1. Structure générale et importations

1. **React + Hooks**
   - `useState` permet de déclarer des **états locaux** (variables réactives) :
     - `loading` (booléen) : si les données sont en cours de chargement
     - `error` (texte) : message d'erreur éventuel
     - `current` (objet) : météo courante
     - `dailyData` (tableau) : prévisions journalières
     - `searchModalVisible` (booléen) : affiche/masque la modal de recherche
     - `searchQuery` (texte) : valeur de l'input de recherche
   - `useEffect` lance une **action de côté** (ici `fetchData('Tizi-Ouzou')`) immédiatement après le montage du composant.
2. **Composants React Native**
   - `SafeAreaView` : gère les zones non utilisables (notches, barres iOS).
   - `StatusBar` : contrôle la couleur et le style de la barre système.
   - `View` : conteneur basique.
   - `Text` : affichage de texte.
   - `Image` : affiche une image à partir d'une URL ou d'un fichier local.
   - `ScrollView` : zone défilante (verticale ou horizontale).
   - `ActivityIndicator` : indicateur de chargement (spinner).
   - `Alert` : boîte de dialogue native pour informer l'utilisateur.
   - `TouchableOpacity` : zone cliquable avec retour visuel.
   - `Modal` : popup superposé à l'application.
   - `TextInput` : champ de saisie de texte.
3. **Icones**

- `MaterialCommunityIcons` (via Expo) fournit des icônes vectorielles (ici loupe pour la recherche).

---

2. Cycle de vie et récupération des données

1. **Initialisation**
   - À l'affichage initial du composant, `useEffect(..., [])` appelle `fetchData('Tizi-Ouzou')`.
   - On entre dans `fetchData`, on active le spinner (`setLoading(true)`) et on réinitialise l'erreur.
2. **Appels réseau**
   - **Météo courante**
     - Requête `GET /weather?q=...&units=metric&appid=API_KEY`
     - Si tout va bien, on extrait :
       - nom de la ville, icône météo, température, description, humidité, vent
     - On les stocke dans `current`.
   - **Prévisions 5 jours / 3 h**
     - Requête `GET /forecast?q=...&units=metric&appid=API_KEY`
     - Le serveur renvoie un tableau d'intervales de 3 heures.
     - On **filtre** pour ne conserver que les entrées à 12:00 (moment clé de la journée)
     - On garde les 7 premiers jours et on transforme chaque entrée pour ne garder que :
       - le jour (`Mon, Tue…` en français), l'icône et la température.
     - On stocke le résultat dans `dailyData`.
3. **Gestion des erreurs**
   - Si l'une des requêtes échoue (`!res.ok`), on intercepte l'erreur et on :
     - stocke le message dans `error`

        - affiche un popup Alert pour en informer l'utilisateur.
    - Dans tous les cas (`finally`), on désactive le spinner (`setLoading(false)`).

---

3. Recherche de ville et modal

1. **Ouverture/fermeture**
    - Le bouton loupe déclenche `setSearchModalVisible(true)` .
    - La modal recouvre l'écran par-dessus le contenu existant.
2. **Saisie et validation**
    - `TextInput` met à jour `searchQuery` à chaque frappe.
    - Appuyer sur "Rechercher" ou valider le clavier appelle `handleSearch()` .
    - Si le champ est vide (`.trim()`), on affiche une alerte pour demander un lieu.
    - Sinon :
        - on relance `fetchData` pour la nouvelle ville
        - on ferme la modal et on réinitialise l'input.

---

4. Rendu conditionnel

- **Si `loading` est vrai** → on affiche seulement le spinner centré.
- **Sinon si `error` est non nul** → on affiche le message d'erreur.
- **Sinon** → on affiche l'interface principale :
    1. **Header**
        - Bouton de recherche + titre "Météo".
    2. **Informations courantes**
        - Nom de la ville, icône, température, description, humidité et vent.
    3. **Prévisions 5 jours**

    - Scroll horizontal de cartes montrant jour/icône/température.

4. **Footer**
    - Date complète (jour de la semaine, jour, mois) et heure actuelle formatées en français.

---

5. Styles et apparence

- Utilisation d'un **StyleSheet** pour définir des styles réutilisables.
- Choix de couleurs pastel (bleu clair, vert d'eau) pour rappeler une ambiance météo.
- Ombrages et arrondis pour donner du relief aux cartes de prévision.
- Taille de police adaptée pour hiérarchiser les informations (32 px pour la ville, 48 px pour la température…).

En résumé

- L'app démarre avec un appel API pour charger la météo de **Tizi-Ouzou**.
- L'utilisateur peut rechercher n'importe quelle ville via la modal.
- Les hooks `useState` pilotent l'affichage dynamique, `useEffect` gère l'appel initial.
- La logique est découpée en deux phases de fetch (current + forecast), avec filtrage et transformation des données.
- Le rendu alterne entre état de chargement, message d'erreur, et affichage complet des données.