**API Implementation Notes**

1. **Authentication**:
   * Use JWT (JSON Web Tokens) for stateless authentication
   * Implement token refresh mechanism to handle expired tokens
   * Set appropriate token expiration times (e.g., 1 hour for access tokens)
   * Store sensitive authentication data in secure HTTP-only cookies
2. **Security**:
   * Use input validation and sanitization for all user inputs
   * Apply proper authorization checks for all protected endpoints
   * Implement CORS policies to restrict API access to authorized domains
3. **Performance**:
   * Use pagination for all endpoints returning lists of items
   * Implement caching for frequently accessed data
   * Optimize database queries to minimize response times
   * Consider implementing GraphQL for complex data requirements
4. **Error Handling**:
   * Use consistent error response format across all endpoints
   * Include proper HTTP status codes (400 for client errors, 500 for server errors)
   * Provide meaningful error messages without exposing sensitive information
   * Implement logging for all API errors for debugging
5. **Media Handling**:
   * Set file size limits for uploaded media
   * Validate file types and formats before processing
   * Generate multiple sizes/qualities of uploaded images for different contexts
   * Implement secure file upload mechanisms
6. **Real-time Features**:
   * Use WebSockets for real-time notifications and messaging
   * Implement proper reconnection strategies for socket connections
   * Consider using server-sent events for one-way communications# Social Media Website Technical Specification

**Database Structure**

**users**

| **Column** | **Description** |
| --- | --- |
| user\_id | Primary key, auto-incrementing integer |
| username | Unique username, varchar(50) |
| email | Unique email address, varchar(100) |
| password | Hashed password, varchar(255) |
| first\_name | User's first name, varchar(50) |
| last\_name | User's last name, varchar(50) |
| profile\_picture | Path to profile image file, varchar(255), nullable |
| bio | User's bio/about text, text, nullable |
| date\_of\_birth | Date of birth, date |
| registration\_date | Account creation timestamp, datetime |
| last\_login | Last login timestamp, datetime, nullable |
| account\_status | Account status (active, suspended, deleted), varchar(20) |
| role | User role (user, admin), varchar(20) |

**Constraints:**

* Primary Key: user\_id
* Unique: username, email
* Not Null: username, email, password, first\_name, last\_name, date\_of\_birth, registration\_date, account\_status, role
* Check: account\_status in ('active', 'suspended', 'deleted')
* Check: role in ('user', 'admin')
* Index: username, email (for fast authentication lookups)

**posts**

| **Column** | **Description** |
| --- | --- |
| post\_id | Primary key, auto-incrementing integer |
| user\_id | Foreign key to users table, integer |
| caption | Post text content, text, nullable |
| post\_type | Type of post (text, photo, video), varchar(20) |
| media\_url | Path to media file, varchar(255), nullable |
| created\_at | Post creation timestamp, datetime |
| updated\_at | Post update timestamp, datetime |
| privacy\_level | Privacy setting (public, friends, private), varchar(20) |
| location\_lat | Latitude coordinate, decimal(10,8), nullable |
| location\_lng | Longitude coordinate, decimal(10,8), nullable |
| location\_name | Text description of location, varchar(255), nullable |

**Constraints:**

* Primary Key: post\_id
* Foreign Key: user\_id references users(user\_id) ON DELETE CASCADE
* Not Null: user\_id, post\_type, created\_at, updated\_at, privacy\_level
* Check: post\_type in ('text', 'photo', 'video')
* Check: privacy\_level in ('public', 'friends', 'private')
* Check: For photo/video posts, media\_url must not be null
* Index: user\_id (for faster retrieval of user posts)
* Index: created\_at (for feed sorting)

**comments**

| **Column** | **Description** |
| --- | --- |
| comment\_id | Primary key, auto-incrementing integer |
| post\_id | Foreign key to posts table, integer |
| user\_id | Foreign key to users table, integer |
| content | Comment text content, text |
| created\_at | Comment creation timestamp, datetime |
| updated\_at | Comment update timestamp, datetime |
| parent\_comment\_id | Self-referencing foreign key for nested comments, integer, nullable |

**Constraints:**

* Primary Key: comment\_id
* Foreign Key: post\_id references posts(post\_id) ON DELETE CASCADE
* Foreign Key: user\_id references users(user\_id) ON DELETE CASCADE
* Foreign Key: parent\_comment\_id references comments(comment\_id) ON DELETE CASCADE
* Not Null: post\_id, user\_id, content, created\_at, updated\_at
* Index: post\_id (for retrieving comments on a post)
* Index: parent\_comment\_id (for retrieving replies)

**likes**

| **Column** | **Description** |
| --- | --- |
| like\_id | Primary key, auto-incrementing integer |
| post\_id | Foreign key to posts table, integer, nullable |
| comment\_id | Foreign key to comments table, integer, nullable |
| user\_id | Foreign key to users table, integer |
| created\_at | Like creation timestamp, datetime |

**Constraints:**

* Primary Key: like\_id
* Foreign Key: post\_id references posts(post\_id) ON DELETE CASCADE
* Foreign Key: comment\_id references comments(comment\_id) ON DELETE CASCADE
* Foreign Key: user\_id references users(user\_id) ON DELETE CASCADE
* Not Null: user\_id, created\_at
* Check: Either post\_id or comment\_id must be not null, but not both
* Unique: Combination of (user\_id, post\_id) to prevent duplicate likes
* Unique: Combination of (user\_id, comment\_id) to prevent duplicate likes
* Index: post\_id (for counting likes on a post)
* Index: comment\_id (for counting likes on a comment)

**friendships**

| **Column** | **Description** |
| --- | --- |
| friendship\_id | Primary key, auto-incrementing integer |
| user\_id\_1 | Foreign key to users table (requestor), integer |
| user\_id\_2 | Foreign key to users table (recipient), integer |
| status | Friendship status (pending, accepted), varchar(20) |
| created\_at | Friendship request timestamp, datetime |
| updated\_at | Friendship status update timestamp, datetime |

**Constraints:**

* Primary Key: friendship\_id
* Foreign Key: user\_id\_1 references users(user\_id) ON DELETE CASCADE
* Foreign Key: user\_id\_2 references users(user\_id) ON DELETE CASCADE
* Not Null: user\_id\_1, user\_id\_2, status, created\_at, updated\_at
* Check: status in ('pending', 'accepted')
* Check: user\_id\_1 != user\_id\_2 (prevent self-friendships)
* Unique: Combination of (user\_id\_1, user\_id\_2) to prevent duplicate friendships
* Index: user\_id\_1, user\_id\_2 (for faster friendship lookups)

**blocks**

| **Column** | **Description** |
| --- | --- |
| block\_id | Primary key, auto-incrementing integer |
| blocker\_id | Foreign key to users table (user who blocked), integer |
| blocked\_id | Foreign key to users table (user who was blocked), integer |
| created\_at | Block creation timestamp, datetime |
| reason | Reason for blocking, text, nullable |

**Constraints:**

* Primary Key: block\_id
* Foreign Key: blocker\_id references users(user\_id) ON DELETE CASCADE
* Foreign Key: blocked\_id references users(user\_id) ON DELETE CASCADE
* Not Null: blocker\_id, blocked\_id, created\_at
* Check: blocker\_id != blocked\_id (prevent self-blocks)
* Unique: Combination of (blocker\_id, blocked\_id) to prevent duplicate blocks
* Index: blocker\_id, blocked\_id (for faster block status checks)

**reports**

| **Column** | **Description** |
| --- | --- |
| report\_id | Primary key, auto-incrementing integer |
| reporter\_id | Foreign key to users table (user who reported), integer |
| reported\_id | Foreign key to users table (user who was reported), integer |
| content\_type | Type of content reported (user, post, comment), varchar(20) |
| content\_id | ID of the reported content, integer |
| reason | Category of report, varchar(100) |
| description | Detailed description of report, text, nullable |
| created\_at | Report creation timestamp, datetime |
| status | Report status (pending, reviewed, action\_taken, dismissed), varchar(20) |
| admin\_notes | Notes from administrator review, text, nullable |
| reviewed\_by | Foreign key to users table (admin who reviewed), integer, nullable |
| reviewed\_at | Report review timestamp, datetime, nullable |

**Constraints:**

* Primary Key: report\_id
* Foreign Key: reporter\_id references users(user\_id) ON DELETE CASCADE
* Foreign Key: reported\_id references users(user\_id) ON DELETE CASCADE
* Foreign Key: reviewed\_by references users(user\_id) ON DELETE SET NULL
* Not Null: reporter\_id, reported\_id, content\_type, content\_id, reason, created\_at, status
* Check: content\_type in ('user', 'post', 'comment')
* Check: status in ('pending', 'reviewed', 'action\_taken', 'dismissed')
* Index: reporter\_id, reported\_id (for viewing user reports)
* Index: status (for admin filtering)
* Index: content\_type, content\_id (for finding reports on specific content)

**conversations**

| **Column** | **Description** |
| --- | --- |
| conversation\_id | Primary key, auto-incrementing integer |
| created\_at | Conversation creation timestamp, datetime |
| updated\_at | Conversation update timestamp, datetime |
| is\_group\_chat | Whether this is a group conversation, boolean |
| group\_name | Name for group conversations, varchar(100), nullable |

**Constraints:**

* Primary Key: conversation\_id
* Not Null: created\_at, updated\_at, is\_group\_chat
* Check: If is\_group\_chat is true, then group\_name should not be null
* Index: updated\_at (for sorting conversations by most recent activity)

**conversation\_participants**

| **Column** | **Description** |
| --- | --- |
| id | Primary key, auto-incrementing integer |
| conversation\_id | Foreign key to conversations table, integer |
| user\_id | Foreign key to users table, integer |
| joined\_at | Timestamp when user joined conversation, datetime |
| left\_at | Timestamp when user left conversation, datetime, nullable |

**Constraints:**

* Primary Key: id
* Foreign Key: conversation\_id references conversations(conversation\_id) ON DELETE CASCADE
* Foreign Key: user\_id references users(user\_id) ON DELETE CASCADE
* Not Null: conversation\_id, user\_id, joined\_at
* Unique: Combination of (conversation\_id, user\_id) where left\_at is null (to prevent duplicate active participants)
* Index: conversation\_id (for retrieving participants)
* Index: user\_id (for retrieving user's conversations)

**messages**

| **Column** | **Description** |
| --- | --- |
| message\_id | Primary key, auto-incrementing integer |
| conversation\_id | Foreign key to conversations table, integer |
| sender\_id | Foreign key to users table, integer |
| content | Message text content, text |
| created\_at | Message creation timestamp, datetime |
| is\_read | Whether message has been read, boolean |
| read\_at | Timestamp when message was read, datetime, nullable |

**Constraints:**

* Primary Key: message\_id
* Foreign Key: conversation\_id references conversations(conversation\_id) ON DELETE CASCADE
* Foreign Key: sender\_id references users(user\_id) ON DELETE CASCADE
* Not Null: conversation\_id, sender\_id, content, created\_at, is\_read
* Check: If is\_read is true, then read\_at should not be null
* Index: conversation\_id, created\_at (for retrieving messages in chronological order)
* Index: sender\_id (for retrieving user's sent messages)

**notifications**

| **Column** | **Description** |
| --- | --- |
| notification\_id | Primary key, auto-incrementing integer |
| recipient\_id | Foreign key to users table (user receiving notification), integer |
| sender\_id | Foreign key to users table (user triggering notification), integer |
| type | Notification type (like, comment, friend\_request, etc.), varchar(50) |
| related\_content\_type | Type of related content (post, comment, etc.), varchar(20) |
| related\_content\_id | ID of related content, integer |
| created\_at | Notification creation timestamp, datetime |
| is\_read | Whether notification has been read, boolean |
| read\_at | Timestamp when notification was read, datetime, nullable |

**Constraints:**

* Primary Key: notification\_id
* Foreign Key: recipient\_id references users(user\_id) ON DELETE CASCADE
* Foreign Key: sender\_id references users(user\_id) ON DELETE CASCADE
* Not Null: recipient\_id, sender\_id, type, related\_content\_type, related\_content\_id, created\_at, is\_read
* Check: type in ('like', 'comment', 'friend\_request', 'friend\_accept', 'mention', 'tag')
* Check: related\_content\_type in ('post', 'comment', 'user', 'message')
* Check: If is\_read is true, then read\_at should not be null
* Index: recipient\_id, is\_read (for retrieving unread notifications)
* Index: created\_at (for chronological ordering)

**Database Implementation Notes**

1. **Security**:
   * Store passwords using bcrypt or Argon2 hashing
   * Implement proper SQL injection prevention (prepared statements, ORM)
   * Use HTTPS for all API communications
   * Implement CSRF protection for form submissions
2. **Scalability**:
   * Consider partitioning large tables (posts, messages) by date
   * Implement caching strategies for frequently accessed data
   * Use database connection pooling
3. **Data Integrity**:
   * Set appropriate cascade delete rules to maintain referential integrity
   * Use transactions for operations affecting multiple tables
   * Implement database triggers for time-sensitive updates
4. **Performance**:
   * Create composite indexes for frequently used query patterns
   * Consider using read replicas for heavy read operations
   * Implement query optimization strategies
5. **File Storage**:
   * Store media files in a dedicated file system or cloud storage service (not in database)
   * Implement file size limits and type validation
   * Consider using CDN for media delivery

**API Endpoints**

**Authentication Endpoints**

**Register New User**

* **Purpose**: Creates a new user account
* **Access**: POST /api/auth/register
* **Request Data**:
  + username (string, required)
  + email (string, required)
  + password (string, required)
  + first\_name (string, required)
  + last\_name (string, required)
  + date\_of\_birth (date, required)
* **Response Data**:
  + success (boolean)
  + message (string)
  + user\_id (integer)

**User Login**

* **Purpose**: Authenticates user and returns access token
* **Access**: POST /api/auth/login
* **Request Data**:
  + email/username (string, required)
  + password (string, required)
* **Response Data**:
  + success (boolean)
  + token (string)
  + user\_id (integer)
  + expiration (datetime)

**User Logout**

* **Purpose**: Invalidates current user token
* **Access**: POST /api/auth/logout
* **Request Data**:
  + token (string, required)
* **Response Data**:
  + success (boolean)
  + message (string)

**Forgot Password**

* **Purpose**: Initiates password reset process
* **Access**: POST /api/auth/forgot-password
* **Request Data**:
  + email (string, required)
* **Response Data**:
  + success (boolean)
  + message (string)

**Reset Password**

* **Purpose**: Updates password using reset token
* **Access**: POST /api/auth/reset-password
* **Request Data**:
  + token (string, required)
  + new\_password (string, required)
* **Response Data**:
  + success (boolean)
  + message (string)

**Verify Email**

* **Purpose**: Verifies user email using token
* **Access**: GET /api/auth/verify-email/:token
* **Request Data**:
  + token (URL parameter, required)
* **Response Data**:
  + success (boolean)
  + message (string)

**User Endpoints**

**Get Current User Profile**

* **Purpose**: Retrieves current user's profile data
* **Access**: GET /api/users/me
* **Request Data**:
  + None (uses authentication token)
* **Response Data**:
  + user\_id (integer)
  + username (string)
  + email (string)
  + first\_name (string)
  + last\_name (string)
  + profile\_picture (string)
  + bio (string)
  + date\_of\_birth (date)
  + registration\_date (datetime)
  + account\_status (string)

**Get User Profile**

* **Purpose**: Retrieves specific user's profile data
* **Access**: GET /api/users/:userId
* **Request Data**:
  + userId (URL parameter, required)
* **Response Data**:
  + user\_id (integer)
  + username (string)
  + first\_name (string)
  + last\_name (string)
  + profile\_picture (string)
  + bio (string)
  + friendship\_status (string, if applicable)

**Update User Profile**

* **Purpose**: Updates current user's profile information
* **Access**: PUT /api/users/me
* **Request Data**:
  + first\_name (string, optional)
  + last\_name (string, optional)
  + bio (string, optional)
  + profile\_picture (file, optional)
* **Response Data**:
  + success (boolean)
  + message (string)
  + updated\_fields (array)

**Update User Password**

* **Purpose**: Updates current user's password
* **Access**: PUT /api/users/me/password
* **Request Data**:
  + current\_password (string, required)
  + new\_password (string, required)
* **Response Data**:
  + success (boolean)
  + message (string)

**Get User Posts**

* **Purpose**: Retrieves posts from a specific user
* **Access**: GET /api/users/:userId/posts
* **Request Data**:
  + userId (URL parameter, required)
  + page (integer, optional)
  + limit (integer, optional)
* **Response Data**:
  + posts (array of post objects)
  + total\_posts (integer)
  + current\_page (integer)
  + total\_pages (integer)

**Search Users**

* **Purpose**: Searches for users by name or username
* **Access**: GET /api/users/search
* **Request Data**:
  + query (string, required)
  + page (integer, optional)
  + limit (integer, optional)
* **Response Data**:
  + users (array of user objects)
  + total\_results (integer)
  + current\_page (integer)
  + total\_pages (integer)

**Get User Suggestions**

* **Purpose**: Gets suggested users for friendship
* **Access**: GET /api/users/suggestions
* **Request Data**:
  + limit (integer, optional)
* **Response Data**:
  + users (array of user objects)
  + mutual\_friends\_count (integer per user)

**Post Endpoints**

**Get Feed Posts**

* **Purpose**: Retrieves posts for user's feed
* **Access**: GET /api/posts
* **Request Data**:
  + page (integer, optional)
  + limit (integer, optional)
  + filter (string, optional)
* **Response Data**:
  + posts (array of post objects with user info)
  + total\_posts (integer)
  + current\_page (integer)
  + total\_pages (integer)

**Get Specific Post**

* **Purpose**: Retrieves a specific post with details
* **Access**: GET /api/posts/:postId
* **Request Data**:
  + postId (URL parameter, required)
* **Response Data**:
  + post (post object)
  + user (user object)
  + comments\_count (integer)
  + likes\_count (integer)
  + user\_has\_liked (boolean)

**Create Post**

* **Purpose**: Creates a new post
* **Access**: POST /api/posts
* **Request Data**:
  + caption (string, optional)
  + post\_type (string, required)
  + media (file, required for photo/video posts)
  + privacy\_level (string, optional)
  + location\_lat (decimal, optional)
  + location\_lng (decimal, optional)
  + location\_name (string, optional)
* **Response Data**:
  + success (boolean)
  + post\_id (integer)
  + message (string)

**Update Post**

* **Purpose**: Updates an existing post
* **Access**: PUT /api/posts/:postId
* **Request Data**:
  + postId (URL parameter, required)
  + caption (string, optional)
  + privacy\_level (string, optional)
  + location\_name (string, optional)
* **Response Data**:
  + success (boolean)
  + message (string)

**Delete Post**

* **Purpose**: Deletes a post
* **Access**: DELETE /api/posts/:postId
* **Request Data**:
  + postId (URL parameter, required)
* **Response Data**:
  + success (boolean)
  + message (string)

**Search Posts**

* **Purpose**: Searches for posts by content
* **Access**: GET /api/posts/search
* **Request Data**:
  + query (string, required)
  + page (integer, optional)
  + limit (integer, optional)
* **Response Data**:
  + posts (array of post objects)
  + total\_results (integer)
  + current\_page (integer)
  + total\_pages (integer)

**Media Upload Endpoint**

**Upload Media**

* **Purpose**: Uploads a media file and returns URL
* **Access**: POST /api/media/upload
* **Request Data**:
  + file (file, required)
  + type (string, required: profile\_picture, post\_media)
* **Response Data**:
  + success (boolean)
  + media\_url (string)
  + file\_type (string)
  + message (string)

**Comments Endpoints**

**Get Post Comments**

* **Purpose**: Retrieves comments for a post
* **Access**: GET /api/posts/:postId/comments
* **Request Data**:
  + postId (URL parameter, required)
  + page (integer, optional)
  + limit (integer, optional)
* **Response Data**:
  + comments (array of comment objects with user info)
  + total\_comments (integer)
  + current\_page (integer)
  + total\_pages (integer)

**Add Comment**

* **Purpose**: Adds a comment to a post
* **Access**: POST /api/posts/:postId/comments
* **Request Data**:
  + postId (URL parameter, required)
  + content (string, required)
  + parent\_comment\_id (integer, optional for replies)
* **Response Data**:
  + success (boolean)
  + comment\_id (integer)
  + message (string)

**Update Comment**

* **Purpose**: Updates an existing comment
* **Access**: PUT /api/comments/:commentId
* **Request Data**:
  + commentId (URL parameter, required)
  + content (string, required)
* **Response Data**:
  + success (boolean)
  + message (string)

**Delete Comment**

* **Purpose**: Deletes a comment
* **Access**: DELETE /api/comments/:commentId
* **Request Data**:
  + commentId (URL parameter, required)
* **Response Data**:
  + success (boolean)
  + message (string)

**Add Reply to Comment**

* **Purpose**: Adds a reply to an existing comment
* **Access**: POST /api/comments/:commentId/replies
* **Request Data**:
  + commentId (URL parameter, required)
  + content (string, required)
* **Response Data**:
  + success (boolean)
  + comment\_id (integer)
  + message (string)

**Likes Endpoints**

**Like Post**

* **Purpose**: Adds a like to a post
* **Access**: POST /api/posts/:postId/like
* **Request Data**:
  + postId (URL parameter, required)
* **Response Data**:
  + success (boolean)
  + message (string)
  + likes\_count (integer)

**Unlike Post**

* **Purpose**: Removes a like from a post
* **Access**: DELETE /api/posts/:postId/like
* **Request Data**:
  + postId (URL parameter, required)
* **Response Data**:
  + success (boolean)
  + message (string)
  + likes\_count (integer)

**Like Comment**

* **Purpose**: Adds a like to a comment
* **Access**: POST /api/comments/:commentId/like
* **Request Data**:
  + commentId (URL parameter, required)
* **Response Data**:
  + success (boolean)
  + message (string)
  + likes\_count (integer)

**Unlike Comment**

* **Purpose**: Removes a like from a comment
* **Access**: DELETE /api/comments/:commentId/like
* **Request Data**:
  + commentId (URL parameter, required)
* **Response Data**:
  + success (boolean)
  + message (string)
  + likes\_count (integer)

**Get Post Likes**

* **Purpose**: Gets users who liked a post
* **Access**: GET /api/posts/:postId/likes
* **Request Data**:
  + postId (URL parameter, required)
  + page (integer, optional)
  + limit (integer, optional)
* **Response Data**:
  + users (array of user objects)
  + total\_likes (integer)
  + current\_page (integer)
  + total\_pages (integer)

**Friendship Endpoints**

**Get Friends**

* **Purpose**: Gets current user's friends
* **Access**: GET /api/friends
* **Request Data**:
  + page (integer, optional)
  + limit (integer, optional)
* **Response Data**:
  + friends (array of user objects)
  + total\_friends (integer)
  + current\_page (integer)
  + total\_pages (integer)

**Get Friend Requests**

* **Purpose**: Gets pending friend requests
* **Access**: GET /api/friends/requests
* **Request Data**:
  + type (string, optional: incoming, outgoing)
* **Response Data**:
  + requests (array of friendship objects with user info)
  + total\_requests (integer)

**Send Friend Request**

* **Purpose**: Sends a friend request to a user
* **Access**: POST /api/friends/request/:userId
* **Request Data**:
  + userId (URL parameter, required)
* **Response Data**:
  + success (boolean)
  + message (string)
  + friendship\_id (integer)

**Accept Friend Request**

* **Purpose**: Accepts a friend request
* **Access**: PUT /api/friends/accept/:userId
* **Request Data**:
  + userId (URL parameter, required)
* **Response Data**:
  + success (boolean)
  + message (string)

**Reject Friend Request**

* **Purpose**: Rejects a friend request
* **Access**: PUT /api/friends/reject/:userId
* **Request Data**:
  + userId (URL parameter, required)
* **Response Data**:
  + success (boolean)
  + message (string)

**Remove Friend**

* **Purpose**: Removes a user from friends
* **Access**: DELETE /api/friends/:userId
* **Request Data**:
  + userId (URL parameter, required)
* **Response Data**:
  + success (boolean)
  + message (string)

**Block/Report Endpoints**

**Block User**

* **Purpose**: Blocks a user
* **Access**: POST /api/users/:userId/block
* **Request Data**:
  + userId (URL parameter, required)
  + reason (string, optional)
* **Response Data**:
  + success (boolean)
  + message (string)

**Unblock User**

* **Purpose**: Unblocks a previously blocked user
* **Access**: DELETE /api/users/:userId/block
* **Request Data**:
  + userId (URL parameter, required)
* **Response Data**:
  + success (boolean)
  + message (string)

**Get Blocked Users**

* **Purpose**: Gets list of users blocked by current user
* **Access**: GET /api/users/blocked
* **Request Data**:
  + page (integer, optional)
  + limit (integer, optional)
* **Response Data**:
  + users (array of user objects)
  + total\_blocked (integer)
  + current\_page (integer)
  + total\_pages (integer)

**Create Report**

* **Purpose**: Creates a new report for post, comment, or user
* **Access**: POST /api/reports
* **Request Data**:
  + reported\_id (integer, required)
  + content\_type (string, required: user, post, comment)
  + content\_id (integer, required)
  + reason (string, required)
  + description (string, optional)
* **Response Data**:
  + success (boolean)
  + report\_id (integer)
  + message (string)

**Get User Reports**

* **Purpose**: Gets reports filed by current user
* **Access**: GET /api/users/:userId/reports
* **Request Data**:
  + userId (URL parameter, required)
  + page (integer, optional)
  + limit (integer, optional)
* **Response Data**:
  + reports (array of report objects)
  + total\_reports (integer)
  + current\_page (integer)
  + total\_pages (integer)

**Messaging Endpoints**

**Get Conversations**

* **Purpose**: Gets user's conversations
* **Access**: GET /api/conversations
* **Request Data**:
  + page (integer, optional)
  + limit (integer, optional)
* **Response Data**:
  + conversations (array of conversation objects with participant info)
  + total\_conversations (integer)
  + current\_page (integer)
  + total\_pages (integer)

**Create Conversation**

* **Purpose**: Creates a new conversation
* **Access**: POST /api/conversations
* **Request Data**:
  + participants (array of user IDs, required)
  + is\_group\_chat (boolean, optional)
  + group\_name (string, optional for group chats)
* **Response Data**:
  + success (boolean)
  + conversation\_id (integer)
  + message (string)

**Get Conversation Details**

* **Purpose**: Gets details of a specific conversation
* **Access**: GET /api/conversations/:conversationId
* **Request Data**:
  + conversationId (URL parameter, required)
* **Response Data**:
  + conversation (conversation object)
  + participants (array of user objects)
  + is\_group\_chat (boolean)
  + group\_name (string, if applicable)
  + created\_at (datetime)

**Get Conversation Messages**

* **Purpose**: Gets messages in a conversation
* **Access**: GET /api/conversations/:conversationId/messages
* **Request Data**:
  + conversationId (URL parameter, required)
  + page (integer, optional)
  + limit (integer, optional)
  + before\_message\_id (integer, optional)
* **Response Data**:
  + messages (array of message objects with sender info)
  + total\_messages (integer)
  + current\_page (integer)
  + total\_pages (integer)

**Send Message**

* **Purpose**: Sends a message in a conversation
* **Access**: POST /api/conversations/:conversationId/messages
* **Request Data**:
  + conversationId (URL parameter, required)
  + content (string, required)
* **Response Data**:
  + success (boolean)
  + message\_id (integer)
  + sent\_at (datetime)

**Mark Message as Read**

* **Purpose**: Marks a message as read
* **Access**: PUT /api/messages/:messageId/read
* **Request Data**:
  + messageId (URL parameter, required)
* **Response Data**:
  + success (boolean)
  + message (string)

**Delete Message**

* **Purpose**: Deletes a message
* **Access**: DELETE /api/messages/:messageId
* **Request Data**:
  + messageId (URL parameter, required)
* **Response Data**:
  + success (boolean)
  + message (string)

**Notification Endpoints**

**Get Notifications**

* **Purpose**: Gets user's notifications
* **Access**: GET /api/notifications
* **Request Data**:
  + page (integer, optional)
  + limit (integer, optional)
  + filter (string, optional: unread, all)
* **Response Data**:
  + notifications (array of notification objects)
  + total\_notifications (integer)
  + unread\_count (integer)
  + current\_page (integer)
  + total\_pages (integer)

**Mark Notification as Read**

* **Purpose**: Marks a notification as read
* **Access**: PUT /api/notifications/:notificationId/read
* **Request Data**:
  + notificationId (URL parameter, required)
* **Response Data**:
  + success (boolean)
  + message (string)

**Mark All Notifications as Read**

* **Purpose**: Marks all notifications as read
* **Access**: PUT /api/notifications/read-all
* **Request Data**:
  + None
* **Response Data**:
  + success (boolean)
  + count (integer)
  + message (string)

**Get Unread Notification Count**

* **Purpose**: Gets count of unread notifications
* **Access**: GET /api/notifications/unread-count
* **Request Data**:
  + None
* **Response Data**:
  + count (integer)

**Privacy Endpoints**

**Get Privacy Settings**

* **Purpose**: Gets user's privacy settings
* **Access**: GET /api/privacy
* **Request Data**:
  + None
* **Response Data**:
  + privacy\_settings (object with various privacy options)

**Update Privacy Settings**

* **Purpose**: Updates user's privacy settings
* **Access**: PUT /api/privacy
* **Request Data**:
  + post\_default\_privacy (string, optional)
  + profile\_visibility (string, optional)
  + friend\_list\_visibility (string, optional)
  + who\_can\_send\_requests (string, optional)
* **Response Data**:
  + success (boolean)
  + message (string)
  + updated\_settings (array)

**Admin Endpoints**

**Get Admin Dashboard**

* **Purpose**: Gets admin dashboard statistics
* **Access**: GET /api/admin/dashboard
* **Request Data**:
  + time\_period (string, optional: day, week, month, year)
* **Response Data**:
  + user\_stats (object with user statistics)
  + post\_stats (object with post statistics)
  + report\_stats (object with report statistics)
  + active\_users (integer)
  + new\_registrations (integer)

**Get All Users**

* **Purpose**: Gets all users for admin
* **Access**: GET /api/admin/users
* **Request Data**:
  + page (integer, optional)
  + limit (integer, optional)
  + filter (string, optional: active, suspended)
  + sort (string, optional)
  + search (string, optional)
* **Response Data**:
  + users (array of user objects with detailed info)
  + total\_users (integer)
  + current\_page (integer)
  + total\_pages (integer)

**Update User (Admin)**

* **Purpose**: Updates a user's information as admin
* **Access**: PUT /api/admin/users/:userId
* **Request Data**:
  + userId (URL parameter, required)
  + first\_name (string, optional)
  + last\_name (string, optional)
  + email (string, optional)
  + role (string, optional)
* **Response Data**:
  + success (boolean)
  + message (string)
  + updated\_fields (array)

**Delete User**

* **Purpose**: Deletes a user account
* **Access**: DELETE /api/admin/users/:userId
* **Request Data**:
  + userId (URL parameter, required)
  + reason (string, optional)
* **Response Data**:
  + success (boolean)
  + message (string)

**Suspend User**

* **Purpose**: Suspends a user account
* **Access**: PUT /api/admin/users/:userId/suspend
* **Request Data**:
  + userId (URL parameter, required)
  + duration (integer, optional: days)
  + reason (string, optional)
* **Response Data**:
  + success (boolean)
  + message (string)
  + suspended\_until (datetime)

**Activate User**

* **Purpose**: Activates a suspended user account
* **Access**: PUT /api/admin/users/:userId/activate
* **Request Data**:
  + userId (URL parameter, required)
* **Response Data**:
  + success (boolean)
  + message (string)

**Get All Reports**

* **Purpose**: Gets all user reports for admin
* **Access**: GET /api/admin/reports
* **Request Data**:
  + page (integer, optional)
  + limit (integer, optional)
  + status (string, optional: pending, reviewed, action\_taken, dismissed)
  + sort (string, optional)
* **Response Data**:
  + reports (array of report objects with detailed info)
  + total\_reports (integer)
  + current\_page (integer)
  + total\_pages (integer)

**Update Report Status**

* **Purpose**: Updates a report's status
* **Access**: PUT /api/admin/reports/:reportId
* **Request Data**:
  + reportId (URL parameter, required)
  + status (string, required)
  + admin\_notes (string, optional)
* **Response Data**:
  + success (boolean)
  + message (string)

**Get Post Analytics**

* **Purpose**: Gets post analytics by timeframe
* **Access**: GET /api/admin/analytics/posts
* **Request Data**:
  + timeframe (string, required: week, month, year)
  + start\_date (date, optional)
  + end\_date (date, optional)
* **Response Data**:
  + post\_count (integer)
  + post\_data (array of objects with dates and counts)
  + engagement\_metrics (object with likes, comments stats)
  + popular\_posts (array of post objects)

**Get User Analytics**

* **Purpose**: Gets user analytics by timeframe
* **Access**: GET /api/admin/analytics/users
* **Request Data**:
  + timeframe (string, required: week, month, year)
  + start\_date (date, optional)
  + end\_date (date, optional)
* **Response Data**:
  + total\_users (integer)
  + new\_users (integer)
  + active\_users (integer)
  + user\_data (array of objects with dates and counts)
  + most\_active\_users (array of user objects with activity stats)

**Search Endpoints**

**Global Search**

* **Purpose**: Searches across users, posts, and comments
* **Access**: GET /api/search
* **Request Data**:
  + query (string, required)
  + page (integer, optional)
  + limit (integer, optional)
  + filter (string, optional: all, users, posts, comments)
* **Response Data**:
  + results (object with categorized search results)
  + total\_results (integer)
  + current\_page (integer)
  + total\_pages (integer)

**Search Users**

* **Purpose**: Searches only for users
* **Access**: GET /api/search/users
* **Request Data**:
  + query (string, required)
  + page (integer, optional)
  + limit (integer, optional)
  + sort (string, optional)
* **Response Data**:
  + users (array of user objects)
  + total\_results (integer)
  + current\_page (integer)
  + total\_pages (integer)

**Search Posts**

* **Purpose**: Searches only for posts
* **Access**: GET /api/search/posts
* **Request Data**:
  + query (string, required)
  + page (integer, optional)
  + limit (integer, optional)
  + sort (string, optional)
  + date\_from (date, optional)
  + date\_to (date, optional)
* **Response Data**:
  + posts (array of post objects with user info)
  + total\_results (integer)
  + current\_page (integer)
  + total\_pages (integer)