

Authsome Service Interface Contracts

Version: 1.0
Last Updated: October 31, 2025
Purpose: This document defines the functional contracts for all Authsome services. Developers can implement these services in any programming language or framework. The communication protocol (REST, gRPC, message queue, etc.) will be decided later.

Table of Contents

- 1. [Overview](#)
 - 2. [Common Data Types](#)
 - 3. [Tenant Service](#)
 - 4. [OTP Service](#)
 - 5. [Notifier Service](#)
 - 6. [Error Handling](#)
-

Overview

Architecture

Each service is independent and must implement the functions defined in this contract. The communication mechanism between services is not yet specified - implementations should be flexible enough to support REST APIs, gRPC, message queues, or direct function calls.

Data Format

All data types should be serializable to JSON for interoperability.

Common Data Types

Timestamps

- **Type:** Long integer (64-bit)
- **Format:** Milliseconds since Unix epoch
- **Example:** 1698710400000

Identifiers

- **Type:** String
- **Format:** UUID (36 characters)
- **Example:** "550e8400-e29b-41d4-a716-446655440000"

Enumerations

IdentityType (Tenant Service)

- EMAIL
- USERNAME
- USER_ID

IdentityType (Notifier Service)

- EMAIL

Note: Future versions may include SMS, PUSH_NOTIFICATION

OtpType

- NUMERIC - Only numbers (0-9)
- ALPHABETIC - Only letters (a-z, A-Z)
- ALPHANUMERIC - Mix of numbers and letters

Tenant Service

Interface Overview

The Tenant Service manages tenant accounts and provides tenant lookup functionality.

Function 1: getTenantByIdentity

Description: Retrieves a tenant by their identity (email, username, or user ID).

Input Parameters:

Parameter	Type	Required	Description
identityType	IdentityType	Yes	Type of identity (EMAIL, USERNAME, USER_ID)
identity	String	Yes	The identity value to search for

Output:

Returns FetchedTenant object or null if not found.

FetchedTenant Structure:



```
{
  id: String (UUID)
  username: String
  createdAt: Long (timestamp in milliseconds)
  updatedAt: Long (timestamp in milliseconds)
}
```

Example:



Input:

```
identityType = EMAIL
identity = "user@example.com"
```

Output (found):

```
{
  id: "550e8400-e29b-41d4-a716-446655440000"
  username: "john_doe"
  createdAt: 1698710400000
  updatedAt: 1698710400000
}
```

Output (not found):

```
null
```

Error Cases:

- Invalid identityType
- Empty or null identity value

Function 2: getTenantByUsername

Description: Retrieves a tenant by their username.

Input Parameters:

Parameter	Type	Required	Description
username	String	Yes	The username to search for

Output:

Returns FetchedTenant object or null if not found.

FetchedTenant Structure:



```
{
  id: String (UUID)
  username: String
  createdAt: Long (timestamp in milliseconds)
  updatedAt: Long (timestamp in milliseconds)
}
```

Example:



```
Input:
username = "john_doe"
```

```
Output (found):
{
  id: "550e8400-e29b-41d4-a716-446655440000"
  username: "john_doe"
  createdAt: 1698710400000
  updatedAt: 1698710400000
}
```

```
Output (not found):
null
```

Error Cases:

- Empty or null username

OTP Service

Interface Overview

The OTP Service generates, stores, validates, and manages one-time passwords for various authentication workflows.

Function 1: generateAndSaveOtp

Description: Generates a new OTP and stores it with associated metadata. Returns the OTP details including the generated code.

Input Parameters:

Parameter	Type	Required	Description
otpType	OtpType	Yes	Type of OTP (NUMERIC, ALPHABETIC, ALPHANUMERIC)
otpLength	Integer	Yes	Length of OTP (typically 4–8, must be positive)
minNumber	Integer	Yes	Minimum numeric characters required (–1 = not specified, 0 = invalid)
minAlphabet	Integer	Yes	Minimum alphabetic characters required (–1 = not specified, 0 = invalid)
maxNumber	Integer	Yes	Maximum numeric characters allowed (–1 = not specified, 0 = invalid)
maxAlphabet	Integer	Yes	Maximum alphabetic characters allowed (–1 = not specified, 0 = invalid)
expiresAfterSecond	Integer	Yes	Expiry time in seconds (typically 300 = 5 minutes, must be positive)
context	String	Yes	Context identifier (e.g., "AUTHSOME_TENANT_SIGNUP")
metadata	Map<String, String>	Yes	Key–value pairs of additional data to store

Output:

Returns FetchedOtp object.

FetchedOtp Structure:



```
{
  id: String (UUID)
  code: String (the actual OTP code)
  context: String
  expiresAt: Long (timestamp in milliseconds)
  metadata: Map<String, Object>
}
```

Example:



```
Input:
otpType = NUMERIC
otpLength = 4
minNumber = -1
minAlphabet = -1
maxNumber = -1
maxAlphabet = -1
expiresAfterSecond = 300
context = "AUTHSOME_TENANT_SIGNUP"
metadata = {
  "identity": "user@example.com",
  "identityType": "EMAIL",
  "username": "john_doe",
  "password": "encryptedPasswordString"
}
```

```
Output:
{
  id: "otp-550e8400-e29b-41d4-a716-446655440000"
  code: "1234"
  context: "AUTHSOME_TENANT_SIGNUP"
  expiresAt: 1698710700000
  metadata: {
    "identity": "user@example.com",
    "identityType": "EMAIL",
    "username": "john_doe",
    "password": "encryptedPasswordString"
  }
}
```

Error Cases:

- Invalid otpType
- Invalid otpLength (must be positive)
- Invalid parameter values (0 is invalid, use -1 for not specified)
- Invalid expiresAfterSecond (must be positive)
- Empty or null context
- Conflicting min/max constraints

Implementation Notes:

- The combination of id + context must be unique
- OTP code should be randomly generated using a cryptographically secure random generator
- Expired OTPs should be cleaned up automatically (recommended: background job)
- Store expiresAt as current timestamp + expiresAfterSecond
- For min/max parameters: -1 = not specified (ignore constraint), 0 = invalid, positive values = apply constraint
- When otpType is NUMERIC, minAlphabet and maxAlphabet are typically -1 (not applicable)

Notifier Service

Interface Overview

The Notifier Service sends notifications to users through various channels (email, SMS, push notifications, etc.).

Function 1: sendNotification

Description: Sends a notification to the specified identity through the appropriate channel.

Input Parameters:

Parameter	Type	Required	Description
identityType	IdentityType	Yes	Type of identity/channel (EMAIL)
identity	String	Yes	Identity value (email address, phone number, etc.)
subject	String	Yes	Subject/title of the notification
content	String	Yes	Body/content of the notification

Output:

Returns void (no return value). Throws error if sending fails.

Example:



```
Input:
identityType = EMAIL
identity = "user@example.com"
subject = "OTP to create authsome account"
content = "Your OTP to create your Authsome account is: 1234"
```

```
Output:
(void - no return value)
```

Error Cases:

- Invalid identityType
- Empty or null identity
- Empty or null subject
- Empty or null content
- Failed to send notification (network error, invalid email, etc.)

Implementation Notes:

- For EMAIL: Send an email using SMTP or email service provider
- Subject becomes email subject line
- Content becomes email body (can be HTML or plain text)
- Should be asynchronous if possible (don't block waiting for email to send)
- Consider retry logic for failed sends
- Log all sent notifications for audit purposes

Error Handling


All service implementations should use consistent error types:

Error Types

Error Type	Description	When to Use
VALIDATION_ERROR	Invalid input data	Bad format, missing required fields, constraint violations
NOT_FOUND	Resource not found	Entity doesn't exist in database
CONFLICT	Resource already exists	Duplicate username, email already registered
EXPIRED	Resource has expired	OTP expired, session timeout
UNAUTHORIZED	Authentication failed	Invalid credentials, invalid token
INTERNAL_ERROR	Unexpected server error	Database errors, network failures, unexpected exceptions

Error Response Structure

When an error occurs, implementations should throw/return an error with:



```
{
  errorType: String (one of the error types above)
  message: String (human-readable error message)
  details: Map<String, String> (optional - additional error details)
}
```

Example:



```
{
  errorType: "CONFLICT"
  message: "Username already exists"
  details: {
    "field": "username",
    "value": "john_doe"
  }
}
```

Implementation Checklist

Tenant Service

- ☐ Implement `getTenantByIdentity(identityType, identity)`
- ☐ Implement `getTenantByUsername(username)`
- ☐ Ensure unique constraints on identity + identityType
- ☐ Ensure unique constraints on username
- ☐ Return proper error types

OTP Service

- ☐ Implement `generateAndSaveOtp(otpType, otpLength, minNumber, minAlphabet, maxNumber, maxAlphabet, expiresAfterSecond, context, metadata)`
- ☐ Use cryptographically secure random generator
- ☐ Implement automatic cleanup of expired OTPs
- ☐ Ensure id + context uniqueness
- ☐ Return proper error types

Notifier Service

- ☐ Implement `sendNotification(identityType, identity, subject, content)`
 - ☐ Configure email SMTP settings
 - ☐ Implement retry logic for failed sends
 - ☐ Log all sent notifications
 - ☐ Return proper error types
-

Security Requirements

Password Storage (Tenant Service)

- NEVER store passwords in plain text
- Use bcrypt (cost factor 10+) or argon2id
- Example bcrypt hash: \$2a\$10\$N9qo8uL0ickgx2ZMRZoMye...

OTP Generation (OTP Service)

- Use SecureRandom (Java) or secrets module (Python) or equivalent
- NEVER use predictable patterns or weak random generators
- OTPs should be unpredictable

Data Encryption (All Services)

- Use TLS/HTTPS for all network communication (when decided)
- Encrypt sensitive data at rest (passwords, OTPs in database)
- Use AES-256 or equivalent

Rate Limiting

- Implement rate limiting on OTP generation (max 5 per identity per hour)
- Implement rate limiting on OTP validation (max 5 attempts per OTP)
- Implement rate limiting on notification sending (max 10 per identity per hour)

Testing Requirements

Each service implementation should include:

1. **Unit Tests**
 - Test each function with valid inputs
 - Test each function with invalid inputs
 - Test error cases
 - Test edge cases (expired OTPs, duplicate usernames, etc.)
2. **Integration Tests**
 - Test service with real database
 - Test service with real email provider (or mock)
3. **Performance Tests**
 - OTP generation should complete in < 100ms
 - Tenant lookup should complete in < 50ms
 - Email sending should not block (async)

Future Enhancements

The following features may be added in future versions:

- SMS notifications (IdentityType.SMS)
- Push notifications (IdentityType.PUSH)
- Multi-factor authentication
- OTP resend functionality
- Tenant password reset
- Tenant profile updates
- Audit logging
- Rate limiting service

End of Document