To optimize the process and reduce the number of API calls from the frontend, moving the logic to the backend is a good approach. This will significantly reduce the overhead caused by multiple API calls from the frontend and improve performance. Here's a revised approach:

1. \*\*Single API Call\*\*: Create a single API endpoint in your Spring Boot backend that handles the entire process: fetching records, checking user roles, fetching user-specific info, and generating the Excel report.

2. \*\*Parallel Processing\*\*: Use parallel processing or asynchronous calls in the backend to handle multiple records simultaneously, which can speed up the processing.

3. \*\*Batch Processing\*\*: Instead of calling the user-specific info API for each record individually, consider using batch processing if the API supports it.

### Step-by-Step Implementation

#### Backend (Spring Boot)

1. \*\*Create a new API Endpoint\*\*:

```java

@RestController

@RequestMapping("/api")

public class ReportController {

@Autowired

private RecordService recordService;

@PostMapping("/download-report")

public ResponseEntity<byte[]> downloadReport(@RequestBody FilterCriteria criteria) {

byte[] report = recordService.generateReport(criteria);

return ResponseEntity.ok()

.header(HttpHeaders.CONTENT\_DISPOSITION, "attachment; filename=report.xlsx")

.contentType(MediaType.APPLICATION\_OCTET\_STREAM)

.body(report);

}

}

```

2. \*\*Service Layer\*\*:

```java

@Service

public class RecordService {

@Autowired

private RecordRepository recordRepository;

@Autowired

private UserService userService;

@Autowired

private ExcelService excelService;

public byte[] generateReport(FilterCriteria criteria) {

List<Record> records = recordRepository.findByCriteria(criteria);

List<EnhancedRecord> enhancedRecords = records.parallelStream()

.map(this::enhanceRecord)

.filter(Optional::isPresent)

.map(Optional::get)

.collect(Collectors.toList());

return excelService.createExcelReport(enhancedRecords);

}

private Optional<EnhancedRecord> enhanceRecord(Record record) {

if (userService.hasRole(record.getUserId(), "PARTICULAR\_ROLE")) {

return Optional.empty();

}

UserSpecificInfo info = userService.getUserSpecificInfo(record.getUserId());

return Optional.of(new EnhancedRecord(record, info));

}

}

```

3. \*\*UserService\*\* (handles the user-related API calls):

```java

@Service

public class UserService {

@Autowired

private RestTemplate restTemplate;

public boolean hasRole(String userId, String role) {

// Call the API to check user role

String url = "http://user-service/api/check-role";

HttpHeaders headers = new HttpHeaders();

headers.setContentType(MediaType.APPLICATION\_JSON);

HttpEntity<String> entity = new HttpEntity<>(userId, headers);

ResponseEntity<Boolean> response = restTemplate.exchange(url, HttpMethod.POST, entity, Boolean.class);

return response.getBody();

}

public UserSpecificInfo getUserSpecificInfo(String userId) {

// Call the API to get user-specific info

String url = "http://user-service/api/user-info/" + userId;

ResponseEntity<UserSpecificInfo> response = restTemplate.getForEntity(url, UserSpecificInfo.class);

return response.getBody();

}

}

```

4. \*\*ExcelService\*\* (handles the creation of the Excel report):

```java

@Service

public class ExcelService {

public byte[] createExcelReport(List<EnhancedRecord> records) {

// Logic to create Excel report from records

// Use Apache POI or any other library to create the Excel file

// Return the byte array of the generated Excel file

}

}

```

#### Frontend (React)

1. \*\*API Call to Download Report\*\*:

```javascript

import axios from 'axios';

const downloadReport = async (filterCriteria) => {

try {

const response = await axios.post('/api/download-report', filterCriteria, {

responseType: 'blob', // Important for handling binary data

});

const url = window.URL.createObjectURL(new Blob([response.data]));

const link = document.createElement('a');

link.href = url;

link.setAttribute('download', 'report.xlsx');

document.body.appendChild(link);

link.click();

} catch (error) {

console.error('Error downloading report:', error);

}

};

// Usage in your component

const handleDownload = () => {

const filterCriteria = {

// Your filter criteria here

};

downloadReport(filterCriteria);

};

```

This approach consolidates the logic into the backend, reducing the number of API calls from the frontend and improving the overall performance and user experience.