
Base URL

```
http://localhost:3000/api
```

---

1. Authentication

1.1 Register a New User

Endpoint

```
POST /auth/register
```

Request Body (JSON)

```
{
  "username": "string",
  "password": "string",
  "email": "string"
}
```

Response (JSON)

```
{
  "id": "uuid",
  "username": "string",
  "email": "string",
  "created_at": "timestamp"
}
```

Details

- Creates a new user.
- Does **not** require authentication.
- Equivalent in functionality to `POST /users` (both are open routes).

1.2 Log In

Endpoint

```
POST /auth/login
```

Request Body (JSON)

```
{
  "username": "string",
  "password": "string"
}
```

Response (JSON)

```
{
  "token": "string"
}
```

Details

- Authenticates the user and returns a JWT token.
- This token must be included in the **Authorization** header for endpoints that require authentication.
  Example:

  ```
  Authorization: Bearer <token>
  ```

  or just

  ```
  Authorization: <token>
  ```

  (Your code supports both styles, but Bearer is the usual convention.)

1.3 Retrieve Current Authenticated User

Endpoint

```
GET /auth/me
```

Headers

```
Authorization: Bearer <token>
```

(or `Authorization: <token>`)

Response (JSON)

```
{
  "id": "uuid",
  "username": "string",
  "email": "string",
  "created_at": "timestamp"
}
```

Details

- Requires a valid token in the `Authorization` header.
- Returns user info for the currently authenticated user.

---

2. Users

2.1 Create a User

Endpoint

```
POST /users
```

Request Body (JSON)

```
{
  "username": "string",
  "password": "string",
  "email": "string"
}
```

Response (JSON)

```
{
  "id": "uuid",
  "username": "string",
  "email": "string",
  "created_at": "timestamp"
}
```

Details

- Functionally similar to `POST /auth/register`.
- Does **not** require authentication.

2.2 Get All Users

Endpoint

```
GET /users
```

Response (JSON Array)

```
[
  {
    "id": "uuid",
    "username": "string",
    "email": "string",
    "created_at": "timestamp"
  },
  ...
]
```

Details

- Returns a list of all users.
- Does **not** require authentication.

2.3 Delete a User

Endpoint

```
DELETE /users/:id
```

Response (JSON)

```
{
  "id": "uuid",
  "username": "string",
  "email": "string"
}
```

Details

- Deletes the user with the provided `id`.
- Does **not** require authentication by default in the provided code (though in production you would likely want to secure it).

---

3. Pictures

3.1 Create a Picture

Endpoint

```
POST /pictures
```

Request Body (JSON)

```
{
  "URL": "string",
  "caption": "string"
}
```

Response (JSON)

```
{
  "id": "uuid",
  "URL": "string",
  "caption": "string",
```

```
  "created_at": "timestamp"
}
```

Details

- Creates a new picture record in the database with the provided URL and caption.
- Does **not** require authentication by default.

3.2 Get All Pictures

Endpoint

```
GET /pictures
```

Response (JSON Array)

```
[
  {
    "id": "uuid",
    "URL": "string",
    "caption": "string",
    "created_at": "timestamp"
  },
  ...
]
```

Details

- Returns all pictures stored in the database.
- Does **not** require authentication.

3.3 Delete a Picture

Endpoint

```
DELETE /pictures/:id
```

Response (JSON)

```
{
  "id": "uuid",
  "URL": "string",
  "caption": "string"
}
```

Details

- Deletes the picture with the given `id`.
- Does **not** require authentication by default in the code (though normally you would secure it).

---

4. Users & Pictures Relationship

4.1 Link a User to a Picture

Endpoint

```
POST /users/:userId/pictures/:pictureId
```

Response (JSON)

```
{
  "id": "uuid",
  "user_id": "uuid",
  "picture_id": "uuid",
  "created_at": "timestamp",
  "updated_at": "timestamp"
}
```

Details

- Links a given user (`:userId`) to a given picture (`:pictureId`) in the join table.
- Does **not** require authentication by default.

4.2 Get Pictures Uploaded by a Specific User

Endpoint

```
GET /users/:userId/pictures
```

Response (JSON Array)

```
[
  {
    "id": "uuid",
    "URL": "string",
    "caption": "string",
    "created_at": "timestamp"
  },
  ...
]
```

Details

- Returns all pictures associated with the given user.

4.3 Remove a User-Picture Link

Endpoint

```
DELETE /users/:userId/pictures/:pictureId
```

Response (JSON)

```
{
  "id": "uuid",
  "user_id": "uuid",
  "picture_id": "uuid"
}
```

Details

- Removes the link between the user and the picture in the join table.

4.4 Get Pictures by **Username** (Alternate)

Endpoint

```
GET /username/:username/pictures
```

Response (JSON Array)

```
[
  {
    "username": "string",
    "user_id": "uuid",
    "picture_id": "uuid",
    "picture_url": "string",
    "picture_caption": "string",
    "picture_createdat": "timestamp"
  },
  ...
]
```

Details

- Similar to GET `/users/:userId/pictures`, but fetches by username string instead of numeric `id`.

---

5. Image Upload to Imgur

5.1 Upload an Image

Endpoint

```
POST /upload
```

Request

- **Content Type**: `multipart/form-data`
- **Form Field Name**: `image` (required)
- Optional form field: `caption`
- **Headers**: Must include a valid token

```
Authorization: Bearer <token>
```

Response (JSON)

```
{
  "message": "Upload successful",
  "picture": {
    "id": "uuid",
    "URL": "string",
    "caption": "string",
    "created_at": "timestamp"
  }
}
```

Details

- **Requires Authentication**.
- Uploads the given `image` to Imgur, then stores the resulting URL in the database as a new `pictures` record.
- Automatically links the newly created `picture` record to the authenticated user.

---

6. Feed

6.1 Fetch the Global Feed

Endpoint

```
GET /feed
```

Response (JSON Array)

```
[
  {
    "user_id": "uuid",
    "picture_id": "uuid",
    "url": "string",
    "caption": "string",
    "created_at": "timestamp",
    "username": "string"
```

```
  },
  ...
]
```

Details

- Fetches a joined view of all user-picture pairs, showing who posted which picture and the picture's details.
- Does **not** require authentication.

---

7. Comments

7.1 Create a Comment

Endpoint

```
POST /createComment
```

Request Body (JSON)

```
{
  "user_id": "uuid",
  "picture_id": "uuid",
  "content": "string"
}
```

Response (JSON)

```
{
  "message": "Comment created",
  "comment": {
    "id": "uuid",
    "content": "string",
    "created_at": "timestamp"
  }
}
```

Details

- Creates a record in `comments`, then links it to the specified user and picture via a join table.

- Currently does **not** enforce authentication in the code; in production, you'd typically require a token or at least validate that `user_id` matches the authenticated user.

7.2 Edit a Comment

Endpoint

```
PUT /editComment
```

Request Body (JSON)

```
{
  "comment_id": "uuid",
  "content": "string"
}
```

Response (JSON)

```
{
  "id": "uuid",
  "content": "string",
  "created_at": "timestamp"
}
```

Details

- Updates the comment text in the `comments` table.
- Returns the updated comment row.

7.3 Delete a Comment

Endpoint

```
DELETE /deleteComment
```

Request Body (JSON)

```
{
  "comment_id": "uuid"
}
```

Response (JSON)

```
{
  "id": "uuid",
  "content": "string",
  "created_at": "timestamp"
}
```

Details

- Deletes the comment record in `comments`.
- Returns the deleted comment.

7.4 Get All Comments for a Picture

Endpoint

```
GET /:pictureId/comments
```

Response (JSON Array)

```
[
  {
    "user_id": "uuid",
    "username": "string",
    "content": "string",
    "created_at": "timestamp"
  },
  ...
]
```

Details

- Returns all comments associated with the specified picture, joined with the user who commented.

- Does **not** require authentication.

---

8. Likes

8.1 Create a Like

Endpoint

```
POST /createLike
```

Request Body (JSON)

```
{
  "user_id": "uuid",
  "picture_id": "uuid"
}
```

Response (JSON)

```
{
  "message": "Like added",
  "like": {
    "id": "uuid",
    "created_at": "timestamp"
  }
}
```

Details

- Inserts a new record into `likes`, and then links that record to the user and picture in a join table.
- Does **not** require authentication by default in the code.

8.2 Delete a Like

Endpoint

```
DELETE /deleteLike
```

Request Body (JSON)

```
{
  "like_id": "uuid"
}
```

Response (JSON)

```
{
  "id": "uuid",
  "created_at": "timestamp"
}
```

Details

- Deletes the `like` record.
- Returns the deleted row.

8.3 Get All Likes for a Picture

Endpoint

```
GET /:pictureId/likes
```

Response (JSON Array)

```
[
  {
    "user_id": "uuid",
    "username": "string",
    "created_at": "timestamp"
  },
  ...
]
```

Details

- Returns all likes for the specified picture, including the user's ID, username, and when the like was created.

---

9. Error Handling

If an error occurs, you will receive a JSON response in this format:

```
{
  "error": "Error message here"
}
```

The HTTP status code will be set appropriately, such as:

- `400` for bad requests
- `401` for unauthorized
- `404` for resource not found
- `500` for server errors etc.

---

10. Important Notes

1. **Authentication**
   - Currently, only `GET /auth/me` and `POST /api/upload` explicitly require a token in the `Authorization` header to function.
   - In a production setting, you would likely add `isLoggedIn` middleware to many of these routes (like creating or deleting comments, likes, pictures, etc.) to ensure the user has permission to act.
2. **Seeding & Initial Data**
   - The application seeds some initial data, but that does not affect how these endpoints should be used.
3. **POST** `/auth/register` and **POST** `/users`
   - Both create new users. They are semantically different but function almost identically.
4. **Bearer Token Format**
   - In the current code, you can pass just the token or `Bearer <token>`. Standard practice is `Bearer <token>`.
5. **Imgur Upload**

- Imgur upload uses `process.env.IMGUR_ACCESS_TOKEN` as the `Client-ID` header. Make sure this environment variable is properly set.

End of Updated Documentation