Sports Reels: System Documentation

Generated by g-h-0-S-t

May 2025

Version 1.0.0
Deployed at: https://sports-reels.onrender.com
Repository: https://github.com/g-h-0-S-t/sports-reels

Introduction

Sports Reels is a web application that allows users to generate and view short video reels about sports celebrities. Users can input a celebrity's name, video title, description, and narration script to create a video montage of images sourced from Unsplash, accompanied by text-to-speech audio. Videos are stored in a GitHub repository and displayed in a scrollable, mobile-friendly interface. The application is built with Next.js for the frontend and backend, Python for video generation, and deployed on Render.

Purpose

The application enables fans to:

- Generate personalized sports celebrity video reels.
- Search and view existing reels by celebrity name, title, or description.
- Enjoy a responsive, Instagram-like interface with auto-playing videos.

Technology Stack

- **Frontend**: Next.js 15.3.1, React 18.3.1, CSS (globals.css with Lato font).
- **Backend**: Next.js API routes, Node.js 18+, Python 3 (moviepy, gTTS).
- **Storage**: GitHub repository for videos and metadata (videos.json).
- **APIs**: Unsplash for images, GitHub API for file access.
- **Deployment**: Render (Node.js environment with Python support).
- **Dependencies**: node-fetch, dotenv, Pillow, numpy, imageio-ffmpeg, requests, tqdm.

System Architecture

Sports Reels follows a client-server architecture with a Next.js frontend and backend, integrated with a Python script for video generation. Videos and metadata are stored in a GitHub repository, accessed via API routes. The system is deployed as a single service on Render.

Components

- **Frontend (pages/index.js)**: Renders the UI, handles user input, and displays videos.
- **Backend API Routes**:
  - `/api/generate-video`: Triggers video generation and pushes to GitHub.
  - `/api/proxy-json`: Fetches videos.json from GitHub.
  - `/api/proxy-video`: Streams video files from GitHub.
  - `/api/refresh-videos`: Refreshes video metadata.
- **Python Script (generate_videos.py)**: Generates videos using Unsplash images and gTTS audio.
- **GitHub Repository**: Stores videos (e.g., `videos/tiger-woods-history.mp4`) and metadata (`videos.json`).
- **Render Deployment**: Hosts the Node.js app and Python environment.

Data Flow

1. User submits a form with celebrity name, title, description, and script via `pages/index.js`.
2. `/api/generate-video` clones the GitHub repo, runs `generate_videos.py`, and pushes the video and updated `videos.json`.
3. `generate_videos.py` fetches images from Unsplash, generates audio with gTTS, and creates a 480p video with moviepy.
4. `pages/index.js` polls `/api/proxy-video` to confirm video availability.
5. `/api/refresh-videos` or `getStaticProps` fetches `videos.json` via `/api/proxy-json` to update the video list.
6. Videos are streamed from GitHub via `/api/proxy-video` and displayed in the UI.

Frontend (pages/index.js)

The frontend, implemented in `pages/index.js`, is a Next.js page that serves as the main interface. It uses React hooks for state management and IntersectionObserver for video autoplay.

Features

- **Start Screen**: Displays a welcome message and "Start Reels" button.
- **Form**: Allows users to input celebrity name, title, description, and narration script. The video URL is auto-generated.
- **Search**: Filters videos by celebrity name, title, or description.
- **Reels Display**: Shows videos in a scrollable, full-screen layout with title and description overlays.
- **Loader**: Displays a spinning golf ball during video generation or refresh.

State Management

- `videos`: Array of video objects from `videos.json`.
- `displayedVideos`: Filtered subset of `videos` for display.
- `isStarted`: Toggles between start screen and main UI.
- `formData`: Stores form inputs (celebrityName, title, description, customScript, videoUrl).
- `searchQuery`: Stores search input.
- `isGenerating`, `isRefreshing`: Control loader visibility.
- `error`: Displays error messages.
- `isFormActive`: Pauses videos when form is focused.
- `refreshKey`: Forces re-render of video list.
- `videoRefs`: References video elements for autoplay.

Key Functions

- `getStaticProps`: Fetches `videos.json` at build time via `/api/proxy-json` with Incremental Static Regeneration (ISR, revalidate: 60s).
- `handleSubmit`: Sends form data to `/api/generate-video`, polls for video availability, and updates the video list.
- `pollVideo`: Polls `/api/proxy-video` (15 attempts, 5s intervals) to confirm video availability.

- `refreshVideos`: Fetches updated `videos.json` via `/api/refresh-videos`.
- `handleSearch`: Filters `displayedVideos` based on `searchQuery`.
- `fetchWithTimeout`: Handles HTTP requests with retries (2 attempts, 2s delay) and optional timeout (skipped for `/api/generate-video`).

UI Behavior

- Videos autoplay when 50% visible (via IntersectionObserver) and pause when out of view or form is active.
- The loader appears during video generation or refresh, with a spinning golf ball animation.
- Errors are displayed below the form (e.g., "Failed to generate video: ...").
- The UI is responsive, with mobile-friendly styles (globals.css).

Backend (API Routes)

The backend consists of Next.js API routes handling video generation, metadata fetching, and video streaming.

/api/generate-video

- **Method**: POST
- **Input**: JSON with `celebrityName`, `title`, `description`, `customScript`.
- **Process**:
    a. Clones the GitHub repo to a temporary directory.
    b. Runs `generate_videos.py` to create a 480p video.
    c. Updates `videos.json` with new video metadata.
    d. Commits and pushes changes to GitHub.
    e. Cleans up the temporary directory.
- **Output**: JSON with `videoUrl` and updated `videos` array.
- **Error Handling**: Returns 400 (missing fields), 500 (script or Git errors).

/api/proxy-json

- **Method**: GET
- **Input**: Query parameter `url` (GitHub API URL for `videos.json`).
- **Process**: Fetches `videos.json` from GitHub with retries (5 attempts, 1s delay).

- **Output**: JSON content of `videos.json`.
- **Error Handling**: Returns 400 (missing URL), 500 (fetch errors).

/api/proxy-video

- **Method**: GET
- **Input**: Query parameter `url` (GitHub raw video URL).
- **Process**: Streams video from GitHub with retries (5 attempts, 5s delay) and GitHub token authentication.
- **Output**: Video stream (Content-Type: video/mp4).
- **Error Handling**: Returns 400 (missing URL), 401 (invalid token), 403 (permissions), 404 (not found), 429 (rate limit), 500 (other errors).

/api/refresh-videos

- **Method**: GET
- **Process**: Clones the GitHub repo, reads `videos.json`, and returns the videos array.
- **Output**: JSON with `videos` array.
- **Error Handling**: Returns 500 (clone or read errors).

Python Video Generation (generate_videos.py)

The `generate_videos.py` script generates videos using Unsplash images and gTTS audio, optimized for Render's limited resources (512MB RAM, 0.1 CPU).

Workflow

1. **Parse Arguments**: Accepts `–celebrity`, `–title`, `–description`, `–script`.
2. **Download Images**: Fetches 10 images from Unsplash via API, resizes to 854x480.
3. **Generate Audio**: Converts script to MP3 using gTTS.
4. **Create Video**: Combines images (4.3s each, 24fps) with audio using moviepy, outputs 480p video (libx264, ultrafast preset).
5. **Cleanup**: Removes temporary files.

Optimizations

- **Low Memory**: Resizes images to 480p, uses `ultrafast` preset, limits threads to 1.
- **Error Handling**: Logs errors to `generate_videos.log`, verifies image integrity.
- **Temporary Directory**: Uses `tempfile` to manage intermediate files.

Storage (GitHub Repository)

Videos and metadata are stored in the GitHub repository `g-h-0-S-t/sports-reels-videos`:

- **videos.json**: Contains an array of video objects (`id`, `celebrityName`, `title`, `description`, `videoUrl`).
- **videos/**: Stores MP4 files (e.g., `tiger-woods-history.mp4`).
- **Access**: Uses GitHub Personal Access Token (GITHUB_TOKEN) with `repo` or `public_repo` scope.

Deployment (Render)

The application is deployed on Render as a web service:

- **Environment**: Node.js with Python 3 support.
- **Build**: Installs Node (npm install) and Python dependencies (pip3 install -r requirements.txt), runs Next.js build.
- **Start**: Runs `npm run start`.
- **Environment Variables**:
  - `NODE_ENV`: production
  - `NEXT_PUBLIC_APP_URL`: https://sports-reels.onrender.com
  - `GITHUB_TOKEN`: GitHub PAT
  - `UNSPLASH_ACCESS_KEY`: Unsplash API key
- **Configuration**: `render.yaml` defines the service.

Workflow Example

1. User visits https://sports-reels.onrender.com, clicks "Start Reels".
2. User enters:
   - Celebrity Name: Tiger Woods
   - Title: Tiger Woods History
   - Description: A short history of Tiger Woods
   - Script: Eldrick Tont Tiger Woods, born December 30, 1975, is an American professional golfer...

3. Form submits to `/api/generate-video`, which:
    - Clones the GitHub repo.
    - Runs `generate_videos.py` to create `videos/tiger-woods-history.mp4`.
    - Updates `videos.json`.
    - Pushes to GitHub.
4. UI polls `/api/proxy-video` until the video is available.
5. Video appears in the reels container, auto-plays when visible.
6. User searches "Tiger" to filter videos.

Error Handling

- **Frontend**: Displays errors (e.g., "Failed to generate video: ...") below the form.
- **Backend**: Returns HTTP status codes (400, 401, 403, 404, 429, 500) with error messages.
- **Python**: Logs errors to `generate_videos.log`, raises exceptions for failures.
- **Retries**: `/api/proxy-json` (5 attempts, 1s delay), `/api/proxy-video` (5 attempts, 5s delay), `fetchWithTimeout` (2 attempts, 2s delay).

Maintenance

- **Monitor Logs**: Check Render logs and `generate_videos.log` for errors.
- **Update Dependencies**: Regularly update `package.json` and `requirements.txt`.
- **GitHub Token**: Rotate `GITHUB_TOKEN` periodically, ensure correct scopes.
- **Unsplash API**: Monitor rate limits (50 requests/hour).
- **Render Resources**: Upgrade from free tier if performance issues arise.

Conclusion

Sports Reels is a user-friendly application for creating and viewing sports celebrity video reels. Its integration of Next.js, Python, GitHub, and Render provides a scalable solution for video generation and display. The codebase is optimized for low-resource environments, with robust error handling and a responsive UI.