I'll explain the architectural flow for generating each type of report (PDF and Excel) in this system.

Overall Architecture

The system follows a **layered architecture** with clear separation of concerns:

```
Controller Layer  Service Layer  Repository Layer  Database

        Exporter Layer (PDF/Excel)
```

Report Generation Flow (Step-by-Step)

1. CLAIMS REPORT

Excel Generation Flow:

```
User clicks "Export to Excel" on Claims Report

GET /hr/reports/claims/export/excel

HrController.exportClaimsReportExcel()

1. Get filter parameters (status, sortBy)
2. Call ReportService.getClaimSummaryByEnrollment(status)

   ReportServiceImpl executes JPQL query:
   - FROM Enrollment e LEFT JOIN e.claims c
   - Filters by status if provided
   - Returns List<ClaimReportDto>

3. Sort the data (by date/status)
4. Call ClaimReportExcelExporter.export(data)

   ClaimReportExcelExporter:
```

```
   - Creates XSSFWorkbook (Excel file)
   - Creates sheet "Claims Summary"
   - Calls createHeaderStyle() from AbstractExcelExporter
   - Calls createCurrencyCellStyle() for amount column
   - Creates header row: ["Enrollment ID", "Claim ID", "Approved Amount", "Claim D
   - Loops through data and creates rows:
     * Cell 0: Enrollment ID
     * Cell 1: Claim ID (or "-")
     * Cell 2: Approved Amount (numeric with currency style)
     * Cell 3: Claim Date (formatted)
     * Cell 4: Status
   - Auto-sizes columns
   - Converts workbook to byte[]

5. Set HTTP response headers:
  - Content-Type: application/vnd.openxmlformats...
  - Content-Disposition: attachment; filename=claims-report.xlsx
6. Write bytes to response stream

Browser downloads claims-report.xlsx
```

PDF Generation Flow:

```
User clicks "Export to PDF" on Claims Report

GET /hr/reports/claims/export/pdf

HrController.exportClaimsReportPdf()

1. Get filter parameters (status, sortBy)
2. Call ReportService.getClaimSummaryByEnrollment(status)
  [Same query as Excel]

3. Sort the data
4. Call ClaimReportPdfExporter.export(data)

   ClaimReportPdfExporter:
   - Creates Document (A4 landscape)
```

- Creates ByteArrayOutputStream
- Creates PdfWriter
- Opens document
- Calls addTitle("Claims Summary by Enrollment")
- Creates PdfPTable with 5 columns
- Gets header font and color from AbstractPdfExporter
- Adds header cells with blue background
- Loops through data:
  * Adds cells with formatted data
  * Uses ReportFormatters for currency/date
- Adds table to document
- Closes document
- Returns byte[]

5. Set HTTP response headers:
   - Content-Type: application/pdf
   - Content-Disposition: attachment; filename=claims-report.pdf
6. Write bytes to response

Browser downloads claims-report.pdf

---

2. EMPLOYEE COVERAGE REPORT (Most Complex)

Excel Generation Flow:

User clicks "Export to Excel" on Employee Coverage Report

GET /hr/employees-report/export/excel

HrController.exportEmployeesReportExcel()

1. Parse filter parameters:
   - status (ACTIVE/NOTICE/EXITED/ALL)
   - category (JUNIOR/SENIOR/ALL)
   - enrollmentState (ENROLLED/NOT_ENROLLED/ALL)
   - sortBy, sortDir

2. Call HrReportService.getEmployeeCoverageReport()

HrReportService.getEmployeeCoverageReport():
a) Fetch all employees for organization:
employeeRepository.findByOrganizationOrganizationId(orgId)

b) Transform to DTOs with enrichment:
employees.stream()
.map(emp -> {
- resolveCategory(emp) calculates JUNIOR/SENIOR
- enrollmentRepository.countByEmployeeAndEnrollmentStatus() gets active
- EmployeeCoverageReportDTO.fromEmployee(emp, category, count)
})

c) Apply status filter (in-memory):
if status != "ALL": filter by matching status

d) Apply category filter (in-memory):
if category != "ALL": filter by matching category

e) Apply enrollment filter (in-memory):
if enrollmentState != "ALL": filter by isEnrolled()

f) Apply sorting (in-memory):
- getComparator(sortBy, sortDir)
- Supports: name, joiningDate, status, category
- Uses whitelisted fields only (security)

g) Pagination (skipped for export - uses pageSize=10000):
- Returns ALL filtered records

Returns EmployeeCoverageReportResult

3. Build filter description string:
buildFilterDescription() "Status: ACTIVE | Category: SENIOR | Sorted by: name (A

4. Call EmployeeCoverageExcelExporter.export(data, filters)

EmployeeCoverageExcelExporter:

- Creates workbook and sheet "Employee Coverage Report"
- Row 0: Filter info cell
- Row 1: Empty (spacing)
- Row 2: Headers with styling
  ["Employee Code", "Name", "Email", "Designation", "Joining Date",
  "Category", "Status", "Enrolled", "Active Policies"]
- Row 3+: Data rows
  * Loop through each DTO
  * Cell values with proper formatting
  * Category/Status use .name() to convert enum to string
  * Enrolled shows "Yes"/"No"
  * Active count as number
- Auto-size all 9 columns
- Convert to byte[]

5. Set response headers and write

Browser downloads employee-coverage-report.xlsx

PDF Generation Flow:

[Same initial flow as Excel until HrReportService]

Call EmployeeCoveragePdfExporter.export(data, filters)

EmployeeCoveragePdfExporter:
- Overrides getHeaderBackgroundColor() returns GREEN
- Creates Document (A4 landscape - needs width for 8 columns)
- Adds title "Employee Coverage Report"
- Adds filter info paragraph (italic, gray)
- Creates PdfPTable with 8 columns (no "Active Policies" column)
- Headers: ["Code", "Name", "Email", "Designation", "Joining Date",
  "Category", "Status", "Enrolled"]
- Creates header cells with GREEN background
- Loops through data:
  * Standard cells for most fields
  * STATUS cell: conditional coloring
    - ACTIVE light green background

```
      - EXITED    light red background
      - NOTICE    light yellow background
    * ENROLLED cell: shows "Yes (2)" or "No"
      - Light blue background if enrolled
      - White if not enrolled
  - Adds table to document
  - Adds footer: "Total Employees: X"
  - Closes and returns byte[]

Set response and download employee-coverage-report.pdf
```

---

3. EMPLOYEE COUNT REPORT (Simple)

Excel Generation Flow:

```
GET /hr/reports/employees/excel

HrController.exportEmployeesReportExcel()

1. Get orgId from current user
2. Call ReportService.getEmployeeCountByOrganization(orgId)

   ReportServiceImpl:
   - JPQL query:
     SELECT new EmployeeReportDto(o.organizationId, o.organizationName, COUNT
     FROM Organization o LEFT JOIN o.employee e
     WHERE :orgId IS NULL OR o.organizationId = :orgId
     GROUP BY o.organizationId, o.organizationName
   - Returns List<EmployeeReportDto>

3. Create EmployeeReportExcelExporter instance
4. Call exporter.export(data)

   EmployeeReportExcelExporter:
   - Creates workbook, sheet "Employees"
   - Headers: ["Org ID", "Org Name", "Employee Count"]
   - Simple data rows (3 columns)
```

- Auto-size columns
- Return byte[]

Download employees-report.xlsx

PDF Generation Flow:

[Same until exporter]

EmployeeReportPdfExporter.export(data)

- Document in PORTRAIT (only 3 columns)
- Title "Employee Report"
- PdfPTable with 3 columns
- BLUE header background (default)
- Simple data cells (converts Long to String)
- Return byte[]

Download employees-report.pdf

---

4. PREMIUM COLLECTION REPORT

Excel Generation Flow:

GET /hr/reports/premium/export/excel

1. Call ReportService.getPremiumCollectedByOrganization(orgId)

   ReportServiceImpl:
   - JPQL query:
     SELECT new PremiumReportDto(o.organizationId, o.organizationName,
                                 COALESCE(SUM(e.premiumAmount), 0))
     FROM Organization o
     LEFT JOIN o.employee emp
     LEFT JOIN emp.enrollments e
     WHERE :orgId IS NULL OR o.organizationId = :orgId

```
    GROUP BY o.organizationId, o.organizationName
  - Returns List<PremiumReportDto>

2. PremiumReportExcelExporter.export(data)

  - Sheet "Premium Report"
  - Headers: ["Organization ID", "Organization Name", "Total Premium Collected"]
  - Premium column uses currency style (numeric with    format)
  - Auto-size
  - Return byte[]

Download premium-report.xlsx
```

PDF Generation Flow:

```
[Same until exporter]

PremiumReportPdfExporter.export(data)

  - Overrides getHeaderBackgroundColor()    returns YELLOW
  - PORTRAIT document (3 columns)
  - Title "Premium Collected by Organization"
  - PdfPTable with 3 columns
  - YELLOW header background
  - Premium formatted with    symbol (string, not numeric)
  - Return byte[]

Download premium-report.pdf
```

Key Architectural Patterns

1. Template Method Pattern

- `AbstractExcelExporter` and `AbstractPdfExporter` provide common methods
- Concrete exporters override specific behaviors (like header colors)

2. Strategy Pattern

- Different exporters for different report types
- Same interface: `export(data)` → `byte[]`

3. Data Flow Separation

```
Data Retrieval (Repository/Service)

Data Transformation (DTOs, Filtering, Sorting)

Data Presentation (Exporters)

Data Delivery (HTTP Response)
```

4. Formatting Utilities

- `ReportFormatters` provides centralized formatting:
  - `formatDate()` → "dd-MM-yyyy"
  - `formatCurrency()` → " 1,234.56" (for PDF)
  - `formatCurrencyRaw()` → 1234.56 (for Excel)

5. Two-Phase Filtering

Employee Coverage Report uses hybrid approach:

1. **Database fetch** (Repository) → Get all org employees
2. **In-memory filtering** (Service) → Filter by status/category/enrollment
3. **In-memory sorting** (Service) → Sort by whitelisted fields
4. **In-memory pagination** (Service) → Slice for display (but export skips this)

Why? Complex business logic (category calculation, enrollment counting) is easier in Java than SQL.

Security Considerations

1. **Sort Field Whitelisting**

```
private static final Map<String, String> SORT_FIELD_MAP = Map.of(
    "name", "employeeName",
    "joiningDate", "joiningDate",
    ...
);
```

Prevents SQL injection via sort parameters

2. **Organization Scoping**
    - All queries filter by current user's organization
    - HR users only see their own org's data
3. **Input Validation**
    - Page size capped at 300
    - Enum validation with try-catch fallback
    - Optional parameters with safe defaults

This architecture provides **clean separation**, **reusability**, and **maintainability** while generating professional reports in both formats!