During the implementation phase, several environment and dependency-related issues arose as a result of conflicting package versions and misconfigurations within the Metro bundler and package manager. These issues, though not critical to functionality, had the potential to cause runtime instability, type conflicts, and degraded developer experience if left unresolved.

**1. Improper TypeScript Dependency (@types/react-native)**

One of the detected issues involved the manual installation of the @types/react-native package, which is unnecessary and potentially problematic in projects using the react-native core package. Since type definitions for React Native are already bundled with the framework itself, this duplication introduced redundant and conflicting type information.

*Resolution*: The issue was resolved by removing the @types/react-native package from the project dependencies using npm uninstall, restoring the integrity of the type system and eliminating warnings during linting and build processes.

**2. Missing Native Peer Dependencies for Navigation**

The application initially failed to include required peer dependencies for the @react-navigation/native-stack library, specifically:

* react-native-safe-area-context
* react-native-screens

These packages are essential for safe rendering across devices with notches or display cutouts and for efficient screen management within stack navigators. Their absence can lead to crashes or degraded UI behaviour, particularly on physical devices outside the Expo Go environment.

*Resolution*: These dependencies were installed using the npx expo install command, which ensures compatibility with the Expo SDK in use. This approach aligns with Expo's best practices and maintains stability across platforms.

**3. Metro Bundler Configuration Conflict**

Another challenge involved an incorrectly customized Metro bundler configuration file (metro.config.js), in which the getDefaultConfig function was inadvertently declared multiple times. This led to runtime errors when attempting to start or validate the project.

*Resolution*: The redundant declaration was removed, and the Metro configuration was updated to extend Expo’s recommended configuration while enabling workspace-level folder watching. This ensured the bundler could resolve dependencies correctly without introducing instability.

**Reflection on Best Practices**

These issues highlight the importance of:

* Adhering to **platform-specific dependency management recommendations** (e.g., Expo’s expo install)
* Avoiding **unnecessary type packages** that duplicate core functionality
* Carefully managing **Metro and project configurations** to prevent conflicts

Had these issues been left unaddressed, they could have led to:

* Unreliable builds or runtime crashes
* Difficulty in debugging during development
* Poor user experience across devices

Proactively resolving them not only stabilized the project but also ensured alignment with industry best practices for maintainable and scalable React Native applications.