ChatGPT

PRD: Gen Z Social Video Platform (APAC)

Product Goals and Vision

The goal is to create a **mobile-first social video platform** tailored to Gen Z in the Asia-Pacific region. This platform aims to combine the **short-form, engaging content** style popularized by TikTok with the **community interaction** features of Bilibili's bullet comments and Discord's group chats. Gen Z users have **short attention spans** and prefer visual content; **short videos with creative, authentic engagement** are most likely to capture their interest [1]. We envision a vibrant ecosystem where **content creators, viewers, and annotators** all actively participate: - **Content creators** share safe, entertaining videos (adhering to content guidelines similar to YouTube's, e.g. no pornography or extreme violence [2]). - **Viewers** enjoy an endless feed of videos without needing to log in, but can join the community to interact. - **Annotators** enrich videos with overlay comments (彈幕/danmu) – a culturally popular feature that makes video watching a **communal, immersive experience** [3].

The vision is to **foster a sense of community** around video content. By enabling real-time overlay commentary and social features, watching videos will feel like **a shared experience** rather than a solitary one. This dynamic, community-driven approach has been a key differentiator for platforms like Bilibili, where the "bullet chat" system (scrolling overlay comments) has become a **defining feature that enhances user engagement and community belonging** [4]. Our platform will similarly strive to blend **entertainment with interaction**, creating a sticky experience that Gen Z users will return to daily. Ultimately, the product should become a **"Gen Z hub"** for APAC – much like Bilibili is in China with over 100 million DAUs spending ~100 minutes daily [5] – by providing an outlet for creative expression, social connection, and trend-driven content consumption.

Key User Roles and Personas

We are designing the MVP to support three **equally important user roles**:

- **Content Creators (Uploaders):** Gen Z individuals or teams who upload videos. They need easy tools to publish content and engage fans. In our platform vision, *"everyone can be a creator"* – lowering barriers to creation (simple capture, editing, upload flows) is crucial [6]. Creators will want profile pages, basic analytics (views, likes), and perhaps moderation control over comments on their videos.
- **Viewers:** Consumers of content who browse and watch videos. They value a **fast, intuitive UI** that delivers immediate entertainment (e.g. TikTok's infinite swipe feed and autoplay algorithm make content instantly accessible [7]). Viewers should be able to discover content through feeds, search, or categories without friction. They can watch anonymously (no login) to encourage onboarding via "lurking," but must sign up to interact.
- **Annotators (Commenters with Overlays):** A subset of viewers who actively engage by adding **overlay comments (彈幕)** or stickers/images on videos at specific timestamps. This role is inspired by Bilibili's bullet-comment culture where users "shoot" comments onto the video in real time, turning viewing into a **shared conversation** [3]. Annotators need a smooth UX for typing comments

(or adding an image/GIF) and maybe choosing its display style. All logged-in users can act as annotators, so this isn't a separate user type per se, but a key activity we support.

All three roles are supported in MVP – creators upload content, viewers consume it, and annotators enrich the experience – to kickstart a **virtuous cycle** of content creation and engagement. Success means a high number of active creators (upload frequency), active annotators (lots of overlay comments), and satisfied viewers (long watch times and retention). For example, Bilibili's ecosystem thrives with 3.8 million monthly creators contributing 15.6 million videos, leading to users spending over 96 minutes a day on the platform [8]. We aim for similar engagement by empowering each role from day one.

MVP Feature Set

The **Minimum Viable Product** will include core features that allow basic operation as a social video app while treating creators, viewers, and annotators equally. Key features for the MVP:

- **Video Upload & Processing:** Users (creators) can upload video files from their device. The app will accept general **safe-for-work videos** (length perhaps up to a few minutes in MVP, focusing on snackable content). Upon upload, the backend will transcode videos into standard formats and multiple resolutions for smooth streaming (e.g. 480p, 720p) [9]. *No explicit or graphic content is allowed* (following YouTube-style policies on nudity, violence, etc. [2]). Videos will have a title, description, and possibly tags for search.

- **Video Playback:** Viewers can play videos in-app with typical controls (play/pause, seek, volume). Videos will display in a **mobile-optimized player**, likely defaulting to vertical full-screen (TikTok style) for short videos or rotated for horizontal content (like YouTube). The player will render **overlay comments** in sync. We'll support **adaptive streaming** so that depending on network, the appropriate video quality loads [9]. **Browsing without login** is permitted: anyone can scroll through content, watch videos, and read comments.

- **Overlay Commenting (弹幕/Danmu System): Logged-in users can post timed overlay comments** on videos. This is a marquee feature differentiating us from many Western platforms. As a video plays, viewers see a stream of user comments flying across the screen or fixed at positions (design TBD, but default to scrolling text across the top or middle). Users can type a short text comment (e.g. max ~50 characters) that will appear at that timestamp for all viewers. Optionally, allow small image or emoji overlays as well (with size limits to avoid obscuring too much content). There will be basic controls: viewers can **toggle off/on** the overlay layer if they find it distracting (just like Bilibili allows hiding bullet chats). This feature brings a **fun, communal feeling** to video watching – viewers react together in real time [3]. (In MVP, these comments will be **post-moderated** or filtered for profanity to prevent abuse.)

Example of bullet comments (弹幕) on Bilibili. Users post text that scrolls over the video in real-time, creating a shared viewing experience. Our MVP will include a similar overlay comment feature to boost community engagement.

- **Core Interactions (Likes, Shares, Basic Comments):** Beyond overlay comments, users can **like (❤)** a video, increasing its like count. They can **share** a video via a link or to social media. Sharing will generate a unique link to view the video (no login needed just to view). We will track share counts.

Additionally, we might include a **basic comment section** under each video (separate from overlay comments) for more persistent discussions or feedback, similar to YouTube's comments. (However, MVP priority is on overlay comments as the primary engagement mode.)

- **User Accounts & Profiles:** Users can **register/login** (required for any posting activity: uploading videos, writing comments, liking, etc.). Sign-up could be via email, phone, or OAuth (Google/Apple) for convenience. Profiles in MVP are simple: a username, avatar, bio, and display of the user's contributions (videos uploaded, perhaps a count of comments made or likes). A creator's profile serves as their channel page listing all their videos. Profile management includes basic settings (change avatar, password, etc.). Privacy setting MVP: profiles are public by default; we won't implement complex privacy controls until later versions.

- **Discovery & Navigation:** The home screen will present a **feed of videos**. For MVP this could be a simple curated feed (e.g. latest videos, or a mix of trending content). Users can **scroll vertically** through videos (like TikTok/Reels) for continuous viewing. Alternatively, a grid or list like YouTube's home could be used – but given Gen Z trends, we lean toward a swipe feed of full-screen videos. Search functionality will be rudimentary in MVP: users can search by video title or tag. We may categorize content by broad genres (e.g. Music, Gaming, Vlogs) with browsable sections. **No advanced recommendation algorithm** is expected in MVP – that will be in full product scope – but we'll prepare the data collection for one.

- **Notifications:** MVP will include basic notifications for user interactions: e.g. if someone likes your video or replies to your comment. This can be in-app notification (a simple bell icon list). Real-time push notifications can be deferred until post-MVP, but the system design will account for them.

- **Moderation & Reporting:** Even in MVP, we need foundational moderation. There will be **Community Guidelines** (mirroring YouTube's policies on prohibited content). Users can **report** videos or comments that violate rules. A simple reporting flow (with reason selection) will be available on each video/comment. Initially, admin tools will be rudimentary (e.g. admins manually reviewing reports and banning content/users). Automated filters for certain banned words in text comments will be implemented. For content, since only SFW allowed, any uploaded video might be scanned by an AI service for porn/violence, or flagged for manual review before publishing (if resources allow). The focus is on having a process in place to handle harmful content quickly to keep the platform safe.

- **Branching Sharing (MVP interpretation):** In the context of MVP, "branching sharing" could mean enabling users to **share videos in various ways** and possibly track share spread. At minimum, a user can share a video link; if a non-user opens it, they see the video in a web view or are prompted to download the app. Internally, we might track if a video was shared by User A to User B (like a referral) – this could evolve into showing how content spreads (a share tree) in full product. MVP will log share actions but not necessarily visualize the "branches" yet.

- **Equal Support for Roles:** In MVP, we ensure **creators have tools to upload/manage content**, **viewers have easy access to watch and browse**, and **annotators have the ability to comment overlay**. No one role's features should feel neglected. For example, at launch a creator can at least upload and title videos (even if advanced editing is later), a viewer can find videos and enjoy them

without hurdles, and an annotator can participate in the overlay chat. This balanced approach is key to jumpstarting engagement.

Full Product Scope (Post-MVP Features)

Beyond MVP, the full product will expand features to fully realize the vision. Future features and enhancements include:

- **Advanced Video Creation Tools:** Integrate an in-app **camera and editor** (like TikTok's creation suite). This includes recording video clips, adding music, filters, AR effects, trimming, stitching multiple clips, and overlaying text/stickers. Lowering the effort to create fun, polished videos will encourage more UGC (TikTok's strategy to *"lower the barriers for content creation"* has directly led to more content and engagement [6]). We may also add **duet or remix features** – e.g. a user can take another's video and add their own video alongside or in sequence (similar to TikTok Duet/Stitch) – this fosters collaborative content and could be what "branching sharing" refers to (content forking into new derivative content).

- **Personalized Feed & Recommendations:** Implement a **recommendation algorithm** using machine learning to show each user a tailored video feed. As the user base and content library grow, a TikTok-style *For You* feed will improve retention. The algorithm would consider watch time, likes, shares, followings, etc., to surface relevant content. This goes hand-in-hand with metrics tracking (so we can measure and improve the feed's performance). Eventually, users might see a mix of content from subscriptions (creators they follow) and algorithmic suggestions. The feed can also feature trending challenges or hashtags popular among Gen Z.

- **Social Graph and Community Features:** Introduce richer social interactions:

- **Follow/Subscriptions:** Users can follow creators to see their new videos in a "following" feed. Creators can gain **subscribers** similar to YouTube channels.
- **Direct Messaging or Group Chats:** To leverage the community aspect (inspired by Discord), allow users to chat. This could start as simple DMs (so users can share videos privately or discuss). Later, we might support **group watch parties** or topic-based channels where fans of a creator or genre chat (text or voice) – essentially bringing a Discord-like community hub into the platform.
- **Collaborative Annotation & Reactions:** Expanding the overlay comments, possibly allow **interactive stickers/emotes** (similar to Twitch's or Discord's emoji culture) that users can fling onto the video. Could also allow creators to pose questions or polls in the video with overlay responses from viewers.

- **Branches of Sharing/Threads:** Build on the "branching sharing" idea – for example, display a **content sharing tree**: if one user shares a video and 5 new users join from that, show the creator or original sharer that impact. Alternatively, implement **video response threads**: a creator posts a prompt video, others reply with their own videos, forming a branching storyline or challenge thread (somewhat like how Twitter threads or TikTok reply chains work).

- **Profile System Enhancements:** Make user profiles more robust:

- Creators: show follower counts, total likes received, and an ability to arrange their videos into playlists or series. Possibly a **verification badge** for notable creators.
- Viewers: allow users to have a **favorite videos** list or collection, visible on their profile if they choose (or a private watch later list).
- Annotators: could display a fun stat like how many overlay comments they've made or a "top commenter" badge on certain videos.
- Profiles will also have privacy options: e.g. the ability to make your liked videos or followers list private, etc., to give users control.

- **Social Login & Integration:** add more login options (e.g. WeChat, LINE, Facebook depending on APAC market needs) to ease onboarding.

- **Monetization & Rewards (Post-launch phase):** Although not part of MVP, eventually incorporate ways for the platform and creators to monetize:

- **Virtual Currency/Tipping:** Like Bilibili's coin system, implement a virtual currency or **in-app coins** that users can purchase or earn and give to creators as tips [10]. This incentivizes creators and engages fans (perhaps tie it to special on-screen effects when someone tips during a live or premiere).
- **Advertising**: Introduce ads carefully (since Gen Z can be ad-averse). Could start with skippable ads or sponsor banners that do not disrupt the community vibe. Ensure moderation and content quality are solid before scaling ads.
- **Premium Features:** Possibly a premium subscription for users that gives perks (like removing ads, exclusive emojis for comments, or early access to new features). Also consider **paid content** where creators can put some content behind a paywall or ticketed live events.

- Note: Monetization will be done in a Gen Z friendly way – as Bilibili did, focusing on user experience first and using *unconventional revenue methods* that suit the community's preferences [11] (like gaming tie-ins, merch, etc. later on).

- **Live Streaming:** In a full scope, add the ability for creators to broadcast **live video streams** with live chat (including overlay comments in real time). Live content drives strong engagement (e.g. gaming streams, Q&As). We'd need real-time chat infrastructure and moderation tools for this. Given Bilibili and others have integrated live streaming (and even exclusive e-sports streams [12]), it's a logical extension once we have a base community.

- **Enhanced Moderation & Safety Tools:** As the platform grows:

- Implement AI-driven content moderation at scale (image/video analysis to detect nudity, violence, hate symbols, etc.) using services like Hive or Amazon Rekognition. These can **detect harmful images/videos (nudity, violence) quickly at scale** [13], which will be crucial as content volume explodes.
- Community moderation: enable trusted users or appointed moderators to help manage comments or flag content (like Discord server mods, or YouTube's community moderators).

- **Age/Content Rating:** Possibly tag videos with content ratings (everyone / teen / mature) and allow users (or parents) to filter out content not suitable for younger audiences. Keep aligning with evolving local regulations in APAC markets regarding content and data.

- **Performance and Scalability Improvements:** Full scope includes migrating from any quick MVP solutions to more robust ones:

- Use of content delivery networks (CDN) across APAC to speed up video load times in each region.
- Support for higher video quality (1080p, 4K) and longer video lengths as needed, with efficient encoding to manage file sizes.

- Fine-tuning of the Flutter app for smoothness (60fps UI even with overlays) and possibly adopting platform-specific optimizations if needed (given Flutter allows adding native modules if necessary).

- **Analytics and Metrics Dashboard:** Internally, a full product would have an analytics dashboard for product teams and for creators:

- Creators get insights: views over time, audience demographics (if available), engagement metrics on their videos.

- Product team monitors: funnel metrics (new user sign-ups, activation rate), content performance, A/B test results on new features, etc. This ties into success metrics – but building dedicated tools for it will come as the platform matures.

- **Localization and Regional Features:** Since APAC is diverse, full product may involve:

- Multi-language support (UI localization into Japanese, Korean, Thai, etc. as needed; also handling multilingual captions or subtitles on videos).
- Region-specific content rules (e.g. more stringent moderation in markets with stricter laws).
- Integration with region-preferred social networks for login/sharing (WeChat in China if ever launching there, LINE in Japan/Taiwan, etc.).
- Possibly region-specific **stickers or filters** that align with local pop culture trends.

Technical Requirements

Platform & Architecture: The app will be built on a **Flutter** front-end with a scalable cloud-based backend. We choose Flutter for a simultaneous iOS and Android launch with one codebase. Flutter is proven viable for large-scale apps (e.g. Alibaba's Xianyu app has 50M+ downloads and 10M DAUs using Flutter) and offers fast development via hot-reload [14]. Key technical requirements and components include:

- **Mobile App (Flutter):**
- Single codebase targeting iOS and Android. Ensure the use of Flutter's **video_player** (or a better video plugin) for efficient video playback with hardware acceleration.
- UI at 60fps: Flutter's Skia rendering should handle animations (swipes, etc.) smoothly, but we must optimize heavy UI elements like the scrolling comment overlay.
- Integration of native code if necessary for advanced media handling (e.g. using platform-specific codecs or DRM in future).
- Plugin usage: camera access (for future recording feature), local storage for caching videos, push notification plugin for notifications, etc.

- Testing on a range of device types common in APAC (including lower-end Android devices, various screen sizes) to ensure consistent performance.

- **Backend Architecture:**

- Likely a **cloud-based microservice architecture**. We can use a scalable provider (AWS, GCP, etc.). The backend should expose a set of **RESTful or GraphQL APIs** that the app calls for actions (fetch feed, upload video, post comment, etc.).
- **Programming language/framework:** Could use Node.js (with Express/Nest) or Python (Django/Flask) for quick development, or Go/Java for performance. Given real-time features, a Node.js + Socket.io or a Go service for WebSockets might be useful for the comment system.
- **Database:** A reliable database for user data and metadata. Likely **SQL (PostgreSQL)** for structured data (users, videos, comments). Use **NoSQL** (like MongoDB or DynamoDB) for unstructured or high-volume data (e.g. logging events, or caching user feed recommendations).
- **Storage & CDN:** Videos and images should be stored in cloud object storage (like AWS S3 or GCP Cloud Storage). We will use a **Content Delivery Network** (CloudFront, Cloudflare, etc.) to distribute video content with low latency across APAC. This is crucial for performance so that, for example, a user in Southeast Asia fetches video from a nearby server.
- **Media Processing Pipeline:** Upon video upload, the backend triggers a **transcoding job**. We can use a service like AWS Elastic Transcoder / MediaConvert or a serverless FFmpeg-based pipeline. The video is converted into multiple resolutions and formats (e.g. 360p, 720p MP4/HLS) and stored in the CDN [9] . Generate thumbnails as well for video previews.
- **APIs for Media Streaming:** For playback, we serve either HLS streams or progressive MP4 with adaptive bitrate. A manifest file (.m3u8 for HLS) might be provided for each video so that the client can adapt quality. Flutter can use HLS through video_player plugin.
- **Realtime Overlays:** For overlay comments, we need low-latency updates. Two approaches:
    1. *Polling:* Simpler for MVP – when a user plays a video, fetch all existing comments for that video with timestamps, then render them in sync. New comments can be polled every few seconds.
    2. *Realtime via WebSockets:* More complex but ideal – maintain a WebSocket channel per video (or use a pub/sub service). When someone posts a comment, the server pushes it to all viewers currently watching that video so they see it instantly. This mimics the live communal feel (like a chat room per video). We may use a service (Firebase Realtime DB or custom Socket server) for this. MVP could start with polling and upgrade later to sockets.
- **Scalability:** Design stateless services where possible behind load balancers, use auto-scaling groups or Kubernetes to handle increasing traffic. For data-heavy components (like feed generation or recommendations), design with caching in mind (Redis for caching popular feed results, etc.). Partition services (e.g. user service, video service, comment service) for clarity and to scale independently.

- **Third-Party Integrations:** Use proven services where it saves time:

    - Authentication: possibly Firebase Auth or AWS Cognito to handle user sign-up/login securely (including OAuth for Google/Apple logins).
    - Analytics: use something like Google Analytics/Firebase Analytics or Mixpanel for user behavior tracking in early stage.
    - Error monitoring: integrate Sentry or similar for app crash reporting and backend error logs.

- **Media Content Rules Enforcement:** Implement backend checks for content:

- Use an **AI content moderation API** on uploads. E.g., Amazon Rekognition or Google Video Intelligence to scan frames for nudity, gore, etc. These services can flag inappropriate content for review [13] . They can also detect text in images (for any hate symbols or disallowed text in overlay images).
- In MVP, possibly put uploads into a "pending" state until scanned by AI (which is usually quick, seconds) or by an automated queue.
- Text moderation: use a profanity filter library for comments. Also consider using a service like Google Perspective API to detect hate speech or harassment in text comments.

- These technical measures enforce the **"safe-for-work content only"** requirement.

- **Performance Requirements:**

- **App Performance:** App should cold-start within ~2 seconds on modern phones. Video load time should be minimal – ideally under 1-2 seconds to start playing on a good network, and gracefully handle slower networks (with a loading indicator and perhaps starting on a lower resolution stream).
- **Overlay Performance:** The overlay comments rendering must be efficient. In Flutter, we might use a widget that rebuilds for each new comment. We have to ensure this doesn't drop frames. Possibly batch draw comments onto a canvas or use an animation layer. Testing needed to handle maybe dozens of simultaneous overlays without lag.
- **Network:** The app will be network-intensive (video streaming). Ensure use of HTTP/2 or QUIC (HTTP/3) for efficient transfers. Also consider a download buffer — pre-fetch the next video in feed while the current one is playing, to reduce wait between videos.

- **Server Performance:** The system should handle surges of traffic (for example, if a video goes viral). We'll use load testing to ensure our video serving can handle at least e.g. 10,000 concurrent viewers per video for MVP scaling (using CDN helps here). The database should handle write bursts (when a popular video gets thousands of comments rapidly – use a scalable approach like writing to a queue or separate comment DB to not overload the main DB).

- **Security:**

- Secure all API endpoints with authentication checks (JWT or similar token for logged-in actions).
- Use HTTPS for all client-server communication to encrypt data in transit.
- Store passwords hashed (if not using external auth) and follow OWASP best practices to prevent common vulnerabilities (SQL injection, XSS in comments – sanitize user inputs, etc.).
- Prevent abuse of APIs (rate-limit comment posts to, say, a few per second per user to prevent spam attacks).

- **Privacy compliance:** Since targeting APAC, ensure compliance with any local data protection laws (e.g. PDPA in Singapore, GDPR if any users in jurisdictions that require it). For now, focus on not collecting unnecessary personal data and providing a way to delete account/data.

- **Backend Stack Summary:** *Example tech stack:* **Flutter** (Dart) for app; **Node.js** with Express for API + Socket.io for realtime; **PostgreSQL** for core data; **Redis** for caching sessions and hot data; **AWS S3/CloudFront** for storage and CDN; **AWS Lambda or MediaConvert** for video processing; **Firebase Cloud Messaging or SNS** for push notifications; **ElasticSearch** or similar for search functionality if needed later (for searching video titles/descriptions).

- **Development Infrastructure:** Use Git for version control (with branching strategies for features). Set up CI/CD pipelines to build and test the app on each commit (possibly using Codemagic or GitHub Actions for Flutter CI). Also set up automated testing frameworks: unit tests for logic, widget tests for UI, and perhaps integration tests that spin up a test backend. For the backend, include automated tests for API endpoints. Ensure we can deploy to a staging environment for testing with a subset of users before full release.

UX Considerations

Design and user experience are critical for adoption by Gen Z. We will follow best practices from leading apps (TikTok's simplicity, Bilibili's community feel, Discord's interactivity) to craft an intuitive UX:

- **Video Feed & Navigation:** The app will likely launch directly into the video feed (especially for new users, mirroring TikTok's approach of throwing you straight into content [15]). The UI should be **distraction-free, focused on content**. For instance, TikTok's UI is famously minimal: one video takes the full screen, with only essential overlay controls (like, comment, share) on the sides [16]. We will adopt a similar *"content-first"* approach – the video is front and center; UI chrome is minimal and intuitive (e.g. swipe up for next video, swipe down for previous). Important controls (like the like button) will be large and thumb-accessible in the lower half of the screen for one-handed use [7] [17].

TikTok's minimalist full-screen video feed is a model example. It autoplays content immediately with easy swipe navigation and simple controls. Similarly, our app's UI will prioritize instant content delivery and intuitive gestures to keep Gen Z users engaged.

- **Overlay Comment UI:** When watching a video, users can toggle the bullet comments on/off via a visible button (e.g. an "overlay" icon). When on, the comments scroll in a semi-transparent layer over the video. We will use readable text (white with black outline perhaps) for visibility. The UX must ensure comments don't completely obscure the video content; possibly limit number of overlapping comments at once or allow them to accumulate at top. Users adding a comment will tap a "Comment" field (perhaps overlaid on the video or an icon that pauses the video and brings up a text input). For timing, by default the comment will tag the current playback timestamp; users might also be able to slide a timeline to choose when their comment should appear (for MVP, current time is used to keep it simple). We may also offer options like choosing the style: scrolling (default) or static top/bottom positions, and maybe text color. In the future, we can let users attach small images or GIFs as comments – UI for that would involve selecting an image from gallery or an emoji menu. The design should make this **fun and quick**, so users do it in the moment (perhaps even during playback without pausing). Also consider a **filter toggle**: some users might want only overlay from people they follow vs. everyone – not MVP, but a future idea.

- **Posting & Creation Flow:** The upload process for creators in MVP will be straightforward: a plus " " button on nav bar -> pick a video from gallery (or record directly in future) -> add details. Ensure the **workflow is short**: Gen Z users might drop off if too many steps. For now, just Title and optional description, then Submit. Show an upload progress indicator. We'll add confirmation when done. We should also prep the UI for possible editing features (e.g. an intermediate screen where they can trim video or add music). But MVP can keep it minimal. Once uploaded, the video appears on their profile and in the feed.

- **Information Density & Visual Design:** Use a **clean, modern aesthetic**. Likely a dark theme UI (many Gen Z apps use dark mode for video apps as it's easier on the eyes when watching media). Use of color highlights (maybe brand color for buttons). The interface should avoid being text-heavy. Instead of lots of buttons or menu options on screen, use gestures and simple icons. For example, swiping left could bring up additional info or comments panel; swiping right maybe opens the uploader's profile. But we must implement such gestures carefully and educate users via a brief tutorial or tooltips, if needed, so they discover them (maybe after a user has watched 3 videos, prompt "Swipe right to view this creator's profile" as a hint).

- **Profiles & Social UX:** Each profile page should clearly show the user's content and stats. For creators, their videos are displayed in a grid or list. Include a prominent "Follow" button. For viewers' profiles, show their liked videos or comments if they choose to make them public. Profiles should also convey status – e.g., if this person is a top creator, maybe a badge. Also consider fun elements like profile avatars could be animated or have frames, appealing to Gen Z's self-expression needs.

- **Onboarding & Login UX:** Since browsing is allowed without login, many users may skip account creation at first. We'll use **just-in-time prompts**: e.g., if they try to like or comment, show a pop-up: "Join to interact – Sign up in seconds to post your comment." Keep the sign-up simple: possibly one-tap via Google/Apple, or minimal fields if manual (username, DOB, etc. – and explain why we ask DOB, e.g. age compliance). We might also show a *"why join"* screen listing benefits (create content, follow creators, join chat) to entice sign-up. For new account onboarding, maybe allow them to pick some interests or follow some suggested top creators to populate their feed (this can improve retention by giving them relevant content early).

- **Localization & Language UX:** Plan the UI to handle multiple languages (both in interface text and user-generated content). For instance, bullet comments might be in various languages – possibly allow filtering by language (future feature: show me only comments in languages I understand). In APAC, using localized language for UI in each market is key. Also, the design should accommodate different text directions if needed (mostly APAC languages are LTR except perhaps Urdu in some regions). Ensure font choices are legible for CJK characters if applicable.

- **Accessibility:** Include basic accessibility in design. For example, support screen readers for visually impaired (the video content itself is visual, but ensure the app navigation is readable by TalkBack/VoiceOver). Provide captions or encourage creators to add subtitles to their videos (maybe future feature to auto-generate captions for accessibility and better understanding in multilingual environment). Ensure color contrast in UI elements is sufficient. Also consider users with limited bandwidth – perhaps allow a setting to default to lower video quality or an option to **download videos offline** (future idea) for those on limited data plans (common in some APAC regions).

- **Privacy Controls:** In future UX, allow users to adjust privacy: e.g., make their account private (approve followers), or only friends can comment on their videos. We should design the settings screen to be clear about these options. Also implement content preferences: e.g., a toggle to **hide overlay comments** by default if some users always find them annoying, or filters to not show potentially sensitive content (like a "restricted mode"). MVP might not have all these, but the architecture should allow adding them.

- **Error Handling & Feedback:** Ensure the app gracefully handles errors (network issues, etc.). If a video fails to load due to connectivity, show a friendly retry UI. If an upload fails, preserve the video and details so the user can retry. Feedback for user actions: liking a video should give a little animation (to feel satisfying), posting a comment overlay might show their comment briefly highlighted for them. These micro-interactions add delight.

- **Influence of Competitor UX:** We actively study best practices:

- From **TikTok**: the infinite scroll feed, easy engagement (big like and follow buttons), and the fact that the app is immediately engaging without requiring setup [15]. We will emulate the *"Don't make me think"* principle – user opens app and is instantly in the experience.
- From **YouTube**: robust controls (maybe a mini-player to allow scrolling while video plays – could add later), and giving users the ability to go deep (descriptions, comments, related videos). Over time, we might add more YouTube-like features (like the ability to schedule video premieres, or end screens that link to another video).
- From **Bilibili**: community culture cues – e.g., Bilibili has an icon to turn on/off danmu, which is well-known to their users. We might even borrow visual cues like a **shooting icon** for sending a comment (to signify "bullet"). Also, Bilibili sometimes gates certain content or features (e.g., they had a community entrance exam for some privileges). We might gamify our community too – for example, require new users to watch 10 videos or have account age X days before they can post unlimited overlay comments, as a spam prevention and to encourage familiarity.
- From **Discord**: emphasize **community and belonging**. Possibly create a feature where each creator can have a **community page** or chat channel where their fans hang out (blurring the line between a content platform and a community forum). The UI for this should be accessible from the creator's profile ("Join XYZ's community") and would resemble a chat or feed of posts. Gen Z likes smaller community feel alongside big platform content [18], so our UX should allow sub-communities to flourish (without leaving the app).

In summary, our UX will be **content-centric, quick, interactive, and community-oriented**. We test prototypes with target users (18-24 in various APAC locales) to ensure the design resonates culturally and is easy to use. Gen Z expectations are high due to polished apps they use, so we aim to meet those from the get-go, focusing on a **delightful, fast, and socially engaging experience**.

Success Metrics

To measure the product's success, we will track key **performance indicators (KPIs)** across user growth, engagement, and content creation. Here are the core metrics and targets:

- **User Growth & Retention:**
- **Daily Active Users (DAU)** and **Monthly Active Users (MAU)**: The count of unique users engaging with the app daily/monthly. We want to see steady growth here. (E.g., target X DAU by 3 months post-launch, Y DAU by end of first year). Growth will indicate market adoption.
- **Retention Rates:** What percentage of users come back to the app after 1 day, 7 days, 30 days, etc. A strong retention means people find value long-term. For reference, Bilibili boasts over **80% 12-month retention** for its users [19] – an extremely high bar indicating a loyal community. Our goal could be, for example, >50% Day-7 retention in early stages, and >30% Day-30 retention, improving

over time with network effects. Also track **cohort retention** by signup month to ensure improvements.

- **Average Session Length / Time Spent:** How long users spend in the app per day on average. This indicates engagement depth. For a video app, high usage is expected (Bilibili users spend ~96–100 minutes daily [8]). If we can reach even 20-30 minutes average per day in early stages, that's promising, aiming higher as content library grows. We'll monitor **video watch time** per session.

- **Content Creation & Supply:**

- **Number of Uploads per Day:** How many videos are uploaded daily. This measures creator activity. We want this to grow in proportion to user growth – a healthy platform has a constant influx of new content. For example, initially it might be small (a few per day in a closed beta), but post-launch perhaps target hundreds per day, scaling to thousands.
- **Active Creators:** The count of unique users uploading videos in a given period (weekly or monthly). We can define an "active creator rate" = % of active users who are contributing content. A higher rate means a more participatory community (TikTok, for instance, turned many passive viewers into creators of at least one video). Even a 5-10% creator rate would be good to start, aiming to increase that by making creation easier (lower barrier).
- **Annotated Comment Activity:** Specifically for annotators, track **number of overlay comments posted** per day and per video. If this feature is engaging, a high percentage of video views will have accompanying comments. We can measure **comments per video view** or **% of videos that receive at least one overlay comment**. Also track **unique annotators per day**. A growing trend here shows our differentiator is working. (Recall that on Bilibili, billions of bullet comments are sent each year [20], showing high engagement).
- **Like/Share Rates:** Average likes per video and shares per video. These indicate content resonating. A video "share rate" (shares per view) is especially valuable – content that people share means they're advocating it, leading to organic growth (word-of-mouth marketing, as shares show something *"resonated enough for them to share"* [21]). We will watch for some percentage of views that result in a share (even if it's a small %, those drive new users).

- **Follow Relationships:** How many users follow creators (when that feature is in). Growth in follow counts indicates community building and that viewers intend to return for specific creators' content.

- **Engagement & Satisfaction:**

- **Video View Count and Completion:** Monitor how many views each video gets, and what percentage watch to completion. Especially for short videos, a high completion rate is desired (e.g. 70%+ viewers finish a 1-minute video) [22]. If we see many dropping off early, it may indicate content quality issues or mismatched recommendations.
- **Average Watch Time per Video:** How long users watch before swiping away. This helps tune the algorithm and also measure content quality.
- **Conversion Funnel:** Track conversion from viewer -> logged-in user -> commenter -> creator. For instance, of 1000 new app downloads, how many register? Of those, how many actually like or comment? Of those, how many eventually upload something? Improving these conversion rates is key (through onboarding improvements, incentives, etc.). A healthy funnel might be something like 50% of downloaders sign up, 20% of those engage with likes/comments, and 5% upload content within first month.

- **Community Health Metrics:** Use moderation stats as a proxy for community health. E.g., **number of reports** per 1000 users (should trend downward or stay low if content quality is good and moderation effective). Also track response time to remove flagged content.

- **User Feedback/NPS:** Though qualitative, we should measure user satisfaction via surveys or Net Promoter Score. If Gen Z users really love the platform, we'd see a high NPS (willingness to recommend). We can survey a sample of users on app stores or in-app prompts about their experience.

- **Infrastructure & Performance Metrics:** These indirectly measure success by ensuring quality of service:

- **App performance**: Crash-free sessions (% of sessions with no crashes) – aim for >99%. Also monitor frame rate drops or memory usage via analytics tools.
- **Latency**: Video start time and rebuffering events. Track median and 95th percentile start times for videos (should be low). Also track if any region is experiencing slower loads (which might mean we need a closer CDN node).

- **Scalability**: Able to handle concurrent load – e.g. monitor CPU/memory on servers, and scale before hitting capacity. Also track cost per user (cloud cost) to ensure we can optimize as we grow.

- **Milestones & Targets:** It's useful to set some concrete goals (which can be refined):

- Within 3 months of launch: X DAU, Y videos uploaded, Z% Day-7 retention.
- Within 6 months: maybe DAU in the hundreds of thousands, MAU crossing 1M, with a library of tens of thousands of videos.
- Long term (1-2 years): become a top social video app in key APAC markets (could measure by ranking in app stores or by share of Gen Z usage). For instance, reaching a point where, say, 10% of 18-24 smartphone users in target country have our app installed and use it monthly.
- **Competitive benchmarks:** Keep an eye on metrics from competitors: e.g., TikTok's average daily usage among Gen Z, or YouTube's mobile watch time, etc., to gauge how we stand. If Gen Z in APAC uses TikTok 48% daily [23], one success indicator is capturing a fraction of that daily habit with our differentiated offering.

Regularly analyzing these metrics will guide product improvements. If creation rate is low, focus on better creation tools or incentives. If retention dips, improve content recommendations or social hooks (like encouraging friend invites or follow suggestions). The ultimate sign of success is **vibrant activity**: a growing user base that *creates, interacts, and returns*, forming a self-sustaining community around the platform.

Non-Functional Requirements

In addition to features, several **non-functional requirements** are critical to ensure the product's viability:

- **Scalability:** The system must handle growth in users and content volume without significant rework. This means using scalable cloud infrastructure and designing stateless, load-balanced services. As traffic grows, we should be able to add servers or increase resources seamlessly. Similarly, the content delivery should scale – e.g., using CDN ensures even if one video suddenly gets 1 million

views, the load is distributed. We design the database with indexing and query optimization in mind so it can scale to millions of records. Partitioning or sharding strategies might be needed as we scale (for example, splitting user database by region).

- **Performance:** Both client and server performance are key:

- The app should be **responsive** – target <100ms response for button taps and smooth scrolling. Video frame rate should remain high even when overlay comments are flying.
- Use lazy loading for lists, preloading of next video, and possibly background pre-fetch of data to reduce perceived wait.
- On the backend, API responses should be quick (most calls under 200ms server processing). We will implement caching for frequently accessed data (like popular video lists) to speed this up.
- We also consider **APAC network variability** – many users may be on slower connections. We could adopt techniques like showing low-res thumbnail or a brief preview GIF while the full video loads, to give feedback. Also ensure the app handles going offline gracefully (maybe allow downloads in future).

- **Optimizing Flutter**: avoid heavy rebuilds, use efficient state management (Bloc or Provider patterns) to only update necessary parts of UI.

- **Reliability & Availability:** Aim for high uptime (e.g. **99.9% uptime** target). This implies no single points of failure: redundant instances of servers, failover for database (replicas), etc. We will implement monitoring (using tools like CloudWatch, New Relic) to detect downtime or spikes. Also have a strategy for backups (regular DB backups, and maybe backup storage for videos if one cloud region fails). If a crash happens (server or client), it should not corrupt user data; for instance, if upload fails mid-way, it should either resume or allow retry without data loss.

- **Security:**

- **Data protection:** All personal data (emails, etc.) stored in DB must be secured. Encrypt sensitive data at rest (using DB encryption or at least hashing for passwords). Ensure secure handling of tokens and keys (no hardcoding secrets in app; use secure storage for tokens on client).
- **Authentication & Authorization:** Only authorized actions should be allowed. E.g., a user can only delete or edit their own videos/comments. Perform server-side checks on every request (never trust client alone for enforcement).
- **Prevention of Unauthorized Access:** Thwart common exploits – e.g., rate limit login attempts to prevent brute force, use captcha if needed on signup to prevent bot accounts, validate inputs to avoid injection attacks.
- **Content Security:** Videos themselves might need protection against unauthorized downloads (though any displayed video can be potentially downloaded by savvy users, we might implement basic measures like obscured URLs or tokenized URLs that expire, to deter casual misuse).
- **Third-party Libraries:** Keep all libs (Flutter plugins, backend packages) up to date for security patches. Also verify any plugin we use is well-maintained.

- **Penetration Testing:** Before full launch, conduct security testing or code audits to catch vulnerabilities.

- **Content Moderation and Community Safety:** As mentioned, robust moderation is non-negotiable:

- For launch, we likely need a small team of moderators or at least a plan for on-call moderation to review flagged content quickly (within X hours). The system should include admin tools (a web interface where mods can review reports, view the video or comment in question, and take actions like remove content or ban user).
- Implement **community guidelines and an enforcement policy**. E.g., three strikes for content violations leading to account suspension. Make sure this is communicated in Terms of Service.
- Utilize AI-based moderation to assist humans. As content scales, rely more on automated filters (with human review mostly for appeals or edge cases). For example, use Hive's API which can scan videos fast for nudity/violence [13], or text sentiment analysis for harassment.
- **Harassment and abuse:** Provide features for users to block or mute others. If someone is being harassed via overlay comments or replies, they should be able to block that user, which stops the harasser's content from showing up for them. Also consider a setting on videos: creators could choose to **approve comments** before they appear (maybe not for overlay because that's real-time, but for normal comments).

- **Age safety:** Ensure users below a certain age have restricted features (like maybe not allowed to upload publicly until 16+ or such, depending on local laws). Possibly require age verification if needed for legal compliance in certain regions.

- **Compatibility:** Since APAC has a range of devices, the app should run well on **older or lower-end phones** (e.g., a budget Android with 2GB RAM). Flutter generally works on Android 5.0+ and iOS 11+, which covers majority. We should test on a variety of screen sizes (small phone, phablet, tablet) – maybe not optimizing for tablet in MVP but ensure it's not broken. Also consider compatibility with upcoming technologies (maybe later a web version or smart TV casting ability) but mobile is priority.

- **Maintainability & Extensibility:** The codebase should be organized to allow fast iteration. Use best practices in Flutter (clear separation of UI and logic, maybe using MVVM or Bloc). Write documentation for APIs and architecture decisions. This makes it easier for new developers to join the project as we scale the team. Also design the system to be modular – e.g., if we need to replace our video encoding service or add a new content type (like images or articles) later, it should be feasible without a total overhaul.

- **Analytics & Logging:** Non-functionally, we need comprehensive logging on backend (with sensitive info scrubbed) to debug issues. Also, analytics events on the client (e.g. track which videos are watched, where users drop off in onboarding) to inform product decisions. This is essential for an iterative development approach guided by data.

- **Compliance & Legal:** Ensure compliance with:

- **Privacy laws** (user data handling and privacy policy must be clear; possibly need parental consent mechanism if targeting minors).
- **Copyright:** Users will upload content – we need a system for copyright complaints (DMCA or local equivalent). Likely a report type for "copyright violation" and a process to take down if valid. In future, possibly content-ID type system if scale demands (to automatically flag copyrighted music, etc.).

- **Terms of Service & Community Guidelines:** Clearly present those during sign-up and have them accessible in-app. They outline what is not allowed (matching the moderation filters we enforce).

- **App Store Compliance:** Follow App Store (Apple) and Google Play guidelines closely since they require content moderation for UGC apps. For example, ensure we have a method to **block and filter objectionable content**, a way for users to report, and a mechanism to take action – which we do have planned (this is something app reviewers will check for). Also ensure the age rating of our app in stores reflects the content allowed (likely 12+ or 17+ if any mild mature content could appear).

In summary, the non-functional aspects (scalability, performance, security, moderation) are as important as the features, especially for a social platform. We will continuously test these: load tests for performance, security audits, and community feedback for moderation effectiveness. The goal is a platform that **runs smoothly, safely, and reliably** – providing a trustworthy environment for Gen Z to express themselves.

Timeline and Development Phases

Building this product will be an iterative process. Below is a **rough timeline** with phases, assuming we start immediately. This timeline can be adjusted based on team size (we assume a reasonably small startup team initially) and any unexpected challenges. The goal is to have a functional MVP on both iOS and Android and then rapidly improve upon it.

1. **Phase 0 – Research & Planning (Month 0-1):**
2. Conduct deep-dive research into user preferences (Gen Z focus groups, competitor analysis in APAC markets). Finalize the PRD (this document) and get alignment on scope.
3. Define the technical architecture in detail (select tech stack, design database schema, outline microservices).
4. Recruit initial team members if not already in place (Flutter developers, backend engineers, UI/UX designer, etc.).

5. Set up project management tools, repository, CI/CD pipeline skeleton.

6. **Phase 1 – Design & Prototype (Month 1-2):**

7. **UI/UX Design:** Create wireframes and high-fidelity mockups for key screens: video feed, video player with comments, upload flow, profile pages, etc. Get feedback from a few target users.
8. **Prototype in Flutter:** Possibly build a throwaway prototype focusing on UI navigation and the video player with a fake data feed, to test the feel of swipe interface and overlay comment rendering. This helps iron out any UX tweaks early.

9. Design review and adjustments based on feedback. By end of Month 2, have a clear design blueprint for MVP.

10. **Phase 2 – Core Development of MVP (Months 2-5):**

11. **Backend Development:** Set up the base project. Implement user authentication (e.g. simple JWT login), video upload API (storing to cloud storage, calling a transcoder), and basic data models for videos and comments. Implement retrieval APIs for feed (initially simple chronological or curated), video playback info (video URLs, etc.), posting/reading comments, and profile data. Also implement a rudimentary moderation module (ability to mark content and a simple admin endpoint to fetch reports).

12. **Flutter App Development:** Implement main screens:
    - Home Feed screen: pulls list of videos from API, enables swipe through them.
    - Video Player screen: uses a video_player widget for playback; overlay comments layer that syncs with current time (this will involve some coding to schedule comment display).
    - Login/Signup screens and logic.
    - Upload screen: allow selecting video file and calling upload API (with a progress bar UI).
    - Profile screen: show user info and their videos.
    - Basic interactions: like button (tapping calls API), share button (invokes OS share sheet), comment button (opens input and sends comment to API).
13. **Iteration:** Likely we'll develop in sprints (say 2-week sprints). By Month 4, we aim to have an internal "alpha" version where one can go through the whole flow: register, upload a video, see it in feed, and another user can view and comment. Continue refining based on internal testing.

14. **Parallel Testing:** Start writing unit tests for critical logic (e.g. comment timing, backend video processing). Also test on different devices for UI issues.

15. **Phase 3 – Testing, QA and Polishing (Month 6):**

16. **Internal QA:** Rigorously test the MVP features. Use test cases to cover: video upload (various sizes/formats), playback on different network speeds, comment sync accuracy, login flows, etc. Fix bugs and polish UI (ensure animations are smooth, fix layout issues).
17. **Limited Beta Release:** Possibly recruit a closed beta group (maybe a few hundred Gen Z users in target region) to try the app in a controlled release (via TestFlight or an APK) under NDA. Collect feedback, especially on UX and any crashes or confusing elements. Monitor analytics from beta usage to see if any metric is way off (e.g. no one using comments -> then UX might need improvement).

18. **Moderation readiness:** Before public launch, ensure moderation process is ready. Train any content moderators on the admin tools. Have community guidelines and FAQ written and accessible in the app.

19. **Phase 4 – Launch MVP (Month 7):**

20. Prepare for **public launch** on App Store and Google Play. Ensure all store listing materials are ready (app description, screenshots, maybe a promo video). Age rating set appropriately.
21. Launch in a target market or several (APAC focus; could be initially English interface in a few countries, then localize soon). Possibly a soft launch in one region to ensure systems scale, then broader.
22. Monitor the launch closely: watch server load, watch for any critical errors (via logging/monitoring). Respond quickly with hotfixes if any major issue arises (Flutter allows relatively quick updates, but App Store review might slow iOS fixes — try to catch issues in beta!).

23. Start marketing efforts in parallel (though not product dev, but noting timeline: around launch we'd do outreach on social media, perhaps leverage influencers to try the app, etc., to seed initial user base).

24. **Phase 5 – Post-Launch Improvements and Full Scope Development (Months 8-12):**

25. Now that MVP is out, gather usage data and feedback. Identify highest priority improvements or missing features. Possibly do updates every 2-4 weeks with enhancements.
26. Likely focus areas:
    - **Algorithm & Feed:** Start implementing a smarter recommendation system based on data (Month 8-9 work on backend machine learning model or at least improved heuristics).
    - **Social Features:** Month 8-9, add follow system and separate "Following" feed, so users can subscribe to creators.
    - **In-app editing tools:** Month 9-10, add basic editing (trim video, add music). This likely requires integrating a video editing library or building custom; might come in stages.
    - **Community & Messaging:** Month 10-11, possibly implement direct messaging or a basic discussion board under videos (if not already in MVP comments).
    - **Monetization planning:** by Month 10+, decide on initial monetization (maybe pilot an ad system by end of year or plan virtual currency). The actual implementation might go into next year unless urgent for revenue.
    - **Localization & Expansion:** Month 9-12, localize app to additional languages, and launch in more countries as needed. Adapt any region-specific requirements.
27. During this phase, also address technical debt from MVP (e.g., improve any quick-and-dirty code, scale the infrastructure if user load is increasing faster than expected).

28. Continue with regular releases. For example, **Version 1.1** in Month 9 with follow/following feature; **v1.2** in Month 10 with improved recommendations and editing; **v1.3** in Month 12 with perhaps group chats or a beta of live streaming.

29. **Phase 6 – Beyond 12 Months (Future Roadmap):**

30. By this time, we should have a solid base and a growing community. Future development might include:
    - Full **live streaming** functionality if metrics show demand (requires separate project planning).
    - **Advanced creator tools** (analytics for creators, monetization features like tipping, partnerships with brands).
    - **Desktop/web version** if expansion to web usage makes sense (Flutter Web could be leveraged).
    - Continuous refinement of the algorithm to boost retention (this is ongoing).
    - Scaling up the team and processes (more formal A/B testing of features, performance optimizations, etc.).
31. Also, implement any **deeper community features** (like interest-based groups, events, etc.) and refine the **moderation with AI** as content volume grows.

Each phase will involve cross-functional work (dev, design, QA, product). We will use **agile methodology**, adapting the roadmap as we learn from users. Success in early phases (MVP adoption) will heavily inform the priorities in later phases. By following this timeline, we expect to go from concept to a live product in ~6-7 months, and then spend the rest of the first year turning the MVP into a feature-rich platform that competes with incumbents by leveraging our unique annotation feature and community focus.

Throughout development, we will keep an eye on **best practices of competitors** and adapt quickly. For example, if TikTok introduces a new popular feature, we consider if it aligns with our vision. Our ultimate timeline goal is not just to launch, but to achieve **product-market fit** within 1 year of launch, indicated by

strong retention and engagement metrics (as outlined in Success Metrics). Frequent iteration and responsiveness to user feedback (a hallmark of modern product development) will be our strategy to hit that target.

---

1 Here’s how Gen Z consumer preference reshapes APAC’s marketing landscape | Singapore Business Review
https://sbr.com.sg/retail/exclusive/heres-how-gen-z-consumer-preference-reshapes-apacs-marketing-landscape

2 YouTube Community Guidelines enforcement
https://transparencyreport.google.com/youtube-policy/removals?hl=en

3 5 Bilibili: Unveiling the Gen Z Powerhouse | China Marketing Corp
https://chinamarketingcorp.com/blog/bilibili-unveiling-the-gen-z-powerhouse/

4 Bilibili Statistics for 2025 | Latest User Counts and More
https://expandedramblings.com/index.php/bilibili-statistics-facts/

6 7 15 16 17 TikTok UI Explained: Why the TikTok UI Is So Good
http://careerfoundry.com/en/blog/ui-design/tiktok-ui/

8 10 11 12 Bilibili : Everything you need to know (Updated 2025)
https://dfc-studio.com/blog/what-is-bilibili-chinese-video-platform/

9 html - how youtube produce different quality video for one uploaded video - Stack Overflow
https://stackoverflow.com/questions/29874922/how-youtube-produce-different-quality-video-for-one-uploaded-video

13 8 Best Content Moderation APIs: Benefits and Features
https://getstream.io/blog/best-moderation/

14 Companies Using Flutter in 2024
https://www.nomtek.com/blog/flutter-app-examples

18 Gen Z Wants Something Different from Social Media | Arena
https://arena.im/audience-engagement/gen-z-social-media-preferences/

19 What is Bilibili? Marketing on One of the Biggest Video Apps in China
https://www.nativex.com/en/blog/6-things-you-should-know-about-bilibili-chinas-youtube/

20 Digitalized deep play and floating emotional public sphere in the ...
https://journals.sagepub.com/doi/10.1177/20594364241302183?int.sj-full-text.similar-articles.5

21 9 social video metrics you need to track in 2025
https://blog.hootsuite.com/social-video-metrics/

22 Video Metrics 101: Everything You Need to Understand Them in 2025
https://www.sendible.com/insights/video-metrics

23 Gen Z Daily Usage of TikTok, Instagram Reels, and Other Short-Form Video Apps
https://pro.morningconsult.com/analysis/gen-z-instagram-reels-tiktok-short-form-video