PRD: React Native Responsive Design Library

Project Name: react-native-responsive-ui Owner: Vincent / Your team Target Users: React Native / Expo developers Status: Planning

---

1. Goal / Problem Statement

React Native developers face challenges in building responsive apps across multiple screen sizes and device types (mobile, tablet, desktop).

Problems we aim to solve: 1. Scaling design sizes (font, spacing, icon) automatically without manual calculations. 2. Adjusting layouts for different breakpoints without duplicating code. 3. Providing an optional mechanism for full layout splitting when mobile and tablet UX diverge. 4. Improving developer experience (DX) with minimal boilerplate and intuitive APIs.

Success Metrics: - Reduce boilerplate code for responsive layouts by 50%. - Consistent UI across device sizes with minimal manual adjustments. - Developers can adopt scaling + responsive helpers without creating extra files for most screens.

---

1. Scope

In Scope: - Scaling system for numeric styles. - Breakpoint detection (mobile, tablet, desktop) using screen width. - Responsive helpers for conditional values within the same component. - Optional layout splitting for screens with fundamentally different designs. - Design tokens for spacing, font sizes, and radius. - Expo + React Native compatibility (iOS, Android).

Out of Scope (v1): - Platform-specific file resolution (Metro bundler hacks). - Automatic orientation-based layouts. - Web support. - Auto-generating responsive components from Figma.

---

1. Key Features

Feature	Description	Priority
Scaling	Automatically scale numeric values (font, padding, margin) based on screen width relative to a design baseline.	High
Design tokens	Predefined tokens (space, font, radius) using scaling.	High
Responsive helper	API to choose values per breakpoint within same component.	High
Optional layout split	API to define full component variations per breakpoint.	Medium

Feature	Description	Priority
Breakpoint detection	Utility useDeviceType() returning "mobile"	"tablet"
Dev-friendly DX	Minimal boilerplate; alias functions (s(), responsive()) for fast adoption.	High

1. Architecture & API Design

4.1 Scaling

```
import { s } from "@ui/responsive";
const styles = {
  padding: s(16),
  fontSize: s(14),
};
```

4.2 Design Tokens

```
import { space, font, radius } from "@ui/tokens";
const styles = {
  padding: space.md,
  fontSize: font.body,
  borderRadius: radius.sm,
};
```

4.3 Responsive Helper

```
import { responsive } from "@ui/responsive";
const columns = responsive({ mobile: 1, tablet: 2, desktop: 3 });
<View style={{ flexDirection: columns === 1 ? 'column' : 'row' }} />
```

4.4 Optional Layout Split

```
import { responsiveComponent } from "@ui/responsive";
import HomeMobile from "./Home.mobile";
import HomeTablet from "./Home.tablet";
export default responsiveComponent({ mobile: HomeMobile, tablet: HomeTablet });
```

4.5 Breakpoint Detection Hook

```
import { useDeviceType } from "@ui/responsive";
const device = useDeviceType(); // "mobile" | "tablet" | "desktop"
```

4.6 Configuration

```
<ResponsiveProvider baseWidth={760} breakpoints={{ tablet: 768, desktop: 1024 }}
>
  <App />
</ResponsiveProvider>
```

---

1. Developer Experience (DX) Principles
2. Default scaling everywhere.
3. Responsive helpers preferred; avoid extra files unless necessary.
4. Layout splitting opt-in; only for complex screens.
5. Tokens first; encourage consistency.
6. Minimal boilerplate; short aliases (s, responsive).

---

1. Roadmap / Implementation Phases

Phase	Description	ETA
Phase 1 – Scaling system	s() function, design tokens, createScaledStyle.	2 weeks
Phase 2 – Breakpoints & responsive helper	useDeviceType(), responsive(), responsive numeric & boolean values.	2 weeks
Phase 3 – Layout split	responsiveComponent(), optional per-screen files.	3 weeks
Phase 4 – DX improvements	ResponsiveProvider, auto-scaled styles, documentation, examples.	2 weeks
Phase 5 – Optional	Orientation support, web support, CLI generator.	Future

---

1. Success Metrics
2. Library can scale font/spacing without wrapping every value.
3. Developers can build responsive screens without creating duplicate files in 80% of cases.
4. Optional layout splitting works for tablet-specific UX.
5. Library is easy to adopt: <20 min setup, intuitive API.
6. Minimal runtime overhead (<1–2ms per render for breakpoint switching).

1. Deliverables
2. npm package: react-native-responsive-ui
3. Documentation / README:
4. Installation
5. Scaling & tokens
6. Responsive helper
7. Layout split examples
8. Best practices
9. Example project:
10. Mobile-first screen, scaled across devices
11. Screen with responsive helper
12. Screen with layout split