Quy trình refractor code và tối ưu thời gian xử lý API

Tài liệu này mô tả quy trình, phương pháp và các nguyên tắc kỹ thuật nhằm thực hiện **refactor code và tối ưu hiệu năng xử lý của các API trong hệ thống backend**. Mục tiêu của tài liệu là cung cấp một hướng dẫn chuẩn để các nhóm phát triển phần mềm có thể cải thiện cấu trúc mã nguồn, giảm độ phức tạp của logic xử lý và nâng cao hiệu năng phản hồi của hệ thống API mà không làm thay đổi hành vi nghiệp vụ hiện tại.

Ngoài việc hướng dẫn các bước thực hiện refactor, tài liệu còn đề cập đến quy trình kiểm thử, đánh giá hiệu năng trước và sau khi tối ưu, cũng như các nguyên tắc đảm bảo tính ổn định của hệ thống trong quá trình triển khai thay đổi.

Tài liệu được sử dụng làm **chuẩn tham chiếu kỹ thuật trong quá trình bảo trì, nâng cấp và tối ưu hệ thống backend**, giúp các lập trình viên và nhóm phát triển có một phương pháp thống nhất khi thực hiện refactor và cải thiện hiệu năng API trong các dự án phần mềm.

- Chương 1: Đặt vấn đề
  - 1.1. Bối cảnh hệ thống
  - 1.2. Mục tiêu của việc refactor và tối ưu API
- Chương 2: Nguyên tắc refactor code
  - 2.1. Không thay đổi hành vi nghiệp vụ của hệ thống
  - 2.2. Thực hiện refactor theo từng bước nhỏ
  - 2.3. Đảm bảo khả năng kiểm thử sau khi refactor
- Chương 3: Triển khai

- 3.1. Triển khai refractor code
- 3.2. Cấu trúc promt và ví dụ

Chương 1: Đặt vấn đề

1.1. Bối cảnh hệ thống

Trong quá trình phát triển phần mềm, hệ thống API thường trải qua nhiều giai đoạn mở rộng chức năng, sửa lỗi và bổ sung nghiệp vụ. Việc thay đổi liên tục có thể dẫn đến:

- Logic xử lý trở nên phức tạp và khó bảo trì

- Code trùng lặp giữa nhiều module

- Query database không tối ưu

- Thời gian phản hồi API tăng dần theo thời gian

Các vấn đề này ảnh hưởng trực tiếp đến:

- Hiệu năng hệ thống

- Khả năng mở rộng

- Trải nghiệm người dùng

- Chi phí vận hành hệ thống

1.2. Mục tiêu của việc refactor và tối ưu API

Cải thiện hiệu năng

- Giảm thời gian phản hồi API

- Giảm tải cho database

Tăng khả năng bảo trì

- Tách rõ logic nghiệp vụ

- Giảm code trùng lặp

- Chuẩn hóa cấu trúc code

Tăng khả năng mở rộng hệ thống

- Dễ bổ sung chức năng mới

- Hạn chế ảnh hưởng dây chuyền khi thay đổi code

Đảm bảo tính ổn định

- Giữ nguyên behavior của API sau khi refactor

- Kiểm soát lỗi thông qua test và logging

Chương 2: Nguyên tắc refactor code

Refactor code là quá trình tái cấu trúc lại mã nguồn nhằm cải thiện chất lượng hệ thống, tăng khả năng bảo trì và tối ưu hiệu năng mà **không làm thay đổi hành vi nghiệp vụ của hệ thống**. Để đảm bảo việc refactor diễn ra an toàn, hiệu quả và không làm phát sinh lỗi mới, cần tuân thủ một số nguyên tắc cơ bản trong quá trình thực hiện.

2.1. Không thay đổi hành vi nghiệp vụ của hệ thống

Nguyên tắc quan trọng nhất của refactor là **không làm thay đổi logic nghiệp vụ hoặc kết quả xử lý của hệ thống**. Mọi thay đổi trong quá trình refactor chỉ nhằm cải thiện cấu trúc code, giảm độ phức tạp và tăng hiệu năng.

Trước và sau khi refactor, API phải đảm bảo:

- Trả về cùng một cấu trúc dữ liệu

- Giữ nguyên logic xử lý nghiệp vụ

- Không làm thay đổi dữ liệu hệ thống

2.2. Thực hiện refactor theo từng bước nhỏ

Refactor nên được thực hiện theo **các thay đổi nhỏ, có kiểm soát**, thay vì thay đổi lớn trong một lần. Điều này giúp dễ dàng kiểm tra và phát hiện lỗi.

Quy trình thường bao gồm:

1. Xác định vấn đề trong code
2. Refactor một phần nhỏ của hệ thống
3. Kiểm thử lại chức năng
4. Đánh giá hiệu năng
5. Tiếp tục refactor các phần tiếp theo

Cách tiếp cận này giúp giảm rủi ro và đảm bảo hệ thống luôn hoạt động ổn định trong quá trình cải tiến.

2.3. Đảm bảo khả năng kiểm thử sau khi refactor

Sau mỗi lần refactor, cần thực hiện kiểm thử để đảm bảo hệ thống vẫn hoạt động đúng như trước. Việc kiểm thử có thể bao gồm:

- Kiểm thử chức năng API
- So sánh dữ liệu trả về trước và sau refactor
- Kiểm tra thời gian xử lý API
- Kiểm tra log hệ thống

Kiểm thử giúp phát hiện sớm các lỗi có thể phát sinh trong quá trình tái cấu trúc code.

Chương 3: Triển khai

3.1. Triển khai refractor code

1. Phân tích logic xử lý của hàm

Ở bước đầu tiên, cần yêu cầu công cụ AI hoặc lập trình viên thực hiện phân tích chi tiết logic xử lý của hàm cần tối ưu. Mục tiêu của bước này là:

- Hiểu rõ toàn bộ luồng xử lý của hàm

- Xác định các bước xử lý chính

- Phân tách các giai đoạn xử lý logic thành các phần độc lập

2. Đánh giá kết quả phân tích

Sau khi AI hoặc lập trình viên đưa ra kết quả phân tích, cần thực hiện bước đánh giá để xác nhận tính hợp lý của việc phân tách logic.

Việc đánh giá cần tập trung vào các yếu tố sau:

- Các phase có phản ánh đúng luồng xử lý của hệ thống hay không

- Mỗi phase có một trách nhiệm rõ ràng hay không

- Có phase nào bị trùng lặp logic hay không

- Có phase nào quá lớn hoặc quá phức tạp hay không

Nếu kết quả phân tích được đánh giá là hợp lý và phù hợp với kiến trúc hệ thống, quy trình sẽ chuyển sang bước refactor mã nguồn.

3. Refactor mã nguồn theo các phase xử lý

Ở bước này, tiến hành tái cấu trúc mã nguồn dựa trên các phase đã được xác định ở bước phân tích.

Quy trình thực hiện gồm:

- Tạo các hàm riêng biệt tương ứng với từng phase xử lý đã được xác định trước đó.

- Di chuyển logic xử lý tương ứng từ hàm gốc sang các hàm mới được tạo.

- Đảm bảo mỗi hàm chỉ thực hiện một nhiệm vụ cụ thể trong quy trình xử lý.

- Bổ sung cơ chế ghi log thời gian xử lý của từng hàm để phục vụ việc đánh giá hiệu năng sau này.

4. Kiểm thử chức năng sau khi refactor

Sau khi hoàn thành việc refactor, cần thực hiện kiểm thử lại toàn bộ chức năng để đảm bảo:

- Logic xử lý không bị thay đổi

- Kết quả trả về của API vẫn giống với phiên bản trước khi refactor

- Không phát sinh lỗi nghiệp vụ hoặc lỗi hệ thống

5. Đánh giá thời gian xử lý của từng hàm

Sau khi hệ thống hoạt động ổn định với cấu trúc code mới, cần tiến hành đo lường và đánh giá thời gian xử lý của từng hàm đã được tách ra.

Thông tin thời gian xử lý được thu thập thông qua các log đã được bổ sung ở bước refactor.

Mục tiêu của bước này là:

- Xác định các hàm có thời gian xử lý lớn

- Xác định các điểm nghẽn hiệu năng trong hệ thống

- Ưu tiên các phần cần tối ưu trước

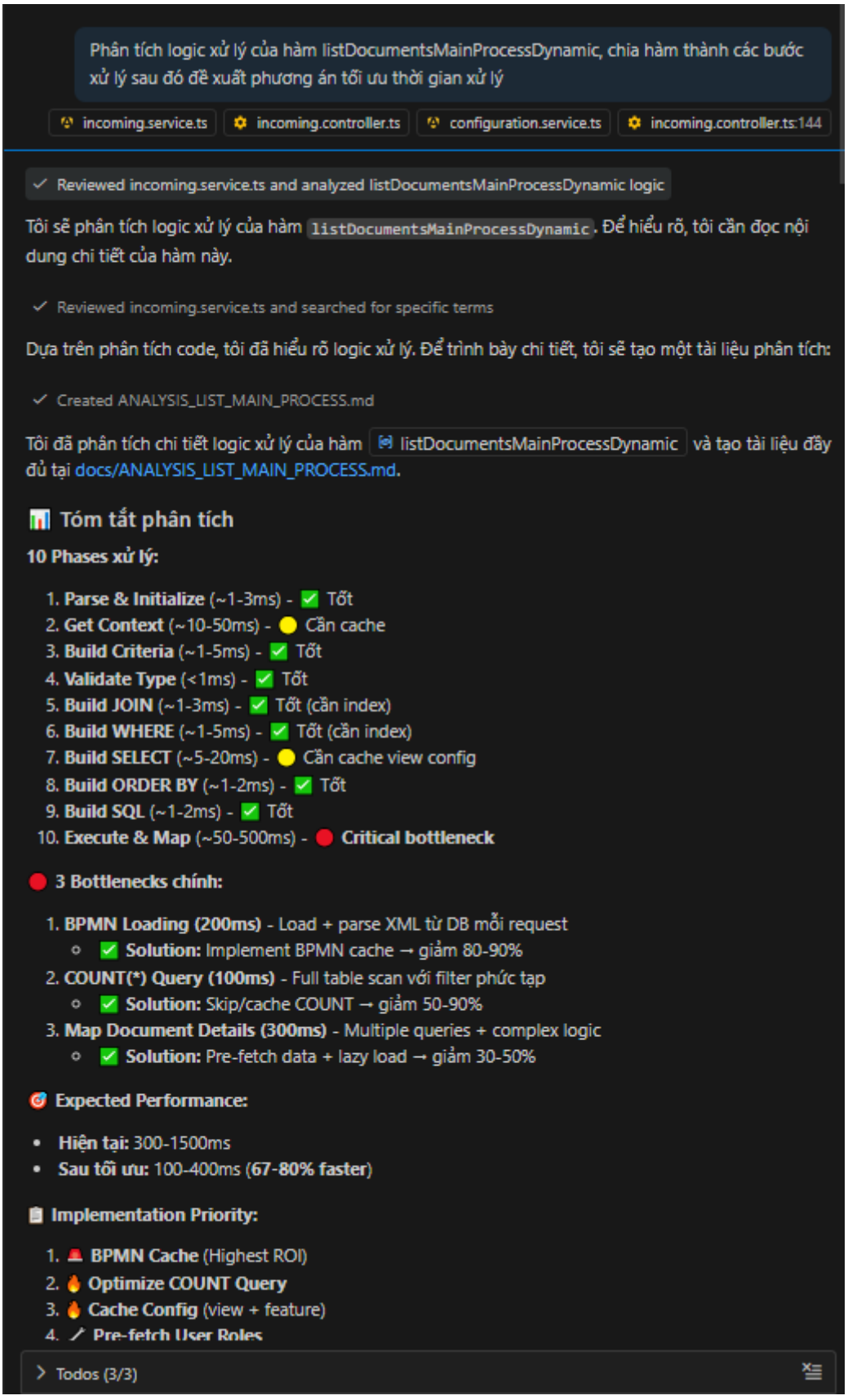
6. Lặp lại quy trình phân tích đối với các hàm có thời gian xử lý lớn

Đối với các hàm được xác định là chiếm nhiều thời gian xử lý, quy trình phân tích và refactor sẽ được áp dụng lại ở cấp độ chi tiết hơn.

3.2. Cấu trúc promt và ví dụ

3.2.1. Các bước thực hiện

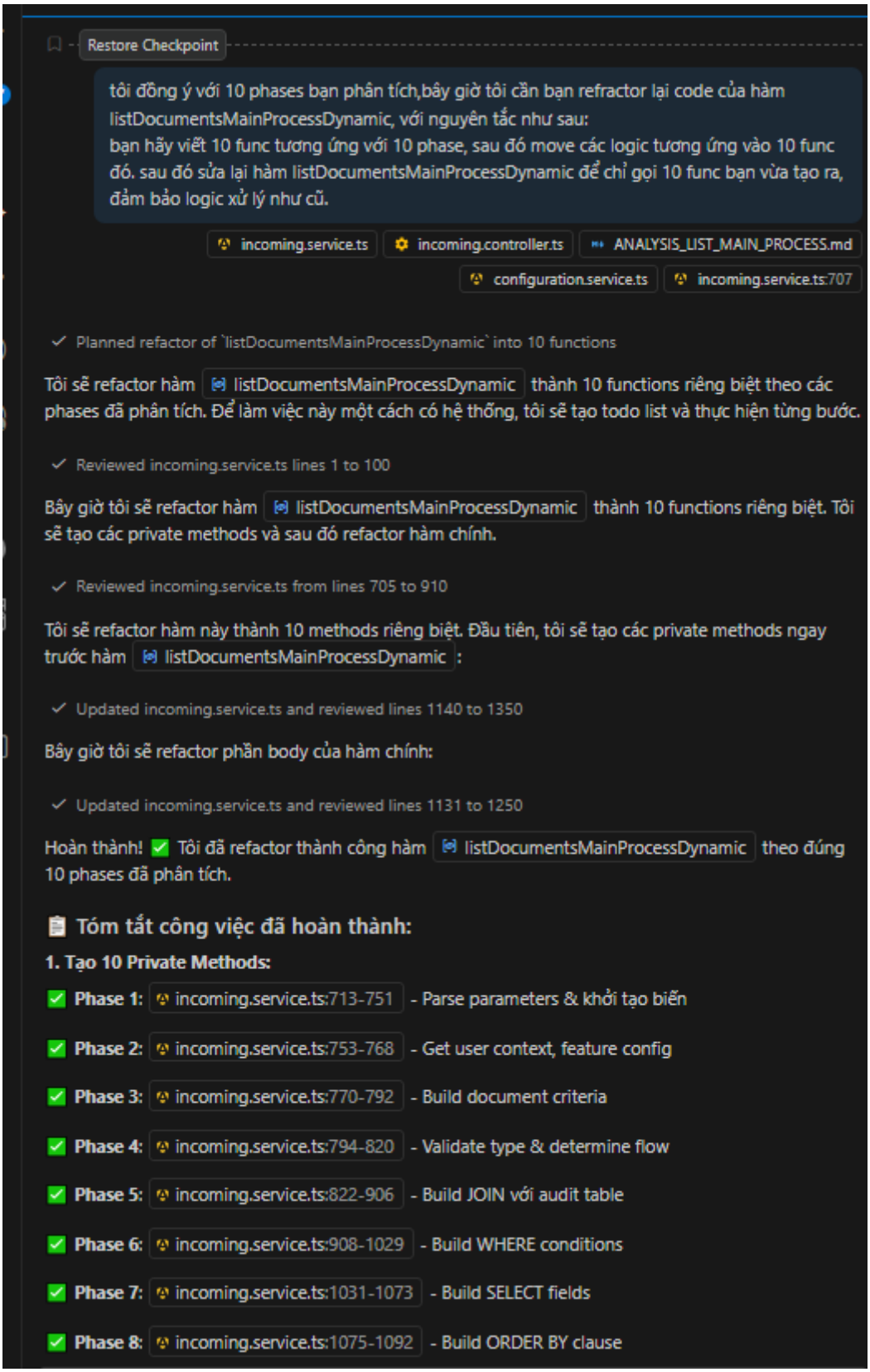
Bước 1: Yêu cầu AI thực hiện yêu cầu sau:
"Phân tích logic xử lý của hàm listDocumentsMainProcessDynamic, chia hàm thành các bước xử lý sau đó đề xuất phương án tối ưu thời gian xử lý"



Bước 2: Đánh giá kết quả phân tích của AI, nếu hợp lý chuyển sang bước 3

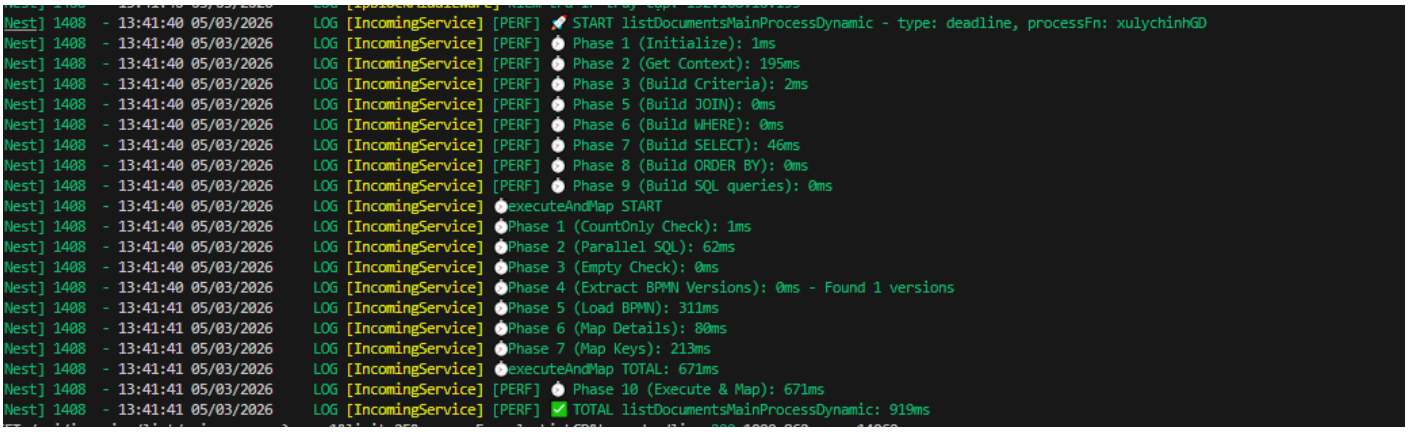
Bước 3: Yêu cầu AI thực hiện yêu cầu sau:
"tôi đồng ý với 10 phases bạn phân tích,bây giờ tôi cần bạn refractor lại code của hàm listDocumentsMainProcessDynamic, với nguyên tắc như sau:

bạn hãy viết 10 func tương ứng với 10 phase, sau đó move các logic tương ứng vào 10 func đó. sau đó sửa lại hàm listDocumentsMainProcessDynamic để chỉ gọi 10 func bạn vừa tạo ra, đảm bảo logic xử lý như cũ và bổ xung các log time xử lý của từng func ở trong chính các func vừa tạo"

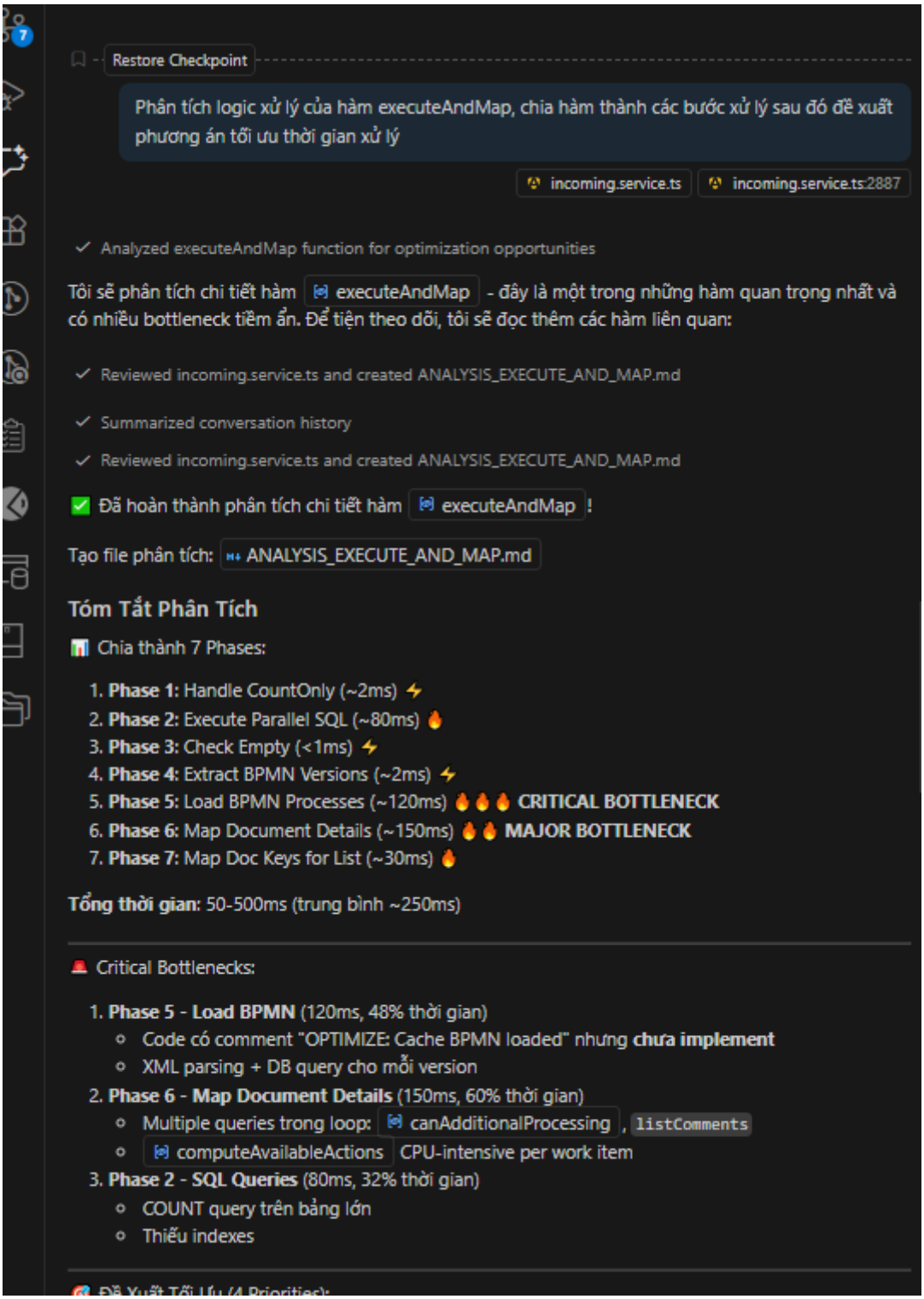


Bước 4: Test lại chức năng sau khi refractor

Bước 5: Đánh gia thời gian xử lý của từng func, ưu tiên các func chiếm tỉ trọng thời gian xử lý lớn



Bước 6: Thực hiện lại từ bước 1 đến bước 5 đối với từng func mà bước 5 chọn được



Bước 7: Thực hiện tối ưu từng func sau khi đã refractor

3.2.2. Kết quả

Trước khi refractor

```ts
// Xử lý chính
async listDocumentsMainProcessDynamic(query: ListDocumentsDto, userId: string, authorId?: string) {      tthieu, 2 months ago • update danh sach văn bản đến ủy quyền _
  const { type, page = 1, limit = 20, filter, sort, processFn, authority, room, isExport, countOnly } = query;
  let limitNum = Math.min(Number(limit) || 20, 100);
  if (isExport === 'true') {
    limitNum = 9999;
  }
  const pageNum = Math.max(Number(page) || 1, 1);
  const offsetNum = (pageNum - 1) * limitNum;
  const pool = await this.getPool();
  const tab = 'processor';
  const where: string[] = [];
  const urgencyValue = 'khn';
  const isRoomFilter = room === 'true' ? 'ONLY_ROOM' : room === 'false' ? 'ONLY_PERSONAL' : 'ALL';
  const roomCondition = isRoomFilter === 'ONLY_ROOM' ? `(af.is_transfer_to_room = 1)` : isRoomFilter === 'ONLY_PERSONAL' ? `(af.is_transfer_to_room = 0 OR af.is_transfer_to_room IS NULL)` : '';

  // Xử lý ủy quyền
  if (authority === 'true' && authorId) {
    userId = authorId;
  }

  const { userContext, featureManagement, receiverUnit } = await this.getListContext(userId, processFn, pool);
  const criteria = this.buildCriteria(filter);

  const featureCriteria = featureManagement?.criteria ?? [];
  const { sql: filterFeature, joins: filterJoins } = buildDocumentCriteriaHelper([...featureCriteria, ...criteria], 'incomming_documents', featureManagement);


  const USER_FLOW_TYPES = ['urgent', 'deadline', 'other', 'processed', 'notComplete', 'waitSign'] as const;
  const DOC_FLOW_TYPES = ['incompleted', 'completed'] as const;
  const ALLOWED_TYPES = [...USER_FLOW_TYPES, ...DOC_FLOW_TYPES];

  if (!type || !ALLOWED_TYPES.includes(type as any)) {
    throw new BadRequestException({
      message: 'Type không hợp lệ',
      allowedTypes: ALLOWED_TYPES,
    });
  }
  type UserFlowType = typeof USER_FLOW_TYPES[number];
  type DocFlowType = typeof DOC_FLOW_TYPES[number];
  const isUserFlow = USER_FLOW_TYPES.includes(type as UserFlowType);
  const isDocFlow = DOC_FLOW_TYPES.includes(type as DocFlowType);
  const safeType = type ?? 'deadline';

  const orderChuaXuLy = `CASE WHEN a.stage_status = '${stageStatusDoc.CHUA_XU_LY}' THEN 1 ELSE 99 END`;
  const auditOrderByByType: Record<string, string> = {
    urgent: orderChuaXuLy,
    deadline: orderChuaXuLy,
    other: orderChuaXuLy,
    processed: `CASE WHEN a.stage_status = '${stageStatusDoc.DA_XU_LY}' THEN 1 WHEN a.stage_status = '${stageStatusDoc.DA_PHAN_CONG}' THEN 2 WHEN a.stage_status = '${stageStatusDoc.HOAN_THANH}' THEN 3 ELSE 99 END`,
    incompleted: `CASE WHEN a.stage_status = '${stageStatusDoc.HOAN_THANH}' THEN 1 ELSE 99 END`,
    completed: `CASE WHEN a.stage_status = '${stageStatusDoc.HOAN_THANH_VAN_BAN}' THEN 1 ELSE 99 END`,
  };

  const auditOrderBy = auditOrderByByType[safeType] ?? ` CASE WHEN a.id IS NOT NULL THEN 1 ELSE 99 END `;

  // Tách JOIN clause theo flow type
```

```ts
  let joinClause = '';

  // Xử lý việc chuyển văn bản sang phòng
  joinClause += ` OUTER APPLY (
    SELECT TOP 1
      CASE
        WHEN a_first.details LIKE '%"transferType":"to_room"%'
          OR a_first.details LIKE '%\\"transferType\\":\\"to_room\\"%'
        THEN 1 ELSE 0
      END AS is_transfer_to_room
    FROM ${this.dbname}.dbo.audit a_first
    WHERE a_first.document_id = incomming_documents.document_id
    ORDER BY a_first.id ASC
  ) af`;

  if (isUserFlow) {
    // Lấy bản ghi theo cá nhân hoặc phòng
    joinClause += ` OUTER APPLY (
      SELECT TOP 1 a.document_id, a.receiver, a.receiver_unit, a.roleProcess, a.stage_status, a.action_code, a.deadline AS user_deadline
      FROM ${this.dbname}.dbo.audit a
      WHERE a.document_id = incomming_documents.document_id
      AND a.roleProcess = '${tab}'
      AND (a.receiver = '${userId}')
      ORDER BY ${auditOrderBy}, a.created_at DESC
    ) au`;
  }

  // Join audit bản ghi mới nhất theo người nhận / phòng nhận
  if (isDocFlow) {
    joinClause += ` OUTER APPLY (
      SELECT TOP 1 a.id AS doc_audit_id, a.stage_status AS doc_stage_status, a.deadline AS user_deadline
      FROM ${this.dbname}.dbo.audit a
      WHERE a.document_id = incomming_documents.document_id
      ORDER BY ${auditOrderBy}, a.created_at DESC
    ) ad`;
  }
  // Văn bản có sao
  if (filter?.isStar === '1' || filter?.isStar === 'true') {
    where.push(` EXISTS ( SELECT 1 FROM document_star ds WHERE ds.document_id = incomming_documents.document_id AND ds.user_id = '${userId}' AND ds.step = '${processFn}' ) `);
  }
  if (filter?.isStar === '0' || filter?.isStar === 'false') {
    where.push(` NOT EXISTS ( SELECT 1 FROM document_star ds WHERE ds.document_id = incomming_documents.document_id AND ds.user_id = '${userId}' AND ds.step = '${processFn}' ) `);
  }

  if (filterJoins) joinClause += ' ' + filterJoins;
  // Loại bỏ văn bản đã hoàn thành
  const excludeCompletedDoc = `
    NOT EXISTS ( SELECT TOP 1 1 FROM ${this.dbname}.dbo.audit a_end WHERE a_end.document_id = incomming_documents.document_id AND a_end.stage_status = '${stageStatusDoc.HOAN_THANH_VAN_BAN}' ) `;

  const typeFilters: Record<string, string[]> = {
    urgent: [
      filterFeature ? `(${filterFeature})` : undefined,
      `incomming_documents.urgency_level IN ('${urgencyValue}')`,
      `( au.stage_status='${stageStatusDoc.CHUA_XU_LY}' )`,
      roomCondition
    ].filter((f): f is string => !!f),
    deadline: [
```

```
  const typeFilters: Record<string, string[]> = {
    ].filter((f): f is string => !!f),
    notComplete: [
      filterFeature ? `(${filterFeature})` : undefined,

      // User hiện tại chưa xử lý
      `( au.stage_status = '${stageStatusDoc.CHUA_XU_LY}' )`,

      // Văn bản đã từng có bước HOAN_THANH_VAN_BAN ở bất kỳ audit nào
      `EXISTS (
        SELECT 1
          FROM ${this.dbname}.dbo.audit ax
        WHERE ax.document_id = incomming_documents.document_id
        AND ax.stage_status = '${stageStatusDoc.HOAN_THANH_VAN_BAN}'
)`,

      roomCondition
    ].filter((f): f is string => !!f),

  };

  if (type && typeFilters[type]) where.push(...typeFilters[type]);

  let whereClause = '';
  if (where.length) {
    whereClause = ' WHERE ' + where.join(' AND ') + ' AND incomming_documents.status = 1';
  } else {
    whereClause = ' WHERE incomming_documents.status = 1';
  }

  const excludeKeys = ['files', 'statusCode', 'status_code', 'userDeadline', 'user_deadline'];
  const { dbKeys, aliases, allViewFields } = await this.configurationService.buildSelectFieldsNew('incomming_documents', excludeKeys, processFn);
  const keyDefaultParts: string[] = [];
  if (allViewFields.includes('userDeadline') || allViewFields.includes('user_deadline')) {
    aliases['userDeadline'] = 'user_deadline';
    aliases['userDeadline'] = 'userDeadline';
  }
  if (allViewFields.includes('statusCode') || allViewFields.includes('status_code')) keyDefaultParts.push(`( SELECT TOP 1 a.action_code FROM ${this.dbname}.dbo.audit a WHERE a.document_id
  if (allViewFields.includes('processors')) keyDefaultParts.push(`(SELECT TOP 1 a.receiver FROM ${this.dbname}.dbo.audit a WHERE a.document_id = incomming_documents.document_id AND a.stage
  keyDefaultParts.push(`CASE WHEN EXISTS ( SELECT 1 FROM document_star ds WHERE ds.document_id = incomming_documents.document_id AND ds.user_id = '${userId}' AND ds.step = '${processFn}')
  aliases['isStar'] = 'is_star';

  const keyDefault = keyDefaultParts.join(', ');
  const selectFieldsArray = [...(keyDefault ? [keyDefault] : []), ...dbKeys];
  const orderBy = ' ORDER BY ' + parseSort(sort, aliases, 'incomming_documents', { user_deadline: 'au.user_deadline', userDeadline: 'au.user_deadline' });
  const selectFields = selectFieldsArray.join(', ');

  const totalSql = `SELECT COUNT(*) AS total FROM ${this.dbname}.dbo.incomming_documents ${joinClause} ${whereClause}`;
  // console.log('[LIST PROCESSOR] TOTAL SQL:', totalSql);

  const rowsSql = `SELECT ${selectFields} FROM ${this.dbname}.dbo.incomming_documents ${joinClause} ${whereClause} ${orderBy} OFFSET ${offsetNum} ROWS FETCH NEXT ${limitNum} ROWS ONLY`;
  console.log('[LIST PROCESSOR] ROWS SQL:', rowsSql);

  return this.executeAndMap(pool, totalSql, rowsSql, pageNum, limitNum, countOnly, userContext, receiverUnit, aliases, authority, type, isExport);
```

Sau khi refractor

```
async listDocumentsMainProcessDynamic(query: ListDocumentsDto, userId: string, authorId?: string) {
  // Handle authorization
  if (query.authority === 'true' && authorId) {
    userId = authorId;
  }

  // Get connection pool
  const pool = await this.getPool();

  // PHASE 1: Parse & Initialize
  const phase1Result = this.phase1_parseAndInitialize(query);
  const { type, page, limit, offset, filter, sort, processFn, authority, isExport, countOnly, tab, urgencyValue, isRoomFilter, roomCondition, startTime } = phase1Result;

  // PHASE 2: Get Context (user roles, feature config, receiver unit)
  const { userContext, featureManagement, receiverUnit } = await this.phase2_getContext(userId, processFn, pool);

  // PHASE 3: Build Criteria & Filters
  const { filterFeature, filterJoins } = this.phase3_buildCriteriaAndFilters(filter, featureManagement);

  // PHASE 4: Validate Type
  const { isUserFlow, isDocFlow, safeType } = this.phase4_validateType(type);

  // PHASE 5: Build JOIN Clause
  const joinClause = this.phase5_buildJoinClause(isUserFlow, isDocFlow, safeType, userId, tab, filterJoins);

  // PHASE 6: Build WHERE Clause
  const whereClause = this.phase6_buildWhereClause(type!, filter, userId, processFn, filterFeature, isRoomFilter, roomCondition, urgencyValue);

  // PHASE 7: Build SELECT Fields
  const { selectFieldsArray, aliases } = await this.phase7_buildSelectFields(processFn, userId);

  // PHASE 8: Build ORDER BY
  const orderBy = this.phase8_buildOrderBy(sort, aliases);

  // PHASE 9: Build SQL Queries
  const { totalSql, rowsSql } = this.phase9_buildSqlQueries(selectFieldsArray, joinClause, whereClause, orderBy, offset, limit);

  // PHASE 10: Execute & Map Results
  return this.phase10_executeAndMapResults(pool, totalSql, rowsSql, page, limit, countOnly, userContext, receiverUnit, aliases, authority, type, isExport, startTime);
}
```

```typescript
private async executeAndMap(pool: sql.ConnectionPool, totalSql: string, rowsSql: string, page: number, limit: number, countOnly: string | undefined, userContext: any, receiverUnit: any, aliases: any, authority?: string, type?: string, isExport?: string) {
  const startTotal = Date.now();
  this.logger.log(`⏱ executeAndMap START`);

  // PHASE 1: Handle CountOnly Mode
  const startPhase1 = Date.now();
  const countOnlyResult = await this.phase1_handleCountOnly(pool, totalSql, countOnly);
  this.logger.log(`⏱ Phase 1 (CountOnly Check): ${Date.now() - startPhase1}ms`);
  if (countOnlyResult) {
    this.logger.log(`⏱ executeAndMap TOTAL: ${Date.now() - startTotal}ms`);
    return countOnlyResult;
  }

  // PHASE 2: Execute Parallel SQL Queries
  const startPhase2 = Date.now();
  const { total, items } = await this.phase2_executeParallelQueries(pool, totalSql, rowsSql);
  this.logger.log(`⏱ Phase 2 (Parallel SQL): ${Date.now() - startPhase2}ms`);

  // PHASE 3: Check Empty Results
  const startPhase3 = Date.now();
  const emptyResult = this.phase3_checkEmptyResults(items, total, page, limit);
  this.logger.log(`⏱ Phase 3 (Empty Check): ${Date.now() - startPhase3}ms`);
  if (emptyResult) {
    this.logger.log(`⏱ executeAndMap TOTAL: ${Date.now() - startTotal}ms`);
    return emptyResult;
  }

  // PHASE 4: Extract Unique BPMN Versions
  const startPhase4 = Date.now();
  const bpmnVersions = this.phase4_extractBpmnVersions(items);
  this.logger.log(`⏱ Phase 4 (Extract BPMN Versions): ${Date.now() - startPhase4}ms - Found ${bpmnVersions.length} versions`);

  // PHASE 5: Load BPMN Processes
  const startPhase5 = Date.now();
  const bpmnEngineMap = await this.phase5_loadBpmnProcesses(bpmnVersions, receiverUnit);
  this.logger.log(`⏱ Phase 5 (Load BPMN): ${Date.now() - startPhase5}ms`);

  // PHASE 6: Map Document Details
  const startPhase6 = Date.now();
  const detailedItems = await this.phase6_mapDocumentDetailsWrapper(items, bpmnEngineMap, userContext, aliases);
  this.logger.log(`⏱ Phase 6 (Map Details): ${Date.now() - startPhase6}ms`);

  // PHASE 7: Map Document Keys for List
  const startPhase7 = Date.now();
  const detailedItemsMapped = await this.phase7_mapDocKeysForListWrapper(detailedItems, aliases, authority, userContext, type, isExport);
  this.logger.log(`⏱ Phase 7 (Map Keys): ${Date.now() - startPhase7}ms`);

  const totalTime = Date.now() - startTotal;
  this.logger.log(`⏱ executeAndMap TOTAL: ${totalTime}ms`);

  return { success: true, items: detailedItemsMapped, total, page, limit, totalPages: Math.ceil(total / limit) };
}
```