routes website

POST register

http://localhost:3001/user/register

Registers a new user account in the system.

Purpose
Create a persistent user profile that can be used to log in, manage reviews, favorites, and groups.

Authentication

- No authentication required (public endpoint).

Request

- **Method:** POST
- **URL:** `/user/register`
- **Body (JSON):**
  - `email` (string, required): New user email address. Must be unique.
  - `password` (string, required): Plain-text password that meets the server's password policy.

Example request body

json

```
{
  "email": "testuser.test@gmail.com",
  "password": "secret123"
}
```

Successful response (201 Created)

json

```
{
  "message": "User created successfully",
  "user": {
    "email": "testuser.test@gmail.com"
  }
```

```
}
```

Common status codes

- `201 Created` – User account successfully created.
- `400 Bad Request` – Missing or invalid fields (e.g., malformed email, weak password).
- `409 Conflict` – Email already in use (if implemented).
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. Call **Register** to create the account.
2. Then call **Login** (`POST /user/login`) to obtain an access token.
3. Use the token to access authenticated routes such as profile, favorites, reviews, and groups.

Body raw (json)

json

```
{
  "email": "testuser.test@gmail.com",
  "password": "secret123"
}
```

POST login

http://localhost:3001/user/login

Authenticates an existing user and returns tokens for subsequent authenticated requests.

Purpose
Verify user credentials and issue an access token (and possibly a refresh token) used to access protected endpoints.

Authentication

- No authentication required (public endpoint).

Request

- **Method:** POST
- **URL:** `/user/login`
- **Body (JSON):**
  - `email` (string, required): Registered user email.
  - `password` (string, required): User password.

Example request body

```json
{
  "email": "testuser.test@gmail.com",
  "password": "secret123"
}
```

Typical successful response

```json
{
  "accessToken": "<JWT or token>",
  "refreshToken": "<refresh-token>",
  "user": {
    "email": "testuser.test@gmail.com"
  }
}
```

Common status codes

- `200 OK` – Authentication successful, tokens returned.
- `400 Bad Request` – Missing required fields.
- `401 Unauthorized` – Invalid email or password.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. Use **Register** (`POST /user/register`) to create an account (once).
2. Use **Login** to obtain tokens for all further authenticated operations (profile, favorites, reviews, groups).
3. Use **Refresh** (`POST /user/refresh`) to renew expired/expiring access tokens.
4. Use **Logout** (`POST /user/logout`) to invalidate the current session.

Body raw (json)

```json
{
  "email": "testuser.test@gmail.com",
  "password": "secret123"
}
```

POST refresh

http://localhost:3001/user/refresh

Refreshes the user's authentication token using a valid refresh token.

Purpose
Issue a new short-lived access token without requiring the user to re-enter credentials.

Authentication

- Typically requires a **refresh token** sent via:
  - HTTP-only cookie, or
  - Authorization header / request body (implementation-specific).
- Access token may not be required if only the refresh token is used.

Request

- **Method:** POST
- **URL:** `/user/refresh`
- **Headers (typical):**
  - `Authorization: Bearer <refresh_token>` **or** refresh token in cookie.

Example request

http

```
POST /user/refresh
Authorization: Bearer <refresh_token>
```

Typical successful response

json

```
{
  "accessToken": "<new-access-token>",
  "refreshToken": "<new-or-same-refresh-token>",
  "expiresIn": 3600
}
```

Common status codes

- `200 OK` – New token issued.
- `401 Unauthorized` – Refresh token missing, invalid, or expired.
- `403 Forbidden` – Refresh token revoked or does not match user/session.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. User logs in via **Login** (`POST /user/login`) and receives initial tokens.

2. When the access token is about to expire or has expired, call **Refresh**.
3. Continue using the new access token for all authenticated endpoints.
4. Use **Logout** (`POST /user/logout`) to terminate the session and invalidate refresh tokens (if implemented).

POST logout

http://localhost:3001/user/logout

Logs out the current user and invalidates their active session and/or tokens.

Purpose
End the user session so that their access and refresh tokens can no longer be used.

Authentication

- Requires a valid user session, typically via:
  - `Authorization: Bearer <access_token>` header, and/or
  - Auth cookies containing session/refresh token.

Request

- **Method:** POST
- **URL:** `/user/logout`
- **Headers (typical):**
  - `Authorization: Bearer <access_token>`

Example request

http

```
POST /user/logout
Authorization: Bearer <access_token>
```

Typical successful response

json

```
{
  "message": "Logged out successfully"
}
```

Common status codes

- `200 OK` – Logout successful; tokens and/or session invalidated.
- `401 Unauthorized` – Missing or invalid authentication.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. User logs in using **Login** (`POST /user/login`).
2. User interacts with authenticated endpoints (profile, favorites, reviews, groups).
3. User calls **Logout** to terminate the session and optionally clear client-side tokens/cookies.

POST favorite create

http://localhost:3001/favorite/share/create

Creates a new shareable favorites list entry or link for the current user.

Purpose
Allow a user to generate a shareable representation of their favorites (e.g., a share link or snapshot) that can be accessed by others.

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>` from the **Login** or **Refresh** flow.

Request

- **Method:** POST
- **URL:** `/favorite/share/create`
- **Headers (typical):**
  - `Authorization: Bearer <access_token>`
- **Body (JSON, inferred):**
  - `title` (string, optional): Custom title or name for this shared favorites view.
  - `description` (string, optional): Description of what is being shared.
  - `items` (array, optional): Explicit list of favorite item IDs to include. If omitted, the server may include all current favorites.

Example request body

json

```
{
  "title": "My top sci-fi picks",
  "description": "Favorites I recommend to friends.",
  "items": ["fav_123", "fav_456"]
}
```

Typical successful response

json

```
{
  "shareId": "abc123",
  "url": "https://example.com/favorites/abc123",
  "title": "My top sci-fi picks"
}
```

Common status codes

- `201 Created` – Shareable favorites entry successfully created.
- `400 Bad Request` – Invalid payload (e.g., items not found or malformed).
- `401 Unauthorized` – Missing or invalid authentication.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. User logs in and manages favorites using **Favorites** endpoints.
2. User calls **Create Favorite Share** to generate a share reference for some or all favorites.
3. Others can read this share using **Get Favorite Share** (`GET /favorite/share/:shareId`).

POST favorite

http://localhost:3001/favorite

Adds a new item to the current user's favorites list.

Purpose
Let authenticated users mark a movie or other media item as a favorite for quick access and sharing.

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>` .

Request

- **Method:** POST
- **URL:** `/favorite`
- **Headers (typical):**
  - `Authorization: Bearer <access_token>`
- **Body (JSON, inferred):**
  - `mediaId` (string | number, required): Unique identifier of the media item (e.g., movie ID).
  - `type` (string, optional): Media type, e.g. `"movie"` , `"tv"` .
  - `metadata` (object, optional): Any additional client metadata such as title, poster path, etc., depending on server implementation.

Example request body

```json
{
  "mediaId": 12345,
  "type": "movie"
}
```

Typical successful response

```json
{
  "favoriteId": "fav_123",
  "mediaId": 12345,
  "type": "movie",
  "createdAt": "2024-01-01T12:00:00.000Z"
}
```

Common status codes

- `201 Created` – Item successfully added to favorites.
- `400 Bad Request` – Missing or invalid `mediaId` or payload.
- `401 Unauthorized` – Missing or invalid authentication.
- `409 Conflict` – Item is already in favorites (if enforced).
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. User discovers a movie via **Search/Now Playing/Movie Details** endpoints.
2. User calls **Add Favorite** to store the item in their favorites list.
3. User can later list favorites via **Get Favorites** (`GET /favorite`) or share via **Create Favorite Share** (`POST /favorite/share/create`).

POST group

http://localhost:3001/group

Creates a new user group.

Purpose
Allow authenticated users to create groups (e.g., watch parties, friend groups) for organizing shared content and membership.

Authentication

- Requires an authenticated user.

- Use `Authorization: Bearer <access_token>`.

Request

- **Method:** POST
- **URL:** `/group`
- **Headers (typical):**
  - `Authorization: Bearer <access_token>`
- **Body (JSON, inferred):**
  - `name` (string, required): Human-readable group name.
  - `description` (string, optional): Description or purpose of the group.
  - `visibility` (string, optional): e.g. `"public"` or `"private"`.
  - `metadata` (object, optional): Implementation-specific data.

Example request body

```json
{
  "name": "Friday Movie Night",
  "description": "Group for our weekly movie night.",
  "visibility": "private"
}
```

Typical successful response

```json
{
  "groupId": "grp_123",
  "name": "Friday Movie Night",
  "description": "Group for our weekly movie night.",
  "ownerId": "user_1"
}
```

Common status codes

- `201 Created` – Group successfully created.
- `400 Bad Request` – Missing or invalid payload fields.
- `401 Unauthorized` – Missing or invalid authentication.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. User logs in and gets an access token.
2. User calls **Create Group** to start a new group.
3. Members can be added via join requests (**Group Requests**) and group content can be managed via **Group Content** endpoints.

POST group content

http://localhost:3001/group/:groupId/content

Adds new content to a specific group.

Purpose
Let group members contribute shared content (e.g., movie suggestions, playlists, notes) associated with a group.

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>`.
- Typically, the user must also be a member of the target group.

Request

- **Method:** POST
- **URL:** `/group/:groupId/content`
- **Path parameters:**
  - `groupId` (string, required): Unique identifier of the group to which content is added.
- **Headers (typical):**
  - `Authorization: Bearer <access_token>`
- **Body (JSON, inferred):**
  - `mediaId` (string | number, optional): ID of an associated movie/media item.
  - `title` (string, required): Content title, e.g. movie name or topic.
  - `notes` (string, optional): Additional comments or description.
  - `type` (string, optional): Content type such as `"movie"`, `"discussion"`, etc.

Example request body

json

```
{
  "mediaId": 12345,
  "title": "Next week's movie",
  "notes": "Let's watch this on Friday.",
  "type": "movie"
}
```

Typical successful response

json

```
{
```

```
  "contentId": "cnt_123",
  "groupId": "grp_123",
  "title": "Next week's movie",
  "mediaId": 12345
}
```

Common status codes

- `201 Created` – Content successfully added to the group.
- `400 Bad Request` – Invalid or missing body fields.
- `401 Unauthorized` – Missing or invalid authentication.
- `403 Forbidden` – User is not a member or lacks permission for this group.
- `404 Not Found` – Group not found.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. A group is created via **Create Group** (`POST /group`).
2. Users join or are added to the group.
3. Members use **Add Group Content** to propose movies, discussions, or other content items for the group.

PATH VARIABLES

groupId

POST group requests

http://localhost:3001/group/:groupId/requests

Submits a request to join a specific group.

Purpose
Allow authenticated users to request membership in an existing group, pending approval from group admins/owners.

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>`.

Request

- **Method:** POST
- **URL:** `/group/:groupId/requests`
- **Path parameters:**
  - `groupId` (string, required): Unique identifier of the group to join.

- **Headers (typical):**
  - `Authorization: Bearer <access_token>`
- **Body (JSON, inferred):**
  - `message` (string, optional): Optional join message or note to group admins.

Example request body

```
json

{
  "message": "Hi, I'd like to join your Friday movie group."
}
```

Typical successful response

```
json

{
  "groupId": "grp_123",
  "userId": "user_1",
  "status": "pending"
}
```

Common status codes

- `201 Created` – Join request created successfully.
- `400 Bad Request` – Invalid group ID or payload.
- `401 Unauthorized` – Missing or invalid authentication.
- `409 Conflict` – Request already pending or user already a member.
- `404 Not Found` – Group not found.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. A group exists (created via **Create Group**).
2. A user calls **Request to Join Group** to become a member.
3. Group admins process requests via **List Group Requests** (`GET /group/:groupId/requests`) and **Accept/Reject Group Request** endpoints.

PATH VARIABLES

groupId

POST review media id

http://localhost:3001/review/:mediaId

Creates a new review for the specified media item.

Purpose
Allow authenticated users to write and submit reviews for movies or other media.

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>`.

Request

- **Method:** POST
- **URL:** `/review/:mediaId`
- **Path parameters:**
  - `mediaId` (string | number, required): Unique identifier of the media item being reviewed.
- **Headers (typical):**
  - `Authorization: Bearer <access_token>`
- **Body (JSON, inferred):**
  - `rating` (number, required): Numeric rating (e.g., 1–5 or 1–10 scale).
  - `title` (string, optional): Short title or summary of the review.
  - `content` (string, required): Full text of the review.
  - `spoilers` (boolean, optional): Whether the review contains spoilers.

Example request body

json

```
{
  "rating": 4.5,
  "title": "Great sci-fi adventure",
  "content": "I really enjoyed the pacing and visuals...",
  "spoilers": false
}
```

Typical successful response

json

```
{
  "reviewId": "rev_123",
  "mediaId": 12345,
  "rating": 4.5,
  "content": "I really enjoyed the pacing and visuals..."
}
```

Common status codes

Common status codes

- `201 Created` – Review successfully created.
- `400 Bad Request` – Missing or invalid rating/content or invalid `mediaId`.
- `401 Unauthorized` – Missing or invalid authentication.
- `404 Not Found` – Target media not found (if validated).
- `409 Conflict` – User already has a review for this media (if only one review per user is allowed).
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. User discovers a movie via **Search/Now Playing/Movie Details**.
2. User calls **Create Review** on the chosen media.
3. Reviews can later be listed via **Get Reviews by Media/User** and edited or deleted via **Update/Delete Review** endpoints.

PATH VARIABLES

mediaId

GET user

http://localhost:3001/user

Returns the currently authenticated user's basic account information.

Purpose
Provide the client with the current user's core profile data (e.g., ID, email) based on the active session/token.

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>`.

Request

- **Method:** GET
- **URL:** `/user`
- **Headers (typical):**
  - `Authorization: Bearer <access_token>`

Example request

http

```
GET /user
Authorization: Bearer <access_token>
```

Typical successful response

json

```
{
  "id": "user_1",
  "email": "testuser.test@gmail.com",
  "createdAt": "2024-01-01T12:00:00.000Z"
}
```

Common status codes

- `200 OK` – Successfully returned user information.
- `401 Unauthorized` – Missing or invalid authentication.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. User logs in and obtains an access token.
2. Client calls **Get Current User** to retrieve identity and basic account information.
3. Additional details can be fetched via **User Profile** (`GET /user/profile`).

GET profile

http://localhost:3001/user/profile

Returns the authenticated user's detailed profile information.

Purpose
Provide extended user profile data beyond basic identity (e.g., preferences, display name, avatar, stats).

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>` .

Request

- **Method:** GET
- **URL:** `/user/profile`
- **Headers (typical):**
  - `Authorization: Bearer <access_token>`

Example request

http

```
GET /user/profile
Authorization: Bearer <access_token>
```

Typical successful response

json

```
{
  "id": "user_1",
  "email": "testuser.test@gmail.com",
  "displayName": "Test User",
  "avatarUrl": "https://example.com/avatar.png",
  "preferences": {
    "language": "en",
    "theme": "dark"
  }
}
```

Common status codes

- `200 OK` – Profile information returned successfully.
- `401 Unauthorized` – Missing or invalid authentication.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. User logs in and gets an access token.
2. Client calls **User Profile** to render account/profile screens.
3. Other endpoints may expose profile update functionality (not shown in this collection).

GET favorite share id

http://localhost:3001/favorite/share/:shareId

Returns a shareable favorites view by its share ID.

Purpose
Allow clients (including unauthenticated viewers, depending on implementation) to see a favorites list that another user has shared.

Authentication

- Often **no authentication required** for public shares.
- Some implementations may restrict access and require an access token.

Request

.

- **Method:** GET
- **URL:** `/favorite/share/:shareId`
- **Path parameters:**
  - `shareId` (string, required): Identifier of the shared favorites resource.

Example request

http

```
GET /favorite/share/abc123
```

Typical successful response

json

```
{
  "shareId": "abc123",
  "owner": {
    "id": "user_1",
    "displayName": "Test User"
  },
  "title": "My top sci-fi picks",
  "items": [
    {
      "favoriteId": "fav_123",
      "mediaId": 12345,
```

Common status codes

- `200 OK` – Shared favorites retrieved successfully.
- `404 Not Found` – Share ID does not exist or has been revoked.
- `410 Gone` – Share existed but has been explicitly deleted/expired (if implemented).
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. An authenticated user creates a share via **Create Favorite Share** (`POST /favorite/share/create`).
2. The generated `shareId` /URL is given to other users.
3. Other users call **Get Favorite Share** to view the shared favorites list, optionally without needing to log in.

PATH VARIABLES

shareId

GET favorite

http://localhost:3001/favorite

Returns the list of items in the current user's favorites.

Purpose
Let authenticated users view all items they have marked as favorites.

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>`.

Request

- **Method:** GET
- **URL:** `/favorite`
- **Headers (typical):**
  - `Authorization: Bearer <access_token>`
- **Query parameters (optional, inferred):**
  - `page` (number, optional): Results page number.
  - `pageSize` (number, optional): Items per page.
  - `type` (string, optional): Filter by media type, e.g. `"movie"`.

Example request

http

```
GET /favorite?page=1&pageSize=20
Authorization: Bearer <access_token>
```

Typical successful response

json

```
{
  "items": [
    {
      "favoriteId": "fav_123",
      "mediaId": 12345,
      "type": "movie",
      "title": "Interstellar"
    }
  ],
  "page": 1,
  "pageSize": 20,
```

Common status codes

- `200 OK` – Favorites list returned successfully.

- `401 Unauthorized` – Missing or invalid authentication.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. Items are added via **Add Favorite** (`POST /favorite`).
2. User calls **Get Favorites** to render the favorites screen.
3. Individual favorites can be removed via **Delete Favorite** (`DELETE /favorite/:favoriteId`).

GET group

http://localhost:3001/group

Returns a list of all groups the user can access.

Purpose
Provide the authenticated user with an overview of groups they own, have joined, or can otherwise see.

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>` .

Request

- **Method:** GET
- **URL:** `/group`
- **Headers (typical):**
  - `Authorization: Bearer <access_token>`
- **Query parameters (optional, inferred):**
  - `membership` (string, optional): Filter groups by relation, e.g. `"owned"` , `"joined"` , `"all"` .
  - `page` (number, optional): Page index.
  - `pageSize` (number, optional): Number of results per page.

Example request

http

```
GET /group?membership=joined&page=1&pageSize=20
Authorization: Bearer <access_token>
```

Typical successful response

json

```
{
```

```
{
  "items": [
    {
      "groupId": "grp_123",
      "name": "Friday Movie Night",
      "description": "Group for our weekly movie night.",
      "membership": "owner"
    }
  ],
  "page": 1,
  "pageSize": 20,
```


Common status codes

- `200 OK` – Groups list returned successfully.
- `401 Unauthorized` – Missing or invalid authentication.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. Groups are created via **Create Group** (`POST /group`).
2. Users join groups via **Request to Join Group** (`POST /group/:groupId/requests`) and admin actions.
3. This endpoint is used by the client to list accessible groups on a dashboard or groups screen.

GET group user

http://localhost:3001/group/user

Returns the list of groups the current user belongs to.

Purpose
Provide a quick view of groups where the authenticated user is a member, as distinct from all accessible or discoverable groups.

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>` .

Request

- **Method:** GET
- **URL:** `/group/user`
- **Headers (typical):**
  - `Authorization: Bearer <access_token>`

Example request

http

```
GET /group/user
Authorization: Bearer <access_token>
```

Typical successful response

json

```
{
  "items": [
    {
      "groupId": "grp_123",
      "name": "Friday Movie Night",
      "role": "member"
    }
  ]
}
```

Common status codes

- `200 OK` – User's groups returned successfully.
- `401 Unauthorized` – Missing or invalid authentication.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. User joins or creates groups.
2. Client calls **Get User Groups** to show groups where the user is a member.
3. For details on a specific group, call **Get Group by ID** (`GET /group/:groupId`).

GET group id

http://localhost:3001/group/:groupId

Returns details of a group by its ID.

Purpose
Fetch a single group's metadata, including name, description, visibility, and possibly counts or summary stats.

Authentication

- Usually requires an authenticated user.
- Use `Authorization: Bearer <access_token>` .
- Public groups may be readable without authentication, depending on implementation.

Request

- **Method:** GET
- **URL:** `/group/:groupId`

- URL: `/group/:groupId`
- **Path parameters:**
  - `groupId` (string, required): Unique identifier of the group.

Example request

http

```
GET /group/grp_123
Authorization: Bearer <access_token>
```

Typical successful response

json

```
{
  "groupId": "grp_123",
  "name": "Friday Movie Night",
  "description": "Group for our weekly movie night.",
  "ownerId": "user_1",
  "visibility": "private"
}
```

Common status codes

- `200 OK` – Group details returned successfully.
- `401 Unauthorized` – Missing or invalid authentication (for protected groups).
- `403 Forbidden` – Group exists but is not visible to this user.
- `404 Not Found` – Group with given ID does not exist.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. Client lists groups via **List Groups** (`GET /group`) or **Get User Groups**.
2. When the user selects a specific group, client calls **Get Group by ID** to show details.
3. Additional endpoints manage members and content for this group.

PATH VARIABLES

groupId

GET group members

http://localhost:3001/group/:groupId/members

Returns the list of members in a specific group.

Purpose

Provide visibility into which users belong to a group, including their roles (owner, admin, member).

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>`.
- Some groups may restrict this list to group members only.

Request

- **Method:** GET
- **URL:** `/group/:groupId/members`
- **Path parameters:**
  - `groupId` (string, required): Unique identifier of the group.

Example request

http

```http
GET /group/grp_123/members
Authorization: Bearer <access_token>
```

Typical successful response

json

```json
{
  "groupId": "grp_123",
  "members": [
    {
      "userId": "user_1",
      "displayName": "Owner User",
      "role": "owner"
    },
    {
      "userId": "user_2",
      "displayName": "Member User",
```

Common status codes

- `200 OK` – Members list returned successfully.
- `401 Unauthorized` – Missing or invalid authentication.
- `403 Forbidden` – User is not allowed to view this group's members.
- `404 Not Found` – Group not found.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. A group exists and may have pending join requests.
2. Group owners/admins list members for management or display purposes.
3. Individual members can be removed via **Remove Group Member** (`DELETE /group/:groupId/members/:userId`).

PATH VARIABLES

groupId

GET group requests

http://localhost:3001/group/:groupId/requests

Returns all pending or historical join requests for a specific group.

Purpose
Allow group owners/admins to review membership requests and process them (accept or reject).

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>`.
- Typically restricted to group owners/admins.

Request

- **Method:** GET
- **URL:** `/group/:groupId/requests`
- **Path parameters:**
  - `groupId` (string, required): Identifier of the group for which to list join requests.
- **Query parameters (optional, inferred):**
  - `status` (string, optional): Filter by request status, e.g. `"pending"`, `"accepted"`, `"rejected"`.

Example request

http

```
GET /group/grp_123/requests?status=pending
Authorization: Bearer <access_token>
```

Typical successful response

json

```
{
  "groupId": "grp_123",
  "requests": [
    {
      "userId": "user_2",
      "status": "pending",
      "message": "Hi, I'd like to join.",
      "createdAt": "2024-01-01T12:00:00.000Z"
    }
  ]
}
```


Common status codes

- `200 OK` – Requests list returned successfully.
- `401 Unauthorized` – Missing or invalid authentication.
- `403 Forbidden` – User is not allowed to manage this group.
- `404 Not Found` – Group not found.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. Users submit join requests via **Request to Join Group** (`POST /group/:groupId/requests`).
2. Group admins list requests using **Get Group Requests**.
3. Individual requests are processed via **Accept Group Request** (`PATCH /group/:groupId/requests/:userId/accept`) or **Reject Group Request** (`DELETE /group/:groupId/requests/:userId/reject`).

PATH VARIABLES

groupId

GET group content id

http://localhost:3001/group/:groupId/content/:contentId

Returns details of a specific content item in a group.

Purpose
Retrieve a single content entry (e.g., a movie suggestion or note) within a group for display or editing.

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>`.
- Typically limited to group members.

Request

Request

- **Method:** GET
- **URL:** `/group/:groupId/content/:contentId`
- **Path parameters:**
  - `groupId` (string, required): Identifier of the group.
  - `contentId` (string, required): Identifier of the content item.

Example request

http

```
GET /group/grp_123/content/cnt_123
Authorization: Bearer <access_token>
```

Typical successful response

json

```
{
  "groupId": "grp_123",
  "contentId": "cnt_123",
  "title": "Next week's movie",
  "mediaId": 12345,
  "notes": "Let's watch this on Friday.",
  "type": "movie"
}
```

Common status codes

- `200 OK` – Content item returned successfully.
- `401 Unauthorized` – Missing or invalid authentication.
- `403 Forbidden` – User is not allowed to view this group's content.
- `404 Not Found` – Group or content not found.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. Content is created via **Add Group Content** (`POST /group/:groupId/content`).
2. Client lists content via **List Group Content** (`GET /group/:groupId/content`).
3. When a specific content item is selected, the client calls **Get Group Content by ID** to render detail view or prepare for further actions (edit/delete if supported).

PATH VARIABLES

groupId

contentId

GET group content

http://localhost:3001/group/:groupId/content

Returns the list of content items in a specific group.

Purpose
List all group content entries (e.g., movie suggestions, discussion topics, notes) in a group.

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>`.
- Typically restricted to group members.

Request

- **Method:** GET
- **URL:** `/group/:groupId/content`
- **Path parameters:**
  - `groupId` (string, required): Identifier of the group.
- **Query parameters (optional, inferred):**
  - `page` (number, optional): Page index.
  - `pageSize` (number, optional): Number of results per page.

Example request

http

```
GET /group/grp_123/content?page=1&pageSize=20
Authorization: Bearer <access_token>
```

Typical successful response

json

```
{
  "groupId": "grp_123",
  "items": [
    {
      "contentId": "cnt_123",
      "title": "Next week's movie",
      "mediaId": 12345,
      "type": "movie"
    }
  ],
```

"page": 1,

Common status codes

- `200 OK` – Content items returned successfully.
- `401 Unauthorized` – Missing or invalid authentication.
- `403 Forbidden` – User is not allowed to view this group's content.
- `404 Not Found` – Group not found.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. Content is added to the group via **Add Group Content** (`POST /group/:groupId/content`).
2. Client calls **List Group Content** to display a feed or list for the group.
3. Individual items can be viewed via **Get Group Content by ID** or removed via **Delete Group Content**.

PATH VARIABLES

groupId

GET movie search

http://localhost:3001/movie/search

Searches for movies by title or other criteria.

Purpose
Provide a search interface over the movie catalog so users can discover titles to review, favorite, or share.

Authentication

- Usually no authentication required (public endpoint), unless restricted by implementation.

Request

- **Method:** GET
- **URL:** `/movie/search`
- **Query parameters (typical):**
  - `query` (string, required): Free-text search query (e.g., movie title).
  - `page` (number, optional): Results page number for pagination.
  - `year` (number, optional): Filter by release year.
  - `language` (string, optional): Filter by language code (e.g., `"en"`).

Example request

http

```
GET /movie/search?query=interstellar&page=1
```

Typical successful response

json


```
{
  "page": 1,
  "results": [
    {
      "id": 12345,
      "title": "Interstellar",
      "overview": "A team of explorers travel through a wormhole...",
      "releaseDate": "2014-11-07"
    }
  ],
  "totalResults": 1,
```


Common status codes

- `200 OK` – Search results returned successfully.
- `400 Bad Request` – Missing `query` parameter or invalid parameters.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. User searches for a movie using **Search Movies**.
2. From the results, client may call **Get Movie Details** (`GET /movie/:movieId`).
3. From there, the user can create **Favorites** or **Reviews** for the selected title.

GET now playing

http://localhost:3001/movie/now-playing

Returns a list of movies that are currently playing in theaters.

Purpose
Provide a curated feed of "now playing" titles users can browse, favorite, or review without searching.

Authentication

- Usually no authentication required (public endpoint), unless restricted by implementation.

Request

- **Method:** GET
- **URL:** `/movie/now-playing`
- **Query parameters (optional, inferred):**

- **Query parameters (optional, inferred):**
  - `page` (number, optional): Results page number.
  - `region` (string, optional): Region/country code.

Example request

```http
GET /movie/now-playing?page=1
```

Typical successful response

```json
{
  "page": 1,
  "results": [
    {
      "id": 56789,
      "title": "Current Blockbuster",
      "overview": "An exciting new release...",
      "releaseDate": "2024-01-10"
    }
  ],
  "totalResults": 10,
```

Common status codes

- `200 OK` – Now-playing list returned successfully.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. Client shows a homepage section using **Now Playing**.
2. User clicks a title to view **Movie Details** (`GET /movie/:movieId`).
3. From there, user can **Favorite** or **Review** the title.

GET reviews id

http://localhost:3001/movie/reviews/:id

Returns all reviews for a specific movie/media item.

Purpose
Show a list of reviews associated with a single movie ID, enabling users to read community feedback.

Authentication

- Usually no authentication required for reading reviews.

- Some implementations may require authentication to view certain content.

Request

- **Method:** GET
- **URL:** `/movie/reviews/:id`
- **Path parameters:**
  - `id` (string | number, required): Identifier of the movie/media item.
- **Query parameters (optional, inferred):**
  - `page` (number, optional): Page index.
  - `pageSize` (number, optional): Number of reviews per page.
  - `sort` (string, optional): Sorting order, e.g. `"newest"`, `"highest-rating"`.

Example request

http

```
GET /movie/reviews/12345?page=1&pageSize=20
```

Typical successful response

json

```
{
  "mediaId": 12345,
  "reviews": [
    {
      "reviewId": "rev_123",
      "userId": "user_1",
      "rating": 4.5,
      "content": "I really enjoyed this movie."
    }
  ],
  "page": 1,
```

Common status codes

- `200 OK` – Reviews list returned successfully (may be empty).
- `404 Not Found` – Media item not found (if validated).
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. User selects a movie via **Search** or **Now Playing**.
2. Client calls **Get Reviews by Movie** to display community reviews.
3. Authenticated users can add their own review via **Create Review** (`POST /review/:mediaId`).

PATH VARIABLES

id

GET movie id

http://localhost:3001/movie/:movieId

Returns detailed information for a single movie by its ID.

Purpose
Provide a full movie detail view (overview, metadata) for display on a movie details page.

Authentication

- Typically no authentication required (public endpoint), unless restricted by implementation.

Request

- **Method:** GET
- **URL:** `/movie/:movieId`
- **Path parameters:**
  - `movieId` (string | number, required): Unique movie identifier.

Example request

http

```
GET /movie/12345
```

Typical successful response

json

```
{
  "id": 12345,
  "title": "Interstellar",
  "overview": "A team of explorers travel through a wormhole...",
  "releaseDate": "2014-11-07",
  "runtime": 169,
  "genres": ["Science Fiction", "Adventure"],
  "language": "en"
}
```

Common status codes

- `200 OK` – Movie details returned successfully.
- `404 Not Found` – Movie not found.

- `500 Internal Server Error` – Unexpected server error.

Flow context

1. User discovers a title via **Search** or **Now Playing**.
2. Client calls **Get Movie Details** to show a detailed page.
3. From this view, user can **Favorite** or **Review** the movie, or see **Movie Reviews** (`GET /movie/reviews/:id`).

PATH VARIABLES

movieId

GET review

http://localhost:3001/review

Returns a list of reviews. Scope is typically configurable via query parameters.

Purpose
Provide a generic listing of reviews, such as all reviews, latest reviews, or reviews filtered by criteria.

Authentication

- Often no authentication required for reading reviews.
- Some filters or views may require authentication.

Request

- **Method:** GET
- **URL:** `/review`
- **Query parameters (inferred examples):**
  - `mediaId` (string | number, optional): Filter by a specific media item.
  - `userId` (string, optional): Filter by author.
  - `page` (number, optional): Results page.
  - `pageSize` (number, optional): Items per page.
  - `sort` (string, optional): e.g. `"newest"` , `"highest-rating"` .

Example request

http

```
GET /review?page=1&pageSize=20&sort=newest
```

Typical successful response

json


```
{
  "items": [
    {
      "reviewId": "rev_123",
      "mediaId": 12345,
      "userId": "user_1",
      "rating": 4.5,
      "content": "Great movie!"
    }
  ],
  "page": 1,
```


Common status codes

- `200 OK` – Reviews list returned successfully.
- `400 Bad Request` – Invalid filter or pagination parameters.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. Client may use **List Reviews** for a general reviews feed.
2. For user-specific reviews, use **User Reviews** (`GET /review/user`) or filter by `userId` .
3. For movie-specific reviews, use **Movie Reviews** (`GET /movie/reviews/:id`).

GET user reviews

http://localhost:3001/review/user

Returns all reviews authored by the currently authenticated user.

Purpose
Let users see and manage reviews they've written across all media items.

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>` .

Request

- **Method:** GET
- **URL:** `/review/user`
- **Headers (typical):**
  - `Authorization: Bearer <access_token>`
- **Query parameters (optional, inferred):**
  - `page` (number, optional): Page index.

- `pageSize` (number, optional): Items per page.

Example request

```http
GET /review/user?page=1&pageSize=20
Authorization: Bearer <access_token>
```

Typical successful response

```json
{
  "items": [
    {
      "reviewId": "rev_123",
      "mediaId": 12345,
      "rating": 4.5,
      "content": "Great movie!"
    }
  ],
  "page": 1,
  "pageSize": 20,
```

Common status codes

- `200 OK` – User's reviews returned successfully.
- `401 Unauthorized` – Missing or invalid authentication.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. User submits reviews via **Create Review** (`POST /review/:mediaId`).
2. Client calls **User Reviews** to show a profile section listing their contributions.
3. Individual reviews can be updated via **Update Review** (`PATCH /review/:reviewId`) or removed via **Delete Review** (`DELETE /review/:reviewId`).

GET review detail

http://localhost:3001/review/:reviewId/detail

Returns detailed information for a specific review.

Purpose
Provide a full detail view for a single review, often including related media info and author details.

Authentication

- Typically no authentication required for reading reviews.
- Implementations may restrict certain reviews to authenticated users.

Request

- **Method:** GET
- **URL:** `/review/:reviewId/detail`
- **Path parameters:**
  - `reviewId` (string, required): Identifier of the review.

Example request

http

```
GET /review/rev_123/detail
```

Typical successful response

json

```
{
  "reviewId": "rev_123",
  "mediaId": 12345,
  "user": {
    "id": "user_1",
    "displayName": "Test User"
  },
  "rating": 4.5,
  "title": "Great sci-fi adventure",
  "content": "I really enjoyed the pacing and visuals...",
  "createdAt": "2024-01-01T12:00:00.000Z"
```

Common status codes

- `200 OK` – Review details returned successfully.
- `404 Not Found` – Review not found.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. Client lists reviews via **List Reviews**, **User Reviews**, or **Movie Reviews**.
2. When a user opens a review detail page, client calls **Get Review Detail** with the review ID.

PATH VARIABLES

reviewId

PATCH user accept

http://localhost:3001/group/:groupId/requests/:userId/accept

Accepts a user's request to join a group.

Purpose
Allow group owners/admins to approve pending join requests and add users as members.

Authentication

- Requires an authenticated user with appropriate permissions (typically group owner/admin).
- Use `Authorization: Bearer <access_token>`.

Request

- **Method:** PATCH
- **URL:** `/group/:groupId/requests/:userId/accept`
- **Path parameters:**
  - `groupId` (string, required): Identifier of the target group.
  - `userId` (string, required): Identifier of the user whose request is being accepted.

Example request

http

```
PATCH /group/grp_123/requests/user_2/accept
Authorization: Bearer <access_token>
```

Typical successful response

json

```
{
  "groupId": "grp_123",
  "userId": "user_2",
  "status": "accepted"
}
```

Common status codes

- `200 OK` – Request accepted and membership granted.
- `400 Bad Request` – No pending request for this user or invalid IDs.
- `401 Unauthorized` – Missing or invalid authentication.
- `403 Forbidden` – Caller lacks permission to manage this group.
- `404 Not Found` – Group or request not found.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. A user submits a join request via **Request to Join Group** (`POST /group/:groupId/requests`).
2. Admin lists pending requests via **Get Group Requests** (`GET /group/:groupId/requests`).
3. Admin calls **Accept Group Request** to approve the user, or **Reject Group Request** to deny it.

PATH VARIABLES

groupId

userId

PATCH review id

http://localhost:3001/review/:reviewId

Updates an existing review by its ID.

Purpose
Allow review authors (and possibly moderators) to edit the content, rating, or title of a review.

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>`.
- Caller is typically required to be the review's author or an admin.

Request

- **Method:** PATCH
- **URL:** `/review/:reviewId`
- **Path parameters:**
  - `reviewId` (string, required): Identifier of the review to update.
- **Headers (typical):**
  - `Authorization: Bearer <access_token>`
- **Body (JSON, inferred):**
  - `rating` (number, optional): New rating value.
  - `title` (string, optional): Updated review title.
  - `content` (string, optional): Updated review text.
  - Any field may be omitted if not changed.

Example request body

```
json
```

```
{
  "rating": 5,
  "title": "Even better on second watch",
  "content": "After rewatching, I appreciated the details even more."
}
```

Typical successful response

json

```
{
  "reviewId": "rev_123",
  "rating": 5,
  "title": "Even better on second watch",
  "content": "After rewatching, I appreciated the details even more."
}
```

Common status codes

- `200 OK` – Review updated successfully.
- `400 Bad Request` – Invalid field values.
- `401 Unauthorized` – Missing or invalid authentication.
- `403 Forbidden` – Caller is not allowed to modify this review.
- `404 Not Found` – Review not found.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. Review is created via **Create Review** (`POST /review/:mediaId`).
2. Author decides to update rating or content and calls **Update Review**.
3. Review can later be removed using **Delete Review** (`DELETE /review/:reviewId`).

PATH VARIABLES

reviewId

DELETE content id

http://localhost:3001/group/:groupId/content/:contentId

Deletes a specific content item from a group.

Purpose
Allow group owners/admins (and possibly the content author) to remove group content entries.

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>`.
- Caller must have permission to delete content (group owner/admin, or content creator if allowed).

Request

- **Method:** DELETE
- **URL:** `/group/:groupId/content/:contentId`
- **Path parameters:**
  - `groupId` (string, required): Identifier of the group.
  - `contentId` (string, required): Identifier of the content item to delete.

Example request

http

```http
DELETE /group/grp_123/content/cnt_123
Authorization: Bearer <access_token>
```

Typical successful response

json

```json
{
  "groupId": "grp_123",
  "contentId": "cnt_123",
  "deleted": true
}
```

Common status codes

- `200 OK` or `204 No Content` – Content deleted successfully.
- `401 Unauthorized` – Missing or invalid authentication.
- `403 Forbidden` – Caller is not allowed to delete this content.
- `404 Not Found` – Group or content not found.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. Content is created via **Add Group Content** and listed via **List Group Content**.
2. Admin or author decides to remove an item and calls **Delete Group Content**.
3. The content will no longer appear in group content lists.

PATH VARIABLES

groupId

contentId

DELETE group id

http://localhost:3001/group/:groupId

Deletes a group by its ID.

Purpose
Allow group owners (and possibly admins) to permanently remove an existing group.

Authentication

- Requires an authenticated user with sufficient permissions (group owner/admin).
- Use `Authorization: Bearer <access_token>`.

Request

- **Method:** DELETE
- **URL:** `/group/:groupId`
- **Path parameters:**
  - `groupId` (string, required): Identifier of the group to delete.

Example request

http

```
DELETE /group/grp_123
Authorization: Bearer <access_token>
```

Typical successful response

json

```
{
  "groupId": "grp_123",
  "deleted": true
}
```

Common status codes

- `200 OK` or `204 No Content` – Group deleted successfully.
- `401 Unauthorized` – Missing or invalid authentication.
- `403 Forbidden` – Caller is not allowed to delete this group.
- `404 Not Found` – Group not found.

- `500 Internal Server Error` – Unexpected server error.

Flow context

1. Group is created via **Create Group** and used by members.
2. Owner decides to remove the group and calls **Delete Group**.
3. Related group content, members, and requests may be cleaned up according to server rules.

PATH VARIABLES

groupId

DELETE review id

http://localhost:3001/review/:reviewId

Deletes a review by its ID.

Purpose
Allow authors (and/or moderators) to remove an existing review from the system.

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>`.
- Caller must be the review author or an admin/moderator.

Request

- **Method:** DELETE
- **URL:** `/review/:reviewId`
- **Path parameters:**
  - `reviewId` (string, required): Identifier of the review to delete.

Example request

http

```
DELETE /review/rev_123
Authorization: Bearer <access_token>
```

Typical successful response

json

```
{
  "reviewId": "rev_123",
  "deleted": true
}
```

Common status codes

- `200 OK` or `204 No Content` – Review deleted successfully.
- `401 Unauthorized` – Missing or invalid authentication.
- `403 Forbidden` – Caller not allowed to delete this review.
- `404 Not Found` – Review not found.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. Review is created via **Create Review** and may be updated via **Update Review**.
2. Author or admin decides the review should be removed.
3. Client calls **Delete Review**; the review no longer appears in listings or detail views.

PATH VARIABLES

reviewId

DELETE group member id

http://localhost:3001/group/:groupId/members/:userId

Removes a specific user from a group.

Purpose
Allow group owners/admins (and possibly users themselves) to leave or be removed from a group.

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>`.
- Caller must be group owner/admin when removing another member; users may be allowed to remove themselves.

Request

- **Method:** DELETE
- **URL:** `/group/:groupId/members/:userId`
- **Path parameters:**
  - `groupId` (string, required): Identifier of the group.

- `userId` (string, required): Identifier of the user being removed.

Example request

```http
DELETE /group/grp_123/members/user_2
Authorization: Bearer <access_token>
```

Typical successful response

```json
{
  "groupId": "grp_123",
  "userId": "user_2",
  "removed": true
}
```

Common status codes

- `200 OK` or `204 No Content` – Member removed successfully.
- `401 Unauthorized` – Missing or invalid authentication.
- `403 Forbidden` – Caller is not allowed to remove this member.
- `404 Not Found` – Group or user membership not found.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. User joins group via join requests or invitations.
2. Group roles change or membership needs to be cleaned up.
3. Admin calls **Remove Group Member**, or user triggers their own removal (leave group) if supported.

PATH VARIABLES

groupId

userId

DELETE group request reject

http://localhost:3001/group/:groupId/requests/:userId/reject

Rejects a pending request from a user to join a group.

Purpose

Purpose

Allow group owners/admins to deny a user's join request and optionally record that decision.

Authentication

- Requires an authenticated user with permission to manage group membership (owner/admin).
- Use `Authorization: Bearer <access_token>`.

Request

- **Method:** DELETE
- **URL:** `/group/:groupId/requests/:userId/reject`
- **Path parameters:**
  - `groupId` (string, required): Identifier of the group.
  - `userId` (string, required): Identifier of the user whose request is being rejected.

Example request

http

```
DELETE /group/grp_123/requests/user_2/reject
Authorization: Bearer <access_token>
```

Typical successful response

json

```
{
  "groupId": "grp_123",
  "userId": "user_2",
  "status": "rejected"
}
```

Common status codes

- `200 OK` or `204 No Content` – Request rejected/removed successfully.
- `400 Bad Request` – Request not in a pending state or invalid IDs.
- `401 Unauthorized` – Missing or invalid authentication.
- `403 Forbidden` – Caller is not allowed to manage requests for this group.
- `404 Not Found` – Group or join request not found.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. Users send join requests via **Request to Join Group**.
2. Admin lists requests via **Get Group Requests**.
3. Admin either **Accepts** (`PATCH /group/:groupId/requests/:userId/accept`) or **Rejects** the request using this endpoint.

PATH VARIABLES

groupId

userId

DELETE user delete

http://localhost:3001/user/delete

Permanently deletes the currently authenticated user's account.

Purpose
Allow a user to delete their account and associated data from the system.

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>`.

Request

- **Method:** DELETE
- **URL:** `/user/delete`
- **Headers (typical):**
  - `Authorization: Bearer <access_token>`
- **Body (optional, implementation-specific):**
  - Some implementations may require confirming data (e.g., password or `confirm: true`).

Example request

http

```
DELETE /user/delete
Authorization: Bearer <access_token>
```

Typical successful response

json

```
{
  "userId": "user_1",
  "deleted": true
}
```

Common status codes

- `200 OK` or `204 No Content` – Account deleted successfully.
- `400 Bad Request` – Additional confirmation required or invalid.
- `401 Unauthorized` – Missing or invalid authentication.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. User is authenticated and optionally confirms deletion within the UI.
2. Client calls **Delete User** to remove the account.
3. After deletion, tokens are invalid, and user can no longer log in with this account; a fresh **Register** is needed to return.

DELETE favorite id

http://localhost:3001/favorite/:favoriteId

Deletes a favorites entry by its ID.

Purpose
Allow users to remove items from their favorites list.

Authentication

- Requires an authenticated user.
- Use `Authorization: Bearer <access_token>`.
- Caller must own the favorite entry.

Request

- **Method:** DELETE
- **URL:** `/favorite/:favoriteId`
- **Path parameters:**
  - `favoriteId` (string, required): Identifier of the favorites entry to delete.

Example request

http

```
DELETE /favorite/fav_123
Authorization: Bearer <access_token>
```

Typical successful response

json

```
{
  "favoriteId": "fav_123",
  "deleted": true
}
```

Common status codes

- `200 OK` or `204 No Content` – Favorite removed successfully.
- `401 Unauthorized` – Missing or invalid authentication.
- `403 Forbidden` – Caller does not own this favorite entry.
- `404 Not Found` – Favorite entry not found.
- `500 Internal Server Error` – Unexpected server error.

Flow context

1. Favorites are created via **Add Favorite** (`POST /favorite`).
2. User views items via **Get Favorites** (`GET /favorite`).
3. User removes an item using **Delete Favorite**; it will no longer appear in their favorites or related shares.

PATH VARIABLES

favoriteId