

# Risk Documentation

## Interactive House – Subgroup 2: Units

### Revision History

Date	Version	Description	Author
2026-02-06	1.0	Initial risk assessment for the Units development subgroup.	Daniel Jönsson
2026-02-18	1.1	Add Risks for Mobile development (R6-R10)	Daniel Jönsson

Risk Description	Priority
R1. Inability to establish stable communication between the mobile/web units and the house server.	High
R2. Difficulty in dynamically rendering User Interfaces (UIs) sent from the server for new devices.	High
R3. Team members failing to complete assigned UI components or speech modules within the iterative cycle.	Medium
R4. Compatibility issues across different mobile platforms	Medium
R5. Inconsistent UI Scaling across Different Display Formats	Medium
R6. Mobile Native module incompatibility within the Expo Go sandbox environment	High
R7. Mobile Significant performance degradation during complex UI re-renders	Medium
R8. Mobile breaking changes during mandatory Expo SDK version upgrades	High

R9. Mobile Over-the-Air update mismatches between JS and Native code	Medium
R10. Mobile Latency and bottlenecks in the React Native Bridge for real-time data	Medium

## 1. General

### **R1. Inability to establish stable communication between the mobile/web units and the house server.**

#### **Impact**

If the mobile phone or web application cannot communicate with the server, users will be unable to observe or control any house devices. This effectively breaks the core purpose of the unit subgroup.

#### **Indications**

Unit test code for server requests returns timeouts or connection errors; the unit fails to receive the initial list of available devices from the server.

#### **Mitigation Strategy**

Establish a clear communication protocol/interface with the Server Subgroup early in the design phase. Develop a Mock Server for early unit testing to ensure the UI can handle data once the real server is ready.

### **R2. Difficulty in dynamically rendering User Interfaces (UIs) sent from the server for new devices.**

#### **Impact**

The system will fail to be scalable; new devices like a coffee machine or media player will not be controllable if the unit cannot upload and display their specific UI components.

#### **Indications**

The unit connects to the server but fails to display the on/off buttons or readouts for certain device types.

#### **Mitigation Strategy**

Standardize the UI distribution format like software components for touch screen GUIs across all subgroups. Ensure the server database correctly stores and pushes these UI components to the unit upon request.

### **R3. Team members failing to complete assigned UI components or speech modules within the iterative cycle**

#### **Impact**

Delays in the unit subgroup increase the workload for other members and hinder integration tests during the iterative RUP-based process.

#### **Indications**

A developer never presents finished code during formal meetings or provides vague updates about progress.

#### **Mitigation Strategy**

Clearly define individual tasks and ensure everyone knows their specific responsibilities. Encourage a team-based culture where members ask for help early to solve technical blockers.

### **R4. Compatibility issues across different mobile platforms**

#### **Impact**

The application may fail to support users with disabilities if it relies solely on touch screens that are unavailable or incompatible with certain hardware.

#### **Indications**

The GUI component is slightly different or non-functional on older phones or specific browser versions.

#### **Mitigation Strategy**

Incorporate alternative interaction techniques like speech recognition and gestures as core requirements early in the project. Use common frameworks like Android or web-based interfaces to ensure a more uniform communication model.

### **R5. Inconsistent UI Scaling across Different Display Formats**

#### **Impact**

Degraded user experience (UX) where buttons may be too small or text overlaps, though core system control remains functional.

#### **Indications**

Visual artifacts or layout distortions appear when switching between small mobile phone screens and larger laptop web interfaces.

#### **Mitigation Strategy**

Strive for context-dependent ways of communicating and utilizing responsive design patterns for all UI components. Test UI rendering on various screen resolutions (laptop, tablet, and phone) during the prototype phase.

## **2. Mobile (React Native + Expo)**

### **R6. Mobile Native module incompatibility within the Expo Go sandbox environment**

**Impact**

Development may stall if a required feature like specific Bluetooth low energy protocols for house devices is not supported by the default Expo Go app.

**Indications**

A third-party library fails to link, or an error stating "Native module cannot be null" appears during testing.

**Mitigation Strategy:** Identify all hardware/native requirements early. If a library is unsupported by Expo Go, immediately transition to Development Builds using expo-dev-client to include custom native code.

**R7. Mobile Significant performance degradation during complex UI re-renders****Impact**

The UI becomes sluggish or unresponsive when many house devices are updated simultaneously, leading to dropped frames (stuttering).

**Indications**

Noticeable delay in button response; "JS thread" frame rate drops below 60fps in the React Native Debugger.

**Mitigation Strategy**

Use React.memo or PureComponent to prevent unnecessary re-renders of device components. Implement FlatList with getItemLayout for efficient rendering of large device lists.

**R8. Mobile breaking changes during mandatory Expo SDK version upgrades****Impact**

As Expo deprecates older SDK versions, the team may be forced to upgrade, which can break existing navigation or UI libraries.

**Indications**

Build errors or "deprecated" warnings can appear in the console after a new SDK release.

**Mitigation Strategy**

Perform incremental upgrades to one version at a time. Use npx expo-doctor after every upgrade to identify and fix dependency conflicts automatically.

**R9. Over-the-Air update mismatches between JS and Native code****Impact**

Pushing a JavaScript update via EAS Update that requires new native code (not yet installed on the user's device) will cause the application to crash.

**Indications**

The app opens but immediately crashes or displays a white screen after an OTA update is downloaded.

**Mitigation Strategy**

Use Runtime Versions in app.json to ensure OTA updates are only delivered to native builds that can support them.

**R10. Mobile Latency and bottlenecks in the React Native Bridge for real-time data****Impact**

High-frequency data can clog the bridge between JavaScript and Native layers, causing the entire UI to freeze.

**Indications**

Real-time data updates appear jumpy or stop entirely when the user interacts with other UI elements.

**Mitigation Strategy**

Offload intensive logic to the native side using Native Modules or utilize libraries like Reanimated that run animations/logic directly on the UI thread, bypassing the bridge.