ChatGPT

Designing a Content Management System with Privacy, Tracking, and Revocation

Introduction

Organizations and creators often need to distribute user-generated content (stories, videos, transcripts, etc.) across multiple platforms while retaining control over privacy, usage, and intellectual property. This report explores how to build a custom content management system that allows: (1) fine-grained privacy settings on stored content, (2) controlled access for different people or groups, (3) cross-posting to social media and blogs, and (4) robust tracking of where content appears and how it performs, with the ability to **revoke** or remove content across the internet when needed. We discuss best practices for database design, content distribution methods, tracking techniques, and revocation mechanisms, providing implementation-level details and references to current approaches.

Secure Content Storage and Privacy Controls

A foundational element is a **central content repository** (e.g. a database + storage service) where all stories and media are stored. In this database, each content item should have metadata defining its privacy level and ownership. Best practices include:

- **Privacy Levels:** Define levels such as *private*, *group-only*, *public*, etc., and store this as a field for each content item. For instance, a story can be marked private (visible only to its author or admins) or public (visible to everyone). The system uses these flags to enforce who can view or share the content. This approach is aligned with Tim Berners-Lee's Solid project goal of giving users full control of their data access [1]. In Solid, users control access to their data pods, and similarly our system will enforce access control on content based on the creator's settings.

- **Access Control Lists (ACLs) or Roles:** Implement row-level security in the database or application-layer checks. Each content item can list which users or groups have access. For example, user Alice's story might be shared with *Bob* explicitly or with all users in *Team X*. The system should check a user's identity and permissions before serving the content. This can be done via role-based access control or attribute-based policies. Modern frameworks or libraries (for example, using OAuth scopes or an identity service) can facilitate managing these permissions.

- **Secure Storage for Media:** For large media files (videos, audio), use a secure object storage (like AWS S3 or Azure Blob) with generated URLs. You can generate time-limited URLs for private content so that only authorized viewers can access them. If content is public, a CDN can be used for performance. Ensure that even within storage, sensitive content is either encrypted or behind access checks if needed (some systems even encrypt files such that only authorized clients can decrypt, though that adds complexity).

- **Metadata and Tagging:** Store content metadata like title, description, author, creation date, etc. Also record a unique Content ID. This ID will be crucial for tracking and linking the content across platforms (as a reference in analytics or when issuing removal commands).

By designing a structured content database with these elements, you set the stage for fine-grained privacy control and selective sharing of content.

User Access Management (Privacy Levels and Sharing)

To “give people access,” the system needs a robust **user management module** and sharing mechanism:

- **Authentication & Accounts:** Each person (content creator, viewer, campaign partner, etc.) should have an account and authenticate (e.g., via email/password or single sign-on). Once logged in, their identity and roles determine what they can do or see.

- **Privacy Settings UI:** Provide creators a simple interface to set the privacy level when they upload or create a story. For example, a dropdown: *Only me*, *Friends/Team*, *Public*. If group-based sharing is needed, the system should allow the creator to pick which group or individuals can access the content. This will update the ACL/permissions in the database for that content.

- **Invitation/Link-Based Access:** In some cases, a creator might want to share content via a secret link (for people who don’t have accounts). Implement the ability to generate a unique, hard-to-guess URL that grants read-access. These links can be revocable (e.g., invalidate the link to revoke access). This is similar to how some cloud document services allow sharing a view link that can later be disabled. Keep track of who was given such links if possible (perhaps the creator enters an email for the person, so you log it).

- **Roles and Restrictions:** Define roles like *Admin*, *Editor*, *Viewer*, *External Partner* etc. An admin might override privacy settings (for moderation), whereas a viewer can only see content shared with them. Role-based access control ensures only authorized modifications (e.g., only content owner or admins can delete or edit a story).

- **Audit Trails:** For accountability, maintain logs of who accessed or modified content and when. This ties into tracking – even internally, you want to know if a sensitive story marked “private” was accessed by someone (to ensure privacy isn’t breached).

By implementing these controls, the system mirrors the kind of privacy options seen on social platforms (like Facebook’s post visibility settings) but tailored to your custom solution.

Cross-Platform Content Distribution

A key requirement is to post the content to various external platforms (Facebook, personal blogs, business blogs, campaign pages, YouTube, news sites) **in a way that it can be tracked and potentially revoked**. There are two primary approaches here:

1. **Embedded Content via a Central Hub:** This approach uses your system as the single source of truth and **embeds** the content on other sites. For example, instead of copying a whole story to each blog or social site, you embed a snippet or player that pulls from your central database. Digital Asset Management (DAM) systems commonly use this method by providing an HTML embed code for an asset [2] [3]. The embed code (often an `<iframe>` or a script) displays the content from your server. This has several advantages:
2. *Central Control:* Updates to the content (or deletion) on your server propagate to all sites using the embed. You can **update, replace, or remove content centrally** without editing each

external post [4]. If a story must be revoked, you simply disable it on your system – all embed instances will either show a “content removed” message or disappear.
3. *Consistency:* The content looks the same everywhere and uses the latest version [5], since it’s pulled from one source.
4. *Analytics:* Embed codes can include tracking parameters or scripts, allowing you to monitor views and interactions across platforms [6]. Your server can log each time the embed is loaded, giving a count of impressions per site.
5. *Access Control:* You can enforce privacy even on embeds. For example, the embed code could require a token or only function on authorized domains. Many DAMs let you restrict which websites can embed the content (preventing unauthorized use) [7] [8]. You might allow a partner’s blog domain but not others.

Implementing Embeds: You would build an API or widget that outputs the content. External sites just include a small code snippet. For text stories, this could be a script that fetches and inserts the story HTML. For videos, an iframe that streams the video from your server or a cloud video service under your control. Ensure the embed is responsive and works across devices.

One challenge: some platforms (like Facebook or certain news sites) may not allow arbitrary third-party embeds for security reasons. In such cases, you might only post a link or a preview on those platforms (discussed next), or negotiate the embed if possible (some sites allow YouTube iframes, etc., but not custom ones easily).

1. **Native Cross-Posting with Tracking Links:** In this approach, your system uses platform APIs to publish the content natively on each platform (e.g., via Facebook Graph API, Twitter/X API, YouTube Data API, etc.). The content then lives on those platforms. To retain tracking and revocation ability:
2. *Include Trackable Links or IDs:* Ensure that the post on the external platform includes a reference to your content ID or a special tracking link back to your system. For example, a Facebook post could have a URL pointing to the full story on your site or an identifier in the text (like a campaign hashtag or code). Similarly, a YouTube video description might include a unique code or link. This way, you can later search for or identify your content on those platforms.
3. *Use Analytics from Platforms:* When posting natively, leverage the platform’s analytics if available. For instance, YouTube provides view counts and other metrics (which you can fetch via API for your video). Facebook pages have Insights data for posts. Your system can periodically call these APIs (with the user’s permission/token) to collect metrics like views, likes, shares from each platform and aggregate them in your database.
4. *Storing Post IDs:* Save the identifier of each external post (e.g., the Facebook post ID, Tweet ID, YouTube video ID) in your database linked to the content record. This is crucial for revocation – it allows your system to later issue a delete command through the API if needed. It also helps for tracking, as you can retrieve stats or check status by ID.

Implementing Cross-Posting: You will need to integrate with each platform’s API and handle authentication (e.g., OAuth tokens for each user’s account or a page access token for Facebook Pages). Your system could have a “Publish” function where the user chooses which platforms to push to. Under the hood, the system formats the content appropriately for each (maybe truncating text for Twitter, choosing an image for Facebook, etc.) and calls the API to post. After posting, it stores the response (IDs, links). Make sure to handle errors (e.g., if a token expired, or if the platform downscales an image).

Note: When posting natively, actual revocation is harder because the content is copied onto external servers. Therefore, consider using a **hybrid**: post only a snippet or summary natively and direct viewers to the full content on your controlled site. This way, revoking the full content is as easy as disabling it on your site, and the external posts only have a teaser (which you can delete or leave as-is). This hybrid approach uses external platforms for reach but the core content remains centrally managed.

Tracking Content Usage Across the Internet

Tracking where and how the content is used is “most important” according to the requirements. This involves both tracking authorized placements (where you intentionally shared the story) and monitoring for any **unauthorized or unexpected reuse** across the web.

Key tracking methods include:

- **Analytics for Authorized Posts:** As mentioned, if using embed codes, your server logs can provide when, where (which domain or which user account) the content was viewed [6]. Implement analytics endpoints that collect events like “content ID 123 viewed on site XYZ at time T”. This could be as simple as a web beacon (an invisible 1x1 image or script in the embed) that pings your server for each view. For richer analytics, integrate something like Google Analytics or a custom dashboard to show views over time, unique visitors, etc., for each content piece. When posting natively on social platforms, use their insight data via API as described to gather view counts, engagement, etc.

- **Unique Identifiers or Watermarks:** To track usage beyond your own distribution, embed unique identifiers in the content itself:

- *Text Stories:* Include a distinctive phrase or a hidden identifier (for example, a deliberate sequence of characters or a stylistic watermark in the text). If someone copies the text to another site, you can search for that identifier. Even a Google search for a unique sentence from the story can reveal copies. (There are services that automate this kind of search.)
- *Images/Videos:* Use **digital watermarking** techniques. Digital watermarks are imperceptible markers embedded in media that do not alter the visual appearance but can be detected by special software [9] [10]. For instance, a subtle pattern in an image or slight variations in video frames can act as a fingerprint. If your content is watermarked, you can employ a service or crawler to scan the internet for that watermark. Vendors like Imatag and Digimarc offer services to track images/videos via watermarks [11] [9]. According to a content tracking overview by FADEL, some tools use web-crawling with image recognition or watermarks to find where your media appears post-distribution [12] [13].

- *Metadata Tags:* Include metadata in media files (EXIF data, or ID3 tags in audio, etc.) with an ID or attribution. This can help if the media is reused without stripping metadata (though savvy users may remove it).

- **Web Crawling and Monitoring:** It may be necessary to actively look for your content on the broader web:

- Set up **Google Alerts** or similar for the story title or unique phrases. This can notify you when that text shows up on new pages.
- Use reverse image search (Google Reverse Image or TinEye) to find copies of your images. FADEL’s blog notes that reverse imaging technology can find public images by matching visual characteristics [14].

- For video, YouTube’s Content ID is an example of a system that fingerprints videos to detect re-uploads. While Content ID is proprietary to YouTube, there are third-party solutions for video fingerprinting that you might integrate if videos are a major concern.
- If resources allow, employ a custom crawler that searches specific sites of interest (e.g., known news sites or blogs relevant to your content) for your keywords or media. This could be combined with an AI that tries to spot similar content (for example, audio fingerprinting if your story was in a podcast, etc.).

- **Tracking Engagement and Outcomes:** Beyond just where the content is, you want metrics like views and even derived outcomes (e.g. donations made via a charity campaign page featuring the story). For each official placement of the content:

- Record the number of views/impressions. If using your own embed, you have impression counts. If on YouTube, you get the view count via API. If on a blog page, you might embed a tracking pixel to count views [15] .
- Track interactions (likes, shares, comments) if possible, via the platform’s data. These indicate engagement level.
- **Conversions/Donations:** If the story is meant to drive an action (like donate to a charity), use trackable URLs for the action. For example, if a story links to a donation page, append a UTM parameter or unique campaign ID for each platform (“?ref=Story123_FB” for Facebook, “...=Story123_Blog” for blog, etc.). The donation platform can record these, and you can later tally how many donations or how much money came via each source. This requires integration with the donation system’s analytics or at least getting a report of referrals. It’s a crucial metric for campaign effectiveness.

In summary, **strong tracking is vital to managing and valuing your content.** As one expert noted, failing to track where your content is used (especially across third-party sites) can weaken your legal position and lower its value, while strong tracking helps protect your rights and document usage for revenue models [16] [17] . There are tools and services that assist in monitoring appearances of your content online, and using them enables rapid response and proof of control [18] .

Content Revocation Mechanisms

Enabling the **revocation** of content (i.e., removing or disabling it everywhere it’s been distributed) is challenging but can be achieved with careful system design:

- **Central Disable/Delete:** If you employed the embed approach, this is straightforward: you can delete or deactivate the content in your central system. All embed instances then stop showing the content (they might display a “content not available” message or nothing at all). This centralized kill-switch is a primary benefit of using embed codes [4] . For example, a video embed could be configured such that when the video is marked private or deleted in the CMS, any attempt to play it returns an error or a placeholder image. Centralized content management thus **“makes it easier to ... remove content without needing to modify the embed code across various platforms”** [4] .

- **API-based Takedowns:** For native posts on external platforms, your system can utilize stored post IDs to remove content:

- Using the platform's API, issue a delete command for the post/video. (E.g., send a DELETE request to `https://graph.facebook.com/{postId}` with proper auth to delete a Facebook post, or use YouTube API to delete a video on the channel if you have rights.)
- If a platform doesn't allow full deletion via API, you might at least be able to **edit** the content (for example, editing a tweet's visibility or a blog post's content to blank). Many social platforms do allow deletion if you have the original credentials.

- It's wise to build a *revocation script* that tries to remove all instances: it goes through all recorded locations (Facebook, YouTube, blog) and attempts the appropriate action. Log the results for any manual follow-up needed.

- **Expiry and Revocation Clauses:** If content is shared with partners or third-party websites (e.g., a news site that featured the story), you should have an agreement or terms of use that allows for revocation. For instance, when providing the content to others, include a clause that the content might be pulled down on request and they should comply. This is more of a legal measure, but it backs your technical efforts. The importance of this is highlighted in copyright management best practices: not defining revocation or license end terms can make it hard to change strategy later [19]. So, structuring the sharing terms to include the possibility of removal is critical.

- **Controlled Access (DRM-like):** Another approach, used typically for sensitive documents, is to require that viewers always fetch the content through an authenticated channel, even if it's downloaded. For example, some DRM systems distribute encrypted files that only open with authorization from a server. If you revoke access, nobody can open the file anymore [20]. In our context (stories on the open web), this strict DRM approach is likely too heavy-handed, but a lighter version is feasible: ensure that the *latest version* of content is always pulled from your server. If someone saved an old copy, you can't erase that, but you can ensure it's no longer available openly or via your channels. In the case of video, for instance, you could host it such that the video player periodically checks a license. However, this may be overkill for general story sharing. It's mentioned for completeness — systems like Thomson Reuters' content platform revoke file access by removing user rights, since each view required user authentication [20].

- **Manual Takedowns for Unauthorized Copies:** If your tracking discovers copies of the content in places you didn't authorize (for example, someone scraped your story onto their blog without permission), revocation is a legal/manual process:

- Send a DMCA takedown notice if applicable (for copyrighted text, images, video).
- Contact the site owner or host to request removal, citing the person's withdrawal of consent or your ownership.
- Use the tracked watermark or ID as evidence that it's the same content you own. While this isn't an automatic process, having a clear, documented content ID and proof of your original will support the request.

In practice, **the most reliable revocation is achieved by not giving out uncontrolled copies in the first place.** That's why using central hosting or API-managed posting is preferred. If done well, a single "Revoke" action by the content owner in your system can trigger removal from all your managed channels within seconds or minutes.

Analytics and Impact Metrics

The system should provide a **dashboard of metrics** for each story, aggregating data from all platforms and locations. Key metrics include:

- **View Counts:** How many times was the story viewed on each platform and in total? E.g., *Facebook: 5,000 impressions; YouTube: 2,000 views; Blog: 1,200 pageviews;* etc. If using embed, your internal analytics will contribute to this. If available, show unique vs repeat views.

- **Engagement Metrics:** Such as likes, shares, comments, retweets, etc., on each platform. These indicate the resonance of the story. If the content was a video, watch-time or completion rate can be insightful (YouTube provides this).

- **Geographic or Demographic Data:** If the platforms provide (or if your own analytics can infer), see where the audience is coming from, or other traits. This might be less critical but is nice-to-have for campaigns (know your audience).

- **Referral and Conversion Data:** As noted, track if the story led to downstream actions. For a charity campaign, a key metric could be "Donations attributed to this story." This can be computed by capturing the UTM codes or referral IDs in the donation process. For example: *Story X brought 50 donors contributing $10k to the campaign (via the unique link used in Story X posts).* Similarly, if the goal was to drive sign-ups or petitions, measure those.

- **Time-on-Page / Read-through:** If the story is long form on a blog, your own site could measure how long users spent on the page, or if they scrolled to the bottom (indicating they read it fully). This helps gauge content effectiveness.

Implementing the analytics collection might involve a combination of your own tracking (for embeds and internal pages) and pulling data from external APIs on a schedule. Since we want near real-time insight, the system could update these stats daily or hourly. Storing these metrics in a separate analytics database or table will help in generating reports without slowing down the main content database.

Finally, present these metrics in a user-friendly dashboard with visualizations (graphs of views over time, pie charts of where the story is most popular, etc.). Short paragraphs or tooltips can explain what each metric means, keeping it understandable to campaign managers or content creators. This fulfills the requirement of seeing metrics like views and places used, and also demonstrates ROI or impact (like funds raised).

Intellectual Property and Legal Considerations

When users entrust their stories or media to the system, and when those stories are disseminated widely, it's crucial to handle intellectual property (IP) carefully:

- **Content Ownership:** Clearly define who owns the content. Often, the creator (e.g., the person telling their story) retains ownership but grants your organization a license to use and distribute it. Make sure you have that license documented (e.g., they sign a consent form or agree to terms) so that your use on various platforms is authorized.

- **Licensing Terms:** As part of the above, set terms that allow revocation. For instance, the storyteller might have the right to withdraw their story in the future. If so, your agreements with partners and your policy on social media posts should reflect that you might need to delete the content. A well-structured license might grant you rights to use the story for certain purposes and timeframes, with the ability to terminate those rights [19]. This avoids complications if they revoke consent or if the campaign ends.

- **Copyright Registration:** If the stories are valuable and original, consider registering copyright (if they are not already protected) to strengthen your legal position. While copyright exists upon creation, registration provides a public record and better remedies in case of infringement [21]. This might be something you assist the storytellers with, or do as an organization if you are assigned rights. It's not directly a tech implementation detail, but it supports the ability to enforce takedowns externally.

- **Attribution and Metadata:** Embed ownership information in the content where possible. For example, an image's metadata could list the creator and your organization. When posting on platforms, include a line like "Story by X, used with permission by Y campaign." This not only gives credit but also signals to others that the content is not just free-for-all. It might deter unauthorized reuse and helps if you need to challenge someone's use later by proving provenance.

- **Privacy and Sensitive Data:** Some stories, especially personal ones, might have private details. Ensure your system complies with privacy laws (e.g., GDPR if applicable). If a story includes personal data (health info, etc.), you may need explicit consent about where it will be posted. Also, implement a way to **anonymize or redact** content if needed for privacy while still using the story (for example, hide real names or faces on certain platforms if required).

- **Audit and Logs for Compliance:** Maintain logs of content distribution and removal actions. If a person asks "Where has my story been shared?" you should be able to list all platforms it went to (from your records). If they request a takedown, log when it was removed from each location. This kind of audit trail is good for accountability and trust.

By covering these IP and legal bases, you reduce the risk involved in sharing content widely and then needing to retract it. It also builds trust with content contributors that their story won't escape their control.

Implementation Considerations and Architecture

Bringing all these elements together requires a thoughtful system architecture. Here are key components and some implementation-level suggestions for a custom build:

- **Database Design:** Use a relational database (e.g., PostgreSQL or MySQL) for structured data like users, content metadata, permissions, and post tracking. For example, have tables such as `Users`, `Content`, `ContentPermissions`, `ExternalPosts`, `Analytics`. The `Content` table stores story text or media references and fields like privacy_level, author_id, status (active/revoked). `ContentPermissions` might map content to user or group IDs for non-public content. `ExternalPosts` can list each platform posting with fields (content_id, platform, post_id, url, timestamp, status). Analytics can either be aggregated in separate tables or stored externally if using a service. Ensure to index by content IDs so you can quickly gather all related info.

- **Media Storage:** For large files, integrate with a cloud storage bucket or a content delivery network. Store only the URL or reference in the database. Use naming conventions or IDs so you know which file belongs to which content record. If revocation is needed, you might need to invalidate CDN caches or remove files from storage (or replace them with a placeholder image). Many CDN/storage systems allow cache invalidation by URL.

- **Backend Application:** This is the core that enforces the business logic. It can be built with frameworks like Django/Flask (Python), Express.js (Node), Ruby on Rails, etc. Key functions of the backend:

- Authentication and session management (perhaps using JWTs or sessions).
- Authorization checks on every content fetch (e.g., an API endpoint `GET /content/123` will verify the requester has access).
- RESTful or GraphQL APIs to allow the front-end (or external consumers) to retrieve content. If you do an embed, an endpoint like `/embed/123` could return an HTML snippet for that content.
- Integration modules for each social platform API (one module for Facebook, one for Twitter, etc.). These handle posting and deleting. Securely store OAuth tokens (encrypted in DB) for accounts that will post. Use the platform SDKs if available to simplify development.
- A scheduler or background job service for tasks like fetching analytics periodically, scanning the web (if you implement automated search), and sending alerts. For instance, a daily job could update view counts from YouTube for each video posted.

- Monitoring: implement logging and maybe alerting (for example, if a revocation command fails on a platform, alert an admin to handle it manually).

- **Front-End Interface:** Provide a web dashboard for users (and admins). This should include:

- Content creation/upload form with privacy settings.
- A “Distribute” or “Publish” panel where the user can select platforms to share to (showing only those they have linked/authorized).
- A tracking dashboard showing all the places the content lives (after publishing, populate a list: e.g., “Facebook Post (link), YouTube Video (link), Blog embed on site X, etc.”). Each with current status (active, removed) and maybe a “Remove” button next to each if they want selective takedown.
- An analytics view with the metrics and graphs as discussed.

- Admin view might have a global list of content and the ability to revoke any (in case a story must be pulled network-wide quickly).

- **Embed Code Provision:** If using embed, the system should generate the snippet for users to copy-paste. For example: `<div id="story123Embed"><script src="https://yourcms.com/embed.js?content=123"></script></div>`. The script would fetch the content in HTML and inject it. Provide options in the embed code for width/height or style if needed, and document how to use it. Also consider providing an oEmbed or embed.ly endpoint so that if someone pastes a link into a platform that supports rich previews, your content could display nicely (this is a nice-to-have).

- **Scalability:** If the content might go viral and get heavy traffic, design for scale. The embed approach means many external site hits will result in calls to your server. Use caching layers (CDNs or in-memory caches) for content that doesn’t change often. Also, if heavy video streaming is expected, a specialized video hosting (like an unlisted YouTube, or a streaming server) might be better than serving raw video from your web host. Ensure your infrastructure can handle spikes from social media attention.

- **Security:** Protect the system from unauthorized access. Harden the APIs (use proper auth tokens, validate inputs to prevent injection attacks, etc.). Since privacy is a concern, any vulnerabilities could expose private stories, so security testing is crucial. Also, log and monitor access patterns to detect if someone is accessing content they shouldn't (could indicate a permission bug or misuse of a sharing link).

- **Testing Revocation:** As this is a critical feature, implement thorough tests. For example, after publishing a piece of content to multiple channels, simulate a revocation: call your revoke function, then verify the content is indeed not accessible via embed (check the embed URL returns a removal message) and via external platform APIs (check that the post ID no longer exists or is marked deleted). This ensures your system's promise of removal is reliable.

By covering these implementation aspects, you can build a custom system that meets the user's needs. It essentially combines features of a **content management system (CMS)**, a **digital rights management (DRM)** solution, and a **social media management** tool, tailored to user-generated campaign stories.

Conclusion

Building a platform for sharing stories widely while retaining control is an ambitious but achievable project. The best solution combines smart technology and policy: store content centrally with rich privacy controls, distribute it via mechanisms that allow tracking (embedding or APIs), and use every tool available – from web analytics to digital watermarks – to monitor its usage. Crucially, design the system such that a single action by the content owner can withdraw the content from all authorized channels, honoring the person's rights and managing the organization's liability.

This approach not only empowers content creators and campaign managers with **full visibility** into where their story travels and how it performs, but also provides confidence that the content can be pulled back if circumstances change. In a world where content is "the new currency" [22] and misuse can have legal and reputational repercussions, having a robust content management and tracking system is an invaluable asset. By implementing the methods discussed – from embed codes with centralized control [4] to active tracking of reuse and swift revocation – we ensure that stories can safely make an impact across the internet without slipping out of the creator's control.

Sources: The design principles and methods outlined above draw on best practices in digital asset management [2] [6], content tracking technologies [12] [13], and lessons from intellectual property management in the digital age [16]. These sources and others have informed the recommendations to ensure a secure, trackable, and user-centric content distribution system.

---

[1] Solid (web decentralization project) - Wikipedia
https://en.wikipedia.org/wiki/Solid_(web_decentralization_project)

[2] [3] [4] [5] [6] [7] [8] What are Embed Codes in Digital Asset Management software?
https://www.orangelogic.com/embed-code-in-digital-asset-management

[9] Digital Watermark | Vobile
https://vobilegroup.com/digital_watermark?lang=en-us

[10] Digital Watermarking: Protect Your Business Sensitive Data - Sealpath
https://www.sealpath.com/blog/digital-watermarking-protect-business-sensitive-data/

[11] IMATAG - Digital Watermarking : Protect your content online
https://www.imatag.com/

[12] [13] [14] [15] Keeping a Watchful Eye on Your Digital Content Tracking
https://fadel.com/resources/keeping-a-watchful-eye-on-your-digital-footprint-with-digital-content-tracking/

[16] [17] [18] [19] [21] [22] Copyright Valuation in the Creator Economy and Digital Media | PatentPC
https://patentpc.com/blog/copyright-valuation-in-the-creator-economy-and-digital-media

[20] Digital rights management
https://www.thomsonreuters.com/en-gb/help/highq/files-module/files-admin/drm