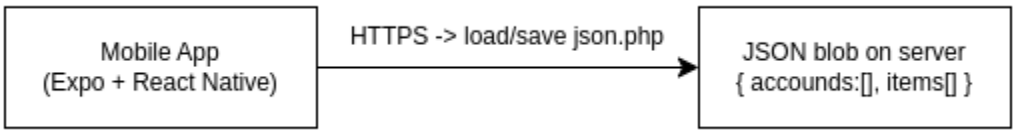
1. **Tech Stack**

Layer	Tooling	Why we chose it
Mobile UI	Expo (React Native)	Fast iteration, cross-platform (iOS + Android) without Xcode/Android Studio setup.
Local state / navigation	React Hooks + plain JS objects	Lightweight; no Redux needed for this size.
Server storage	Instructor-hosted PHP endpoints loadjson.php / savejson.php	Course-provided hosting; keeps everything in a single JSON file so no extra backend work.
Image & map utilities	expo-image-picker, react-native-maps	Expo-friendly libraries with minimal native configuration.

2. **Static Architecture**
    a. Single source of truth: one JSON file that holst accounts and items
    b. No separate backend code: the same PHP endpoints read/overwrite the blob whenever the app calls loadAll() or saveAll()



3. **Data Model**
    a. Item ownership and contact are embedded in each item so the detail screen never needs to re-fetch user data


```
{
  "accounts": [
    { "username":"u1", "password":"p1", "phone":"1112223333" }
  ],
  "items": [
    {
      "title":"Wallet",
      "description":"Black",
      "type":"lost",
      "latitude":43.60,
```

```
    "longitude":-116.20,
    "location":"SU",
    "imageUri":"file:file",
    "owner":"u1"
    "phone":"1112223333"
   }
  ]
}
```


4. **Runtime Flow**
    a. When the app launches, App.js immediately calls loadAll to fill the in-memory dataObj, and a loaded flag ensures the app never overwrites the server file with an empty object during this first fetch. Sign-up or login occurs in AuthScreen, which checks the entered credentials against dataObj.accounts; if they match, the user is stored in currentUser and the full data blob is returned to App.js. Whenever a user creates a post, AddItemScreen adds a complete item, already containing its owner and phone number, to dataObj.items, and a watching effect then triggers saveAll to persist the change. In the browsing views, LostScreen and FoundScreen simply filter that in-memory array; their FlatList components handle scrolling, and a long-press on a card deletes it only if the current user owns the item. Tapping a card opens ItemDetailScreen, which shows a header such as “Wallet lost by u1” (or “found by u1”). If the viewer is not the owner, a Contact u1 button appears; if they are, they see Delete Post instead. Finally, the Home screen’s red Log Out button clears currentUser and returns the app to AuthScreen for the next sign-in.
5. **Key Design Decisions**
    a. Single JSON blob instead of REST API
    Pros: zero backend code, trivial deploy.
    Cons: potential race conditions—mitigated by saving only after initial load and by the app being single-user during testing.
    b. Owner-only mutations entirely on the client.
    Because every item stores its owner, the client can enforce delete rights without extra server logic.
    c. Minimal state library
    React Hooks (useState, useEffect) handle all state; navigation is a simple string so no React Navigation dependency.