Comprehensive Guide to Passing Apple App Store Review (Expo React Native Apps)

Note: Apple's App Store Review Guidelines are extensive and updated frequently, so always refer to the latest official documentation. The tips below compile legal, content, design, and technical requirements – with special attention to Expo/React Native apps – to help ensure your iOS app is approved on the first try. *(Expo's own docs caution that there's no guaranteed formula for approval, but following best practices and addressing reviewer feedback will get you there[1].)*

Preparing for Initial Submission

Developer Account & App Registration: Enroll in the Apple Developer Program (cost is \$99/year) and set up App Store Connect access. You'll need a unique Bundle Identifier for your app (set in Expo's app.json under ios.bundleIdentifier) and proper version numbers for release[2]. Use Expo's EAS Build or expo build:ios to generate your app's IPA file for submission. Before uploading, double-check basic settings: app name, version, icons, and other metadata in app.json are correct[3][2].

App Store Connect Metadata: When you create a new app listing in App Store Connect, fill out all required information accurately[4]. This includes:

- App Name and Subtitle – must be unique and not infringe trademarks.
- Description – a clear, honest description of what your app does. Avoid hype or misleading claims. The description should match the app's actual features and stay away from superlatives like "#1 app" unless you can substantiate them[5]. Any mention of prices or subscriptions in the description must match the in-app pricing exactly (inconsistencies here have caused rejections)[6].

- Categories and Keywords – choose an appropriate category and relevant keywords without keyword stuffing or using competitor names inappropriately (Apple will reject metadata that seems spammy or irrelevant[7]).
- Screenshots and Preview – upload high-quality screenshots for all required device sizes. Ensure the UI shown in screenshots is up-to-date and exactly what the app delivers[8]. Outdated screenshots (e.g. showing an old button name or a feature you removed) can lead to rejection for misleading visuals. Every image should reflect the current UI layout and features of the app.
- App Icon – prepare a 1024x1024px icon (no alpha transparency) that meets Apple's design rules. Avoid using Apple trademarked images or too-generic designs. Expo will use the icon specified in app.json (expo.icon) for the build, so make sure it's the final version. (Apple is strict about icons – they must be visually appealing and not easily confused with other apps[9].)
- Privacy Policy URL – a *Privacy Policy* is mandatory for all apps now[10]. Even if your app doesn't collect sensitive data, you must provide a URL to a privacy policy. Ensure the policy is accessible and covers what data your app collects (if any) and how it's used.
- Support URL & Contact Info – provide a support website or contact email. Apple requires a contact in case users (or reviewers) need help, and you should also keep your developer contact info up to date in your account[4].

App Privacy Questionnaire: Apple will prompt you to answer the *App Privacy* questions (Data Safety) in App Store Connect[11]. Answer these truthfully, describing what data types (if any) your app collects and for what purpose (e.g. analytics, crash reports, user info). For example, if you use Expo's OTA updates via expo-updates, mark that you collect "Crash Data" in the privacy questionnaire[12] – Expo's update module does collect error/crash information, so failing to disclose that could lead to rejection. It's crucial that the data usage info you provide is accurate and consistent with your app's behavior.

App Build and Versioning: In Expo, increment your app version and build number before submission. Apple treats each upload with a new version, so update the expo.version

and ios.buildNumber in app.json accordingly[2]. The binary's internal version must match the metadata you enter on App Store Connect (otherwise you'll get an ITMS error, though not a review rejection). Also ensure you've disabled any development flags (Expo development mode, debug menus) – build a release variant for App Store.

Testing on Devices: Before submitting, test your app extensively on real iOS devices (and simulators). Run it on different iPhone models and iPad if possible. No crashes or obvious bugs should remain – Apple will reject apps that crash or have broken functionality on review[13]. Test all user flows (account signup, purchasing, feature access, etc.) to ensure they work end-to-end. If your app requires login, see the next point.

Demo Account for Review: If your app requires an account login or special hardware (e.g. a Bluetooth accessory or a QR code to scan), provide the reviewer with a way to fully use the app[14]. The App Review notes field is the place to give a demo login (username & password for a test account) or to enable a "demo mode" in the app for reviewers[14]. Failing to supply login credentials will likely result in a rejection for "account-based features not accessible." Make sure any backend services or APIs are running and populated with test data during the review period[15], so the reviewer can experience the app as a normal user would. Apple advises to enable all backend services and provide any needed resources (like a sample QR code) for testing[14].

"Before You Submit" Checklist: Apple themselves list common issues that slow down approval. Ensure the following are done[4]:

- *Test for crashes and bugs* (stability is essential – even one crash can mean rejection[16]).
- *Complete and accurate metadata* (every field in App Store Connect filled correctly, screenshots and descriptions matching the app).
- *Updated contact info* (in case Apple needs to reach you about issues).
- *Provide full access* (demo logins or features accessible without requiring Apple to create an account[14]).

- *Backend services live* (no “under maintenance” or dummy data during review[17]).
- *Explain special features in review notes* – if any feature might be unclear, especially around in-app purchases or permissions, clarify them for the reviewer[18]. For instance, if your app uses background location or has a complex multi-step setup, write a note about why and how the feature is used. This proactive communication can prevent misunderstandings[19].

By thoroughly preparing these aspects before hitting “Submit for Review,” you set a solid foundation for approval.

Legal and Privacy Requirements

Apple places a strong emphasis on user privacy and legal compliance. Many app rejections stem from privacy issues, so pay close attention to this section.

- Privacy Policy: As noted, a Privacy Policy is required for all iOS apps[10]. Host it on a URL (your website or even a GitHub gist) and make sure it's linked in App Store Connect. In your app, it's good practice (especially if you collect personal data) to provide a way for users to view the privacy policy, but the critical part for review is the URL in the submission metadata. The policy should clearly explain what data you collect, how it's used, and how users can contact you or request deletion of their data. Apps without a privacy policy will be rejected, and since 2018 Apple hasn't budged on this[10].

- User Data & Permissions: If your app collects personal data or uses device services (camera, microphone, location, contacts, etc.), you must request user permission at runtime and explain the purpose of that access. In Expo/React Native, this means adding the appropriate `NS...UsageDescription` keys to your app's Info.plist via `app.json` (under `ios.infoPlist`) for each permission. Do not use the default boilerplate text – provide a specific, clear reason for the access, or Apple may deem it insufficient[20]. For example, instead of "This app uses the camera," say *"$(PRODUCT_NAME) needs camera access to scan QR codes for adding new contacts"*. Apple reviewers check these permission strings, and vague descriptions (or missing keys) will result in rejection ("doesn't sufficiently explain the use of the ... in the purpose string"). Expo's documentation warns that the default messages many libraries supply are generic and likely need to be tailored to your use case for App Store acceptance[20]. In fact, an Expo app was recently rejected because the location usage string was not showing the custom message properly, implying the explanation didn't appear to the reviewer[21][22]. Always verify that your permission prompts display the exact text you intend.

- App Tracking Transparency (ATT): If your app (or any included SDKs) tracks users across other apps or websites (for example, an ad network SDK or analytics that combines data for ad targeting), you must implement the ATT prompt (the “Allow this app to track?” dialog) and get user consent for tracking. Also, declare the tracking in your App Store privacy questionnaire. Failing to do so will lead to rejection under guideline 5.1.2 (privacy compliance).

- Data Use and Sharing: Only collect data that is necessary for your app’s core functionality. Apple’s guidelines require data collection to be transparent and minimal. If you collect sensitive data (health, financial info, etc.), ensure you have user consent and secure handling. For health/medical apps, include a disclaimer that it’s not a substitute for professional care. For apps targeting kids, do not collect personal info at all in most cases. In general, avoid asking for login or personal details unless truly needed; apps that seem to harvest data without justification raise red flags with Apple[23].

- Third-Party SDK Compliance: Starting in 2024, Apple requires third-party SDKs to provide a *privacy manifest* describing their data collection[24]. As a developer, you should update any SDKs in your Expo app to versions that include these manifests by the May 2024 deadline[24]. If you use popular SDKs (analytics, push notifications, etc.), check their documentation for compliance. For example, the Customer.io React Native SDK needed an update to add a privacy manifest; failing to update could result in App Store rejection[25]. In Expo, this might mean updating to a newer Expo SDK or specific library versions. Also, ensure that any data collected by third-party SDKs is properly disclosed in your App Privacy section. Remember, you are responsible for what all code in your app does – Apple expects you to vet SDKs and services for compliance[26].

- Sign-In and Account Deletion: If your app offers user account creation, as of early 2022 Apple requires that apps also offer a way for users to delete their account from within the app (for GDPR/California compliance). Make sure there’s a

feature (even if it's just contacting support) to delete accounts, and mention it in your privacy policy. Also, if your app uses social login (Sign-in with Google, Facebook, etc.), be aware of Apple's rules on Sign in with Apple. Guideline 4.8 says that if an app uses a third-party login service for creating the primary account, you must also offer an equivalent option that respects privacy (formerly this meant Sign in with Apple in most cases)[27]. As of 2025, Apple allows an alternative login *if* it only collects name/email and allows private email relay, etc.[27]. The simplest way to comply is: if you use Google/Facebook login, just implement Sign in with Apple as well (Expo has `expo-apple-authentication` for this). This keeps you safe from any interpretation issues – Apple has explicitly mandated Sign in with Apple in the past, so it's better to include it when social logins are present.

- Intellectual Property and Copyright: Legally, you must have rights to all content in your app. Apple will reject apps that use unlicensed material – e.g. copyrighted music, images, or brands – if you don't prove you're authorized. Avoid any use of Apple's own trademarks (like Apple logos, iPhone images) within your app or metadata unless specifically allowed, as that's a common legal rejection reason. For user-generated content apps, ensure your users' content doesn't violate IP (and respond quickly to any takedown requests). If your app features third-party content (such as user-uploaded videos or music), you should have a mechanism to handle copyright disputes and filters for IP-infringing content[28].

- Regulatory Compliance: Certain app categories have legal requirements. If your app involves health or medical data, follow privacy laws (like HIPAA if applicable) and include proper disclaimers (e.g. "This app is not medical advice"). If it's a financial app, ensure compliance with financial regulations and that your descriptions don't promise unrealistic outcomes (which could be seen as fraud or misrepresentation). If the app is a contest/sweepstakes, make sure it doesn't violate gambling laws and include official rules. Kids apps (if you opt into the Kids category) have *very strict rules*: no behavioral advertising, no external links

out of the app (unless behind a parental gate), and compliance with COPPA (children's privacy laws)[29][30]. Apple will heavily scrutinize apps for children, so if your app is for under-13 users, study the Kids Category guidelines in detail[29].

- Export Compliance (Encryption): If your app uses encryption (even the standard HTTPS calls that most apps use), you will need to answer "Yes" to using encryption in the App Store Connect questionnaire. Most apps qualify for exemption (if you're just using HTTPS and not doing custom encryption algorithms) – you'll answer a few questions to self-certify that you're exempt under category 5 part 2 of U.S. export law. Be sure to complete this, as an incorrect or missing encryption info can delay approval. Expo apps using HTTPS, for example, typically check "uses standard encryption" and fall under the exemption.

In summary, be transparent and thorough about privacy and legal compliance. Provide all required documentation (policy URLs, etc.), describe why you need any sensitive data, and ensure your app *respects user privacy by design*. Privacy violations have been the #1 cause of App Store rejections in recent years[31], so this is a section to triple-check.

Content and Safety Guidelines

Apple's Safety guidelines (section 1 of the review rules) focus on the content of your app and its potential impact on users. To avoid rejections:

- Objectionable Content: Do not include any content that is offensive, upsetting, or in poor taste[32]. This includes pornography, graphic violence, hate speech, or extreme political or religious content that could be deemed derogatory. Even user-generated content is subject to this – if users can post content, you need filters and moderation. For example, apps cannot have hate speech or overly sexual material. If your app would be considered "shocking" or "extreme," it's likely not suitable for the App Store[33]. Keep the content family-friendly or appropriately age-rated. Apple explicitly lists disallowed content: defamatory or discriminatory speech, realistic violence toward people or animals, encouragement of illegal drug use or weapon violence, overtly sexual content (like pornography or "hook-up" apps), and so on[34][35]. Review those examples in Apple's 1.1 guidelines and ensure your app doesn't cross the line.

- Age Ratings: When you fill out the content rating questionnaire in App Store Connect, answer truthfully about any instances of violence, sexuality, profanity, etc., in your app. The generated age rating must align with your content. If you say there's no mature content but the reviewer finds some, it's a problem. Also, if your app is targeted to kids, it should not have any of the disallowed elements (and likely will get a 4+ or 9+ rating, depending on cartoon violence or such). Misrating your app (like marking it 4+ when it has mild violence) can lead to rejection or reclassification.

- User-Generated Content (UGC): If your app allows users to post content (text, images, videos, etc.), Apple requires that you moderate that content. Guideline 1.2 lists the requirements: you need a method to filter objectionable material, a mechanism for users to report inappropriate content, and a way to block or remove abusive users[28]. Additionally, provide contact info (in-app or in your

metadata) so users can reach you with concerns[36]. Without these safeguards, UGC apps are often rejected. For example, a social feed in your Expo app should have profanity filters or manual moderation, and an obvious “Report” button on user posts. Document these measures in your App Review notes to preempt questions. Apps that are primarily UGC but are overrun with things like porn or bullying can be removed even after approval[37], so it’s both a review and operational concern.

- Realistic Dummy Content: Avoid using any kind of fake data that might be misconstrued. Apple forbids “trick or joke” functionality that could deceive users (e.g., a fake virus scanner or a faux fingerprint scanner for laughs)[38]. If your app shows sample data, make sure it’s clearly identified as such and not real personal info. Also, ensure any placeholder text is removed – don’t ship the app with lorem ipsum or debug messages, as that looks unprofessional and could cause confusion during review.

- Personal Safety: If your app involves user interaction (like messaging, dating, or forums), have policies to prevent harassment or harm. Apps that facilitate bullying, stalking, or any physical danger to users will be rejected under safety grounds. For certain categories like dating apps, Apple might ask for more information on how you ensure user safety (e.g., content moderation, user verification). Make sure to implement features accordingly (like the ability to block other users, etc., as mentioned).

- Health and Medical: For apps providing health-related content or guidance (common with fitness or wellness apps), include appropriate disclaimers such as “Always consult a doctor” if it borders on medical advice. Avoid any claim that could be seen as misleading health advice or that your app alone can treat or diagnose (that could fall under 1.1 misleading content or even 5.2.1 if it infringes on medical device regulations). If your app uses Apple’s HealthKit or CareKit data, follow Apple’s rules for those (e.g., you can’t use HealthKit data for

marketing or share it with third parties). Expo apps can integrate with Health via certain libraries – ensure compliance if you do that.

- Illegal Content and Geo-Restrictions: Make sure your app doesn't facilitate illegal activity. For example, an app that allowed users to download YouTube videos (violating YouTube's terms) might be flagged. Or if it allows something not legal in all regions (like a betting app), you need to geo-fence it to allowed regions and provide the necessary licensing info to Apple. Apps involving gambling or contests must have proper licensing and cannot use in-app purchases for real-money gaming rewards. If your Expo app includes something like a sweepstakes, read guideline 5.3 carefully (which covers gambling and contests). You may need to provide evidence of compliance during review.
- Physical Safety: If your app might lead to physical activities (e.g., an AR game that has people walking around), consider adding a caution for users to be aware of surroundings. Apple doesn't want apps causing accidents. A famous guideline: apps shouldn't encourage reckless behavior, like looking at the screen while driving, etc. So if applicable, include warnings or design your app to lock out certain features when in motion (for CarPlay apps, etc.). Expo apps likely won't integrate with CarPlay, but consider general safety.

Overall under safety: keep content appropriate, moderated, and transparent. If in doubt about a piece of content or feature's appropriateness, err on the side of caution or be ready to explain it to App Review in your notes.

Design and User Experience

Apple is known for its high design standards – your app's look and feel, as well as how it conforms to platform norms, are a significant part of the review.

- Polished UI & UX: A well-designed, intuitive user interface is crucial. If an app appears unfinished or has a poor layout, Apple may reject it for not meeting the "Design" guidelines (section 4). Common issues include: clipped or overlapping UI elements, text that's hard to read, or controls that are too small to tap. Test on multiple screen sizes, especially small devices like iPhone SE and larger ones like iPhone 14/15 Pro Max. Also test on iPad, because even if you don't specifically target iPad, your iOS app will run in compatibility mode on iPads. Apple has been known to reject apps that render poorly on iPad (e.g., layout breaks or blank space), even if you set ios.supportsTablet = false[39]. In practice, ensure no critical content is cut off and that the app is usable on iPad. If you truly cannot support iPad, at least make sure it runs without crashing or grotesque UI issues there – otherwise Apple might ask you to support iPad properly or fix the layout.

- Follow the Human Interface Guidelines (HIG): Apple's HIG provides recommendations on navigation, controls, feedback, etc. While not all HIG deviations are rejection reasons, some are. For example, using platform-consistent UI elements (like iOS-standard gestures and icons) makes a good impression. Avoid Android-esque design patterns that might confuse iOS users. Ensure that status bar text is visible (e.g., don't put black text on a dark status bar without adjusting style). Use safe area insets so content isn't hidden by notches or the home indicator. As a React Native/Expo developer, leverage libraries or components that respect these iOS conventions.

- No Placeholder or Beta Elements: Remove any "coming soon" labels or blank screens in the app. Apple wants a complete, fully functional app. If a feature isn't ready, disable it entirely or hide it until a future update. Also, do not label your app as "beta" or "preview" in the UI or metadata – apps must be production-ready for

App Store. (TestFlight is available for beta distribution; the App Store itself is only for final releases.)

- Sufficient Functionality (No Spam/Clones): Guideline 4.2 requires apps to offer value and a unique experience. If your app is basically a website wrapped in WebView or a very simple app that doesn't do much, it could be rejected for not being "useful enough." Make sure your Expo app has native features or a polished experience beyond what a mobile website would offer. Similarly, if you create multiple apps that are very similar (like reskinned clones with the same content for different clients), Apple can reject them under the spam rule (4.3)[40]. Put your best foot forward with one great app rather than many low-quality ones. For Expo specifically, since it's easy to churn out multiple builds, take care not to spam the store with variants of the same app.

- Consistent Metadata vs App Content: Apple will cross-check your App Store listing against the app. Ensure that: the features you tout in the description are *actually present* in the app, the screenshots show actual app screens (and not, say, a mockup or an unrelated image), and even things like the app name and icon don't mislead. For instance, if your app name implies functionality that the app doesn't deliver, that's a problem. One example of a rejection: An app subtitle said "Free and Easy" but the app required a purchase immediately – that was flagged for misleading users[41]. So be accurate and honest in all representations.

- Accessibility & Compatibility: While not strictly required for approval, it's wise to follow accessibility best practices (support Dynamic Type fonts, VoiceOver, etc.) – Apple has been increasingly encouraging this. There have been cases where apps got feedback about contrast or text size. It usually won't block approval unless it makes the app unusable for some users, but demonstrating care for accessibility can help your case if an edge issue arises. Also ensure your app works on the latest iOS version and doesn't use deprecated APIs (Expo keeps

React Native updated, but if you've ejected or added custom native code, keep it current)[42].

- Responsive and Adaptive Design: Expo apps often run on both platforms, but for iOS, ensure that you comply with things like the iPhone notch (use proper padding at top/bottom). Make sure rotating the device (if your app supports landscape) doesn't break layouts. If your app is portrait-only, lock it to portrait in the Xcode project/Expo config to avoid rotation issues.

- In-App Navigation: The app should not be confusing to navigate. If it has multiple sections, provide a clear menu or tab interface. Avoid situations where the user can get stuck with no way to go back (iOS has conventions like a Back button in the top-left for hierarchical nav). Also, do not mimic system alerts or UI in a misleading way – e.g., don't show a fake "Apple Security" popup in your app.

- Ads and Overlays: If your app includes advertisements, ensure they are placed in a way that doesn't ruin the user experience. Ads must not be too frequent or impossible to close. Apple's guidelines say ads should not interrupt the user in an unskippable way, and must have a clear close button[43][44]. Also ensure ads are appropriate for the app's age rating (no 18+ ads in a kids app, for example)[45]. If using any ad SDK in Expo, be cautious to configure it to follow these rules.

In essence, aim for a clean, professional, iOS-native feel in your Expo app. Many React Native apps pass review just fine – the key is that the reviewer shouldn't be able to tell it's cross-platform from a quality standpoint. No glaring UI bugs, no non-iOS patterns, and a smooth, intuitive experience will tick the right boxes.

Technical Performance and Stability

Apple's Performance guidelines (section 2) cover how your app behaves technically. Here's how to make sure your Expo app meets them:

- No Crashes or Major Bugs: This bears repeating – *if the reviewer encounters a crash, your app will be rejected.* It's one of the most straightforward and common rejection reasons[16]. Test thoroughly with release builds (not just the Expo Go app). Use TestFlight to do a final round of testing on an actual device. Check logs for any errors or warnings. Fix any bug that could degrade the experience (e.g., buttons not working, features not loading). Also ensure decent memory usage; if your app is too heavy and gets killed by the system, that's a problem.

- Launch Time and Loading: Your app should load reasonably quickly and not hang. If initial content loading takes more than a few seconds, consider adding a loading indicator or splash screen (Expo provides a splash screen configuration). Apple may reject apps that take too long to load content or that appear broken while loading[13]. If your app requires an internet connection for basic use, handle no-connection scenarios gracefully (show an error message rather than freezing).

- Compatibility: Apps must support the latest iOS version and ideally the latest devices at the time of submission. In 2025, that means ensuring compatibility with iOS 18/17 (whatever is current) and new iPhones/iPads. Expo usually keeps up with iOS changes if you're on a recent SDK. Also, Apple tests apps on IPv6-only networks (this has been a requirement for years) – make sure your networking libraries handle IPv6 (most modern libraries do)[46]. Expo apps using fetch/Axios etc. are typically fine, but if you use any low-level networking, test on a Wi-Fi that's IPv6.

- Public APIs Only: Your app should only use Apple's public APIs – no private frameworks or undocumented functions[47]. With Expo/React Native, if you stick

to standard libraries, you're unlikely to hit private API issues. This is more a concern if you write custom native modules or use obscure plugins. Apple's automated scans will flag usage of private APIs. So avoid any hacky native code changes (like trying to call iOS internals). Also ensure you've removed any debug code that might call non-public endpoints.

- No Downloading New Code (Within Rules): Apple's guideline 2.5.2 prohibits apps from downloading executable code after release that changes the app's features[48]. However, interpreted code like JavaScript is allowed *with conditions*. Expo's OTA updates (via expo-updates) are permitted as long as you don't violate the spirit of 2.5.2. Specifically, any remote JS updates must not change the app's primary purpose or introduce completely new features that weren't in the initial version[49]. You can fix bugs or tweak UI via updates, but you shouldn't, say, turn a calculator app into a chat app via OTA update – that would "change the primary purpose" and violate Apple's policy. Also, ensure you're not downloading any other code or scripts from random servers. Apple allows React Native code push because the JavaScript runs in Apple's provided JavaScriptCore engine and *cannot* access new native functionality that wasn't already in the binary[50]. In summary: use Expo's update feature responsibly and *within Apple's guidelines* – no sneaky feature additions, no loading of unreviewed plugins, and definitely no running of new native code.

- Background Modes & Services: If your app uses background capabilities (e.g., background location updates, VOIP, audio playback, etc.), you must declare the appropriate background modes in Xcode/Expo config and ensure your usage is legit. Apple will scrutinize these. For instance, if you enable background location, your app should clearly need it (e.g., a turn-by-turn navigation app). If not, you might be asked to remove background usage. Similarly, background audio is only for apps actually playing audio when the app is backgrounded (e.g., music or podcast apps). Don't abuse background privileges as it will lead to rejection under guideline 2.5.4[51].

- Push Notifications: If you include push notification functionality (Expo can handle push via Expo's push service), a few rules: You must prompt the user for permission (which iOS does automatically via `Notifications.requestPermissionsAsync()` in Expo). In your App Store Connect, enable the Push Notifications capability. Apple requires that push notifications are not used for unsolicited advertising or irrelevant spam. If you plan to send promotional pushes, users must opt-in to that in some way. Also, if you use the APNs VoIP push for calling apps, ensure it's only for internet calling purposes. For standard push, Apple's fine as long as you respect user consent and frequency. Additionally, it's good to mention in your privacy policy if you use push tokens and that they're only used for delivering notifications.

- App Performance: Keep your app responsive. Avoid obvious lag or freezing. If Apple reviewers find the app frequently unresponsive, they might reject it under performance issues. This ties in with testing on lower-end devices. For React Native/Expo, ensure any heavy computations are done efficiently (perhaps off the JS thread if needed). Optimize images and resources to keep the app size reasonable – while app size alone isn't usually a rejection reason, an extremely large app might get extra scrutiny (plus it's bad UX).

- External Links and Browsers: If your app opens web content (perhaps via an in-app browser or by launching Safari), ensure those work properly. Use SafariViewController or Safari (Expo's WebBrowser module uses SFSafariViewController under the hood). If your app has external links, Apple requires that they open in Safari or an SFSafariViewController, not in a WebView that tries to replicate the App Store or other apps. Also, if you link to an external site for account sign-up or content purchase (which is allowed only for certain apps like "reader" apps – see Business section), ensure there's no mention of non-App Store purchasing in a way that violates rules.

- Device-Specific Features: If your app declares usage of certain hardware features, ensure to handle cases where that hardware isn't present. For example, if an iPad-only feature or you require a gyroscope, and a device doesn't have it, handle that gracefully or restrict device compatibility in your App Store settings. Apple might test on different devices to see if the app crashes or misbehaves when a feature is missing.

- IPv6 Network Testing: Apple *will* test your app on an IPv6-only network environment[52]. Ensure all network calls in your app (API requests, etc.) resolve via IPv6. Usually, if your server has a proper DNS (aaaa records) and you're using high-level APIs (NSURLSession or fetch), you're fine. But if you had any hard-coded IP addresses (not common with Expo apps) that are IPv4, that would fail.

To sum up, focus on stability, compliance with Apple's technical restrictions, and thorough testing. A technically sound app that runs smoothly on all intended devices and iOS versions has a much higher chance of quick approval.

Monetization and Business Model Rules

Apple's Business guidelines (section 3) govern how you can charge users and what you can/cannot do with payments and subscriptions. This is critical to avoid rejection, especially if your app involves purchases or subscriptions.

- In-App Purchase (IAP) for Digital Goods: If your app offers any digital goods or premium content/features for sale, you must use Apple's in-app purchase system. This is a strict rule: *"If you want to unlock features or functionality within your app... you must use in-app purchase"*[53]. This includes things like premium features, game currency, subscription content, or unlocking a "Pro" version. Do not implement your own payment mechanism or direct users to an external payment for these things. Expo apps can use the expo-in-app-purchases module or similar to integrate Apple's IAP. Common pitfalls that cause rejection:
- Mentioning in the app or description that users can pay you on a website or another app to get features. (Apple will reject you for trying to bypass their IAP and 30% cut).
- Using wording like "Subscribe on our website for full access" – you cannot do this unless you have an entitlement from Apple (rare cases) or you're a "Reader app" (explained below).
- Even having non-functional purchase buttons that don't use IAP can get you flagged. If something should be an in-app purchase, implement it properly or remove it.

- Physical Goods vs Digital Goods: To clarify, physical goods or services (like buying clothes, or ordering a taxi) are NOT required to use IAP – those can use regular credit card payments outside Apple. But digital content (e.g., an e-book PDF, a virtual coin, a premium account in an app) must go through IAP[53]. If your Expo app is an e-commerce for physical items, you can integrate Stripe or another payment gateway – that's fine. Just make sure nothing in the app hints at digital purchases outside IAP.

- Reader Apps Exception: Apple allows certain apps (“reader” apps like Netflix, Kindle, Spotify) to let users access content subscriptions bought elsewhere, as long as they don’t offer purchases in-app or directly prompt signups. They also recently allowed “external link account” entitlements for some apps (like you can have a link to your website to manage account in specific cases). But these are niche and require permission. As a rule of thumb, if your app’s primary purpose is to consume content that users may have purchased elsewhere (news, cloud storage, courses), you can let existing users login, but you cannot tell users how to sign up or purchase the content in the app. For example, a magazine app can let users log in to view their subscription, but it shouldn’t say “Go to www.magazine.com to subscribe” inside the app (that was disallowed until recently, and now might require an entitlement in certain countries). Keep the app focused on content consumption, not selling. If you’re unsure, err on side of offering IAP or contacting Apple for clarification via the entitlement process.

- Subscriptions: If you offer subscriptions (auto-renewable or non-renewing), follow Apple’s rules closely. Make sure you:

- Use in-app purchase for subscriptions as required.
- Clearly describe subscription pricing and terms within the app (Apple likes to see the price, duration, and what’s included, before the user commits). A common rejection is not clearly explaining how a subscription works or hiding the price until after the user taps subscribe[54][55].
- Provide a restore purchases mechanism (so users who reinstall can get their subscription back).
- Don’t try any “scammy” subscription practices (Apple will ban apps that bait-and-switch or hide the true cost in fine print[56]). For instance, don’t pre-select an expensive plan without informing the user, or trick them into trials that auto-bill unexpectedly. Apple reviews subscription signup flows carefully now.

- If you converted previously paid features into subscription, do not remove those features from users who already paid under old model (Apple considers that unfair).

- Pricing and Value: Apple doesn't set your prices, but they will reject apps that *"try to cheat users with irrationally high prices"*[57] or which appear to be rip-offs. For example, if your app charges an exorbitant amount for minimal functionality (like \$99 for a wallpaper app), expect scrutiny. Make sure your pricing is clear and your app delivers value for that price. Also, don't advertise something as "Free" and then immediately require payment without any free functionality – that's misleading. If you say "Free with in-app purchases," give at least some base functionality for free.

- No External Payment Links: We touched on this, but to emphasize: do not include any buttons or links that direct the user to pay outside the app for digital content. Apple expanded guidelines to even forbid mentioning better deals outside (except with those special entitlements for certain regions)[58][59]. For instance, an app was rejected for having a sentence like "You can also subscribe on our website." This is not allowed in most cases. Remove such language. If you must inform users due to business model (again, like a reader app), follow Apple's specific guidance or apply for the external link entitlement if eligible[60].

- Donations and Fundraising: If your app collects charitable donations, Apple requires that they be done via Safari web links to the charity's site (not in-app purchase, since Apple's cut on charity is problematic) *or* using Apple Pay with the nonprofit's approval. You cannot use IAP for donations to third parties, and you can't collect via your own in-app form either (unless you're the charity itself). Make sure to handle this correctly if applicable.

- Advertising: If you monetize with ads, ensure your app's primary purpose isn't just to display ads. Apple may reject apps that appear to be made just for ad revenue with little content (they consider that low-quality). Ads must comply with privacy

(if using ad networks, answer the privacy questions about data used for tracking). Also, avoid excessive full-screen ads or anything that could be construed as clickbait or malicious. If you use IDFA (Identifier for Advertisers) via ads, you must show the ATT prompt and only use it after user consent.

- Rewards and Contests: You can't use IAP to sell raffle tickets or any chance-based items that have real-world value. If your app has some kind of reward system, ensure it's virtual. Loot boxes or gacha mechanisms in games are allowed if odds are disclosed to users[61][62].

- Crypto and NFTs: If by chance your Expo app deals with cryptocurrencies or NFTs (some RN apps do), note Apple's rules: you can display NFTs and let users browse them, but any NFT sales or unlocking content via NFTs must go through IAP if it's digital content[63]. Apps aren't allowed to direct users to external purchase of NFTs except in specific contexts (and not for most regions). Cryptocurrency trading apps are allowed if they come from approved exchanges and follow local laws; but you cannot, for example, build an app that just lets users transfer crypto outside of an exchange context without proper legal backing.

- Avoiding Scams: Apple monitors for apps that might be scammy or defraud users. Ensure your in-app purchases aren't misleading (e.g., say exactly what the user gets). Don't use names or icons that impersonate other popular apps or services. If your app deals with money (payments, loans, etc.), extra diligence is needed to show you're legitimate. They may ask for documentation for finance-related apps or medical apps (e.g., proof of partnership with a medical institution, etc., if you claim such).

In summary, for business model: if it's digital and in the app, use IAP. Don't try to sidestep Apple's payment system or rules on subscriptions. Be transparent with users about costs, and deliver what you promise. Many rejections (and even post-approval

app removals) happen due to attempts to bypass these rules or “clever” workarounds – Apple is very experienced at catching those.

Expo & React Native Specific Considerations

Developing with Expo and React Native means most of the above guidelines apply the same as for any native app, but there are a few extra points to consider to ensure a smooth review:

- Expo Permissions and Config: Expo makes it easy to include various native features (camera, location, etc.), but you must configure the usage descriptions in app.json as mentioned. If you add an Expo module like expo-camera or expo-location, the build will include default Info.plist messages. Those defaults might be generic ("Allow app to access your location"); take the time to override them with a more specific reason via ios.infoPlist or the module's config plugin options[20][64]. This is a frequent oversight in Expo apps that leads to rejections. Always test that your permission dialogs show the custom text you expect.

- Over-The-Air Updates (Expo Updates): As discussed, using Expo's OTA updates is generally fine under Apple's rules if used for bug fixes or minor improvements. Do not use OTA to add entirely new features or content that might surprise Apple. Apple's policy on interpreted code[49] essentially allows JS updates *only if they don't change the app's intended purpose or introduce unreviewed functionality*. For example, pushing a small UI fix or enabling a feature that was dormant is okay; pushing a whole new game mode or significant content expansion might cross the line. Also, whenever you do release a major update via the App Store, include in the review notes that your app uses expo-updates for minor patches so the reviewer is aware (this is not required, but can be a courtesy if you've had any prior issue).

- Bundle Size & SDK Stripping: Expo apps sometimes include native libraries even if you don't use them (though this has improved with EAS Build and tree-shaking of unimodules). Still, ensure you remove any unused permissions (on Android) and avoid including unnecessary packages that bloat the binary. While Apple won't reject purely for large size, a very bloated app that requests lots of

permissions by default might raise questions. For instance, older Expo builds would include camera or location permissions even if not used unless you opt-out. Use android.permissions and ios.infoPlist in app.json to strictly declare only what you need[65][20]. For iOS, unused permission keys generally won't be included unless a library adds them, but double-check that you're not asking for permissions you don't actually use, as this confuses reviewers (they might wonder "why does this note-taking app ask for GPS access?").

- Detaching / Custom Native Code: If you "eject" from Expo (to the bare workflow) for some reason and add native modules, be aware you need to manage all the native config yourself (entitlements, plist, etc.). Keep any custom native code within Apple's rules (no private APIs, etc.). Most Expo apps remain in the managed workflow though, which aligns well with guidelines by default.

- Expo SDK Version and iOS Compatibility: Use a relatively recent Expo SDK when submitting. Expo SDKs are typically supported for a year; using an obsolete SDK could mean your app is using older iOS APIs. For instance, ensure you're not targeting any deprecated UIWebView (this was a big issue a couple years ago – Apple deprecated UIWebView and would reject apps still containing it). Expo has since moved fully to WKWebView, but if you have older packages, be cautious. Always test on the latest iOS to catch any compatibility issues.

- Push Notification Configuration: If you use Expo's Push Notifications service, you'll need to upload your push key/credentials in Expo's dashboard or via eas credentials. That's a setup step, but in terms of guidelines: make sure your app gracefully handles notification permissions. Don't assume the user enabled them; handle the case where they deny. Also, Apple requires that you don't auto-subscribe users to unrelated promotional push messages. With Expo, you typically only get a push token after asking permission, so respect the user's choice. In your app's settings (if you have any), you might allow users to toggle certain notification types – not required, but good practice.

- Linking to Expo or Debug Info: Remove any references to Expo Dev menus or debug screens. In older Expo builds, shaking the device in development brings up a dev menu – ensure that is disabled in production. The user (and reviewer) should never see an Expo loading screen that shows an expo URL or anything. Customize your splash screen (via app.json expo.splash) so it's branded for your app, not the default Expo logo. Expo's build process does this by default now (no more expo watermark), but just double-check.

- Performance on Device: Sometimes React Native apps perform differently on simulators vs devices. Test the release build on an actual device for jank or slowdowns. For example, heavy animations or large lists – ensure they run smoothly. Expo has tools like Flipper for debugging performance if needed. A poorly performing app might not be explicitly rejected unless it's extreme, but Apple could flag apps that appear unresponsive or incomplete.

- Handling App Lifecycle: Expo abstracts some native behaviors, but ensure your app doesn't do weird things on pause/resume. For example, if your app requires a persistent connection (like a game), handle interruptions (phone calls, etc.) gracefully. Apple checks that the app doesn't misbehave if you background it and return. With Expo, test backgrounding the app and coming back to ensure it doesn't reset unexpectedly or hang.

- Expo Store Review API: If you use the expo-store-review to prompt users for App Store ratings, be mindful of Apple's rules: you can only prompt for a review at most 3 times in a year per user, and it must use Apple's official SKStoreReviewController (Expo's API does use that). Don't put the prompt on first launch or in an annoying spot – Apple might not catch this, but it's good practice to ask for reviews politely after the user has had some positive interaction.

- EAS Submit and Build Automation: Expo provides EAS Submit to send your binary to App Store Connect. This can streamline the process and reduce human error (like forgetting a setting in Xcode). Just ensure that after submission, you log

into App Store Connect to fill out all the requisite metadata we discussed. EAS won't handle the App Privacy questions or app description for you – that's on you to complete.

- React Native Specific (if not fully Expo managed): If you included any native modules or other frameworks, verify they don't use prohibited APIs or flags. For example, some RN libraries might use the UIBackgroundModes in Info.plist (like audio or location) – remove those if not needed, because having them declares to Apple you use those background modes, and they will expect your app to function accordingly. Another example: If your RN app uses Bluetooth, Apple requires a purpose string and might also expect that you don't misuse Bluetooth (some apps got rejected for using Bluetooth solely to identify devices for tracking, which Apple disallows without user benefit).

In short, Expo simplifies a lot of iOS compliance (by using sensible defaults and official APIs), but you as the developer must still configure the app properly and not assume "Expo takes care of it all." Use the Expo documentation on building for app stores[66][67], and remember that an Expo app is indistinguishable from a native app to Apple's review – it must meet all the same criteria. Many developers successfully ship Expo apps on iOS, so there's no inherent issue with the framework as long as you follow the guidelines discussed.

Submission Process and What to Expect

Once you have addressed all the guidelines above and are ready to submit, here's a brief overview of the submission and review process, along with tips to handle the outcome:

- Submitting to App Store: After uploading your build (via EAS submit or Xcode Application Loader/Transporter), go through App Store Connect and ensure all fields are completed (as covered in the Preparation section). Save your entries and submit the app for review. You'll choose whether it's manual or auto-release upon approval, etc., according to your preference.

- Review Timeline: Typically, Apple reviews new app submissions within 1 to 3 days[68]. Sometimes it can be faster (even under 24 hours), or slower if there's a high volume (after holidays, big events, etc.). App updates often get reviewed a bit quicker than first submissions. Be prepared for possible delays and don't promise an exact release date to your users until approval comes.

- Communication from Apple: You will get an email when the app changes status – if it goes to "In Review," and then either "Approved" or "Rejected." If approved, congrats! It will be released according to your settings (immediately or on a set date). If rejected, don't panic. Apple will include a message citing which guideline(s) were violated (e.g., "Guideline 5.1.1 - Legal - Privacy - Data Collection and Storage"). They often provide a brief note about the issue and maybe a screenshot showing the problem if applicable.

- Responding to a Rejection: Read Apple's notes carefully. Sometimes the message is cryptic, but the guideline number gives a clue. For example, if they said 2.1, that's "App Completeness," which often means something didn't work as expected (a button, a login, a feature). Retrace the reviewer's steps: Apple's message might say "Upon launch, the app required login and no demo credentials were provided," or "we encountered a crash when tapping X." Once you identify the issue, fix it promptly. You can upload a new build with the fix and

submit again. In App Store Connect's Resolution Center, write a reply explaining what you fixed or clarifying the misunderstanding. Always be polite and precise – e.g., "We apologize for the issue. We have added a demo login (username: demo, password: 1234) and ensured the app no longer crashes on launch. Please let us know if there are any further issues."[14]

- If You Disagree with a Rejection: Occasionally, you might feel the reviewer made a mistake or misinterpreted something. You can use the Resolution Center to politely argue your case or provide additional info. For example, if they thought your app has a bug but it actually requires a certain step, you might explain how to use it. Provide screenshots or videos if that helps them understand. If that fails, there's an appeal process to the App Review Board (a link is provided in the rejection notice). Use that only if you truly believe you're in the right and the reviewer is mistaken or being unfair. The board can overturn rejections in some cases, but it's slower.

- Common Rejection Reasons Recap: To ensure you caught everything, here's a quick bullet list of frequent rejection causes and how to avoid them:

- *Crashes and Bugs:* Any crash or obvious bug = likely rejection[13]. Avoid by thorough testing and fixing all crashes.
- *Privacy Violations:* Missing privacy policy, improper data handling, or insufficient permission explanation = rejection[10][20]. Avoid by providing policy and clear consent/usage strings.
- *Incomplete/Misleading Metadata:* Screenshots or description not matching the app content, or using forbidden phrases (like "Download on Android too") = rejection[8][5]. Avoid by keeping metadata accurate and up to date.
- *Unimplemented Features:* Buttons or sections that do nothing or say "coming soon" = rejection under App Completeness[13]. Avoid by removing or hiding unfinished features.

- *UI/Design Issues:* Very poor UI, unreadable text, or an app that looks like a rough draft = possible rejection[69]. Avoid by polishing design, fixing layout issues (especially on iPad).
- *Spam/Duplicate Apps:* If your app is one of many similar ones or is just a wrapped website, it could be rejected[40]. Avoid by providing unique value and not duplicating existing apps.
- *In-App Purchase Rule Breaches:* Using external payment for digital content or mentioning it = rejection[53]. Avoid by using IAP for all digital goods and removing outside purchase links.
- *Sign-In with Apple (if applicable):* If you offer social logins and no Apple login or equivalent = potential rejection (guideline 4.8)[27]. Avoid by including Sign in with Apple for compliance, or meeting the new criteria if using your own login.
- *Permission Misuse:* Requesting a permission that you don't use or not explaining why = rejection[70][71]. Avoid by only asking for what you need and giving a reason in Info.plist.
- *Content Violations:* Any inappropriate content (porn, hate, self-harm, etc.) = rejection[72][35]. Avoid by removing such content or raising age rating appropriately (some content just shouldn't be in the app at all).
- *Intellectual Property:* Using someone else's trademarks or content without permission = rejection[73]. Avoid by using original or licensed content only.
- *Excessive Data Collection:* If your app gathers more data than it needs (especially on kids) or doesn't respect user control = rejection. Avoid by minimizing data collection and being transparent (also implement account deletion if accounts are used).
- *Not Providing Demo Login:* As mentioned, not giving access to a reviewer for locked content = rejection[14]. Avoid by always providing a way in for them.

- After Approval – Stay Compliant: Getting approved once doesn't mean you can slack off later. Future updates will also be reviewed. And Apple periodically re-scans apps; if they later find an issue (say, you added a new SDK that violates

something), they can remove your app. Also, if laws change or guidelines tighten (like the privacy manifest in 2024), you need to comply to stay on the store. So keep an eye on Apple's developer news and update your app accordingly. For example, Apple sometimes announces new requirements (like the account deletion requirement or new rules for kids' apps) – they usually give a deadline for compliance.

- Expedited Reviews: In some cases, you might urgently need an app reviewed (perhaps a critical bug fix or an event). Apple offers an expedited review request process. It's not guaranteed, but if you have a legitimate reason, you can fill out a form explaining why you need faster review. Use sparingly; it's typically granted for emergencies or time-sensitive events.

- App Store Guidelines Updates: Apple updated the guidelines in 2025, and they do so often. It's wise to skim through the "What's New" in App Store Review Guidelines[74] each year. For instance, new rules about social media apps in certain regions or new features (maybe AR, AI content guidelines, etc.) could appear. Stay informed so you're not caught off guard. Subscribing to Apple's developer news or checking Expo's blog for any policy-related posts (Expo sometimes alerts developers about Apple policy changes relevant to RN/Expo, like the privacy manifest issue).

Finally, remember Apple's review is not adversarial – the reviewers' goal is to ensure quality and safety for users. If you've done your homework and your app is well-made and compliant, there's a very high chance you'll get approved quickly. Many apps pass on the first try. And if you don't, use the feedback to improve – Apple even allows you to appeal or ask for clarification if something is unclear. As Expo's guidelines note, you can always address issues and re-submit[1], so a rejection is usually just a bump on the road, not the end of it.

In summary, by covering legal requirements (privacy policy, data use), app content (no objectionable material, proper moderation), design (polished UI, matching metadata),

technical aspects (stability, correct use of APIs), and monetization rules (using in-app purchases, etc.), you significantly increase the likelihood of sailing through App Store review. Use the above as a comprehensive checklist while preparing your Expo/React Native app, and you'll be well on your way to App Store approval. Good luck with your submission!

Sources:

- Apple App Store Review Guidelines (Official)[48][53][4]
- Expo Documentation – App Store Submission Best Practices[75][10]
- Expo Documentation – Configuring iOS Permissions[20]
- Apple Developer Forums / Stack Overflow – Common Rejection Cases[70][27]
- App Industry Insights – Top Reasons for Rejection and Avoidance Tips[76][77]
- AppFollow Blog – App Store Review Guide and Checklist[8][19]

---

[1] [9] [10] [11] [12] [39] [66] [75] App stores best practices - Expo Documentation

https://docs.expo.dev/distribution/app-stores/

[2] [3] [67] [68] Publish Expo React Native App to App Stores

https://codebrahma.com/publish-expo-react-native-app-to-app-stores/

[4] [14] [15] [17] [18] [26] [28] [29] [30] [32] [33] [34] [35] [36] [37] [38] [42] [43] [44] [45] [46] [47] [48] [51] [52] [53] [56] [57] [58] [59] [60] [61] [62] [63] [72] App Review Guidelines - Apple Developer

https://developer.apple.com/app-store/review/guidelines/

[5] [6] [7] [8] [13] [19] [23] [41] [74] iOS App Store Review Guidelines: How to Pass & Avoid Rejection

https://appfollow.io/blog/app-store-review-guidelines

[16] [31] [54] [55] [69] [73] [76] [77] Apple App Store Rejection Reasons In 2025 (And Fixes)

https://twinr.dev/blogs/apple-app-store-rejection-reasons-2025/

[20] [64] [65] Permissions - Expo Documentation

https://docs.expo.dev/guides/permissions/

[21] [22] [70] [71] react native - App Store rejection due to insufficient location usage explanation in Expo app - Stack Overflow

https://stackoverflow.com/questions/77987117/app-store-rejection-due-to-insufficient-location-usage-explanation-in-expo-app

[24] [25] Mobile and App Store Privacy | Customer.io Docs

https://docs.customer.io/accounts-and-workspaces/mobile-privacy/

[27] authentication - Fall 2025 Does Apple Require you to offer Sign in with Apple if you offer sign in with Google and your own login system? - Stack Overflow

https://stackoverflow.com/questions/79819590/fall-2025-does-apple-require-you-to-offer-sign-in-with-apple-if-you-offer-sign-i

[40] Help with App Store Rejection - Guideline 4.0 - Design : r/reactnative

https://www.reddit.com/r/reactnative/comments/1cjhtev/help_with_app_store_rejection_guideline_40_design/

[49] [50] Why does Apple allow Expo's remote upgradable code feature? : r/reactnative

https://www.reddit.com/r/reactnative/comments/1649ge8/why_does_apple_allow_expos_remote_upgradable_code/