**Deepfake Detection Platform — Full Technical Documentation**

**Base URL:** https://deepfake-51fk.onrender.com  
**Stack:** MERN (React frontend, Node/Express backend, MongoDB), Socket.IO, Hugging Face inference API, FFmpeg for video processing.  
**Hugging Face models used:** default to umm-maybe/AI-image-detector (AI/image detector) and dima806/deepfake\_vs\_real\_image\_detection (deepfake classifier). These are configurable with env variables.

**1. Environment variables (required)**

Create .env (locally) or set environment variables in Render.

# Server

PORT=5000

MONGODB\_URI=mongodb+srv://<user>:<pass>@cluster.mongodb.net/dbname

JWT\_SECRET=supersecretjwtkey

# Hugging Face

HF\_API\_KEY=hf\_xxxYourHfApiKeyxxxx

HF\_MODEL\_AI=umm-maybe/AI-image-detector

HF\_MODEL\_DF=dima806/deepfake\_vs\_real\_image\_detection

**How to set on Render**

1. Open your Render service → **Environment** → **Environment Variables**.
2. Add keys: HF\_API\_KEY, HF\_MODEL\_AI, HF\_MODEL\_DF and other for cloudinary.
3. Save and redeploy.

**Important:** Keep HF\_API\_KEY secret. Do not expose in frontend.

**2. System architecture (high level)**

Clients (Browser) -> Frontend (React/Vite)

| (REST) POST /api/detect/file |

v v

Backend (Express) -> Queue/Worker (optional) -> HuggingFace inference & ffmpeg -> MongoDB (Upload results)

| socket.io events to user (analysis\_complete)

v

Admin/Analyst UI (React) -> /api/admin/\* endpoints

Components:

* **Frontend**: Upload UI, user dashboard, admin dashboard.
* **Backend**: Express API; routes for auth, detect, admin, history.
* **Hugging Face**: inference API calls using HF\_API\_KEY.
* **FFmpeg**: extract frames from video (/usr/bin/ffmpeg via apt.txt).
* **MongoDB**: store User, Upload,
* **Socket.IO**: send real-time analysis\_complete notifications to specific users.

**3. Key Mongoose Schemas (examples)**

**models/User.js**

const mongoose = require('mongoose');

const UserSchema = new mongoose.Schema({

username: {type: String, required: true, unique: true},

email: {type: String, required: true, unique: true},

password: {type: String, required: true}, // hashed

role: {type: String, enum: ['user','admin'], default: 'user'},

isSuspended: {type: Boolean, default: false},

lastLogin: {type: Date},

uploadsCount: { type: Number, default: 0 },

}, {timestamps: true});

module.exports = mongoose.model('User', UserSchema);

**models/Upload.js**

const UploadSchema = new mongoose.Schema({

filename: { type: String, required: true },

originalName: String,

mimetype: String,

size: Number,

url: String, //cloudinary url

user: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },

results: mongoose.Schema.Types.Mixed,

verdict: String,

confidence: Number,

}, { timestamps: true });

module.exports = mongoose.model('Upload', UploadSchema);

**4. Backend API Reference (detailed)**

All endpoints base: https://deepfake-51fk.onrender.com/api

**Auth**

* **POST /api/auth/register** — Register
  + Body: { username, email, password }
  + Response: { success, token, user } (token = JWT)
* **POST /api/auth/login**
  + Body: { email, password }
  + Response: { token, user }
* **GET /api/auth/me**
  + Header: Authorization: Bearer <token>
  + Response: { user }

**File detection**

* **POST /detect/file** — (Protected) upload & analyze
  + Headers: Authorization: Bearer <token>
  + Body: multipart/form-data with file
  + Behavior:
    - If image: run both Hugging Face models on the image.
    - If video: extract frames using ffmpeg (configurable fps), run detectors on frames, aggregate results.
    - Save Upload doc in MongoDB.
    - Upload image and video to cloudinary
    - Emit analysis\_complete to the user via Socket.IO (if registered).
  + Response example:

{

"success": true,

"results": {

"AI Image Detector": [

{ "label": "artificial", "score": 0.87856 },

{ "label": "human", "score": 0.12143 }

],

"Deepfake Detector": [

{ "label": "Real", "score": 0.8476 },

{ "label": "Fake", "score": 0.15236 }

]

},

"verdict": { "verdict": "Real", "confidence": 86 }

}

* **GET /history** — user upload history
  + Headers: Authorization: Bearer <token>
  + Query: ?page=1&limit=20
  + Response: list of Upload docs (with results and metadata)

**Admin endpoints (require admin role)**

* **GET /api/admin/users** — list users, supports page, limit, search, role, status
* **PATCH /api/admin/user/:id** — update role or suspend: {"role":"admin"} or {"suspend": true}
* **DELETE /api/admin/user/:id**
* **GET /api/admin/dashboard** — stats used by admin dashboard (KPIs, trends)

**5. How the Hugging Face calls are performed (backend helper)**

Example helper using axios (must use HF\_API\_KEY env var):

const axios = require('axios');

async function hfImageDetect(modelId, filePath) {

const imageData = fs.readFileSync(filePath);

const res = await axios.post(

`https://api-inference.huggingface.co/models/${modelId}`,

imageData,

{

headers: {

Authorization: `Bearer ${process.env.HF\_API\_KEY}`,

"Content-Type": "application/octet-stream",

},

timeout: 120000,

}

);

return res.data; // array of {label, score} or model-specific

}

**Model selection:** The backend uses:

const models = {

ai: process.env.HF\_MODEL\_AI || "umm-maybe/AI-image-detector",

df: process.env.HF\_MODEL\_DF || "dima806/deepfake\_vs\_real\_image\_detection",

};

To change models, update HF\_MODEL\_AI or HF\_MODEL\_DF in env and redeploy.

**Note:** Hugging Face model outputs may have different label casing/format — parse accordingly (e.g., artificial, human, Real, Fake).

**6. Combining results → final verdict**

Example algorithm (already in your backend; documented) — tweak thresholds as needed:

1. Extract scores:
   * aiArtificial = score for label matching /artificial|fake|synthetic/
   * aiHuman = score for label matching /human|real/
   * dfFake = score for label matching /fake|manipulated/
   * dfReal = score for label matching /real|authentic/
2. Rules (example):

* If aiArtificial > 0.8 && dfReal > 0.6 → verdict = "AI-Generated", confidence aiArtificial.
* If dfFake > 0.6 → verdict = "Deepfake/Manipulated".
* If aiHuman > 0.7 && dfReal > 0.7 → verdict = "Real".
* Else → verdict = "Uncertain".

Store verdict and confidence in Upload doc.

**7. Socket.IO / Real-time notifications**

**Server-side**

* Maintain userSocketMap (map userId → socketId).
* On Socket connect, client emits register\_user with userId.
* Server stores mapping: userSocketMap[userId] = socket.id.
* On analysis completion, server emits to that socket id:

io.to(socketId).emit('analysis\_complete', {

id: uploadDoc.\_id,

filename: uploadDoc.originalName,

verdict: uploadDoc.verdict,

confidence: uploadDoc.confidence,

url: uploadDoc.url,

createdAt: uploadDoc.createdAt,

});

**Client-side (example)**

const socket = io(BACKEND\_URL, { auth: { token } });

socket.on('connect', () => socket.emit('register\_user', user.\_id));

socket.on('analysis\_complete', (data) => {

// sound, update history list

});

Event names: register\_user, analysis\_complete.

**8. File size & upload limits, chunked upload**

* Default limits: images ≤ 5MB, videos ≤ 50MB.
* Then call /api/detect/file with merged file or trigger analysis server-side.
* And upload the file to cloudinary to get the url

**9. FFmpeg (video frame extraction) usage**

* Ensure ffmpeg available at /usr/bin/ffmpeg (installed via apt.txt on Render).

**10. Security recommendations**

* Hash passwords with bcrypt.
* Use JWT with strong secret and expiry.
* Validate file types & sanitize filenames.
* Scan uploaded files for malware (optional).
* Use HTTPS everywhere (Render provides TLS by default).
* Protect admin endpoints with role-based middleware.
* Rate-limit uploads (e.g., express-rate-limit) to protect from abuse.
* Store secrets (HF\_API\_KEY) in Render environment variables — never in frontend.

**11. Cost & rate-limit considerations**

* Hugging Face Inference API may have usage limits and costs — batch frames, limit fps to reduce calls, cache results.
* FFmpeg CPU and memory usage can be high for large videos — consider moving video processing to a worker queue or running in a larger instance.

**12. Sample requests**

**cURL — upload image**

curl -X POST "https://deepfake-51fk.onrender.com/api/detect/file" \

-H "Authorization: Bearer <TOKEN>" \

-F "file=@/path/to/image.jpg"

**cURL — admin get stats**

curl -H "Authorization: Bearer <ADMIN\_TOKEN>" \

"https://deepfake-51fk.onrender.com/api/admin/dashboard"

**13. Notes on models you used**

* umm-maybe/AI-image-detector — identifies AI-generated images vs human photos. Output labels typically something like artificial and human.
* dima806/deepfake\_vs\_real\_image\_detection — trained for face-swap style deepfakes; outputs Real/Fake labels and confidence.

**14. Short developer quickstart (local)**

1. Clone repo, go to backend:

cd backend

cp .env.example .env

# fill .env keys incl HF\_API\_KEY, HF\_MODEL\_AI, HF\_MODEL\_DF

npm install

# ensure ffmpeg installed locally (apt/brew)

node server.js

1. Frontend:

cd frontend

npm install

npm run dev