📘 API Documentation: Authentication and User Routes (authRoutes.js)

This document provides a complete A-Z description of all functions and route handlers established in the `authRoutes.js` file of the Secure File Transfer Application.

🔐 Authentication Flow

1. Register a New User `POST /register`

- **Validates input** using `express-validator`
- **Generates RSA key pair** for encryption
- **Hashes the password** using bcrypt
- **Temporarily stores user data and OTP** in memory (otpStorage)
- **Sends OTP via email** using Nodemailer

2. Verify OTP and Save User `POST /verify-registration-otp`

- Validates OTP
- Retrieves stored data from `otpStorage`
- Saves new user to the MongoDB `User` collection

3. Login `POST /login`

- Accepts email or username
- Validates password
- Generates and emails an OTP
- Stores OTP temporarily in memory

4. Verify Login OTP `POST /verify-login-otp`

- Validates OTP
- If successful, generates a JWT token
- Sets default `privacySetting` to `public` if not already set

5. Update Public Key `PATCH /update-public-key`

- Allows users to upload a new RSA public key

👤 User Profile

6. Get Profile `GET /profile`

- Returns user data (excluding password)
- Also resolves profile picture from the `uploads/profile-pictures/` directory

7. Update Profile `PUT /profile`

- Allows update of user fields: `email`, `username`, `name`, `mobileNumber`, `password`
- Validates data

8. Upload Profile Picture `POST /upload-profile-picture`

- Accepts image files (jpeg, jpg, png, gif)
- Deletes old profile picture
- Stores new image in `/uploads/profile-pictures/`
- Updates `User.profilePicture` path

9. Update Privacy Setting `PATCH /update-privacy`

- Updates `privacySetting` field to either `public` or `private`
- Creates a notification about the update

🔍 Search and Discovery

10. Search Users `GET /search`

- Searches users by `name`, `username`, or `email` (case-insensitive regex)

🔔 Notifications

11. Get Notifications `GET /notifications`

- Returns the 25 most recent notifications for the user

🧰 Utilities & Helpers

🔹 OTP Generation

```
function generateOTP() { return Math.floor(100000 + Math.random() * 900000);
}
```

🔹 RSA Key Pair Generation

```
function generateRSAKeys() { ... }
```

🔹 OTP Email Sender

```
async function sendOTP(email, otp, message) { ... }
```

🔹 Multer Configuration

- Handles profile picture uploads
- Deletes old picture based on filename match

🔹 Rate Limiting

- `authLimiter` limits requests to prevent brute-force attacks

✅ Middleware Used

- **`verifyToken`**: Validates JWT for protected routes
- **`express-validator`**: Ensures form field validity
- **`bcryptjs`**: Password hashing
- **`jsonwebtoken`**: Token-based authentication
- **`nodemailer`**: Email OTP delivery
- **`multer`**: File uploads (profile pictures)

📝 Notes

- OTPs are stored **in-memory** in `otpStorage` and expire manually (no automatic expiry logic here).
- RSA keys are generated per user at registration time.
- Private keys are not stored on the server (security best practice).

📁 API Documentation: File Management Routes (fileRoutes.js)

This document provides a complete A-Z reference for all file-related API endpoints in the **Secure File Transfer Application**. These routes handle encryption, secure uploads, downloads, file management (trash, restore, permanent delete), sharing, and access control.

🔐 Core Security Concepts Implemented

- **AES-256 CBC Encryption**: All files are encrypted before being stored using AES symmetric encryption.
- **RSA Key Wrapping**: The AES key is encrypted using the recipient's RSA public key.
- **SHA-256 Hashing**: Used to ensure file integrity and detect tampering.
- **Chunked Uploads**: Files are segmented and encrypted per chunk for large file support.
- **2FA-Integrated Access**: JWT-based authentication using `verifyToken` middleware.
- **Storage Monitoring**: User storage usage is calculated and warnings sent via notifications.

📤 Upload File

POST /api/file/upload

- Accepts a file upload using `multer`.
- Segments the file into 256 KB chunks and encrypts each using AES.
- Generates a random AES key and IV.
- Encrypts AES key with:
  - Uploader's RSA public key.
  - Each friend's RSA public key (if shared with friends or close friends).
- Stores encrypted chunks in `fileChunks`.
- Computes SHA-256 hash of original file for integrity checks.
- Sends upload success notification to uploader.

📥 Download File

POST /api/file/download/:fileId

- Requires user's private RSA key in the body.
- Verifies if user has access via:
  - Ownership
  - Visibility (public/friends/closeFriends)
  - Encrypted key availability
- Decrypts AES key using RSA private key.
- Decrypts all file chunks and reassembles into full file.
- Verifies integrity using SHA-256.
- Sends download success notification to file owner.
- Logs file in `downloadedFiles` of downloader.

✅ Verify File Integrity

GET /api/file/verify-file/:fileId

- Reads file from disk.
- Recalculates SHA-256 hash.
- Compares against stored hash.
- If mismatch is found:
  - Notifies the file owner and all shared users.
- Returns success or integrity compromised message.

📦 Get All Files (Accessible to User)

GET /api/file/files?userId=targetUserId

- Fetches files based on:
  - Ownership
  - Public visibility
  - Explicit sharing
  - Friend/close friend status
- Ensures duplicate files are removed before sending.
- Returns metadata only (no file content).

📉 Storage Usage

GET /api/file/storage-usage

- Calculates total uploaded and downloaded file sizes.
- Warns user via notification if usage exceeds 90% of 5MB limit.
- Returns current usage and limit info.

📥 Downloaded Files History

GET /api/file/downloaded

- Returns metadata of all files downloaded by the user.

🔐 Re-Encrypt Files for New Friend

POST /api/file/re-encrypt-for-friend/:friendId

- Accepts user's private RSA key.
- Finds all files owned by the user and re-encrypts the AES key for the new friend.
- Adds encrypted AES key under `encryptedKeysForRecipients` field.

🗑 Soft Delete File (Move to Trash)

PUT /api/file/soft-delete/:fileId

- If user is file owner:
  - Adds file to `removedUploads`
- If file was downloaded:
  - Removes from `downloadedFiles`
  - Adds to `removedDownloads`

♻️ Restore File from Trash

PUT /api/file/restore/:fileId

- Removes file from both `removedUploads` and `removedDownloads` arrays.
- Optionally re-adds to `downloadedFiles`.

❌ Permanently Delete File

DELETE /api/file/permanent-delete/:fileId

- If user is owner:
  - Deletes file from disk and DB.
  - Removes file reference from all users.
- If user is not owner:
  - Removes from `removedDownloads` only.

🗃️ Get Trashed Files

GET /api/file/trash

- Retrieves both uploaded and downloaded files marked for deletion.
- De-duplicates and sends simplified metadata.

🤝 API Documentation: Friend System (friendRoutes.js)

This document describes the complete friend management system used in the Secure File Transfer Application. All routes are protected by JWT authentication (`verifyToken` middleware).

📤 1. Send a Friend Request

POST `/api/friends/friend-request/:userId`

- Sends a friend request to a user.
- Prevents duplicate requests.
- Notifies the recipient with a `Friend Request` notification.

✅ 2. Accept a Friend Request

POST `/api/friends/accept-friend-request/:userId`

- Removes the incoming friend request from current user.
- Adds the requester to current user's friends list and vice versa.
- Sends a `Friend Request Accepted` notification.

❌ 3. Reject a Friend Request

POST `/api/friends/reject-friend-request/:userId`

- Simply removes the pending friend request.
- No notification sent.

🔍 4. Search for Users

GET `/api/friends/search?search=`

- Searches users by name, username, or email (case-insensitive).
- Returns a list of matched users excluding sensitive fields.

📄 5. View Friends List

GET `/api/friends/friends`

- Returns the authenticated user's full friends list.
- Uses `.populate()` to retrieve `username` and `email`.

🗑️ 6. Remove a Friend

DELETE `/api/friends/remove-friend/:userId`

- Removes the user from both sides (bi-directional delete).
- Updates both users' `friends` arrays.

⭐ 7. Toggle Close Friend Status

POST `/api/friends/add-close-friend/:userId`

- Adds or removes a user from `closeFriends` array.
- Only possible if the user is already a friend.
- Returns updated close friends list.

👤 8. Get a Friend's Profile

GET `/api/friends/profile/:userId`

- Returns profile metadata for a friend.
- Shows friend status:
  - `isFriend`, `isCloseFriend`, `hasRequestedYou`, `youRequested`
- If profile is private and not a friend, sensitive data is hidden.
- Used for rendering user cards or profile pages.

🔐 Notes on Access & Privacy

- Profiles respect user-defined privacy (`public` or `private`).
- Friend requests are stored in `friendRequests` with `{ userId }`.
- Relationships (friends/closeFriends) are stored directly in the `User` model for quick access.
- Notifications are triggered during key events.