup>
              <EuiFlexItem>
                <EuiFieldSearch
                  placeholder="Search accounts..."
                  value={searchValue}
                  onChange={(e) => setSearchValue((e.target as HTMLInputElement).value)}
                  onSearch={(value) => {
                    setPageIndex(0);
                    fetchAccounts({ q: value, page: 0, pageSize });
                  }}
                  isClearable
                  aria-label="Search accounts"
                />
              </EuiFlexItem>
              <EuiFlexItem grow={false}>
                <EuiButton
                  iconType="refresh"
                  onClick={() => fetchAccounts({ page: pageIndex, pageSize, q: searchValue })}
                >
                  Refresh
                </EuiButton>
              </EuiFlexItem>
            </EuiFlexGroup>
          </EuiPageContentBody>
        </EuiPageContent>
      </EuiPageBody>
    </EuiPage>
  </>
);

```

```

        </EuiFlexGroup>
        <EuiSpacer />
        <EuiBasicTable
          items={accounts}
          columns={columns}
          loading={loading}
          pagination={{
            pageIndex,
            pageSize,
            totalItemCount: totalItems,
            pageSizeOptions: [10, 20, 50],
          }}
          onChange={onTableChange}
        />
      </EuiPageContentBody>
    </EuiPageContent>
  </EuiPageBody>
</EuiPage>

{deleteModalAccount && (
  <EuiConfirmModal
    title="Delete Account"
    onCancel={() => setDeleteModalAccount(null)}
    onConfirm={() => deleteAccount(deleteModalAccount._id)}
    cancelButtonText="Cancel"
    confirmButtonText="Delete"
    buttonColor="danger"
  >
    <EuiText size="s">
      Are you sure you want to delete the account for <strong>{deleteModalAccount.name}</strong>?
    </EuiText>
  </EuiConfirmModal>
)}
</>
);
};

```

Conclusion

You now have a complete client UI wired to server routes for search, paging, cloning, and deleting account documents in Kibana 9.0.1. The code is intentionally compact and consistent with the server-indexing approach used in earlier parts, giving you a solid base to extend the plugin with editing, richer filters, or advanced pagination later.

This article was published on the Search Guard blog: <https://search-guard.com/blog/writing-kibana-plugins-5/>
 Search Guard is Security and Alerting for Elasticsearch and Kibana — search-guard.com · [Start a free trial](#)