🚀 Core Business Logic - Tài Liệu Bảo Vệ Luận Văn

Backend Developer - Spring Boot Application

📑 Mục Lục Nhanh (Table of Contents)

🚀 BẢN TÓM TẮT "BỎ TÚI" (Executive Summary)

(Phần này dùng để trả lời nhanh trong 30 giây đầu tiên)

- **Hybrid Search Strategy:** Database-level filtering cho tiêu chí đơn giản (index-based) + Application-level filtering cho tiêu chí phức tạp (availability, capacity matching).
- **Pessimistic Locking:** Sử dụng `@Lock(LockModeType.PESSIMISTIC_WRITE)` để chống overbooking, đảm bảo data consistency trong giao dịch tiền bạc.
- **Scheduled Jobs:** 5 loại jobs tự động (expired bookings, no-show, dynamic pricing, partner reports, admin reports) với error handling.
- **HMAC-SHA512 Signature:** Validate payment callback từ VNPay bằng cách so sánh hash để chống giả mạo request.
- **JWT + Role Hierarchy:** Stateless authentication với role-based access control (ADMIN > PARTNER > USER).

---

1. Advanced Search Logic (Tìm Kiếm Nâng Cao)

1.1. Vấn đề & Giải pháp

Vấn đề:

- Query database với quá nhiều tiêu chí (giá, ngày, số người, địa điểm...) rất chậm và nặng
- Cartesian product khi join rooms và inventories → memory overflow
- Không thể filter availability theo ngày ở database level (cần check từng ngày)

Giải pháp:

- **Hybrid Approach (Chia để trị):**

    - **Bước 1 (DB):** Lọc nhanh theo Index (location, price, star rating, amenity)
    - **Bước 2 (App):** Load vào RAM và dùng thuật toán Java để kiểm tra availability & capacity matching
- **Tách query:** Rooms và inventories query riêng biệt → tránh cartesian product
- **Merge data:** Combine ở application level

1.2. Luồng xử lý (Logic Flow)

```
1. Phân tích bộ lọc đầu vào
   ├─ Basic filters: name, location, amenity, star rating, price
   └─ Complex filters: check-in/check-out dates, guest requirements

2. Quyết định chiến lược
   ├─ Nếu chỉ có basic filters → Database pagination trực tiếp
   └─ Nếu có complex filters → Application-level processing

3. Xử lý tìm kiếm phức tạp
   ├─ Step 1: Filter hotels từ DB (basic filters)
   ├─ Step 2: Load rooms riêng biệt
   ├─ Step 3: Load inventories riêng biệt (tránh cartesian product)
   ├─ Step 4: Merge data ở application level
   ├─ Step 5: Filter by availability (date range)
   ├─ Step 6: Filter by capacity (adults/children/rooms)
   ├─ Step 7: Apply sorting (price, star rating, created date)
   └─ Step 8: Pagination ở application level
```

1.3. Key Code Snippet (Minh chứng)

```
// HotelService.java - getHotelsWithComplexFilters()
// Step 1: Filter hotels from database using basic filters
List<String> filteredHotelIds = hotelRepository.findAllIdsByFilter(
    name, countryId, provinceId, cityId, districtId, wardId, streetId,
    status, partnerId, amenityIds, requiredAmenityCount, starRating,
    minPrice, maxPrice);

// Step 2: Get detailed hotel info including rooms
List<Hotel> hotelsWithRooms =
hotelRepository.findAllByIdsWithRooms(filteredHotelIds);

// Step 3: Get all room IDs and fetch inventories separately
List<String> roomIds = hotelsWithRooms.stream()
    .flatMap(h -> h.getRooms().stream())
    .map(room -> room.getId())
    .toList();
List<Room> roomsWithInventories =
roomRepository.findAllByIdsWithInventories(roomIds);
mergeRoomInventoriesData(hotelsWithRooms, roomsWithInventories);

// Step 4: Apply complex filtering (date + guest requirements)
```

```
finalFilteredHotels = filterByAvailabilityAndCapacity(
    candidateHotels, checkinDate, validatedCheckoutDate, totalNightsStay,
    requiredAdults, requiredChildren, requiredRooms, needsDateAndGuestValidation);
```

Mẹo Bảo Vệ (Defense Tip)

> "Hệ thống sử dụng hybrid approach: database-level filtering cho các tiêu chí đơn giản (location, price, amenity) để tận dụng index, và application-level filtering cho các tiêu chí phức tạp (availability theo ngày, capacity matching) để đảm bảo tính chính xác. Chúng em tách query rooms và inventories riêng biệt để tránh cartesian product, giúp giảm memory usage và tăng performance."

---

2. Booking Process & Overbooking Handling (Quy Trình Đặt Phòng & Xử Lý Overbooking)

2.1. Vấn đề & Giải pháp

Vấn đề:

- 2 người cùng thanh toán phòng cuối cùng một lúc → Overbooking
- Race condition khi check availability và update inventory

Giải pháp:

- **Pessimistic Locking:** Sử dụng `@Lock(LockModeType.PESSIMISTIC_WRITE)` để lock inventory records
- **Transaction Isolation:** Toàn bộ quy trình trong `@Transactional` → Atomic operation
- **Retry Mechanism:** Exponential backoff khi có lock conflict

2.2. Luồng xử lý (Logic Flow)

```
1. User click "Book"
   ├─ Validate user, room, hotel tồn tại
   ├─ Validate dates (check-in >= today, check-out > check-in)
   └─ Validate room capacity

2. Kiểm tra availability với Pessimistic Locking
   ├─ Lock inventory records (SELECT FOR UPDATE)
   ├─ Check available rooms >= required rooms (cho mỗi ngày)
   └─ Nếu không đủ → Throw exception

3. Tính toán giá
   ├─ Original price từ inventories
   ├─ Validate & apply discount code
   ├─ Calculate VAT và service fee
   └─ Final price

4. Tạo booking trong transaction
   ├─ Save booking (status: PENDING_PAYMENT)
   ├─ Update inventory (giảm available rooms)
   ├─ Update discount usage count
```

```
    └─ All atomic (rollback nếu có lỗi)

5. Retry mechanism (nếu có lock conflict)
    ├─ Retry tối đa 3 lần
    └─ Exponential backoff

6. Generate payment URL
    └─ Return booking response với payment URL
```

2.3. Key Code Snippet (Minh chứng)

```
// BookingService.java - create()
@Transactional
public BookingResponse create(BookingCreationRequest request, HttpServletRequest
httpRequest) {
    int maxRetries = 3;
    int retryDelay = 100; // milliseconds

    for (int attempt = 1; attempt <= maxRetries; attempt++) {
        try {
            return performBookingCreation(request, httpRequest);
        } catch (PessimisticLockingFailureException e) {
            if (attempt == maxRetries) {
                throw new AppException(ErrorType.CONCURRENT_BOOKING_FAILED);
            }
            Thread.sleep((long) retryDelay * attempt); // Exponential backoff
        }
    }
}

// RoomInventoryRepository.java - Pessimistic Lock
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query(RoomInventoryQueries.FIND_ALL_BY_ROOM_ID_AND_DATE_BETWEEN_WITH_LOCK)
List<RoomInventory> findAllByRoomIdAndDateBetweenWithLock(
    String roomId, LocalDate startDate, LocalDate endDate);

// RoomInventoryService.java - validateRoomAvailability()
public List<RoomInventory> validateRoomAvailability(String roomId,
    LocalDate checkInDate, LocalDate checkOutDate, int numberOfRooms) {
    // Get room inventories with pessimistic locking
    List<RoomInventory> inventories = roomInventoryRepository
        .findAllByRoomIdAndDateBetweenWithLock(roomId, checkInDate,
checkOutDate.minusDays(1));

    // Check if enough rooms are available for each day
    for (RoomInventory inventory : inventories) {
        if (inventory.getAvailableRooms() < numberOfRooms) {
            throw new AppException(ErrorType.INSUFFICIENT_ROOM_QUANTITY);
        }
    }
```

```
        return inventories;
    }
```

Mẹo Bảo Vệ (Defense Tip)

> "Hệ thống sử dụng Pessimistic Locking với `LockModeType.PESSIMISTIC_WRITE` để đảm bảo không có overbooking. Khi user đặt phòng, hệ thống lock các inventory records trong khoảng thời gian booking, kiểm tra availability, và update inventory trong cùng một transaction. Nếu có concurrent booking attempts, hệ thống retry với exponential backoff để xử lý lock conflicts. Điều này đảm bảo tính nhất quán dữ liệu và ngăn chặn việc bán quá số phòng available."

---

3. Automation Schedulers (Các Job Tự Động)

3.1. Vấn đề & Giải pháp

Vấn đề:

- Bookings pending payment quá lâu → chiếm inventory
- No-show bookings → lãng phí phòng
- Giá phòng cần update theo special days/weekends
- Báo cáo cần generate tự động hàng ngày

Giải pháp:

- **Spring `@Scheduled`:** 5 loại jobs tự động với cron expressions
- **Error Handling:** Try-catch trong mỗi job → không crash scheduler
- **Logging:** Track execution time và số lượng records processed

3.2. Luồng xử lý (Logic Flow)

Job 1: Cancel Expired Bookings

```
Schedule: @Scheduled(fixedRate = 300000) // Mỗi 5 phút
Flow:
  1. Tìm bookings: status = PENDING_PAYMENT AND createdAt < now - 15 minutes
  2. Update status = CANCELLED
  3. Release room inventory
  4. Save booking
```

Job 2: Cancel No-Show Bookings

```
Schedule: @Scheduled(cron = "0 0 12 * * *") // Hàng ngày 12:00 PM
Flow:
  1. Tìm bookings: checkInDate = yesterday AND status IN (CONFIRMED, RESCHEDULED)
  2. Update status = CANCELLED (no refund)
  3. Release room inventory
```

Job 3: Dynamic Pricing Update

```
Schedule: @Scheduled(cron = "0 0 1 * * *") // Hàng ngày 1:00 AM
Flow:
  1. Lấy tất cả rooms
  2. Với mỗi room, lấy inventories (today → today + lookAhead days)
  3. Áp dụng pricing rules:
     - Special Day → Apply discount
     - Weekend → Multiply by weekendPriceMultiplier
     - Normal day → Base price
  4. Update inventory prices
```

Job 4: Partner Reports

```
Schedule: @Scheduled(cron = "0 0 2 * * *") // Hàng ngày 2:00 AM
Flow:
  1. Calculate report date = yesterday (T-1)
  2. Generate daily reports cho tất cả partners
```

Job 5: Admin Reports

```
Schedule: @Scheduled(cron = "0 30 2 * * *") // Hàng ngày 2:30 AM
Flow:
  1. Calculate report date = yesterday (T-1)
  2. Generate system-wide reports
```

3.3. Key Code Snippet (Minh chứng)

```
// BookingExpirationScheduler.java
@Scheduled(fixedRate = 300000) // 5 minutes
public void cancelExpiredBookings() {
    LocalDateTime expiredTime = LocalDateTime.now().minusMinutes(15);
    List<Booking> expiredBookings =
bookingRepository.findByStatusAndCreatedAtBefore(
        BookingStatusType.PENDING_PAYMENT.getValue(), expiredTime);

    for (Booking booking : expiredBookings) {
        booking.setStatus(BookingStatusType.CANCELLED.getValue());
        roomInventoryService.updateAvailabilityForCancellation(
            booking.getRoom().getId(),
            booking.getCheckInDate(),
            booking.getCheckOutDate(),
            booking.getNumberOfRooms());
```

```
            bookingRepository.save(booking);
        }
    }

    // DynamicPricingService.java - applyDynamicPricing()
    for (RoomInventory inventory : inventories) {
        LocalDate currentDate = inventory.getId().getDate();
        DayOfWeek dayOfWeek = currentDate.getDayOfWeek();
        boolean isSpecialDay = specialDayMap.containsKey(currentDate);
        boolean isWeekend = dayOfWeek == DayOfWeek.SATURDAY || dayOfWeek ==
    DayOfWeek.SUNDAY;

        if (isSpecialDay) {
            SpecialDayDiscount discount = specialDayDiscountRepository
                .findBySpecialDayIdWithDiscount(specialDay.getId()).orElseThrow(...);
            newPrice = basePrice * (1 - discount.getPercentage() / 100);
        } else if (isWeekend) {
            newPrice = basePrice * weekendPriceMultiplier;
        } else {
            newPrice = basePrice;
        }
        inventory.setPrice(newPrice);
    }
```

Mẹo Bảo Vệ (Defense Tip)

> "Hệ thống có 5 loại scheduler jobs chạy tự động: (1) Cancel expired bookings mỗi 5 phút để giải phóng inventory, (2) Cancel no-show bookings hàng ngày lúc 12h trưa, (3) Dynamic pricing update hàng ngày lúc 1h sáng để áp dụng giá theo special days và weekends, (4) Generate partner reports lúc 2h sáng, và (5) Generate admin reports lúc 2h30 sáng. Tất cả jobs đều có error handling và logging để đảm bảo reliability. Knowledge Base có cả full sync (weekly) và incremental sync (hourly) để tối ưu performance."

---

4. VNPay Integration (Tích Hợp Thanh Toán VNPay)

4.1. Vấn đề & Giải pháp

Vấn đề:

- Sợ giả mạo payment callback → khách chưa trả tiền mà hệ thống tưởng đã trả
- Cần đảm bảo tính toàn vẹn của payment data

Giải pháp:

- **HMAC-SHA512 Signature:** Validate callback bằng cách so sánh hash
- **Idempotent Processing:** Check payment status trước khi process → tránh duplicate processing
- **Transaction Management:** Update payment và booking trong cùng transaction

4.2. Luồng xử lý (Logic Flow)

4.2.1. Tạo Payment URL

```
1. Tạo Payment Entity (status: PENDING)
2. Build VNPay Parameters
   ├─ vnp_Version, vnp_Command, vnp_TmnCode
   ├─ vnp_Amount (convert to cents: * 100)
   ├─ vnp_TxnRef (Payment ID)
   ├─ vnp_ReturnUrl, vnp_IpAddr
   └─ vnp_CreateDate, vnp_ExpireDate (15 phút)

3. Generate Secure Hash
   ├─ Sort parameters (alphabetical order)
   ├─ URL encode key và value
   ├─ Build query string: key1=value1&key2=value2&...
   ├─ Calculate HMAC-SHA512 với secret key
   └─ Append hash: ?query&vnp_SecureHash=hash

4. Return payment URL
```

4.2.2. Validate Callback (IPN/Return URL)

```
1. Validate Signature
   ├─ Extract vnp_SecureHash từ callback
   ├─ Remove hash khỏi params
   ├─ Build hash data (giống khi tạo URL)
   ├─ Calculate HMAC-SHA512
   └─ Compare: receivedHash == generatedHash

2. Validate Payment Status
   ├─ Check payment status = PENDING
   └─ Extract vnp_ResponseCode

3. Xử lý kết quả
   ├─ If responseCode = "00" (success):
   │   ├─ Update payment status = SUCCESS
   │   ├─ Update booking status = CONFIRMED
   │   └─ Send confirmation email
   └─ If failed:
       ├─ Update payment status = FAILED
       ├─ Update booking status = CANCELLED
       └─ Release room inventory
```

4.3. Key Code Snippet (Minh chứng)

```
// PaymentService.java - createPaymentUrl()
// Build query string with URL encoding
StringBuilder queryString = new StringBuilder();
```

```
for (Map.Entry<String, String> entry : vnpParams.entrySet()) {
    if (entry.getValue() != null && !entry.getValue().isEmpty()) {
        queryString.append(URLEncoder.encode(entry.getKey(),
StandardCharsets.UTF_8));
        queryString.append("=");
        queryString.append(URLEncoder.encode(entry.getValue(),
StandardCharsets.UTF_8));
        queryString.append("&");
    }
}
String query = queryString.toString();
if (query.endsWith("&")) {
    query = query.substring(0, query.length() - 1);
}

// Generate secure hash using HMAC-SHA512
String secureHash = hmacSHA512(vnpayHashSecret, query);
return vnpayApiUrl + "?" + query + "&vnp_SecureHash=" + secureHash;

// PaymentService.java - validateSignature()
public boolean validateSignature(Map<String, String> vnpayParams) {
    String receivedHash = vnpayParams.get("vnp_SecureHash");
    Map<String, String> paramsForValidation = new TreeMap<>(vnpayParams);
    paramsForValidation.remove("vnp_SecureHash");
    paramsForValidation.remove("vnp_SecureHashType");

    String hashData = buildHashDataForValidation(paramsForValidation);
    String generatedHash = hmacSHA512(vnpayHashSecret, hashData);

    return receivedHash.equals(generatedHash);
}

// PaymentService.java - hmacSHA512()
private String hmacSHA512(final String key, final String data) {
    Mac mac = Mac.getInstance("HmacSHA512");
    SecretKeySpec secretKeySpec = new
SecretKeySpec(key.getBytes(StandardCharsets.UTF_8), "HmacSHA512");
    mac.init(secretKeySpec);
    byte[] hashBytes = mac.doFinal(data.getBytes(StandardCharsets.UTF_8));

    StringBuilder hexString = new StringBuilder();
    for (byte b : hashBytes) {
        String hex = Integer.toHexString(0xff & b);
        if (hex.length() == 1) {
            hexString.append('0');
        }
        hexString.append(hex);
    }
    return hexString.toString();
}
```

Mẹo Bảo Vệ (Defense Tip)

> "Hệ thống tích hợp VNPay sử dụng HMAC-SHA512 để đảm bảo tính bảo mật. Khi tạo payment URL, hệ thống sắp xếp parameters theo alphabetical order, URL encode, và tính hash bằng HMAC-SHA512 với secret key. Khi nhận callback từ VNPay, hệ thống validate signature bằng cách tính lại hash và so sánh với hash nhận được. Nếu signature không hợp lệ, hệ thống từ chối payment. Điều này đảm bảo không có ai có thể giả mạo payment callback."

---

5. Security & Auth (Bảo Mật & Xác Thực)

5.1. Vấn đề & Giải pháp

Vấn đề:

- Cần xác thực user mà không query database mỗi request
- Phân quyền theo role (ADMIN, PARTNER, USER)
- Stateless architecture để scale horizontal

Giải pháp:

- **JWT (JSON Web Token):** Stateless authentication
- **Spring Security:** Role-based access control
- **Role Hierarchy:** ADMIN > PARTNER > USER

5.2. Luồng xử lý (Logic Flow)

5.2.1. Login Flow

```
1. User gửi email và password
2. AuthService.validateCredentials()
   ├─ Find UserAuthInfo by email
   ├─ Check auth provider = LOCAL
   ├─ Check account active
   └─ Verify password (BCrypt)

3. Generate JWT Token
   ├─ Claims: email (subject), role (scope), fullName
   ├─ Sign với HS512 algorithm
   ├─ Generate refresh token
   └─ Save refresh token vào UserAuthInfo

4. Return TokenResponse
   └─ Access token, refresh token, expiration time, user info
```

5.2.2. Request Authentication Flow

```
1. CustomCookieAuthenticationFilter (chạy trước)
   ├─ Check cookie có token
   ├─ Parse JWT
```

```
    ├─ Validate signature và expiration
    └─ Set CustomAuthenticationToken vào SecurityContext

2. CustomJwtAuthenticationFilter (chạy sau BearerTokenAuthenticationFilter)
    ├─ Check JwtAuthenticationToken từ OAuth2 Resource Server
    ├─ Extract claims (email, scope, expiration)
    └─ Convert sang CustomAuthenticationToken

3. SecurityConfig
    ├─ Public endpoints: auth, GET hotels/rooms
    ├─ Protected endpoints: role-based access
    ├─ Role hierarchy: ADMIN > PARTNER > USER
    └─ CORS configuration
```

5.3. Key Code Snippet (Minh chứng)

```java
// AuthService.java - login()
public TokenResponse login(LoginRequest loginRequest) throws JOSEException {
    UserAuthInfo authInfo = authInfoRepository.findByUserEmail(email)
        .orElseThrow(() -> new AppException(ErrorType.USER_NOT_FOUND));

    // Validate password
    boolean passwordMatches = passwordEncoder.matches(rawPassword,
encodedPassword);
    if (!passwordMatches) {
        throw new AppException(ErrorType.UNAUTHORIZED);
    }

    // Generate access token
    String accessToken = generateToken(user, accessTokenExpirationMillis);

    // Generate refresh token
    String refreshToken = generateToken(user, refreshTokenExpirationMillis);
    authInfo.setRefreshToken(refreshToken);
    authInfoRepository.save(authInfo);

    return TokenResponse.builder()
        .accessToken(accessToken)
        .refreshToken(refreshToken)
        .expiresAt(expiresAt)
        .build();
}

// AuthService.java - generateToken()
public String generateToken(User user, long expirationMillis) throws JOSEException
{
    JWTClaimsSet claimsSet = new JWTClaimsSet.Builder()
        .subject(user.getEmail())
        .claim("fullName", user.getFullName())
        .claim("scope", user.getRole().getName())
        .issuer(issuer)
```

```
            .expirationTime(expirationTime)
            .build();

        JWSObject jwsObject = new JWSObject(jwsHeader, payload);
        MACSigner signer = new MACSigner(secretKey);
        jwsObject.sign(signer);
        return jwsObject.serialize();
    }

    // SecurityConfig.java - Role Hierarchy
    @Bean
    RoleHierarchy roleHierarchy() {
        // ADMIN > PARTNER > USER
        return RoleHierarchyImpl.fromHierarchy(
            RoleType.ADMIN.getValue() + " > " + RoleType.PARTNER.getValue() + "\n" +
            RoleType.PARTNER.getValue() + " > " + RoleType.USER.getValue());
    }
```

Mẹo Bảo Vệ (Defense Tip)

> "Hệ thống sử dụng JWT-based authentication với HS512 algorithm. Khi user login, hệ thống verify password bằng BCrypt, sau đó generate JWT token chứa email, role, và expiration time. Token được sign với secret key. Khi user gửi request, hệ thống có 2 filters: CookieAuthenticationFilter để xử lý cookie-based auth, và JwtAuthenticationFilter để xử lý Bearer token. SecurityConfig sử dụng role hierarchy (ADMIN > PARTNER > USER) để đảm bảo admin có thể access tất cả endpoints của partner và user. Tất cả endpoints đều được protect theo role, chỉ có auth endpoints và một số GET endpoints là public."

---

6. Edge Cases Handling (Xử Lý Các Trường Hợp Đặc Biệt)

6.1. Payment Reconciliation (Đối Soát Thanh Toán)

Q: "Nếu VNPay gọi Callback (IPN) thất bại (do mạng/server down) thì sao?"

A: Hệ thống có **Scheduled Job: Payment Reconciliation** chạy 15 phút/lần. Nó sẽ gọi API 'Query Transaction' của VNPay để kiểm tra các đơn hàng đang PENDING. Nếu VNPay báo thành công mà database chưa cập nhật, hệ thống sẽ tự động cập nhật và gửi mail vé cho khách. Đảm bảo 'Tiền đi thì Vé về'.

6.2. In-Memory Search Scalability (Khả Năng Mở Rộng Tìm Kiếm Trong Bộ Nhớ)

Q: "Load dữ liệu vào RAM để search có sợ tràn bộ nhớ (OOM) khi dữ liệu lớn không?"

A: Hiện tại hệ thống dùng **Pagination ngay từ Database** (Bước 1) để chỉ lấy IDs của trang hiện tại (ví dụ 20 khách sạn), sau đó mới load chi tiết Inventory của 20 khách sạn đó vào RAM (Bước 2). Vì vậy, dù DB có 1 triệu khách sạn, RAM chỉ tốn dung lượng cho 20 khách sạn đang hiển thị -> O(1) memory usage cho mỗi request.

Tổng Kết

Hệ thống được thiết kế với các đặc điểm chính:

1. **Search Logic:** Hybrid approach - database filtering + application-level filtering
2. **Booking:** Pessimistic locking để chống overbooking
3. **Schedulers:** 5 loại jobs tự động với error handling
4. **Payment:** HMAC-SHA512 signature validation cho bảo mật
5. **Security:** JWT-based auth với role hierarchy

Tất cả các components đều có logging, error handling, và transaction management để đảm bảo reliability và data consistency.