💻 Complete Code Explanation - Part 4

Complex Page Components (EmployeeList & EmployeeDetail)

---

14. src/pages/EmployeeList.tsx

Purpose: Employee list page with search, filter, sort, and pagination

This is one of the most complex components in the application. Let me break it down section by section.

```typescript
```

```
import { useState, useEffect, useMemo, ChangeEvent } from 'react';
import { useNavigate } from 'react-router-dom';
import {
  Container,
  Paper,
  Table,
  TableBody,
  TableCell,
  TableContainer,
  TableHead,
  TableRow,
  TablePagination,
  TableSortLabel,
  TextField,
  Button,
  Box,
  Typography,
  Checkbox,
  FormControlLabel,
  IconButton,
  Alert,
  CircularProgress,
} from '@mui/material';
import { Edit as EditIcon, Add as AddIcon } from '@mui/icons-material';
import Layout from '../components/Layout';
import { employeeAPI } from '../services/api';
import type { Employee } from '../types';

function EmployeeList() {
  // Employee data state
  const [employees, setEmployees] = useState<Employee[]>([]);
  const [loading, setLoading] = useState(true);
const [error, setError] = useState<string | null>(null);

// Search and filter state
const [searchKeyword, setSearchKeyword] = useState('');
const [showRetired, setShowRetired] = useState(false);

// Sort state
const [order, setOrder] = useState<'asc' | 'desc'>('asc');
const [orderBy, setOrderBy] = useState<keyof Employee>('EmployeeId');

// Pagination state
const [page, setPage] = useState(0);
const [rowsPerPage, setRowsPerPage] = useState(10);

const navigate = useNavigate();

// Load employees on component mount
useEffect(() => {
  loadEmployees();
}, []);

const loadEmployees = async () => {
  setLoading(true);
  setError(null);

  try {
    const response = await employeeAPI.getAll();
    setEmployees(response.data.employees);
  } catch (err: any) {
    setError('従業員データの読み込みに失敗しました');
    console.error(err);
  } finally {
      setLoading(false);
    }
  };

  // Filter and sort employees
  const filteredEmployees = useMemo(() => {
    let filtered = [...employees];

    // Filter by retire flag
    if (!showRetired) {
      filtered = filtered.filter(emp => !emp.RetireFlg);
    }

    // Filter by search keyword
    if (searchKeyword) {
      const keyword = searchKeyword.toLowerCase();
      filtered = filtered.filter(emp =>
        emp.EmployeeId.toLowerCase().includes(keyword) ||
        emp.Name.toLowerCase().includes(keyword) ||
        (emp.KanaName && emp.KanaName.toLowerCase().includes(keyword))
      );
    }

    // Sort
    filtered.sort((a, b) => {
      const aValue = a[orderBy];
      const bValue = b[orderBy];

      if (aValue === undefined || aValue === null) return 1;
      if (bValue === undefined || bValue === null) return -1;

      if (order === 'asc') {
```

```
      return aValue > bValue ? 1 : -1;
    } else {
      return aValue < bValue ? 1 : -1;
    }
  });

  return filtered;
}, [employees, showRetired, searchKeyword, order, orderBy]);

// Paginate filtered employees
const paginatedEmployees = useMemo(() => {
  const startIndex = page * rowsPerPage;
  const endIndex = startIndex + rowsPerPage;
  return filteredEmployees.slice(startIndex, endIndex);
}, [filteredEmployees, page, rowsPerPage]);

// Handle sort
const handleSort = (property: keyof Employee) => {
  const isAsc = orderBy === property && order === 'asc';
  setOrder(isAsc ? 'desc' : 'asc');
  setOrderBy(property);
};

// Handle page change
const handleChangePage = (_event: unknown, newPage: number) => {
  setPage(newPage);
};

// Handle rows per page change
const handleChangeRowsPerPage = (event: ChangeEvent<HTMLInputElement>) => {
  setRowsPerPage(parseInt(event.target.value, 10));
  setPage(0);
```

```
};

// Navigate to add employee
const handleAddEmployee = () => {
  navigate('/employees/new');
};

// Navigate to edit employee
const handleEditEmployee = (id: number) => {
  navigate(`/employees/${id}`);
};

return (
  <Layout>
    <Container maxWidth="lg">
      <Box sx={{ mb: 4 }}>
        <Typography variant="h4" component="h1" gutterBottom>
          従業員一覧
        </Typography>

        {/* Search and Filter Controls */}
        <Box sx={{ display: 'flex', gap: 2, mb: 2, flexWrap: 'wrap' }}>
          <TextField
            label="検索"
            placeholder="社員番号、氏名、カナ氏名"
            value={searchKeyword}
            onChange={(e) => setSearchKeyword(e.target.value)}
            sx={{ flexGrow: 1, minWidth: '200px' }}
          />

          <FormControlLabel
            control={
```

```
          <Checkbox
            checked={showRetired}
            onChange={(e) => setShowRetired(e.target.checked)}
          />
        }
        label="退職者を表示"
      />

      <Button
        variant="contained"
        startIcon={<AddIcon />}
        onClick={handleAddEmployee}
      >
        新規登録
      </Button>
    </Box>

    {error && (
      <Alert severity="error" sx={{ mb: 2 }}>
        {error}
      </Alert>
    )}

    {loading ? (
      <Box sx={{ display: 'flex', justifyContent: 'center', py: 4 }}>
        <CircularProgress />
      </Box>
    ) : (
      <Paper>
        <TableContainer>
          <Table>
            <TableHead>
    <TableRow>
     <TableCell>
      <TableSortLabel
       active={orderBy === 'EmployeeId'}
       direction={orderBy === 'EmployeeId' ? order : 'asc'}
       onClick={() => handleSort('EmployeeId')}
      >
       社員番号
      </TableSortLabel>
     </TableCell>
     <TableCell>
      <TableSortLabel
       active={orderBy === 'Name'}
       direction={orderBy === 'Name' ? order : 'asc'}
       onClick={() => handleSort('Name')}
      >
       氏名
      </TableSortLabel>
     </TableCell>
     <TableCell>性別</TableCell>
     <TableCell>電話番号</TableCell>
     <TableCell>部署</TableCell>
     <TableCell>状態</TableCell>
     <TableCell>操作</TableCell>
    </TableRow>
   </TableHead>
   <TableBody>
    {paginatedEmployees.length === 0 ? (
     <TableRow>
      <TableCell colSpan={7} align="center">
       従業員が見つかりません
      </TableCell>
```

```
          </TableRow>
         ) : (
          paginatedEmployees.map((employee) => (
           <TableRow key={employee.id} hover>
            <TableCell>{employee.EmployeeId}</TableCell>
            <TableCell>{employee.Name}</TableCell>
            <TableCell>
             {employee.Sex === 1 ? '男性' : '女性'}
            </TableCell>
            <TableCell>{employee.PhoneNumber || '-'}</TableCell>
            <TableCell>{employee.Department || '-'}</TableCell>
            <TableCell>
             {employee.RetireFlg ? '退職済み' : '在職中'}
            </TableCell>
            <TableCell>
             <IconButton
              size="small"
              color="primary"
              onClick={() => handleEditEmployee(employee.id!)}
             >
              <EditIcon />
             </IconButton>
            </TableCell>
           </TableRow>
          ))
         )}
        </TableBody>
       </Table>
      </TableContainer>

      <TablePagination
       component="div"
```

```
          count={filteredEmployees.length}
          page={page}
          onPageChange={handleChangePage}
          rowsPerPage={rowsPerPage}
          onRowsPerPageChange={handleChangeRowsPerPage}
          rowsPerPageOptions={[5, 10, 25, 50]}
          labelRowsPerPage="表示件数:"
          labelDisplayedRows={({ from, to, count }) =>
           `${from}-${to} / ${count}`
          }
         />
        </Paper>
       )}
      </Box>
     </Container>
    </Layout>
  );
}

export default EmployeeList;
```

Detailed Line-by-Line Explanation:

Line 1: `import { useState, useEffect, useMemo, ChangeEvent } from 'react';`

- **useState**: Manage component state
- **useEffect**: Side effects (API calls on mount)
- **useMemo**: Performance optimization (memoize expensive calculations)
- **ChangeEvent**: TypeScript type for input change events

Line 30-33: Employee Data State

```typescript
const [employees, setEmployees] = useState<Employee[]>([]);
const [loading, setLoading] = useState(true);
const [error, setError] = useState<string | null>(null);
```

- **employees**: Array of all employees from API
- **Employee[]**: TypeScript ensures array contains Employee objects
- **loading**: Show spinner while fetching data
- **error**: Store error messages

Line 35-37: Search & Filter State

```typescript
const [searchKeyword, setSearchKeyword] = useState('');
const [showRetired, setShowRetired] = useState(false);
```

- **searchKeyword**: User's search input
- **showRetired**: Toggle to show/hide retired employees
- Default: hide retired employees

Line 39-41: Sort State

```typescript
```

```
const [order, setOrder] = useState<'asc' | 'desc'>('asc');
const [orderBy, setOrderBy] = useState<keyof Employee>('EmployeeId');
```

- **order**: Sort direction (ascending or descending)
- **'asc' | 'desc'**: TypeScript union type (only these two values allowed)
- **orderBy**: Which column to sort by
- **keyof Employee**: TypeScript ensures orderBy is a valid Employee property
- Default: Sort by EmployeeId ascending

Line 43-45: Pagination State

```typescript
const [page, setPage] = useState(0);
const [rowsPerPage, setRowsPerPage] = useState(10);
```

- **page**: Current page number (0-indexed)
- **rowsPerPage**: How many rows to show per page
- Default: Page 0, 10 rows per page

Line 49-52: useEffect for Data Loading

```typescript
useEffect(() => {
  loadEmployees();
}, []);
```

- **useEffect**: Runs after component mounts
- **[]**: Empty dependency array = run only once
- Immediately loads employee data when page loads

Why empty dependency array?

```typescript
// Run once on mount:
useEffect(() => {
  loadData();
}, []);

// Run every time 'count' changes:
useEffect(() => {
  console.log(count);
}, [count]);

// Run on every render (bad!):
useEffect(() => {
  console.log('Render');
});
```

Line 54-66: loadEmployees Function

```typescript
```

```
const loadEmployees = async () => {
  setLoading(true);
  setError(null);

  try {
    const response = await employeeAPI.getAll();
    setEmployees(response.data.employees);
  } catch (err: any) {
    setError('従業員データの読み込みに失敗しました');
    console.error(err);
  } finally {
    setLoading(false);
  }
};
```

Flow:

1. Set loading = true (show spinner)
2. Clear any previous errors
3. Try to fetch employees from API
4. On success: Update employees state
5. On error: Set error message
6. Finally: Set loading = false (hide spinner)

Line 68-104: filteredEmployees useMemo

This is a **performance optimization**. Let me explain step by step:

```
typescript
```

```
const filteredEmployees = useMemo(() => {
  // Expensive calculation here
}, [dependencies]);
```

What is useMemo?

- Memoizes (caches) the result of a calculation
- Only recalculates when dependencies change
- Prevents unnecessary re-calculations on every render

Without useMemo (BAD):

typescript

```typescript
// This runs on EVERY render, even if data hasn't changed!
const filteredEmployees = employees
  .filter(...)
  .sort(...);

// User types in search box
// ↓ Component re-renders
// ↓ filteredEmployees recalculates (slow!)
```

With useMemo (GOOD):

typescript

```
const filteredEmployees = useMemo(() => {
  return employees.filter(...).sort(...);
}, [employees, searchKeyword, showRetired, order, orderBy]);

// Only recalculates if one of these changes:
// - employees
// - searchKeyword
// - showRetired
// - order
// - orderBy
```

Line 69-71: Copy Array

```typescript
let filtered = [...employees];
```

- **Spread operator (...)**: Creates a new array copy
- Why? Don't mutate original employees array
- **Immutability**: Important in React

Line 73-76: Filter by Retire Flag

```typescript
if (!showRetired) {
  filtered = filtered.filter(emp => !emp.RetireFlg);
}
```

- If checkbox unchecked: filter out retired employees
- **emp.RetireFlg**: true = retired, false = active
- **!emp.RetireFlg**: only keep active (false) employees

Line 78-87: Filter by Search Keyword

```typescript
if (searchKeyword) {
  const keyword = searchKeyword.toLowerCase();
  filtered = filtered.filter(emp =>
    emp.EmployeeId.toLowerCase().includes(keyword) ||
    emp.Name.toLowerCase().includes(keyword) ||
    (emp.KanaName && emp.KanaName.toLowerCase().includes(keyword))
  );
}
```

Breakdown:

1. Convert search keyword to lowercase (case-insensitive search)
2. Check if keyword matches:
   - Employee ID
   - Name
   - Kana Name (if exists)
3. Keep employee if ANY field matches

Why toLowerCase()?

```typescript
// Without toLowerCase():
"YAMADA".includes("yamada") // false ❌

// With toLowerCase():
"YAMADA".toLowerCase().includes("yamada") // true ✅
```

Why (emp.KanaName && ...)?

```typescript
// If KanaName is undefined:
emp.KanaName.toLowerCase() // Error! Cannot read property of undefined

// Safe check:
(emp.KanaName && emp.KanaName.toLowerCase())
// If KanaName undefined → short-circuit, returns false
// If KanaName exists → check if includes keyword
```

Line 89-101: Sorting

```typescript
```

```
filtered.sort((a, b) => {
  const aValue = a[orderBy];
  const bValue = b[orderBy];

  if (aValue === undefined || aValue === null) return 1;
  if (bValue === undefined || bValue === null) return -1;

  if (order === 'asc') {
    return aValue > bValue ? 1 : -1;
  } else {
    return aValue < bValue ? 1 : -1;
  }
});
```

Sort function explained:

- **Comparator function**: `(a, b) => number`
- Return **positive**: a comes after b
- Return **negative**: a comes before b
- Return **0**: keep order

Handle undefined/null:

```typescript
if (aValue === undefined) return 1;  // Push a to end
if (bValue === undefined) return -1; // Push b to end
```

Ascending sort:

```
typescript

if (order === 'asc') {
  return aValue > bValue ? 1 : -1;
}
// If aValue > bValue: return 1 (a comes after b)
// If aValue <= bValue: return -1 (a comes before b)
```

Example:

```
typescript

// Array: [3, 1, 2]
// Sorting (ascending):

compare(3, 1):
  3 > 1 → return 1 → [1, 3, 2]

compare(3, 2):
  3 > 2 → return 1 → [1, 2, 3]

// Result: [1, 2, 3] ✅
```

Line 106-110: Pagination with useMemo

```
typescript
```

```
const paginatedEmployees = useMemo(() => {
  const startIndex = page * rowsPerPage;
  const endIndex = startIndex + rowsPerPage;
  return filteredEmployees.slice(startIndex, endIndex);
}, [filteredEmployees, page, rowsPerPage]);
```

Pagination Math:

typescript

```typescript
// page = 0, rowsPerPage = 10:
startIndex = 0 * 10 = 0
endIndex = 0 + 10 = 10
slice(0, 10) → items 0-9 (first page)

// page = 1, rowsPerPage = 10:
startIndex = 1 * 10 = 10
endIndex = 10 + 10 = 20
slice(10, 20) → items 10-19 (second page)

// page = 2, rowsPerPage = 5:
startIndex = 2 * 5 = 10
endIndex = 10 + 5 = 15
slice(10, 15) → items 10-14 (third page, 5 per page)
```

Line 112-116: handleSort Function

typescript

```
const handleSort = (property: keyof Employee) => {
  const isAsc = orderBy === property && order === 'asc';
  setOrder(isAsc ? 'desc' : 'asc');
  setOrderBy(property);
};
```

Toggle Logic:

```
typescript
```

```
// Current: Sort by Name, ascending
// User clicks Name column header again

isAsc = (orderBy === 'Name') && (order === 'asc')
      = true && true
      = true

setOrder(true ? 'desc' : 'asc')
       = 'desc'

// Result: Sort by Name, descending (toggled!)

// User clicks EmployeeId header

isAsc = (orderBy === 'EmployeeId') && (order === 'asc')
      = false && true
      = false

setOrder(false ? 'desc' : 'asc')
       = 'asc'

setOrderBy('EmployeeId')

// Result: Sort by EmployeeId, ascending (new column, default asc)
```

Line 118-121: handleChangePage

```
typescript
```

```
const handleChangePage = (_event: unknown, newPage: number) => {
  setPage(newPage);
};
```

- **_event**: Underscore prefix = unused parameter (TypeScript convention)
- **newPage**: Material-UI provides the new page number
- Simply update page state

Line 123-127: handleChangeRowsPerPage

```typescript
const handleChangeRowsPerPage = (event: ChangeEvent<HTMLInputElement>) => {
  setRowsPerPage(parseInt(event.target.value, 10));
  setPage(0);
};
```

- **parseInt(event.target.value, 10)**: Convert string to number (base 10)
- **setPage(0)**: Reset to first page when changing rows per page
  - Otherwise might land on invalid page

Example:

```typescript
```

```
// Current: page 5, 10 rows per page
// Total: 100 items → 10 pages

// User changes to 50 rows per page
// Total: 100 items → 2 pages

// If we stay on page 5 → out of bounds!
// So we reset to page 0 ✅
```

Line 129-131: handleAddEmployee

```typescript
const handleAddEmployee = () => {
  navigate('/employees/new');
};
```

- Navigate to add employee page
- URL: `/employees/new`
- EmployeeDetail component will detect "new" and show add form

Line 133-136: handleEditEmployee

```typescript
const handleEditEmployee = (id: number) => {
  navigate(`/employees/${id}`);
};
```

- Navigate to edit employee page
- Example: `/employees/42`
- EmployeeDetail component will load employee with ID 42

Line 138-285: JSX Return (UI)

I'll break down the complex parts:

Line 152-160: Search TextField

```typescript
<TextField
  label="検索"
  placeholder="社員番号、氏名、カナ氏名"
  value={searchKeyword}
  onChange={(e) => setSearchKeyword(e.target.value)}
  sx={{ flexGrow: 1, minWidth: '200px' }}
/>
```

- **Controlled component**: value tied to state
- **onChange**: Updates state on every keystroke
- **flexGrow: 1**: Takes remaining space
- **minWidth: '200px'**: Minimum width on small screens

Real-time search:

```
User types 'Y'
 ↓
onChange fires
 ↓
setSearchKeyword('Y')
 ↓
Component re-renders
 ↓
filteredEmployees useMemo recalculates
 ↓
Table updates with filtered results
```

Line 162-170: Show Retired Checkbox

```typescript
<FormControlLabel
  control={
    <Checkbox
      checked={showRetired}
      onChange={(e) => setShowRetired(e.target.checked)}
    />
  }
  label="退職者を表示"
/>
```

- **checked={showRetired}**: Checkbox state tied to showRetired
- **e.target.checked**: Boolean value (true/false)
- **Toggling checkbox updates state → triggers re-render → table updates**

Line 172-178: Add Button

```typescript
<Button
  variant="contained"
  startIcon={<AddIcon />}
  onClick={handleAddEmployee}
>
  新規登録
</Button>
```

- **startIcon**: Icon before text
- **variant="contained"**: Filled button
- Clicking navigates to `/employees/new`

Line 186-190: Loading Spinner

```typescript
{loading ? (
  <Box sx={{ display: 'flex', justifyContent: 'center', py: 4 }}>
    <CircularProgress />
  </Box>
) : (
  // Table
)}
```

- **Conditional rendering**
- While loading: Show spinner

- After loading: Show table

Line 193-215: Table Header with Sort

```typescript
<TableCell>
  <TableSortLabel
    active={orderBy === 'EmployeeId'}
    direction={orderBy === 'EmployeeId' ? order : 'asc'}
    onClick={() => handleSort('EmployeeId')}
  >
    社員番号
  </TableSortLabel>
</TableCell>
```

- **TableSortLabel**: Clickable column header with sort arrow
- **active**: Highlight if currently sorted by this column
- **direction**: Show up/down arrow
- **onClick**: Toggle sort on click

Visual:

```
Initial:
社員番号↓ (ascending)

After click:
社員番号↑ (descending)

After another click:
社員番号↓ (ascending again)
```

Line 219-233: Empty State vs Data Rows

```typescript
{paginatedEmployees.length === 0 ? (
  <TableRow>
    <TableCell colSpan={7} align="center">
      従業員が見つかりません
    </TableCell>
  </TableRow>
) : (
  paginatedEmployees.map((employee) => (
    <TableRow key={employee.id} hover>
      {/* Table cells */}
    </TableRow>
  ))
)}
```

- **If no employees**: Show "No employees found" message
- **colSpan={7}**: Span across all 7 columns
- **Else**: Map over employees and render rows

Why key={employee.id}?

```typescript
// React needs unique key to identify elements
paginatedEmployees.map((employee) => (
  <TableRow key={employee.id}>
    {/* ... */}
  </TableRow>
))

// Helps React:
// 1. Efficiently update DOM
// 2. Preserve component state
// 3. Track which items changed
```

Line 241-243: Conditional Cell Rendering

```typescript
<TableCell>
  {employee.Sex === 1 ? '男性' : '女性'}
</TableCell>
```

- **Ternary operator**: condition ? true : false
- Sex = 1 → "男性" (Male)
- Sex = 2 → "女性" (Female)

```typescript
```

```
<TableCell>{employee.PhoneNumber || '-'}</TableCell>
```

- **Logical OR (||)**: If PhoneNumber empty/null → show '-'
- Handles optional fields gracefully

Line 252-259: Edit Icon Button

```typescript
<IconButton
  size="small"
  color="primary"
  onClick={() => handleEditEmployee(employee.id!)}
>
  <EditIcon />
</IconButton>
```

- **onClick arrow function**: Pass employee.id to handler
- **employee.id!**: Non-null assertion (TypeScript)
    - We know id exists here (came from database)

Line 264-277: Pagination Component

```typescript
```

```
<TablePagination
  component="div"
  count={filteredEmployees.length}
  page={page}
  onPageChange={handleChangePage}
  rowsPerPage={rowsPerPage}
  onRowsPerPageChange={handleChangeRowsPerPage}
  rowsPerPageOptions={[5, 10, 25, 50]}
  labelRowsPerPage="表示件数:"
  labelDisplayedRows={({ from, to, count }) =>
    `${from}-${to} / ${count}`
  }
/>
```

- **count**: Total number of items (filtered, not all)
- **page**: Current page (0-indexed internally, shown as 1-indexed to user)
- **rowsPerPageOptions**: Dropdown options
- **labelDisplayedRows**: Custom display text
  - Example: "1-10 / 45" (showing items 1-10 out of 45 total)

Complete Data Flow Visualization:

```
1. Component Mounts
   ↓
2. useEffect runs
   ↓
3. loadEmployees() calls API
   ↓
```

```
4. API returns employees
  ↓
5. setEmployees(data)
  ↓
6. Component re-renders
  ↓
7. filteredEmployees useMemo calculates:
  ├─ Filter by retire flag (showRetired)
  ├─ Filter by search keyword
  └─ Sort by orderBy + order
  ↓
8. paginatedEmployees useMemo calculates:
  └─ Slice array for current page
  ↓
9. Table renders paginatedEmployees
```

User Interaction Flows:

Search Flow:

```
User types "山田"
 ↓
setSearchKeyword("山田")
 ↓
Component re-renders
 ↓
filteredEmployees recalculates (useMemo)
  ├─ Checks each employee:
  │  ├─ EmployeeId.includes("山田")? No
  │  ├─ Name.includes("山田")? Yes! Keep
  │  └─ KanaName.includes("山田")? -
 ↓
paginatedEmployees recalculates
 ↓
Table shows only matching employees
```

Sort Flow:

```
User clicks "氏名" header
  ↓
handleSort('Name')
  ├─ Check: Currently sorting by Name + asc?
  │   ├─ Yes → Toggle to desc
  │   └─ No → Set to Name + asc
  ↓
setOrder() + setOrderBy()
  ↓
Component re-renders
  ↓
filteredEmployees recalculates with new sort
  ↓
Table shows sorted employees
```

Pagination Flow:

```
User clicks "Next Page" button
  ↓
handleChangePage(event, 1)
  ↓
setPage(1)
  ↓
Component re-renders
  ↓
paginatedEmployees recalculates:
  ├─ startIndex = 1 * 10 = 10
  ├─ endIndex = 10 + 10 = 20
  └─ slice(10, 20)
  ↓
Table shows items 10-19
```

This is a feature-rich, production-grade list component with search, filter, sort, and pagination - all working together seamlessly!

Would you like me to continue with the EmployeeDetail component (Add/Edit/Delete) next? That one is even more complex!