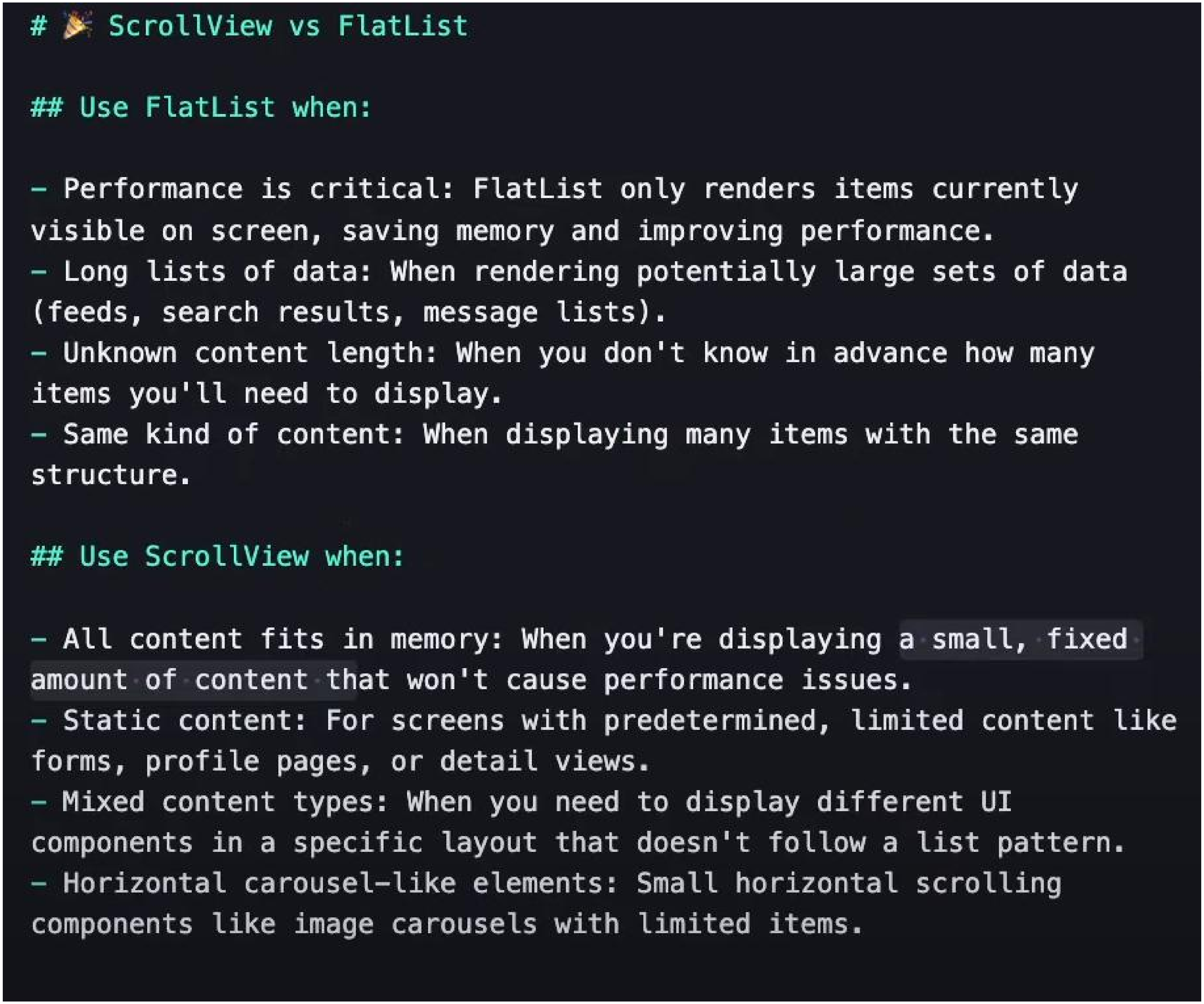
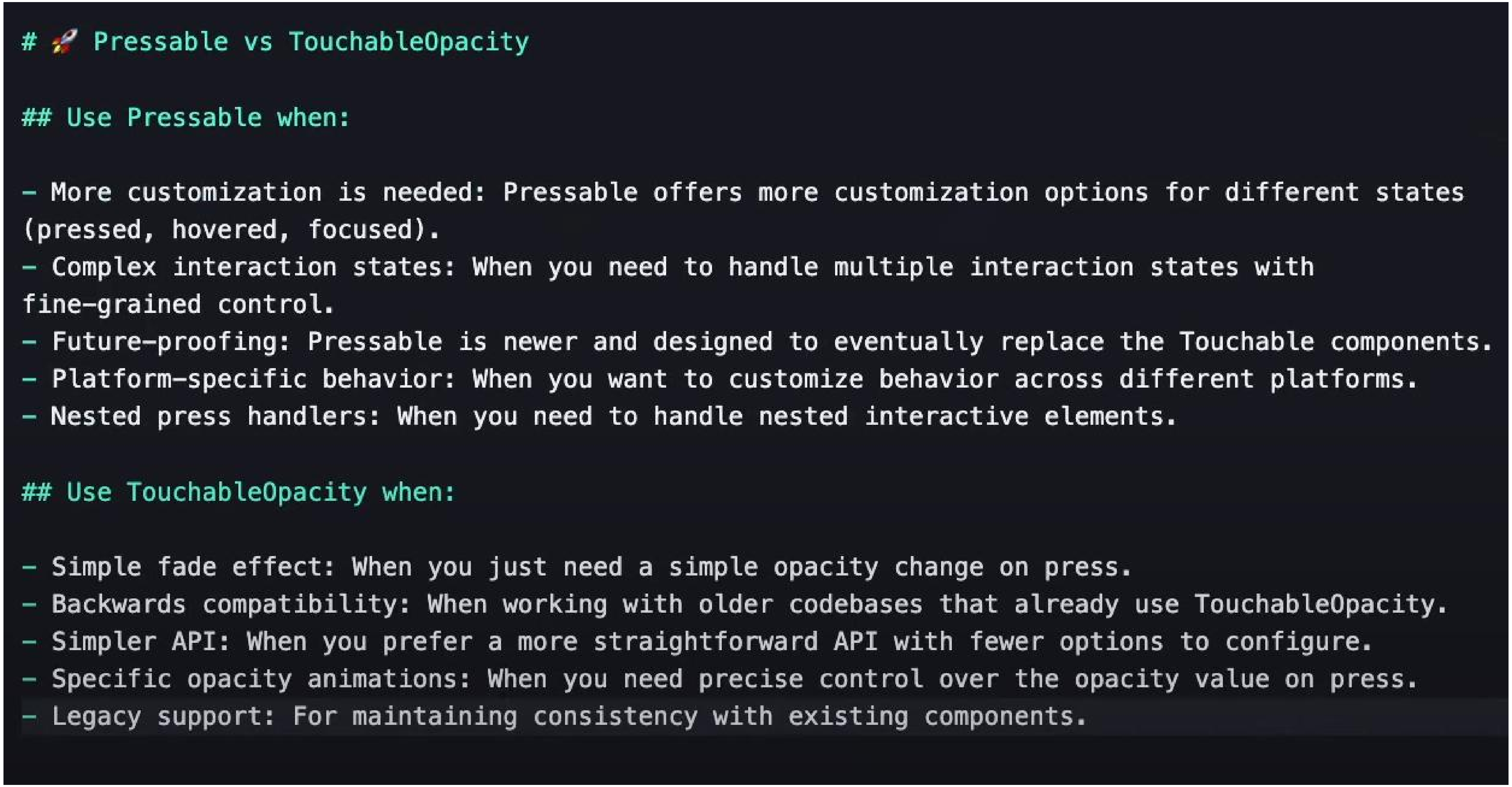


```
# 🎉 ScrollView vs FlatList

## Use FlatList when:

- Performance is critical: FlatList only renders items currently
visible on screen, saving memory and improving performance.
- Long lists of data: When rendering potentially large sets of data
(feeds, search results, message lists).
- Unknown content length: When you don't know in advance how many
items you'll need to display.
- Same kind of content: When displaying many items with the same
structure.

## Use ScrollView when:

- All content fits in memory: When you're displaying a small, fixed
amount of content that won't cause performance issues.
- Static content: For screens with predetermined, limited content like
forms, profile pages, or detail views.
- Mixed content types: When you need to display different UI
components in a specific layout that doesn't follow a list pattern.
- Horizontal carousel-like elements: Small horizontal scrolling
components like image carousels with limited items.
```



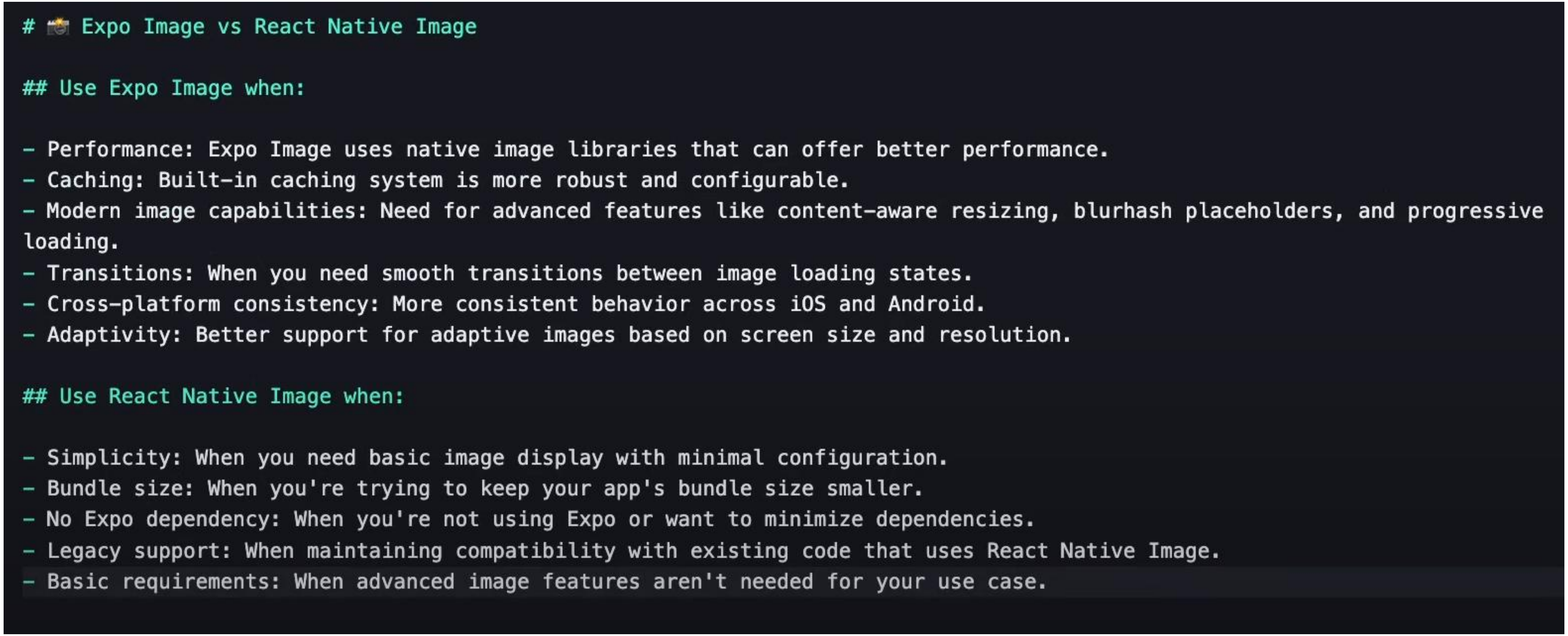
# 🚀 Pressable vs TouchableOpacity

## Use Pressable when:

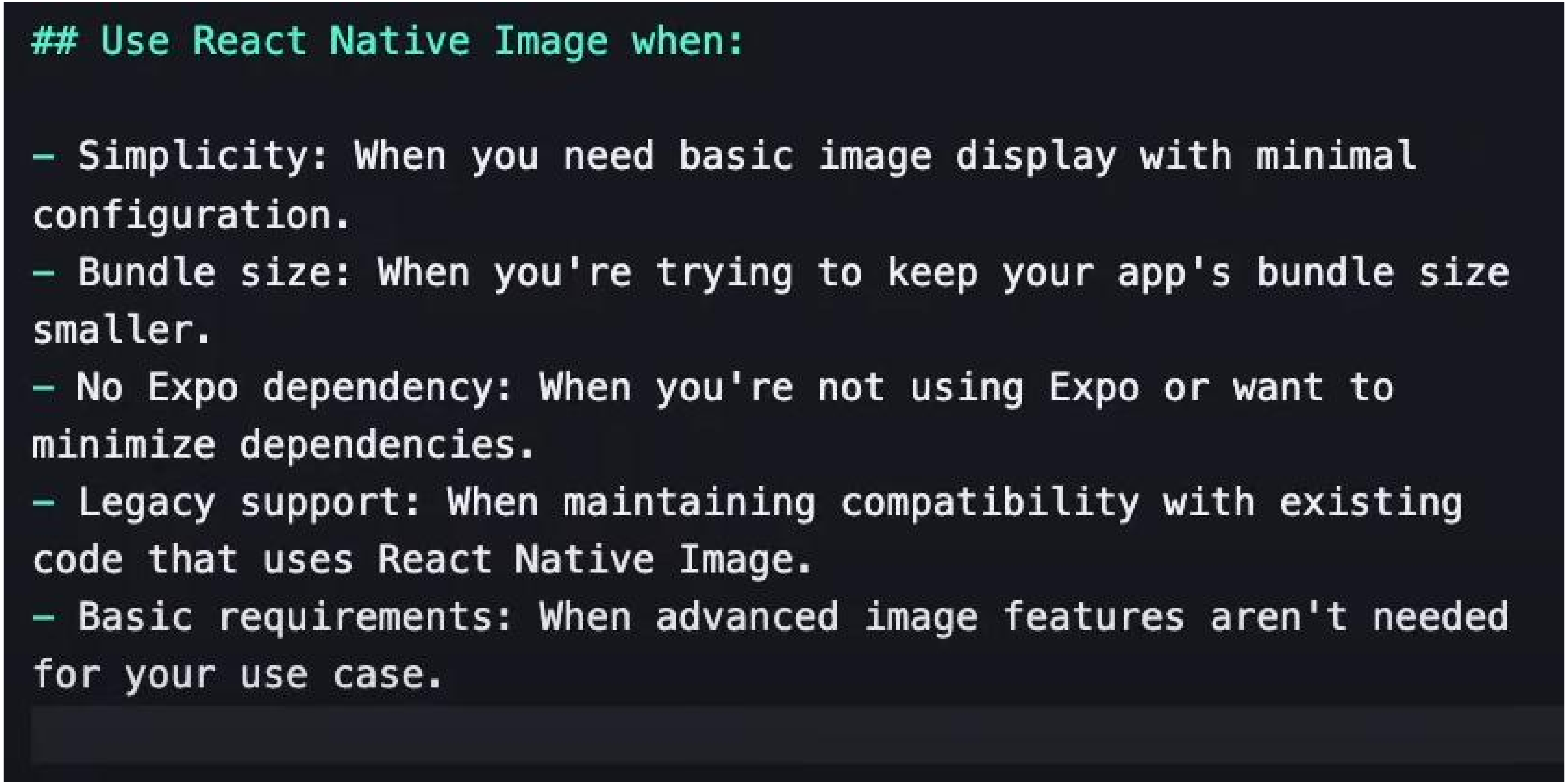
- More customization is needed: Pressable offers more customization options for different states (pressed, hovered, focused).
- Complex interaction states: When you need to handle multiple interaction states with fine-grained control.
- Future-proofing: Pressable is newer and designed to eventually replace the Touchable components.
- Platform-specific behavior: When you want to customize behavior across different platforms.
- Nested press handlers: When you need to handle nested interactive elements.

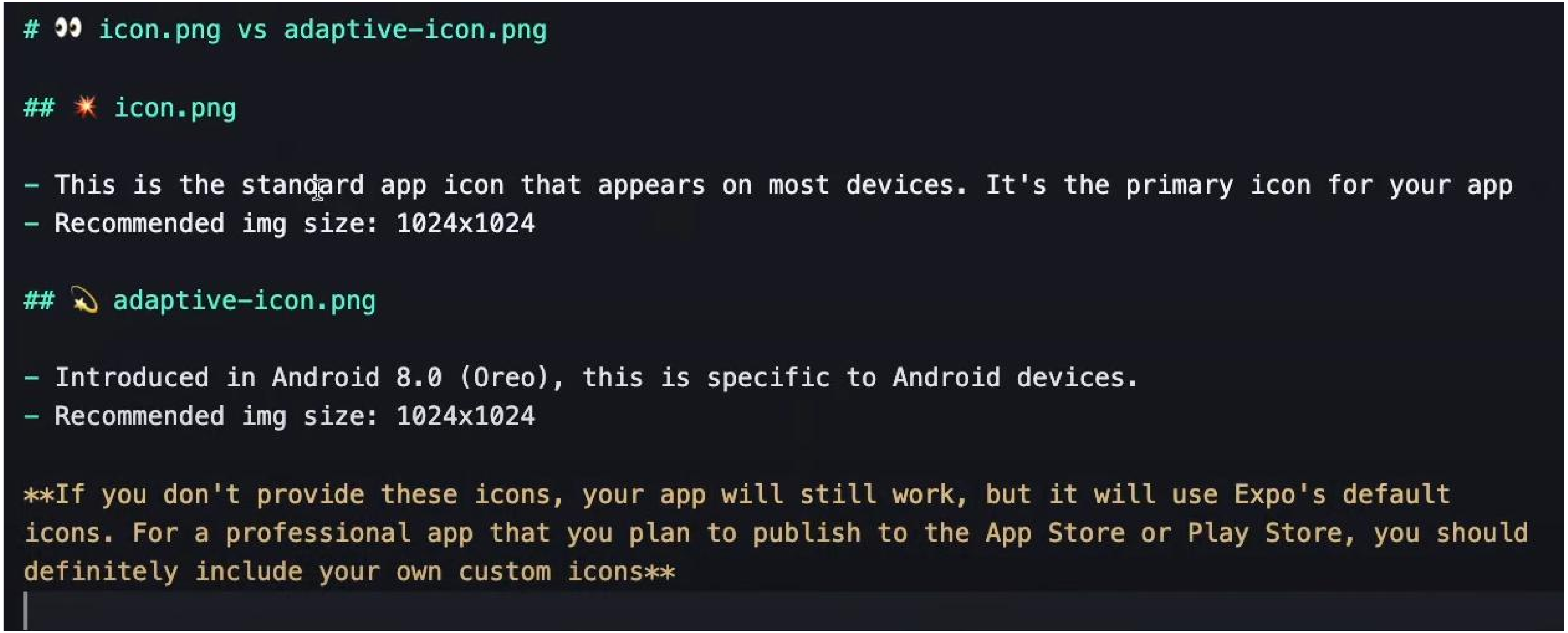
## Use TouchableOpacity when:

- Simple fade effect: When you just need a simple opacity change on press.
- Backwards compatibility: When working with older codebases that already use TouchableOpacity.
- Simpler API: When you prefer a more straightforward API with fewer options to configure.
- Specific opacity animations: When you need precise control over the opacity value on press.
- Legacy support: For maintaining consistency with existing components.

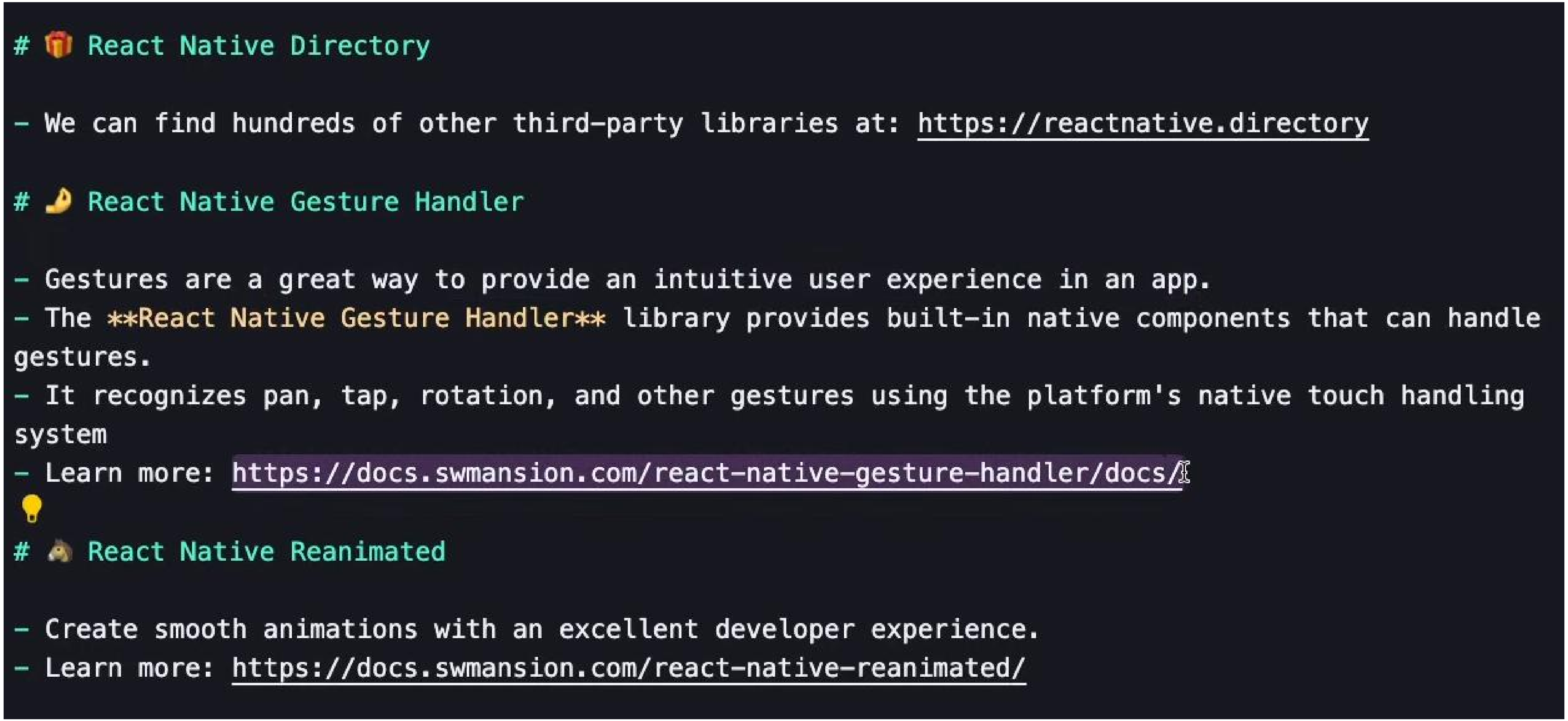


```markdown
# 📸 Expo Image vs React Native Image

## Use Expo Image when:

- Performance: Expo Image uses native image libraries that can offer better performance.
- Caching: Built-in caching system is more robust and configurable.
- Modern image capabilities: Need for advanced features like content-aware resizing, blurhash placeholders, and progressive
loading.
- Transitions: When you need smooth transitions between image loading states.
- Cross-platform consistency: More consistent behavior across iOS and Android.
- Adaptivity: Better support for adaptive images based on screen size and resolution.

## Use React Native Image when:

- Simplicity: When you need basic image display with minimal configuration.
- Bundle size: When you're trying to keep your app's bundle size smaller.
- No Expo dependency: When you're not using Expo or want to minimize dependencies.
- Legacy support: When maintaining compatibility with existing code that uses React Native Image.
- Basic requirements: When advanced image features aren't needed for your use case.
```

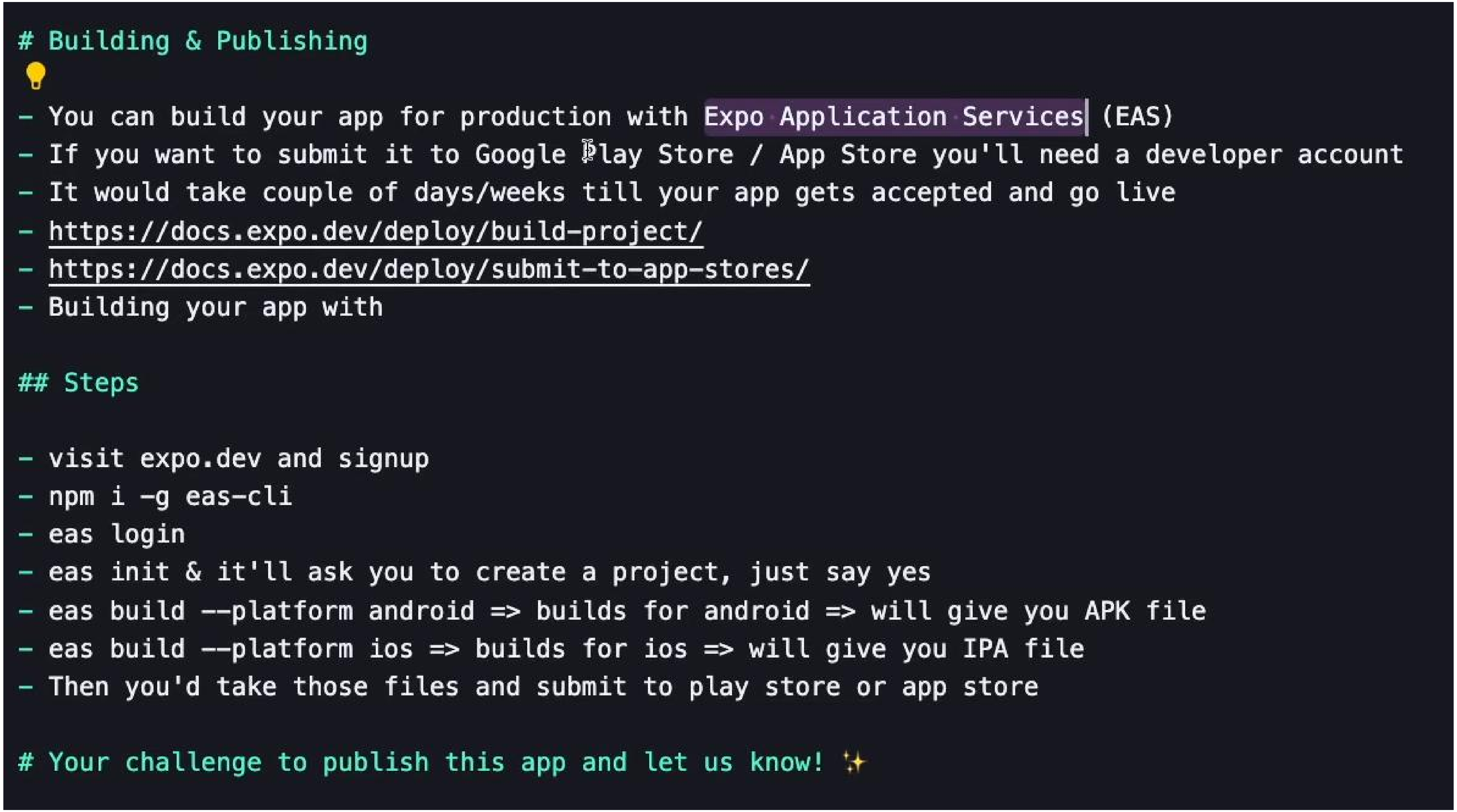

```
## Use React Native Image when:

- Simplicity: When you need basic image display with minimal
configuration.
- Bundle size: When you're trying to keep your app's bundle size
smaller.
- No Expo dependency: When you're not using Expo or want to
minimize dependencies.
- Legacy support: When maintaining compatibility with existing
code that uses React Native Image.
- Basic requirements: When advanced image features aren't needed
for your use case.
```



```markdown
# 👀 icon.png vs adaptive-icon.png

## 💥 icon.png

- This is the standard app icon that appears on most devices. It's the primary icon for your app
- Recommended img size: 1024x1024

## 💫 adaptive-icon.png

- Introduced in Android 8.0 (Oreo), this is specific to Android devices.
- Recommended img size: 1024x1024

**If you don't provide these icons, your app will still work, but it will use Expo's default
icons. For a professional app that you plan to publish to the App Store or Play Store, you should
definitely include your own custom icons**
```



```markdown
# 🎁 React Native Directory

- We can find hundreds of other third-party libraries at: https://reactnative.directory

# 🤌 React Native Gesture Handler

- Gestures are a great way to provide an intuitive user experience in an app.
- The **React Native Gesture Handler** library provides built-in native components that can handle gestures.
- It recognizes pan, tap, rotation, and other gestures using the platform's native touch handling system
- Learn more: https://docs.swmansion.com/react-native-gesture-handler/docs/
💡
# 🐴 React Native Reanimated

- Create smooth animations with an excellent developer experience.
- Learn more: https://docs.swmansion.com/react-native-reanimated/
```



# Building & Publishing
💡
- You can build your app for production with Expo Application Services (EAS)
- If you want to submit it to Google Play Store / App Store you'll need a developer account
- It would take couple of days/weeks till your app gets accepted and go live
- https://docs.expo.dev/deploy/build-project/
- https://docs.expo.dev/deploy/submit-to-app-stores/
- Building your app with

## Steps

- visit expo.dev and signup
- npm i -g eas-cli
- eas login
- eas init & it'll ask you to create a project, just say yes
- eas build --platform android => builds for android => will give you APK file
- eas build --platform ios => builds for ios => will give you IPA file
- Then you'd take those files and submit to play store or app store

# Your challenge to publish this app and let us know! ✨