Spiral Take-Home Coding Challenge — Execution Plan

This document outlines a clean, production-oriented plan to implement the **Thirsty Drink Finder App**, aligned with Spiral's frontend standards and tech stack. It is intentionally structured to be copyable, extensible, and defensible during a lead engineer review.

---

1. Objectives & Evaluation Alignment

Primary goals - Demonstrate strong React + TypeScript fundamentals - Show pragmatic architectural decisions - Follow Spiral's frontend stack and best practices - Deliver a working, readable, and maintainable solution within scope

Secondary goals (extra credit) - Clear separation of concerns - Testability and composability - Thoughtful data normalization - Clean UI implementation using Chakra UI

---

2. Proposed Tech Stack (Spiral-Aligned)

This stack mirrors Spiral's production environment as closely as possible.

Core

- **Next.js (App Router)** — modern routing, server/client boundaries
- **TypeScript** — strict typing enabled
- **React 18**

UI & Styling

- **Chakra UI** — primary component system
- **Emotion** — Chakra's underlying styling engine

Data Fetching

- **@tanstack/react-query** — caching, loading states, request deduplication
- **Fetch API** — simple REST calls (no axios required)

Visualization

- **recharts** — lightweight, React-first pie chart support

Utilities

- **uuid** — stable keys where needed
- **clsx** — conditional class handling (minimal use)

Testing (Extra Credit)

- **Jest**
- **React Testing Library**

Tooling

- **ESLint + Prettier** — consistent formatting
- **Storybook** (optional) — component isolation

---

3. Project Initialization (Clean Template)

```
npx create-next-app thirsty --typescript --eslint --app
cd thirsty
```

Install dependencies

```
npm install @chakra-ui/react @emotion/react @emotion/styled
npm install @tanstack/react-query recharts uuid
npm install -D @testing-library/react @testing-library/jest-dom jest
```

---

4. Application Structure

```
src/
├── app/
│   ├── layout.tsx
│   ├── page.tsx                # Search screen
│   └── drinks/
│       └── [id]/page.tsx       # Details screen
│
├── components/
│   ├── SearchBar.tsx
│   ├── DrinkList.tsx
│   ├── DrinkListItem.tsx
│   ├── IngredientLegend.tsx
│   ├── IngredientsPieChart.tsx
│   └── LoadingState.tsx
│
├── hooks/
│   ├── useDrinkSearch.ts
│   └── useDrinkDetails.ts
│
├── lib/
│   ├── api.ts                 # API calls
│   ├── ingredientUtils.ts     # unit conversion, filtering
│   └── colorUtils.ts          # pastel color generation
│
├── types/
│   └── cocktail.ts
│
├── theme/
│   └── index.ts               # Chakra theme overrides
│
├── __tests__/
│   └── ingredientUtils.test.ts
│
└── README.md
```

---

5. Data Model & API Handling

API

- Source: `thecocktaildb.com`
- Endpoint:

```
/search.php?s={query}
```

Normalization Strategy

- Convert API's `strIngredientX` + `strMeasureX` fields into:

```
interface Ingredient {
  name: string;
  amount: number; // normalized unit
  unit: 'ml';
}
```

- Filter invalid or unsupported measurements early
- Centralize conversion logic in `ingredientUtils.ts`

---

6. Search Screen (Main Screen)

Behavior

- Search input triggers refetch on each keystroke
- Default placeholder: **"Find a drink"**

Components

- `SearchBar`
- `DrinkList`
- `DrinkListItem`

Implementation Notes

- Use `useQuery` with query key `["drinks", searchTerm]`
- Debounce input lightly (optional, but defensible)
- Fixed row height (60px) using Chakra `Flex`

---

7. Details Screen (Drink View)

Sections

1. Drink image + title
2. Ingredients legend (color-coded)
3. Pie chart (ingredient ratios)
4. Instructions (scrollable)

Pie Chart

- Only include ingredients with convertible units
- Normalize all values to **ml**
- Assign stable pastel colors per ingredient

Color Strategy

- Random pastel generator with deterministic seed (ingredient name)

---

8. State Management Approach

- **React Query** handles:
- Loading
- Caching

- Refetching

- Local component state for:

- Search input
- UI-only concerns

No global state library required.

---

9. Error & Edge Case Philosophy

- Follow instructions: no need for exhaustive error handling
- Gracefully handle:
- Empty results
- Partial ingredient data
- Missing images

---

10. Testing Strategy (Extra Credit)

Focus on **pure logic**, not UI snapshots.

Example

- Unit test `convertToMl()`
- Test unsupported units are excluded

---

11. README Checklist

README should include: - Project overview - Setup instructions - API reference - Architectural decisions - Trade-offs & assumptions - Deployed URL (if applicable)

---

12. Clarifying Questions to Ask Spiral

These demonstrate senior-level judgment:

1. Is the App Router preferred, or is Pages Router equally acceptable?
2. Should ingredient ratios be visually exact or approximate?
3. Is unit conversion accuracy more important than coverage?
4. Is client-side fetching preferred over server components for this exercise?
5. Are accessibility considerations part of evaluation?

---

13. Delivery Checklist

- [ ] App builds locally
- [ ] TypeScript strict mode enabled
- [ ] No console errors
- [ ] README complete
- [ ] Public GitHub repo
- [ ] (Optional) Deployed link

This plan prioritizes clarity, realism, and maintainability—closely mirroring how Spiral engineers ship production frontend features.