Service Contracts

Description of Services in the System

Team : CacheME

Project : Secure File Sharing Platform

Version 2

Table of Contents

1. Overview

This document defines the service contracts between major components of the secure file sharing platform. The contracts specify APIs, data formats, communication protocols, and error handling to ensure reliable integration between services.

2. API Gateway Service

API Specification

- **Base URL**: /api
- **Protocol**: REST over HTTP/HTTPS
- **Authentication**: JWT in Authorization header

NOTE: The parts lead to the service endpoints discussed in other sections.

Routes

```
/api/users/*
/api/files/*
/api/contact/*
/api/vault/*
/api/notifications/*
/api/health
```

Data Format

- **Request/Response**: JSON
- **Error Responses**:

-
  ```
  {
    "success": false,
    "message": "Error description",
  ```

```
  "code": "ERROR_CODE"
}
```

Error Handling

- **400**: Bad Request
- **401**: Unauthorized
- **403**: Forbidden
- **404**: Not Found
- **500**: Internal Server Error
- **Timeout**: 30 seconds

3. File Service

API Specification

- **Base URL**: `http://file-service:8081` (or via API Gateway)
- **Protocol**: REST over HTTP/HTTPS

Endpoints

- **Notification handling**
  - `POST /notifications` - gets all user notifications
  - `POST /notifications/markAsRead` - marks a specific notification as read
  - `POST /notifications/respond` - adds a rejected or accepted response for a file transfer.
  - `POST /notifications/clear` - clears all the user's notifications
  - `POST /notifications/add` - adds a new notification to a user.
- **File Metadata**
  - `POST /metadata` - gets metadata for a specific user
  - `POST /getFileMetadata` - gets file metadata for a specific user
  - `POST /getNumberOfFiles` - gets the number of files associated with a specific user.

- `POST /addPendingFiles` - adds a sent file into partially received (receiver can reject the file).
- `POST /getPendingFiles` - gets all the files that are partially received.
- `POST /deleteFile` - permanently deletes a file from server
- `POST /addTags` - adds tags about a file such as received etc.
- `POST /addUser` - adds a specific user to the user file sharing database.
- `POST /removeTags` - removes tags relating to specific file.

- **File Send and receive**
  - `POST /send` - Send file to recipient
  - `POST /sendByView` - Send view-only file
  - `POST /download` - downloads a specific file to user computer
  - `POST /downloadSentFile` - Download sent file
  - `POST /downloadViewFile` - Download view-only file
  - `POST /getSharedViewFiles`- gets view only files for specific user.
  - `POST /addSentFiles`- adds a sent file for specific receiver
  - `POST /getSentFiles`- gets sent files for specific receiver
  - `POST /changeMethod`- changes a view-only file to full access and vice-versa.
- **File access and revocation**
  - `POST /addAccesslog` - adds access log to a specific file
  - `POST /getAccesslog` - gets access log for specific file
  - `POST /revokeViewAccess` - revokes access to a specific file for a specific user.
  - `POST /getViewFileAccessLogs` - gets access logs for a view-only file
- **Folder Creation and upload**
  - `POST /createFolder` - Create folder
  - `POST /updateFilePath` - Move file
  - `POST /upload` - uploads a file to storage

Data Format

- **File Uploads**: Multipart/form-data
- **Other Requests**: JSON
- **Headers**:
  - `x-nonce`: Base64-encoded nonce for file encryption
  - `Content-Type`: `application/json` or `multipart/form-data`

Example Request (Send File)

```
POST /api/files/send
Content-Type: multipart/form-data

form-data:
  fileid: "file123"
  userId: "user123"
  recipientUserId: "user456"
  metadata: JSON.stringify({
    fileNonce: "base64...",
    keyNonce: "base64...",
    ikPublicKey: "base64...",
    spkPublicKey: "base64...",
    ekPublicKey: "base64...",
    opk_id: "opk123",
    encryptedAesKey: "base64...",
    signature: "base64...",
    fileHash: "base64...",
    viewOnly: false
  })
  encryptedFile: <binary data>
```

Error Handling

- **400**: Missing required fields
- **403**: Access denied (for view-only files)
- **404**: File not found
- **500**: Internal server error
- **Timeout**: 60 seconds for file operations

4. Vault Service (Key Management)

API Specification

- **Base URL**: `http://vault-service:8443` (or via API Gateway)
- **Protocol**: REST over HTTP/HTTPS

Endpoints

- `GET /health` - Service health check

- `POST /store-key` - Store key bundle
- `GET /retrieve-key` - Retrieve key bundle
- `DELETE /delete-key` - Delete key bundle

Data Format

- **Request/Response**: JSON
- **Key Bundle Structure**:

-
  ```
  {
    "encrypted_id": "user123",
    "spk_private_key": "base64...",
    "ik_private_key": "base64...",
    "opks_private": [
      {"opk_id": "opk1", "private_key": "base64..."},
      ...
    ]
  }
  ```

Error Handling

- **400**: Invalid key bundle
- **404**: Key not found
- **500**: Vault operation failed
- **Timeout**: 10 seconds

5. User Service

API Specification

- **Base URL**: `http://user-service:3000` (or via API Gateway)
- **Protocol**: REST over HTTP/HTTPS

Endpoints

Method	Endpoint	Auth Required	Description
POST	/register	No	Register a new user
POST	/login	No	User login
POST	/logout	Yes	User logout
GET	/profile	Yes	Get user profile
DELETE	/profile	Yes	Delete user profile
POST	/token_refresh	Yes	Refresh authentication token
PUT	/profile	Yes	Update user profile
POST	/verify-password	Yes	Verify user password
POST	/send-reset-pin	Yes	Send password reset PIN
POST	/change-password	Yes	Change user password
GET	/public-keys/:userId	Yes	Get public keys for a user
GET	/getUserId/:email	Yes	Get user ID from email
POST	/get-token	Yes	Get user token
GET	/token-info	Yes	Get user info from token
GET	/notifications	Yes	Get notification settings
PUT	/notifications	Yes	Update notification settings
POST	/avatar-url	Yes	Update avatar URL

Data Format

- **Request/Response**: JSON
- **Registration Example**:

-
  ```
  {
    "username": "user1",
    "email": "user@example.com",
    "password": "securepassword",
    "ik_public": "base64...",
    "spk_public": "base64...",
    "opks_public": ["base64..."],
    "nonce": "base64...",
    "signedPrekeySignature": "base64...",
    "salt": "base64...",
    "ik_private_key": "base64...",
    "spk_private_key": "base64...",
    "opks_private": ["base64..."]
  }
  ```

Error Handling

- **400**: Missing required fields
- **409**: User already exists
- **500**: Database operation failed
- **Timeout**: 15 seconds

5. Admin Service

API Specification

- **Base URL**: `http://user-service:3000` (or via API Gateway)
- **Protocol**: REST over HTTP/HTTPS

Endpoints

Method	Endpoint	Auth Required	Description
POST	/login	No	Admin login
GET	/dashboard/stats	Yes	Get dashboard statistics (active users, blocked users, reports)
GET	/announcements	Yes	Get all announcements
POST	/announcements	Yes	Create a new announcement
DELETE	/announcements/:id	Yes	Delete an announcement
GET	/announcements/user/:email	Yes	Get announcements created by a specific admin
GET	/users	Yes	Get all users (for admin dashboard)
GET	/users/blocked	Yes	Get blocked users
GET	/reports/pending	Yes	Get pending user reports
PUT	/users/:id/block	Yes	Block a user
PUT	/users/:id/unblock	Yes	Unblock a user

Data Format

- **Request/Response**: JSON
- **Login Example**:

```
{
  "email": "user@example.com",
  "password": "securepassword",
}
```

- **Dashboard Stats Example:**
```
{
        "success": true,
        "stats":
        {
                "totalUsers": 1234,
                "blockedUsers": 12,
                "pendingReports": 5
        }
}
```

Error Handling

- **400**: Missing required fields
- **409**: User already exists
- **500**: Database operation failed
- **Timeout**: 15 seconds

6. Cross-Service Communication

Protocols

- **Primary**: REST (JSON)
- **Alternative**: gRPC (for performance-critical paths)

Data Flow Examples

File Sharing Flow

1. UI -> API Gateway: `POST /api/files/download`
2. API Gateway -> File Service: Download encrypted file
3. UI -> API Gateway: `GET /api/users/public-keys/{recipientId}`
4. UI -> API Gateway: `POST /api/files/send` (with encrypted payload)
5. API Gateway -> File Service: Store sent file
6. File Service -> Metadata DB: Record transaction

Key Storage Flow

1. UI -> API Gateway: `POST /api/vault/store-key`
2. API Gateway -> Vault Service: stores key bundle
3. Vault Service -> HashiCorp Vault: stores secrets

Key Retrieval Flow

1. UI -> API Gateway: `GET /api/vault/retrieve-key`
2. API Gateway -> Vault Service: Retrieve key bundle
3. Vault Service -> HashiCorp Vault: Get secrets

Register Flow

1. UI -> API Gateway: `POST /api/users/register`
2. API Gateway -> Supabase: stores the credentials
3. Then Key Storage Flow

Login Flow

1. UI -> API Gateway: `POST /api/users/register`
2. API Gateway -> Supabase: retrieves the credentials
3. Then Key Retrieval Flow

7. Testing Requirements

Each service contract must be verified with:

1. Unit tests for individual endpoints
2. Integration tests for cross-service workflows
3. Performance tests for file operations
4. Security tests for encryption/decryption flows

Test cases should verify:

- Correct data formats
- Proper error handling
- Authentication/authorization
- Encryption/decryption correctness
- Performance under load

8. Versioning

All APIs would follow semantic versioning (v1, v2, etc.) with:

- Version in URL path (/v1/api/files)
- Backward compatibility for at least one previous version
- Deprecation notices in documentation