**5. Implementation**

# 目录

5. Implementation ........................ 2

I. Algorithms ........................ 3

1. Service Provider Search Algorithm ........................ 4

2. Booking Slot Availability Algorithm ........................ 5

3. Service Provider Recommendation Algorithm ........................ 6

4. Review Sentiment Analysis Algorithm ........................ 7

5. Dynamic Pricing Algorithm ........................ 8

6. Service Provider Matching Algorithm ........................ 9

7. Real-time Chat Message Routing Algorithm ........................ 10

8. Booking Conflict Resolution Algorithm ........................ 11

9. Payment Processing Algorithm ........................ 12

10. Service Provider Ranking Algorithm ........................ 13

11. User Authentication and Authorization Algorithm ........................ 14

12. Notification Dispatch Algorithm ........................ 15

*[在Microsoft Word中，您可以右键点击上方目录区域并选择'更新域'来自动生成准确的页码]*

# **5. Implementation**

## **I. Algorithms**

This document outlines the key algorithms implemented in the ArtisanConnect platform. These algorithms are essential for the core functionality of the application, including search, matching, recommendation, booking, and other critical operations.

### **1. Service Provider Search Algorithm**

The search algorithm enables users to find relevant service providers based on multiple criteria.

Algorithm: ServiceProviderSearch

Input: searchCriteria (category, location, price range, rating, availability)

Output: rankedListOfProviders

1. Initialize empty result list

2. If category is specified:

   a. Filter providers by exact category match

3. If location is specified:

   a. Calculate distance between user location and provider service areas

   b. Filter providers within specified radius

4. If price range is specified:

   a. Filter providers with services in the specified price range

5. If rating is specified:

   a. Filter providers with ratings >= specified minimum

6. If availability is specified:

   a. Filter providers with available slots on specified dates

7. Calculate relevance score for each provider:

   score = w1\*categoryRelevance + w2\*locationProximity + w3\*priceMatch + w4\*ratingScore + w5\*availabilityMatch

8. Sort providers by relevance score in descending order

9. Return paginated list of providers

**Time Complexity**: O(n log n) where n is the number of service providers (dominated by the sorting operation)  
**Space Complexity**: O(n) for storing the filtered and sorted results

### **2. Booking Slot Availability Algorithm**

This algorithm determines available booking slots for a service provider based on their schedule, existing bookings, and service duration.

Algorithm: AvailableBookingSlots

Input: providerId, date, serviceDuration

Output: listOfAvailableTimeSlots

1. Retrieve provider's working hours for the specified date

2. Retrieve all existing bookings for the provider on the specified date

3. Initialize availableSlots as empty list

4. For each working period (start time to end time):

   a. Generate potential time slots with intervals of serviceDuration

   b. For each potential slot:

      i. Check if slot overlaps with any existing booking

      ii. If no overlap, add slot to availableSlots

5. Return availableSlots sorted by start time

**Time Complexity**: O(n\*m) where n is the number of working periods and m is the number of existing bookings  
**Space Complexity**: O(p) where p is the number of potential time slots

### **3. Service Provider Recommendation Algorithm**

This algorithm recommends service providers to users based on their past bookings, preferences, and provider ratings.

Algorithm: RecommendProviders

Input: userId, limit

Output: rankedListOfRecommendedProviders

1. Retrieve user's booking history and preferences

2. Identify categories user has previously booked

3. Retrieve highly-rated providers in those categories

4. Calculate recommendation score for each provider:

   score = w1\*categoryMatch + w2\*ratingScore + w3\*bookingFrequency + w4\*userPreferenceMatch

5. Filter out providers already used by the user (optional)

6. Sort providers by recommendation score in descending order

7. Return top 'limit' providers

**Time Complexity**: O(n log n) where n is the number of potential providers  
**Space Complexity**: O(n) for storing recommendation scores and sorted results

### **4. Review Sentiment Analysis Algorithm**

This algorithm analyzes the sentiment of user reviews to provide insights on service quality.

Algorithm: ReviewSentimentAnalysis

Input: reviewText

Output: sentimentScore, keyPhrases

1. Preprocess review text:

   a. Convert to lowercase

   b. Remove punctuation and special characters

   c. Remove stop words

2. Tokenize text into words/phrases

3. For each token:

   a. Look up sentiment score in sentiment lexicon

   b. Adjust score based on negation words and intensifiers

4. Calculate overall sentiment score as weighted average of token scores

5. Identify key phrases with highest positive or negative sentiment

6. Return overall sentiment score and key phrases

**Time Complexity**: O(n) where n is the number of tokens in the review text  
**Space Complexity**: O(n) for storing tokens and their sentiment scores

### **5. Dynamic Pricing Algorithm**

This algorithm adjusts service prices based on demand, availability, and other market factors.

Algorithm: DynamicPricing

Input: basePrice, serviceId, date, timeSlot

Output: adjustedPrice

1. Calculate demand factor:

   a. Retrieve booking density for the specified date and time slot

   b. Compare to historical average booking density

   c. Calculate demand ratio = current density / average density

2. Calculate seasonality factor based on historical data for the date

3. Calculate time slot premium:

   a. Assign premium values to peak hours

   b. Assign discount values to off-peak hours

4. Calculate service popularity factor based on booking frequency

5. Compute adjusted price:

   adjustedPrice = basePrice \* (1 + w1\*(demand factor - 1) + w2\*(seasonality factor - 1) +

                   w3\*(time slot premium) + w4\*(popularity factor - 1))

6. Apply minimum and maximum price constraints

7. Return adjustedPrice

**Time Complexity**: O(1) assuming pre-calculated factors  
**Space Complexity**: O(1) for storing calculation variables

### **6. Service Provider Matching Algorithm**

This algorithm matches customer service requests with the most suitable service providers.

Algorithm: MatchServiceRequest

Input: serviceRequest (service type, location, requirements, budget, timing)

Output: rankedListOfMatchedProviders

1. Filter providers by service type match

2. Filter by location proximity to customer

3. Filter by availability during requested timing

4. For each remaining provider:

   a. Calculate expertise match score based on provider skills and request requirements

   b. Calculate price match score based on provider rates and customer budget

   c. Calculate rating match based on provider ratings for the specific service

   d. Calculate response time score based on provider's historical response time

   e. Compute overall match score:

      matchScore = w1\*expertiseMatch + w2\*priceMatch + w3\*ratingMatch + w4\*responseTimeScore

5. Sort providers by match score in descending order

6. Return top ranked providers

**Time Complexity**: O(n log n) where n is the number of providers after initial filtering  
**Space Complexity**: O(n) for storing match scores and sorted results

### **7. Real-time Chat Message Routing Algorithm**

This algorithm ensures messages are properly routed between customers and service providers in the real-time chat system.

Algorithm: ChatMessageRouting

Input: message (senderId, recipientId, content, attachments)

Output: deliveryStatus

1. Validate message content and structure

2. Check if conversation exists between sender and recipient

   a. If not, create new conversation

3. Store message in database with status "sending"

4. Check if recipient is online:

   a. If online, push message via WebSocket connection

   b. If offline, queue message for delivery and prepare notification

5. Wait for delivery acknowledgment or timeout

6. Update message status to "delivered" or "pending"

7. If message contains action items (e.g., booking request), trigger appropriate workflow

8. Return delivery status and message ID

**Time Complexity**: O(1) for message routing operations  
**Space Complexity**: O(m) where m is the message size including attachments

### **8. Booking Conflict Resolution Algorithm**

This algorithm detects and resolves conflicts in service provider booking schedules.

Algorithm: ResolveBookingConflicts

Input: newBooking (providerId, startTime, endTime, serviceId)

Output: bookingStatus, conflictResolution

1. Retrieve all existing bookings for the provider within the time range

2. Check for direct conflicts (overlapping time slots)

3. If no direct conflicts:

   a. Check for buffer time violations (insufficient time between bookings)

   b. If buffer time violated, calculate alternative time slots

4. If direct conflicts exist:

   a. Identify the nature of the conflict (double booking, overrun risk)

   b. Calculate severity of conflict

   c. Generate resolution options:

      i. Suggest alternative time slots

      ii. Suggest alternative providers

      iii. Adjust service duration if possible

5. Return booking status and resolution options

**Time Complexity**: O(n) where n is the number of existing bookings in the relevant time range  
**Space Complexity**: O(m) where m is the number of resolution options generated

### **9. Payment Processing Algorithm**

This algorithm handles the secure processing of payments for bookings.

Algorithm: ProcessPayment

Input: paymentDetails (bookingId, amount, paymentMethod, customerInfo)

Output: transactionResult

1. Validate payment details and amount against booking

2. Apply security checks:

   a. Verify customer identity

   b. Check for suspicious patterns

3. Select payment processor based on payment method

4. Initialize payment transaction with processor

5. Apply any applicable discounts or promotions

6. Process payment through secure channel:

   a. If successful, record transaction and update booking status

   b. If failed, log error and provide failure reason

7. Generate payment receipt

8. Trigger post-payment workflows (notifications, commission calculations)

9. Return transaction result with reference ID

**Time Complexity**: O(1) for the core payment processing (dependent on external payment processor)  
**Space Complexity**: O(1) for transaction data

### **10. Service Provider Ranking Algorithm**

This algorithm ranks service providers for display in search results and category browsing.

Algorithm: RankServiceProviders

Input: providers, userContext (location, preferences, history)

Output: rankedProviders

1. For each provider:

   a. Calculate base score from average rating

   b. Apply weighting based on number of reviews (confidence factor)

   c. Calculate recency factor based on recent activity and reviews

   d. Calculate completion rate factor from successful vs canceled bookings

   e. Calculate response time factor from average response time to inquiries

   f. If userContext available:

      i. Calculate personalization factor based on user preferences and history

      ii. Calculate proximity factor based on user location

   g. Compute final rank score:

      rankScore = w1\*weightedRating + w2\*recencyFactor + w3\*completionRate +

                 w4\*responseTimeFactor + w5\*personalizationFactor + w6\*proximityFactor

2. Sort providers by rank score in descending order

3. Return ranked list of providers

**Time Complexity**: O(n log n) where n is the number of providers  
**Space Complexity**: O(n) for storing rank scores and sorted results

### **11. User Authentication and Authorization Algorithm**

This algorithm handles secure user authentication and role-based access control.

Algorithm: AuthenticateAndAuthorize

Input: credentials, requestedResource, requestedAction

Output: authorizationResult

1. Authenticate user:

   a. Verify credentials against stored user data

   b. If invalid, return authentication failure

   c. If valid, generate session token

2. Retrieve user role and permissions

3. Check if user role has permission for requested action on resource:

   a. Get resource access control list

   b. Check if user role is in the allowed roles for the action

   c. Check for specific user permissions overrides

4. If authorized:

   a. Log access attempt

   b. Return success with session token

5. If not authorized:

   a. Log unauthorized access attempt

   b. Return authorization failure with reason

**Time Complexity**: O(1) for authentication and permission checking  
**Space Complexity**: O(1) for authorization data

### **12. Notification Dispatch Algorithm**

This algorithm manages the sending of notifications to users through appropriate channels.

Algorithm: DispatchNotification

Input: notification (userId, type, content, urgency)

Output: deliveryStatus

1. Retrieve user notification preferences

2. Determine appropriate notification channels based on:

   a. Notification type and urgency

   b. User preferences

   c. User's recent activity and current status

3. Format notification for each selected channel

4. For each channel:

   a. Attempt to deliver notification

   b. Track delivery status

   c. If delivery fails, queue for retry or fallback to alternative channel

5. Update notification status in database

6. Return aggregated delivery status

**Time Complexity**: O© where c is the number of notification channels  
**Space Complexity**: O(n) where n is the notification content size

These algorithms form the core of the ArtisanConnect platform’s functionality, enabling efficient service provider discovery, booking management, user engagement, and platform operations. Each algorithm is implemented with consideration for performance, scalability, and user experience.