The All In One CheatSheat

This is my knowledge...

JavaScript

- Configuing of absolute imports i.e using `@/components/my-component` instead of `../../../components/my-component`.
- If a function (closure) survives ***beyond its original context*** and it holds onto large or unnecessary data — you have the potential for a memory leak.
- Use margin for external spacing (space outside an element). Use padding for internal spacing (space inside an element, between its content and its border.)

React

- `children` prop is the most basic and easiest way to optimize your components.
- Pass `...props` to custom components to inherit properties, to avoid redeclaring them.
- If you don't like Tailwind CSS's default sizes, just modify them in the config or inline... I promise, it isn't a taboo.
- For a FlatList with a header, use the `ListHeaderComponent` prop.
- If you ever wanna deploy on GitHub Pages and you're using Vite React + React Router, just save yourself the headache and follow this guide to the letter. And DO NOT FORGET to clear cache and data of the website after build, trust me!
- If you're using a custom `base` in `vite.config.js` (like `/portfolio/`), **all paths to public assets must include that base**.

  ```
  <img src="/images/avatar.png" /> //Wrong
  <img src="/portfolio/images/avatar.png" /> //Right
  ```

-

React Native

- React Native Starter Guide
- React Native Starter Pack to create an Expo app.
- For authentication structure, react-navigation docs explained it well.
- To add cursor selection position:
  - In `<TextInput />`, use `onSelectionChange` prop to get the clicked position:

    ```
    selection={selectionPosition}
    onSelectionChange={(event) => event.nativeEvent.selection}
    ```

    - Then use a state to keep track, then set the cursor to the current position using the `selection` prop in `<TextInput />`

- Use the SafeAreaView component on the screens you register with a React Navigation navigator.
- Use useSafeAreaInsets hook from react-native-safe-area-context instead of SafeAreaView component
- For setting up themes
- To persist in a current screen when developing, this could come in handy.
- JavaScript performance is slower in dev mode due to extra checks, so always test in release builds for accurate performance.
- Use `TouchableWithoutFeedback` to wrap the entire screen to dismiss the keyboard when tapping outside the `TextInput` by calling `Keyboard.dismiss()`.
- If you need faster local storage than AsyncStorage, choose MMKV—it's 30x faster, though it's synchronous.
- Use hairlineWidth instead of 1px borders to achieve a cleaner, more native look, but double-check for unexpected gaps in layout.
- Use `<Modal>` with presentation="pageSheet" to have a native-feeling iOS modal with swipe-to-dismiss functionality, but note it may require a custom fallback for Android compatibility.
- If react-native-bottom-sheet isn't providing a desired native feel, consider checking out react-native-true-sheet.
- Using simple animations like FadeIn and FadeOut for transitions from react-native-reanimated can make the app feel more native.

```
// If you want something to move around smoothly
<Animated.View layout={LinearTransition} />

If you want something to appear and disappear smoothly
<Animated.View entering={FadeIn} exiting={FadeOut} />
```

- Use a slight scale-on-press animation for buttons (use a custom `<Pressable />`) to give a native squishy feel for interactions:

```
const DEFAULT_TARGET_SCALE = 0.98;
const AnimatedPressable = Animated.createAnimatedComponent(Pressable);

function SquishyButton() {
  const scale = useSharedValue(1); // Initial scale value
  const animatedStyle = useAnimatedStyle(() => ({
    transform: [{ scale: scale.value }], // Animated scaling
  }));

  return (
    <AnimatedPressable
      accessibilityRole="button"
      onPressIn={() => {
        cancelAnimation(scale); // Cancel any ongoing animation
        scale.value = withTiming(targetScale, { duration: 100 }); //
Animate to target scale
      }}
      onPressOut={() => {
        cancelAnimation(scale);
        scale.value = withTiming(1, { duration: 100 }); // Animate back
to original scale
      }}
    >
      <Text>Press Me!</Text> {/* Render children */}
    </AnimatedPressable>
  );
}
```

- For native navigation, use Native Stack Navigator in React Navigation or Stack in Expo Router.
- Avoid custom headers for better native interactions, like back-button history, smooth transitions, and dynamic backdrops.
- For all out native bottom tabs look, try react-native-bottom-tabs library. It works with Expo and React Navigation.
- Ensure all interactive elements are at least 44x44 pixels by using the hitSlop prop to increase the touch area without altering the layout.
- Remove `console.log` statements before bundling to prevent slowing the JavaScript thread; use `babel-plugin-transform-remove-console` in the `.babelrc` to automatically eliminate them in production.

- Wrap all list items in a ScrollView for a native feel (i.e the smooth dragging animation at the top or bottom end).
- To modify the default header's height, use `Header` from `react-navigation/elements` and remember to realign header elements to default:

```
import { Header } from "@react-navigation/elements";
import { useSafeAreaInsets } from "react-native-safe-area-context";

export default function Layout() {
  const insets = useSafeAreaInsets();
  return (
    <Stack
      screenOptions={{
        header: ({ options }) => (
          <Header
            {...options}
            headerStyle={{
              height: 200, // custom header height
            }}
            headerLeftContainerStyle={{
              justifyContent: "flex-start", // Align to the top
              paddingTop: insets.top + 8, // Adjust for status bar + 8
to match default look
              paddingLeft: 16, // Left padding; 16, matches default
look
            }}
          />
        ),
      }}
    />
  );
}
```

- If you only need toasts on **Android**, you can use the ToastAndroid API provided by React Native.
- Use `.jsx` or `.tsx` for files containing React components (JSX) and `.js` or `.ts` for regular JavaScript or TypeScript files.
- Memoization isn't always helpful! `useCallback` and `useMemo` have a cost, so only use them if you notice performance issues.
- Dev mode slows things down, so for accurate performance testing, turn off JS Dev Mode on Android and run Metro on iOS.
- Memory leaks usually happens when closures hold onto data your app no longer needs, especially if you store those closures in arrays or global references. To check

leaks, open *React Native DevTools → Memory tab → Allocation instrumentation*, and start a recording session. Stop the recording and look out for blue bars (those mean the objects are still stuck in memory). Ideally, you want to see gray bars—those mean the memory was successfully released. Functions (Closures) that look tiny but with a massive retained size are your culprits.

- With TextInput, avoid letting React control the `value` during `onChangeText` to prevent race conditions. Instead opt for uncontrolled component by removing the value prop, but still store the state for later use, like onSubmit or verification.
- The perception of how quickly (and smoothly) apps load and respond to user interaction is sometimes more important than raw performance. For instance, it's usually better to provide a quick response and regular status updates than make the user wait until an operation is completed.
- Try to **minimize unnecessary `<View>` wrappers** to reduce the renderer's workload. The less the renderer has to flatten, the faster the app will be.

Expo

- To delay the app's startup until resources are ready, check here.
- In expo router, something must be rendered to the screen while loading the initial auth state.
- Instead of using fonts through assets you can opt in for getting the fonts straight from expo-google-fonts
- To start expo app without cache, run `npx expo start --clear`
- So apparently, there's expo secure store and react native async storage, each can both used to store values locally on the device. When to use which:

> *Use react native async storage to store non-sensitive data, such as user preferences, user preferences, app state, or caching.*
>
> *Use expo se`cure store to store sensitive data like passwords, tokens, or personal information*

- Using Expo Secure store as the custom persistence manger for firebase auth *(untested)*
- Expo vector icon supports using the icon as a button. For instance:

```
<Ionicons.Button name={"people"} size={24} color="#ffffff">
  This is a button icon
</Ionicons.Button>
```

- Expo Router automatically adds react-native-safe-area-context support.
- To preload/cache expo icons, expo google fonts or local fonts in the `useFonts` hook:

```
const [fontLoaded, fontError] = useFonts({
  BarlowSCRegular: require("RELATIVE_PATH_HERE"), // Load local fonts
  Inter_900Black, // Load fonts from expo-google-fonts
  ...Ionicons.font, // Load icon fonts from @expo/vector-icons
});
```

- In Expo Router, keep only the initial screens in the `(tabs)` folder. Move any nested stacks outside to avoid issues with hiding the tab bar and routing between stacks in different tabs.
- For keyboard handling, check out this thorough guide.
- To prevent bottom tabs from moving above the keyboard on Android, set `softwareKeyboardLayoutMode` to `pan` in app config or tabBarHideOnKeyboard
- Use `npx expo-doctor` in your project's root to diagnose and fix common issues in your Expo app.
- For native module support, view app from user perspective, and a full development experience, switch from Expo Go to a Development Build.
- To review and upgrade dependencies, run `npx expo install --check`
- To make the status bar match your app's background color, try setting `userInterfaceStyle` in the app config to `automatic`, `light`, or `dark`
- To configure eslint for Expo, run `npx expo lint`

- To switch tabs without performing any navigation, you should use a TabTrigger
- Use the `reset` prop on `TabTrigger` to control how a tab handles its state (options: `always`, `onLongPress`, or `never`); for example, setting `reset="always"` will return a tab with nested stacks to its index route when navigating.
- Run your Expo app locally in production mode using `npx expo start --no-dev --minify` to catch production-specific errors.
- Use `npx expo install` instead of `npm install` to ensure library compatibility and get warnings about issues.
- Use a .env file to define variables like `EXPO_PUBLIC_[NAME]=VALUE` & access them via `process.env`

```
// env
EXPO_PUBLIC_API_URL=https://staging.example.com

// jsx
const apiUrl = process.env.EXPO_PUBLIC_API_URL;
```

- You can use standard `.env` files like `.env`, `.env.local`, or `.env.production`, which load based on priority, and it's recommended to add `.env.local` files to `.gitignore`
- Never store sensitive keys in your app code or `AsyncStorage`; instead, use expo-secure-store or react-native-keychain, and always keep secrets server-side when possible.
- You can use Expo's Linking API to open links in the default web browser, or the expo-web-browser library for in-app browsing.
- To prebuild for one platform, run `npx expo prebuild --platform ios` or `--platform android`.
- To build and run native files, run `npx expo run:android` or `npx expo run:ios` - Expo will run `npx expo prebuild` first if native directories don't already exist.
- To reset and rebuild native files, run `npx expo prebuild --clean`, followed by `npx expo run` (or `npx expo run:ios` or `npx expo run:android`).

> Optional: It is recommended to run `npx expo install --fix` before proceeding with the prebuild. This solves a lot of simulator build warnings/errors

- Must we use Index file? Yes. Using index file in folders is essential in Expo Router for defining main routes, as it treats folders as their respective routes.
- Avoid editing native files (AndroidManifest.xml or Info.plist) directly; use a config plugin instead since direct edits will be lost after running prebuild.
- Add `.expo`, `android`, and `ios` to your `.gitignore`, to keep local Expo configurations and platform-specific files from committing between prebuilds.
- Use Expo Atlas to analyze and optimize the libraries in your project, affecting your JavaScript bundle size.
- Follow these steps to set up Android Simulator or iOS Simulator.
- `navigate` no longer replaces screens—it pushes them (Expo Router v4). Use `dismissTo`, which dismisses screens until the target is reached or replaces the current screen if not found.
- Instead of using `<Stack.Screen>`, you can configure a route's options inside the component with `navigation.setOptions()`, like hiding the header dynamically.
- Set route options dynamically with `<Stack.Screen>`, or use `router.setParams()` to update things like the screen title while the app is running
- Stack navigator ignores pushing multiple instances of the same screen; to change this, use `getId()` in `<Stack.Screen>` to generate a unique ID for each push.
- To remover stack screens: `dismiss()` removes one screen, `dismiss(n)` removes multiple screens, `dismissTo('/route')` goes back to a specific screen (not in history, it pushes it), and `dismissAll` clears the stack back to the first screen.
- `initialRouteName` ensures a specific screen loads first when a deep link is used, but it doesn't impact regular in-app navigation.
- Expo Router automatically generates a /_sitemap for debugging, but in SDK 52 and later, you can remove it by adding `sitemap: false` to the expo-router config.

- To fix `Error: Attempted to navigate before mounting the Root Layout component. Ensure the Root Layout component is rendering a Slot, or other navigator on the first render.`, check here
- To fix missing back buttons on certain screens, add `unstable_settings` in your layout and set `initialRouteName` to ensure a proper starting route for navigation.
- Notifications may not always trigger the response listener when reopening a killed app. Use `useLastNotificationResponse` or `getLastNotificationResponseAsync` for reliability.
- To generating different config files in Expo, run `npx expo customize`
- For a built-in, secure UUID generator in Expo, use `Crypto.randomUUID()` from `expo-crypto` instead of installing an external package.
- Its always best to prepare your codebase using the **React Compiler ESLint plugin**. It flags any violations of the Rules of React in your code.
- To prevent a (white) flash during screen routing transition renders especially on Android, you can set the app's native `backgroundColor` in app.json to match your app's background theme.

Prettier

- To format code with Prettier and sort Tailwind classes:
  - *Install the packages*:

```
npm install -D prettier prettier-plugin-tailwindcss
```

  - *Create a `.prettierrc` file, then add*:

```
{
  "plugins": ["prettier-plugin-tailwindcss"]
}
```

- *As a plus, to format all specific files (e.g `.js` and `.jsx` files) and reformat Tailwind classes in one go, run*:

```
npx prettier "**/*.{js,jsx}" --write
```

Chrome Extensions

- If you're coding with Chrome APIs, use the `chrome-types` package to get smart auto-completion — it updates itself whenever Chromium updates.
- Since service workers can't use `window.localStorage` or rely on global variables (because they shut down often), use `chrome.storage.local` instead to keep data safe across sessions.
- Event listeners must be statically registered in the global scope of your service worker, so avoid placing them inside async functions — this helps with restoring them after a reboot.
- Let users decide what data your extension can access by using optional permissions when possible — it's cleaner, safer, and more respectful.
- With WebSockets, make sure your extension doesn't get suspended prematurely — use heartbeats.
- If using React + Vite, make sure to place the manifest.json file into the public folder, so building it is copied into the root folder (dist) automatically.
- Use this Jonghakseo's chrome extension template for fast building of any chrome extension needed or even to learn indepthly about chrome extensions

NativeWind

- ClassNames can be used in a Flatlist component.
- By default NativeWind maps className->style, but it can handle the mapping of complex components.
- To create a component with default styles

```
function MyComponent({ className }) {
  const defaultStyles = "text-black dark:text-white";
  return <Text className={`${defaultStyles} ${className}`} />;
```

```
}

<MyComponent className="font-bold" />;
```

- To handle components with multiple style props:

```
function MyComponent({ className, textClassName }) {
  return (
    <View className={className}>
      <Text className={textClassName}>Text Component</Text>
    </View>
  );
}
```

- To style a child element based on its parent's state, mark the parent with the `group` class and apply `group-*` modifiers.
- Don't use `space-{n}` anymore, instead use `gap-*`
- `Divide Width` has temporarily been removed.
- Outline doesn't work in NativeWind
- Transitions and Animations are experimental features. Use with caution.
- NativeWind remapProps lets you map Tailwind classes to third-party component style props or override them for easier customization.

```
remapProps(Component, { "new-prop": "existing-prop" });
remapProps(Component, { prop: true });
```

- Use NativeWind cssInterop to tag components, allowing `className` strings to resolve into styles, ideal for custom or third-party components needing precise prop handling.

Firebase

- In Firestore, addDoc() adds a new document with an auto-generated ID. setDoc() creates or replaces a document with a specific ID (pass `{ merge: true }` to

avoid overwriting document). updateDoc() updates specific fields in an existing document (and only if the document exists).

- Currently, Firebase does not support providers authentication (e.g Google) using signInWIthPopup / signInWIthRedirect f**or React Native.** Here is another way to set it up.

- To reduce storage costs, exclude long string fields not used for querying (e.g., notes, comments) from indexing.

- In Firestore, serverTimestamp() is preferred to ensure all records have a consistent timestamp.

- All Firebase Auth Error Codes

- In Firebase, prefer initializeAuth() over getAuth().

- When paginating query results in batches, always use `orderBy()`.

- To batch upload data to Firestore Database:

```
async function batchUploadToFirestore(dataArray, collectionName) {
  const batch = writeBatch(db);
  try {
    dataArray.forEach((dataItem) => {
      const docRef = doc(collection(db, collectionName));
      batch.set(docRef, { ...dataItem, id: docRef.id });
    });

    await batch.commit();
    console.log(
      `All data uploaded to Firestore collection: ${collectionName}`
    );
  } catch (error) {
    console.error(
      `Failed to upload data to Firestore collection:
${collectionName}`,
      error
    );
  }
}
/*
 * Note:
 * - This method is efficient for creating multiple documents with
auto-generated IDs.
 * - If your dataItem already has a meaningful 'id', use:
 *   batch.set(doc(db, collectionName, dataItem.id), dataItem);
 */
```

- Firestore snapshots aren't local storage-friendly due to metadata and non-serializable objects. For accurate pagination and queries, store essential data like IDs from the snapshot and retrieve a fresh snapshot when needed.

```
const lastVisitedId = "lastDocumentId"; // save this ID locally
// When needed, retrieve a fresh reference
const lastVisitedDoc = await getDoc(doc(db, "collectionPath",
lastVisitedId));
// Pass this fresh snapshot to startAfter()
const query = collection(db, "collectionPath")
  .orderBy("someField")
  .startAfter(lastVisitedDoc)
  .limit(batchSize);
```

-

Supabase

- To delete all tables in the `public` schema:

  > WARNING: This command is irreversible and will delete all data and table definitions.

  - Go to the SQL Editor
  - Run:

```
do $$ declare
  r record;
begin
  for r in (select tablename from pg_tables where schemaname =
'public') loop
    execute 'drop table if exists ' || quote_ident(r.tablename) ||
' cascade';
  end loop;
end $$;
```

-

Gorhom BottomSheet

- If the BottomSheet isn't rendering as expected, Ensure your app's root layout is wrapped with `BottomSheetModalProvider`.
- To render childrenor components in a BottomSheet, use the versions provided by BottomSheet - learned that one the hard way.
- If BottomSheet is adding extra snapPoints, disable dynamic sizing i.e `enableDynamicSizing={false}`.
- For better UX, keep BottomSheetModal's drag-to-close behavior, but disable it with `enablePanDownToClose={false}` if necessary.

React Native Gifted Charts

- To dynamically size the `BarChart` to its parent width, use the parent `onLayout` prop to get the width, store in a state, and pass it to `BarChart` `parentWidth`.
- To fit bars to the chart's width, use `adjustToWidth`; won't work if `barWidth` or `spacing` is set.
- To have a fully dynamic responsive chart, set `parentWidth` and enable `adjustToWidth`.
- To remove horizontal rules (lines) behind bars, enable the `hideRules` prop.
-

Commit Messages Guide

This was gotten from Gitmoji.dev

- 🐛 [bug]: Fix a bug.
- ✨ [sparkles]: Introduce new features.
- 🚀 [rocket]: Deploy stuff.
- ♻️ [recycle]: Refactor code.
- 🎨 [art]: Improve structure / format of the code.
- ⚡ [zap]: Improve performance.
- 📝 [memo]: Add or update documentation.
- 🚑 [ambulance]: Critical hotfix.
- 💡 [bulb]: Add or update comments in source code.
- 🚧 [construction]: Work in progress.
- 👷 [technologist]: Improve developer experience.
- 🚸 [children_crossing]: Improve user experience / usability.

- ⬆️ [arrow_up]: Upgrade dependencies.
- 💄 [lipstick]: Add or update the UI and style files.
- ♿ [wheelchair]: Improve accessibility.
- 👽 [alien]: Update code due to external API changes.
- 🚚 [truck]: Move or rename resources (e.g.: files, paths, routes).
- ➕ [heavy_plus_sign]: Add a dependency.
- ➖ [heavy_minus_sign]: Remove a dependency.
- 📱 [iphone]: Work on responsive design.
- 🔥 [fire]: Remove code or files.
- 💬 [speech_balloon]: Add or update text and literals.
- 🍱 [bento]: Add or update assets.
- 💫 [dizzy]: Add or update animations and transitions.
- ⬇️ [arrow_down]: Downgrade dependencies.
- ✏️ [pencil2]: Fix typos.
- 🙈 [see_no_evil]: Add or update a .gitignore file.
- 🎉 [tada]: Begin a project.
- 🗑️ [wastebasket]: Deprecate code that needs to be cleaned up.
- 💩 [poop]: Write bad code that needs to be improved.

AI Prompts

- To explain and teach code concepts:

  > [Content to explain]
  >
  > You are an expert tutor in [content field]. I'm a complete beginner with no prior knowledge of the content provided above. Explain the content above in a simple, descriptive, and detailed explanations covering every aspect, WITHOUT SKIPPING ANY PART, using simple language. Explain each piece in a way that connects the dots. Feel free to Include extra context and examples to ensure a thorough deeper understanding.

- To clean, optimize and refactor code:

  > [INPUT CODE]
  >
  > Deeply analyze the provided code step-by-step to Refactor and Refine. Specifically:

> - Code Readability & Maintainability
> - Fixing any potential bugs, performance issues, or memory leaks.
> - Use/Swap any potential improvements that could make the code more efficient.
> - Descriptive naming and better styling practices.

- To review codebase:

> [INPUT CODE]
>
> Review the code above step by step in detail. Check for bugs, issues, mistakes, inefficiencies, or vulnerabilities. Provide a straight to the point review of issues in the code with fixes

- To generate commit messages:

> Using these commit guide emojis: 🐛 (Fix a bug), ✨ (Introduce new features), 🚀 (Deploy stuff), ♻️ (Refactor code), 🎨 (Improve structure / format of the code), ⚡ (Improve performance), 📝 (Add or update documentation), 🚑 (Critical hotfix), 👷 (Improve developer experience), ⬆️ (Upgrade dependencies), 💄 (Add or update the UI and style files), 🚚 (Move or rename resources), ➕ (Add a dependency), ➖ (Remove a dependency), 🔥 (Remove code or files), 🍱 (Add or update assets), ⬇️ (Downgrade dependencies), ✏️ (Fix typos), 🙈 (Add or update a .gitignore file), 🎉 (Begin a project), 🗑️ (Deprecate code that needs to be cleaned up).
>
> Generate 5 different instances of a concise commit messages (max 72 chars for the subject) based on the provided code changes. Strictly adhering to the Conventional Commits specification and follow best practices. The output should be in the format: **"[Emoji] [Commit message]"**

- To generate Vector illustration:

> Generate outline vector illustration with white background showing a group of people going through a breakup

- To summarize or shorten

> "[Include content]"
>
> Summarize the content above into a single, conversational sentence that highlights only the most essential information, keeping it as simple and short as

> possible for easy, at-a-glance understanding. Give 5 options.

- To create database structure for an app:

> Create a [Firestore] data structure for my app. Here are the core features of the app so you'd use that to construct a full most optimal data structure for the app firestore database:
>
> [- add app feature in list format]

- To generate better prompts:

> [Input Prompt]
>
> Act as a prompt engineer. Review the prompt above and optimize it to make it better for the real use. You can ask any questions before proceeding with optimizing the prompt. REMEMBER, THE PROMPT ABOVE IS NOT BEING USED NOW, YOU'RE TO OPTIMIZE IT ONLY.

- To generate top results:

> [INSERT CONTENT]
>
> Review the provided content and determine the most efficient, optimized, and recommended approach to achieve it. Ensure the solution aligns with real-world best practices.

- To generate app designs or UI/UX:

> You are a professional UI/UX product designer with 20+ years of experience and expertise in building intuitive, user-friendly [VS Code Extensions].
>
> [Insert app description].
>
> Using your expertise, provide thoughtful recommendations for UX Flow, UI layout concepts, color schemes. Ensure the design is optimized for a native [mobile app] experience.

- To redesign an existing page

> You're a professional product designer and developer with 20+ years of experience in [building saas that converts]. Above is [the ___ page for my saas]. Improve and redesign it + the uiux and everything i'm not seeing.

> Remember, this is a very important page... it should make user [feel good to do it... it should make it feel like this would solve whatever pain they have on twitter... it should justify them paying $9/month]

- To brainstorm ideas:

> Hey! I'd like to brainstorm an idea together. It's about:
>
> [What are we brainstorming about?]
>
> I have some rough thoughts, but I'd love to build on them gradually with you:
>
> | [Include rough thoughts or ideas] |
>
> Let's keep it casual, short and engaging, like a back-and-forth conversation between friends. Can you start by sharing one idea or question related to the idea to kick things off? From there, we can bounce ideas off each other little by little, step by step.

- To convert UI to JSON to Code

> **UI to JSON**
>
> Deeply analyze the attached UI image and describe this in as much detail as you can to hand over from a UI/UX designer to a [React Native (Expo)] developer. Output characteristics as JSON [and end with a prompt explaining how to implement the UI for a code agent].
>
> The attached image is a mockup/wireframe UI design for a mobile app, [DishSpot (meal recommendation app)]. Extract every visual token, describe design and UX intent from it, infer reasonable missing details, and surface assumptions clearly.
>
> Tech stack is: React Native (Expo) - JS/JSX only. Styling: twrnc. State management: zustand. Icons: Expo Vector Icons — Ionicons. Prefer Expo SDKs and recommended libraries based on app needs.
>
> JSON must contain at least these top-level keys (order matters, list them in this order): meta, designTokens, atoms, molecules, organisms, screens, componentLinkage, dataModels, assumptions_and_missing_details, implementationNotes

> Start from atomic tokens, then atoms→molecules→organisms→screens. Use** `a_`, `m_`, `o_`, **`s_` prefixes for ids. Each lists the components used and navigation links to other screens for easy assembling files deterministically.

> **JSON to Code**

You are an elite, senior-level React Native (Expo) developer. Your sole function is to act as a “Spec-to-Code" engine. You will be given detailed UI specifications in JSON format, and you convert into clean, production-ready, and performant code for the [“DishSpot”] mobile app.

Operate with extreme precision and adhere to these rules:

- Build the smallest, most reusable components first.
- Break code into multiple files when it improves maintainability.
- Prioritize Flexbox system for responsiveness.
- Tech Stack:
-
  - Framework: React Native (Expo); JavaScript/JSX only.
  - Styling: You MUST use** **twrnc styling library. Apply inline via the `style` prop (e.g. `style={tw...}`}>). If a style property cannot be expressed as twrnc class, apply it as an inline object merged in an array:** **`style={[tw..., {...}]}`.
  - Icons: Expo Vector Icons - Ionicons.
  - State Management: For simple local state, use the** **`useState` hook. For shared/global state, add a placeholder comment where an assumed Zustand store would be used.
- Convert specified colors, font sizes, spacings etc to twrnc classes when possible; if a value is not available as a tw class, apply the most appropriate tw combination and add a short inline comment.
- Infer reasonable details only when missing and necessary.
- Use functional arrow components.
- Use JSDoc comments to clearly define prop types.

This is the JSON Spec:

```
[Input JSON Spec]
```

```
[Paste optional context code here]
```

> **Creative Mode**
>
> You know what, this time, i don't have a JSON info description of how i want it to be like. I want you to take the wheel. Design the profile screen for [Dishspot]. Use the design of [Dishspot] and know it takes inspiration from [Spotify] UI/UX.
>
> [Include optional details]
>
> Do your thing.

Overall

- To create design blueprint/wireframes on a board, use Milanote.
- For color theme generation, use UI Colors or Realtime Colors or Color Mind or better still ask AI (Copilot or Claude)
- For UI component libraries, use Tamagui or gluestack (formerly NativeBase).
- For toast notification, use react-native-toast-message, react-native-notifier, react-native-toast, react-native-root-toast (expo-recommended)
- In Zustand, when combining smaller stores into a single bounded store, use `create` only for the bounded store, not for each individual stores.

  > Note: When combining stores into one, make sure each store has **unique state property names**. If state properties overlap (e.g., both stores having `isLoading`), this can lead to serious side effects, as one state might overwrite the other unexpectedly.

- For free public APIs, check out public-apis, Public-APIs or RapidAPI.

UX Tips

- Follow the design guidelines for the specific platform (the Human interface guideline for Apple, and Material design guideline for Google)
- When brainstorming with AI regarding building an app, ask it to design each app screen from a ui ux expert pov.

- Place controls in the middle or bottom of the screen for easy reach, and enable swipe gestures for smoother navigation.
- Use platform-specific features for only that platform to make your app feel truly native. Don't mimic effects like iOS's liquid glass on other platforms. Instead, create components suited to each platform.

App Essentials Checklist

- ☐ Design wireframes and mockups for all screens.
- ☐ Integrate navigation (e.g., React Navigation, Expo router).
- ☐ Implement user authentication (Firebase or custom backend).
- ☐ Ensure form validation with instant feedback.
- ☐ Add loading spinners for sign-in, sign-out, and data fetching.
- ☐ Use skeleton loaders for placeholder content while images or videos load.
- ☐ Set keyboard management to auto-dismiss and avoid covering input fields.
- ☐ Check navigation flows to ensure smooth back-and-forth between screens.
- ☐ Implement smooth animations for screen transitions and interactions.
- ☐ Use lazy loading for images, videos, or heavy content to improve performance.
- ☐ Enable offline support for critical parts of the app.
- ☐ Add a splash screen for a polished startup experience.
- ☐ Add pull-to-refresh gestures for lists and content-heavy pages.
- ☐ Add toast messages for non-critical notifications.
- ☐ Add password visibility toggles for password fields.
- ☐ Test on multiple devices and screen sizes for compatibility.
- ☐ Support biometric authentication (Face ID, fingerprint).
- ☐ Ensure high contrast and readability for text, especially in dark and light modes.
- ☐ Design for one-handed usage by placing important actions near the bottom.
- ☐ Add sync indicators for reconnecting after being offline.
- ☐ Auto-logout feature after a session timeout for security.
- ☐ Implement auto-complete or suggestions to make forms easier to fill out.
- ☐ Setup color themes for light and dark modes if needed.

Developer Essentials Checklist

- ☐ Use ESLint and Prettier for code formatting and error prevention.
- ☐ Use TypeScript to catch errors early during development.

- ☐ Store secrets securely with environment variables or SecureStore.
- ☐ Regularly cleanup dependencies not in use.
- ☐ Clean up unnecessary console.log statements.
- ☐ Use Git & regular commit codes.
- ☐ Clean up resources like intervals or listeners to prevent memory leaks.
