

School of Electronic Engineering and  
Computer Science

## **Final Report**

### **Programme of study:**

Degree of Bachelor of Electronic  
Engineering

### **Project Title:**

**Aware Phone Interface**

### **Supervisor:**

Vasileios Klimis

### **Student Name:**

Catriona Leila Anderson Borazjani  
Gholami

Final Year

Undergraduate Project 2024/25



Date: 06/05/2025

## ABSTRACT

This project presents the design and development of *The Aware Phone Interface*, a smartphone system feature that supports safer device use during intoxicated states. Aimed at reducing accidental communication, improving decision-making and enhancing safety, the prototype introduces a context-aware “*Drunk Mode*” with features such as auto-correction for drunk texting, app and contact restrictions, health recommendations, and activity tracking.

Based on inclusive UX principles and supported by theories of distributed cognition and phenomenology, the interface adapts dynamically to user’s cognitive and motor limitations. The system was implemented using React Native, Supabase, and Expo Go, with an MVVM architecture to maintain separation of concerns. Testing involved ongoing feature testing throughout development, alongside heuristic usability evaluation and informal user feedback.

The outcome is a working prototype that shows how built-in phone features could better support users when they’re intoxicated or otherwise impaired, offering a meaningful contribution to the wider discussion on inclusive and responsible design.

## Table of Contents

<i>Introduction.....</i>	<i>6</i>
1.1 Background of Project.....	6
1.2 Problem Statement .....	6
1.3 Project Aim.....	7
1.4 Objectives.....	7
<i>Background Research .....</i>	<i>8</i>
2.1 Literature Review .....	8
2.2 Existing Tools and Apps .....	10
2.3 Identified Gaps .....	10
<i>System Requirements .....</i>	<i>12</i>
3.1 Requirements Analysis.....	12
3.2 Proposed Solution.....	13
3.3 Functional Requirements.....	14
3.4 Non-Functional Requirements .....	15
3.5 Software & Hardware Requirements .....	16
3.6 Key Feature List .....	16
<i>Design .....</i>	<i>18</i>
4.1 System Flow Diagrams .....	18
4.2 Component Diagram .....	22
4.3 State Diagram.....	23
4.4 System Architecture .....	24
4.5 Design Pattern .....	25
4.6 Design Principles.....	25
4.7 Justification for System-Level Integration .....	26
4.8 Wireframes .....	27
4.9 Summary of Key Design Objectives.....	28
<i>Implementation .....</i>	<i>29</i>
5.1 Development Environment.....	29
5.2 Architecture and MVVM Design Pattern.....	30
5.3 Feature Implementation.....	32
5.4 Supabase Integration .....	44
5.5 Theoretical Grounding in Design .....	45
<i>Testing.....</i>	<i>47</i>
6.1 Testing Approach .....	47
6.2 Informal Usability Feedback.....	47
6.3 Feature Testing.....	48

6.4 Testing Tools, Environments & Bug Handling .....	50
<i>Evaluation</i> .....	51
7.1 Heuristic Evaluation .....	51
7.2 Strengths and Achievements .....	54
7.3 Limitations.....	54
7.4 Ethic, Legal, Social, and Sustainability Considerations.....	55
<i>Conclusion</i> .....	56
8.1 Learning and Achievements .....	56
8.2 Challenges Faced.....	57
8.3 Future Work .....	58
8.4 Summary .....	58
<i>Appendix</i> .....	60
Appendix A – Existing Systems Comparison Table:.....	60
Appendix B – User Survey and Results: .....	60
Appendix C – Implementation – Code and Screenshots:.....	64
Appendix D – Anonymous User Testing Results .....	65
Appendix E – Risk Assessment Analysis.....	65
<i>References</i> .....	67

## List of Figures

Figure 1: Drunk Mode System Flow Diagram .....	14
Figure 2 : Set Up Flow Diagram.....	19
Figure 3: Active Drunk Mode Flow Diagram.....	20
Figure 4: Exit Drunk Mode Flow Diagram.....	21
Figure 5: Component Diagram .....	22
Figure 6: Drunk Mode State Diagram .....	23
Figure 7: System Architecture .....	24
Figure 8: Model-ViewModel-Model Diagram .....	25
Figure 10: Drunk Mode Settings Page.....	27
Figure 9: Normal Home Screen UI.....	27
Figure 11: Drunk Mode Home Screen UI.....	28
Figure 12: Activity Overview Modal.....	28
Figure 13: Drunk Mode on and Off Toggle.....	33
Figure 15: Context-Aware Prompt.....	33
Figure 14: Drunk Mode Deactivation Prompt .....	33

Figure 17: Interface for setting App Restrictions .....	35
Figure 16: Interface for Setting Contact Restrictions .....	35
Figure 18: Interface for Setting Notification Restrictions .....	35
Figure 19: Locked Drunk Mode Settings Prompt.....	36
Figure 20: Alert Setting Screen.....	37
Figure 21: Health Recommendation Screen .....	38
Figure 22: BAC Estimator Modal.....	39
Figure 23: Drunk Mode Autocorrect .....	40
Figure 24: Drunk Mode Activated Home Screen .....	41
Figure 25: Normal Home Screen .....	41
Figure 26: Confirmation Prompt.....	41
Figure 27: Quick Access Buttons .....	41
Figure 28: Safety Settings Screen .....	42
Figure 31: Activity Overview Modal.....	43
Figure 30: Past-Session Activity Data .....	43
Figure 29: Activity Overview Screen .....	43
Figure 32: Supabase Database Schema.....	44

### List Of Tables

Table 1: Functional Requirements .....	15
Table 2: Non-Functional Requirements .....	15
Table 3: Feature Testing Results Table .....	50
Table 4: Heuristic Scoring and Observations Table .....	53

## Chapter: 1

# Introduction

---

### 1.1 Background of Project

Intoxication impairs motor skills, attention, memory, and decision-making, significantly reducing the ability to safely and effectively use smartphones (Fillmore, M.T. 2003). Despite the prevalence of mobile phone use while intoxicated – particularly in nightlife or social events – standard interfaces are not designed to accommodate users in these impaired states. As a result, users may experience confusion, make accidental communication (Trub, L. *et al.* 2021), struggle with emergency actions, compromise their safety and take risks (Fillmore, M.T. 2003).

These interactions can lead to social embarrassment, financial loss, or even physical danger (Steele and Josephs, 1990). While digital wellbeing features exist, they are rarely made for intoxicated users (Trub and Starks, 2017). The lack of system-level support for impaired users highlights a critical gap in inclusive interface design (Clarkson et al., 2003). An adaptive, context-aware interface (such as a customisable “Drunk Mode”) could reduce risks by aligning with users’ cognitive limitations when under the influence, providing safety features and helping users stay in control without making the interface harder to use.

### 1.2 Problem Statement

There is currently a lack of smartphone interfaces designed to accommodate the cognitive and motor impairments associated with intoxication (Trub and Starks, 2017). Existing third-party apps offer limited solutions, often focusing on specific actions such as blocking texts or apps, without integrating across the phone’s core functions.

This project addresses that gap by developing a customisable “Drunk Mode” system interface, which demonstrates how a system-wide feature could support users in impaired states. The prototype showcases core functionalities – such as user interface (UI) adjustments, safety prompts, app and contact restrictions, and activity logging – to demonstrate how a built-in focus mode (Apple, n.d) could reduce harm and improve usability during intoxicated phone use.

### 1.3 Project Aim

The aim of this project is to design and prototype a customisable smartphone interface that supports users when intoxicated. By demonstrating how a “Drunk Mode” focus setting could be integrated into smartphone operating systems, the project seeks to reduce user error, improve safety, and enhance usability. The design applies inclusive UX principles to address common impairments caused by intoxication, such as reduced vision, impaired decision-making, and motor difficulties (Fillmore, 2003). This project also aims to contribute to the broader discourse on inclusive and context-aware interface design.

### 1.4 Objectives

1. Design a context-aware smartphone interface tailored for intoxicated users.
2. Prototype key functionality including a customisable “Drunk Mode” with restricted access, simplified UI, activity tracking, safety features, and error prevention.
3. Implement algorithms for auto-correction, health recommendations, and context-based activation prompts.
4. Apply inclusive UX and HCI principles.
5. Test and evaluate usability and technical performance through heuristic evaluation, feature testing, and informal user testing.
6. Demonstrate how the system could be integrated as a built-in smartphone feature, rather than a third-party app.

## Chapter: 2

### Background Research

---

This literature review examines three key studies that inform different aspects of the project. The first explores the cognitive and visual impairments associated with alcohol consumption, offering insight into the user's physical experience while intoxicated. The second focuses on inclusive design for individuals with cognitive decline, highlighting principles relevant to accessibility and usability and their importance. The third presents a case study on the development of an app for intoxicated users, providing practical context and identifying current design limitations.

#### 2.1 Literature Review

##### 2.1.1 The Effects of Alcohol on Visual Attention and Memory

Alcohol consumption impairs cognitive processing, particularly in relation to attention and memory. Steele and Josephs (1990) introduced the concept of alcohol myopia, describing how intoxicated individuals focus on central stimuli while neglecting peripheral details – a phenomenon with significant behavioural and safety implications.

Harvey, Kneller, and Campbell (2013) investigated this effect by comparing eye movements and recall between intoxicated and sober participants while viewing visual scenes. Results showed that intoxicated individuals focused significantly on more central areas of the image, regardless of the emotional impact of the scene, and had reduced memory recall for both central and peripheral details compared to sober participants. This provided strong support for the alcohol myopia theory, suggesting that intoxicated users visually process less information overall.

While the study was well designed, the authors noted limitations, such as fixed image position and the absence of a placebo group. They recommended that future research would use more advance eye-tracking methods to assess covert attention.

These findings are directly relevant to interface design, particularly for mobile devices. If users under the influence are visually focused on the centre of the screen, peripheral navigation or small UI elements may be overlooked. This supports the need for centrally



focused, simplified UI design, large touch targets, and minimal peripheral distractions in systems designed for intoxicated users.

### **2.1.2 Inclusive UX Design for Cognitive Decline**

Designing accessible technology requires consideration for users with varying cognitive abilities, including those experiencing cognitive decline. While inclusive design is widely applied in sectors like healthcare and education, cognitive accessibility remains underexplored in mainstream UX practice.

Pokinko (2015) addressed this gap in a major research project aimed at improving mobile UX for older adults with cognitive decline. The study produced a practical guide of inclusive design considerations through literature review, expert feedback, and content analysis. The main recommendations were simplifying layouts, providing meaningful feedback, enabling personalisation, and designing with trust-building in mind.

Though the study was limited by its small expert panel and lack of real-world user testing, it highlights a critical shift in UX – towards creating “living documents” that evolve with both technology and user needs.

These principles strongly inform this project’s design, where users may be less aware due to intoxication. Features including easy navigation, clear prompts, big buttons, and personalised restrictions follow Pokinko’s (2015) ideas around making interfaces feel safe and easy to use, especially helpful when users might be less focused due to drinking.

### **2.1.3 Wingman: A Companion App for Intoxicated Users**

Srivastava et al., (2021) developed *Wingman*, a mobile application paired with a Breathalyzer, designed to detect alcohol intoxication and trigger a “Drunk Mode” that restricts access to risky phone functions. The app aimed to support young users during nights out by enabling safe transport booking, limiting interactions, and connecting them with emergency contacts.

The research identified a clear gap in the market: most existing solutions focused on tracking drinking behaviour instead of integrated, safety-focused interventions. Through user research, the team uncovered key risks users face while intoxicated, including poor judgement, unsafe

social interactions, and limited control over phone use. Their findings support this project's goal of using technology to help users with impaired thinking and behaviour.

However, usability testing revealed significant drawbacks. Users found Wingman's interface unintuitive and overly controlling. Some doubted its practicality, especially as it relied on the external breathalyser and the need for users to actively use the app while intoxicated. The system also lacked system-level integration, limiting its ability to restrict core functions. While Wingman highlights the demand for tools that assist intoxicated users, this project builds on those findings by focusing on a system-integrated, customisable interface designed to simplify user experience and support independence during intoxicated states.

## 2.2 Existing Tools and Apps

Several mobile applications aim to reduce the risks of using a phone while intoxicated, including *Drunk Mode*, *Drunk Locker*, and *Drunk Mode Locker*. These apps typically offer features like app or text blocking, location tracking, or puzzle challenges that prevent usage while drunk. However, all current features are third-party apps, meaning they cannot fully integrate with system-level functions such as messaging or calling.

Most use manual activation, which can be impractical when intoxicated if there is no alternative way of activating the system. They offer limited accessibility, with small buttons, complex interfaces, or confirmation puzzles that can be confusing for users. Additionally, they are often easy to bypass by simply uninstalling the app or closing the app in the background.

While these tools demonstrate demand, they highlight a key flaw: the lack of seamless, inclusive, and system-integrated solutions. This project addresses that gap by proposing a built-in, operating system-level interface that adapts the phone's core functionality to support safe and easy to use.

*(See Appendix A for a full comparison of existing systems and their limitations)*

## 2.3 Identified Gaps

Current research and tools demonstrate the challenges intoxicated users face but lack a fully inclusive, system-level solution. Existing apps often lack accessibility features, are overly

complex or controlling, and can be easily bypassed. Many rely on external hardware, which may not be practical for intoxicated users. Importantly, most tools cannot integrate with core phone functions like calling or messaging, limiting their impact. There remains a gap for an interface that is natively integrated, designed for temporary cognitive and motor impairments, and focused on safety, usability, and independence.

## Chapter: 3

# System Requirements

Before drafting the systems requirements, it was important to understand real-life behaviour when users are intoxicated and using their phones. This section brings together insights from the literature review and a user questionnaire to shape the prototype requirements so that it feels genuinely helpful. The requirements offer a clear structure to build from, making sure the final interface actually matches what users need.

### 3.1 Requirements Analysis

#### 3.1.1 Literature Review Insights

Insights from the literature review were used to guide the system's design and features. Harvey et al. (2013) highlighted that intoxicated users have limited peripheral attention and memory recall, supporting the need for a simplified layout with clear central focus and minimal distractions. Pokinko (2015) emphasised the value of inclusive, confidence-building UX design, therefore informing the use of clear feedback, error prevention (e.g., confirmation prompts), and larger interactive elements. Srivastava et al., (2021) demonstrated the importance of practical features like emergency access and transport help, while also revealing the drawbacks of overly restrictive and hardware-dependent systems. Together, these studies helped shape the system's core requirements, especially in relation to impaired cognitive and motor conditions.

#### 3.1.2 User Questionnaire Insights

To support the literature and ensure the system meets actual user needs, an online questionnaire was conducted with 32 participants. The survey investigated patterns of phone usage while intoxicated, common challenges of safety concerns, and preferences for potential features.

##### Key findings:

1. **High Usage While Intoxicated:** 84.4% of respondents reported using their phone "always" or "often" while drunk (Appendix B, Figure 1).
2. **Common Activities:** The most frequent actions included texting (71.9%), taking photos/videos (84.4%), using social media (62.5%), and booking transport (56.3%) (Appendix B, Figure 2).

3. **User Errors:** 83.3% reported texting or calling the wrong person, and nearly half reported posting on social media unintentionally or struggling with transport booking (Appendix B, Figure 3).
4. **Emotional Frustrations:** 68.8% of participants reported feeling confused or frustrated when using their phones while intoxicated (Appendix B, Figure 4).
5. **Safety Concerns:** A significant portion expressed concern over potential consequences of drunk phone use (e.g., safety risks or financial loss) (Appendix B, Figure 5), with some reporting being unable to complete essential tasks like calling someone or booking a ride (Appendix B, Figure 6).
6. **Strong Interest in Supportive Features:** 68.8% were interested in a Drunk Mode setting, and 84.4% wanted supportive notifications and reminders (Appendix B, Figures 7-8). Preferred features included a simplified UI, emergency contact access, app restrictions, and usage summaries (Appendix B, Figure 9)

Participants also suggested new features including:

- A “confirmation before action” prompt before making sensitive decisions.
- Location-sharing with specific contacts during intoxication

These responses demonstrate a strong user need for system-level assistance that reduces errors, supports safety, and respects user independence.

### 3.2 Proposed Solution

This project proposes **Drunk Mode**, a built-in smartphone feature designed to support users while intoxicated. Before use, users set up preferences such as app and contact restrictions, notifications, emergency options, and health recommendations.

When activated, Drunk Mode enforces the user’s selected preferences and applies system-defined changes: it restricts access, simplifies the UI, enables safety features, sends relevant alerts, and other supportive adjustments. Once deactivated, system restores access and shows a summary of the user’s activity during the session.

Figure 1 shows how the system handles setup, activation, and exit.

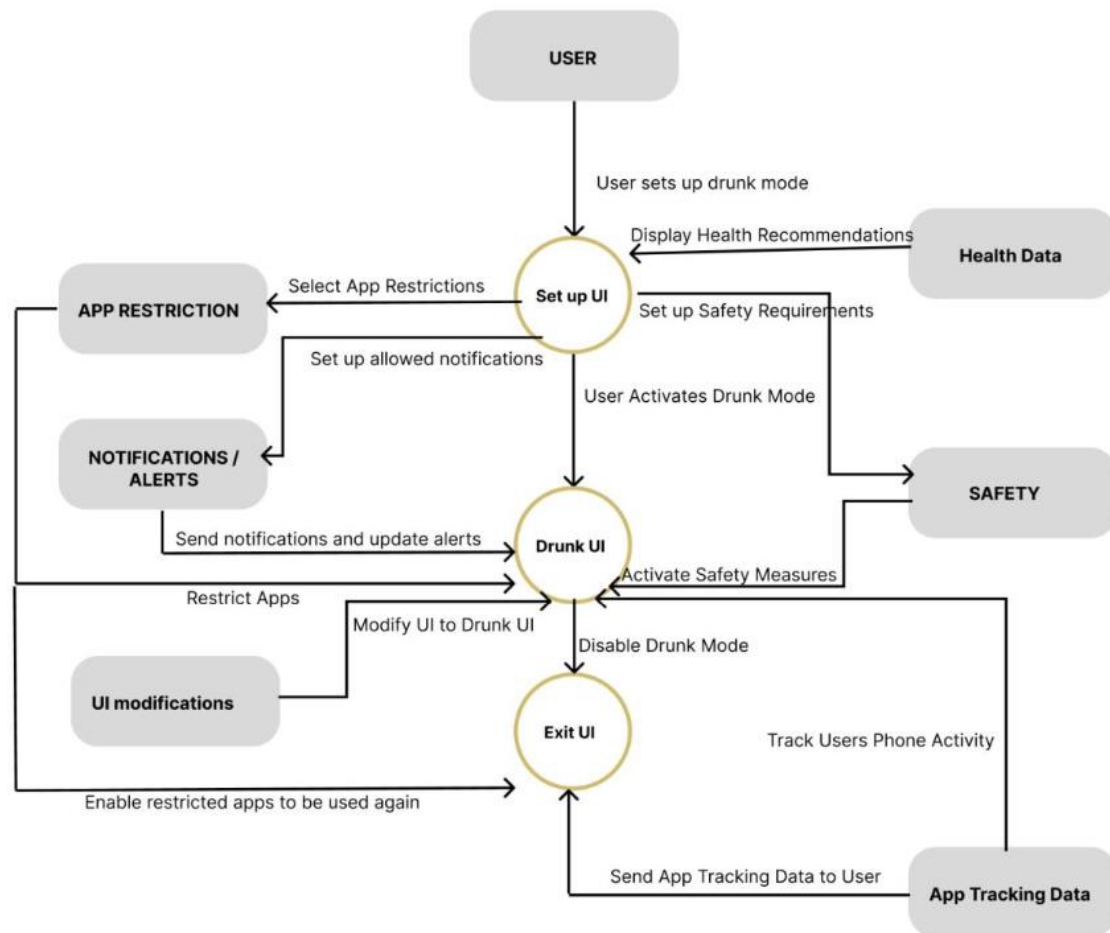


Figure 1: Drunk Mode System Flow Diagram

### 3.3 Functional Requirements

#	Requirement
1	<b>Drunk Mode Activation:</b> Users must be able to manually toggle Drunk Mode, with the system also prompting activation based on context (e.g., time, movement, typos).
2	<b>Simplified Interface:</b> When active, the interface must shift to a simplified layout with large visual elements, minimal peripheral distractions, and easy navigation.
3	<b>App and Contact Restrictions:</b> Users must be able to pre-select specific apps/contacts to be hidden or disabled during Drunk Mode (e.g. messaging, social media).
4	<b>Error Prevention:</b> The system must include confirmation prompts before sensitive actions (e.g. emergency calls) to reduce errors.
5	<b>Emergency Contact Access:</b> Users must be able to access emergency contacts or safety actions quickly and easily when Drunk Mode is active.

6	<b>Activity Overview:</b> After exiting Drunk Mode, the system must display a summary of user activity during the session.
7	<b>Customisable Setup:</b> Users must be able to configure Drunk Mode during setup, including preferences for restrictions, emergency contacts, personalised alerts, and designated drivers.
8	<b>Reminders &amp; Notifications:</b> The system must deliver useful alerts during Drunk Mode, such as reminders to drink water and check for their wallet.

Table 1: Functional Requirements

### 3.4 Non-Functional Requirements

#	Requirement
1	<b>Usability:</b> The system must be intuitive and simple to navigate, with a layout that remains easy to understand even in a cognitively impaired state.
2	<b>Reliability:</b> The system must function consistently, including in high-risk context where the user is intoxicated and carrying out essential tasks.
3	<b>Accessibility:</b> The system must meet accessibility standards to suit cognitively impaired users (e.g., large text, reduced steps, cognitive offloading)
4	<b>Scalability:</b> The architecture should support future expansion, including potential integration with OS-systems and external services such as Uber or health tracking tools.
5	<b>Maintainability:</b> The codebase must be modular and easy to update or debug to support long-term development and testing.
6	<b>User Satisfaction:</b> The system must provide a supportive, trustworthy experience to reduce frustration and encourage repeated use in future intoxicated states.

Table 2: Non-Functional Requirements

## 3.5 Software & Hardware Requirements

### Hardware Requirements

- iPhone: Used to test the Drunk Mode prototype in a realistic mobile context.
- Mac: Required for development, code compilation, and native device testing via Expo Go
- Internet Access: Necessary for downloading dependencies, syncing with Supabase, and previewing the app on physical devices.

### Software Requirements

- Figma: Designed low-fidelity wireframes of the Drunk Mode interface.
- React Native: Core framework used to build the cross-platform mobile prototype.
- Node.js & npm: Enabled project setup, dependency installation, and script execution.
- Expo: Simplified development and testing of the app on real devices; used for live preview and access to native APIs (e.g. notifications, location.)
- Supabase: Provide real-time backend functionality, including data storage for user settings, alerts, and session logs.
- JavaScript: Primary programming language used to write logic and UI components.

## 3.6 Key Feature List

These features are based on user needs identified through research and follow core usability and feedback principles outlined in Norman's (2013) design framework.

1. Customisable Setup: Users can configure their preferences before enabling Drunk Mode – including restricted apps and contacts, safety options, designated drivers, and alerts.
2. Health Recommendations: Based on entered weight and drink count, the system estimates Body Mass Index (BMI), calculates Blood Alcohol Content (BAC), and suggests safe alcohol limits.
3. Drunk Mode Activation: A toggle enables Drunk Mode, which applies user preferences and system-defined changes.
4. Simplified Interface (UI): The phone UI transforms to support impaired cognition and motor control: enlarged buttons and text, intuitive colour palette, reduced clutter.
5. Access Restriction: Selected apps and contacts are hidden or blocked during Drunk Mode to reduce risk.



6. Alerts & Notifications: The system sends reminders during intoxication (e.g. to drink water, check for wallet), and users can customise alerts with their own text.
7. Safety Features: Quick actions including call 999, contact a designated driver, send live location to an emergency contact.
8. Activity Logging: While active, the system calls, messages, and notifications. Upon exit, a summary is shown to the user for review and users can review past Drunk Mode session activity.
9. Typo Detection & Auto-Correction: When Drunk Mode is active, the system detects and auto-corrects common typing errors, sending a notification to let the user know.
10. Context-Aware-Detection Algorithm: A background scoring system checks for signs of intoxication (e.g. late-night usage, typos, phone shaking) and prompts the users to turn on Drunk Mode if needed.

## Chapter: 4

### Design

---

This chapter outlines the design of the proposed Drunk Mode interface, including the user experience flow, component structure, and system architecture. The system was designed with impaired cognitive and motor ability in mind, using inclusive design principles and the MVVM pattern to separate logic from UI. The diagrams in this section demonstrate how users interact with the interface, how data is structured and flows through the system, and how key modules work together to deliver a responsive, safe experience.

#### 4.1 System Flow Diagrams

The following diagrams illustrate the internal flow of the system across its three primary states: Set-Up, Drunk Mode, and Exit. Instead of focusing on user navigation, these charts represent how data, logic, and system components interact at each state. They outline how user preferences are configured, how each part of the system functions during an active Drunk Mode session, and how the session concludes with an after-session summary. These diagrams give a clear overview of how the system works, showing its structure and how it behaves.

### 4.1.1 Set-Up Flow Diagram

This diagram shows how users configure Drunk Mode by setting restrictions, safety settings, alerts, and health data. Each step customises system behaviour, storing preferences in the database for retrieval when Drunk Mode activates. This setup saves user choices in advance so Drunk Mode can apply them automatically when needed.

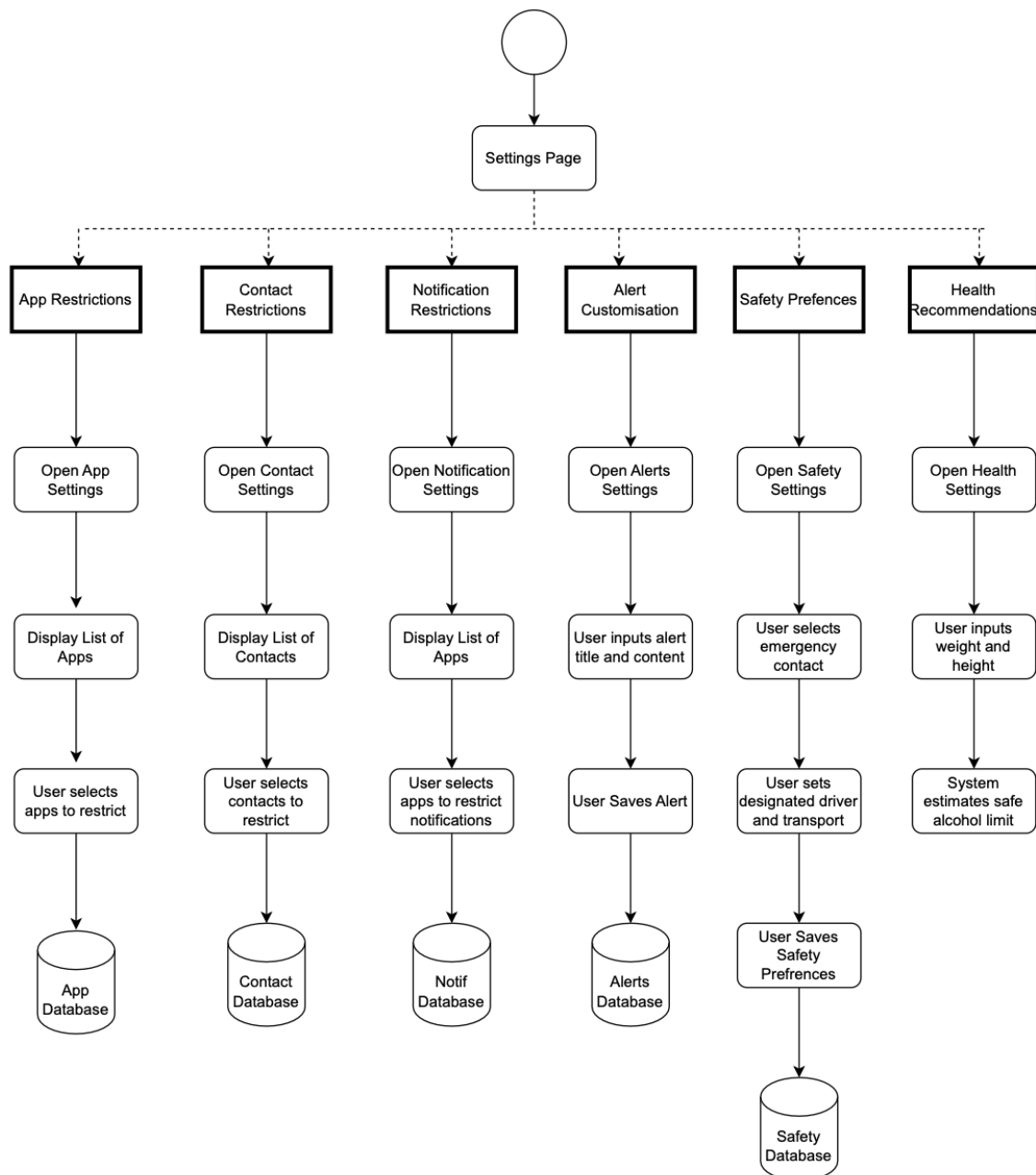


Figure 2 : Set Up Flow Diagram

### 4.1.2 Active Drunk Mode Flow Diagram

The diagram below shows what happens in the system when Drunk Mode is activated. It maps out how different components handle user preferences and pre-set logic – including tracking, alerts, travel safety, app restrictions, interface adjustments, assistive features, and a BAC calculator. Each module works together to support the user in a safer, more intuitive way during intoxicated phone use.

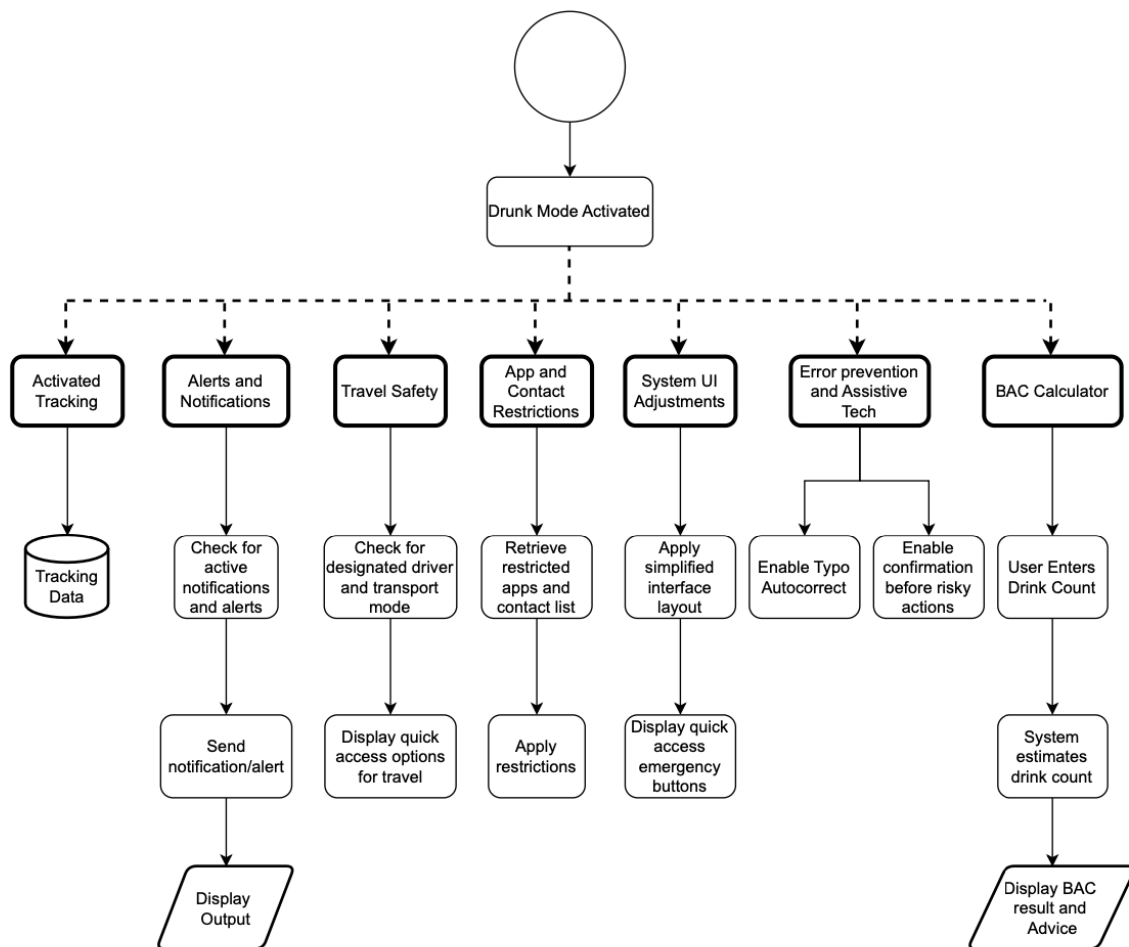


Figure 3: Active Drunk Mode Flow Diagram

### 4.1.2 Exit Drunk Mode Flow Diagram

This diagram illustrates the process followed once Drunk Mode is turned off. The system restores the original phone interface, accesses the tracking data collected during the session, and generates an overview summary for the users. The diagram shows how data is captured and processed across the last stage of the session.

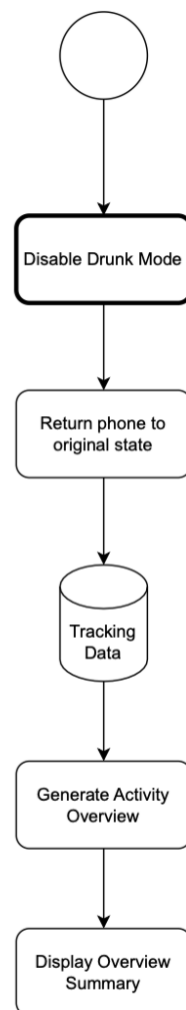


Figure 4: Exit Drunk Mode Flow Diagram

## 4.2 Component Diagram

This diagram breaks down the system into key components, highlighting how the user interfaces, core logic modules, and data storage layers interact. It visually represents the architecture behind Drunk Mode, including services for tracking, notifications, UI modification, BAC calculation, and error handling.

The core system handles the main logic, and coordinates between different modules, while external UI components allow the user to configure and interact with Drunk Mode. All data, such as user preferences, health info, and activity tracking, is stored in corresponding databases or local storage.

Dashed arrows represent dependencies between components, indicating how modules communicate, trigger actions, or exchange data.

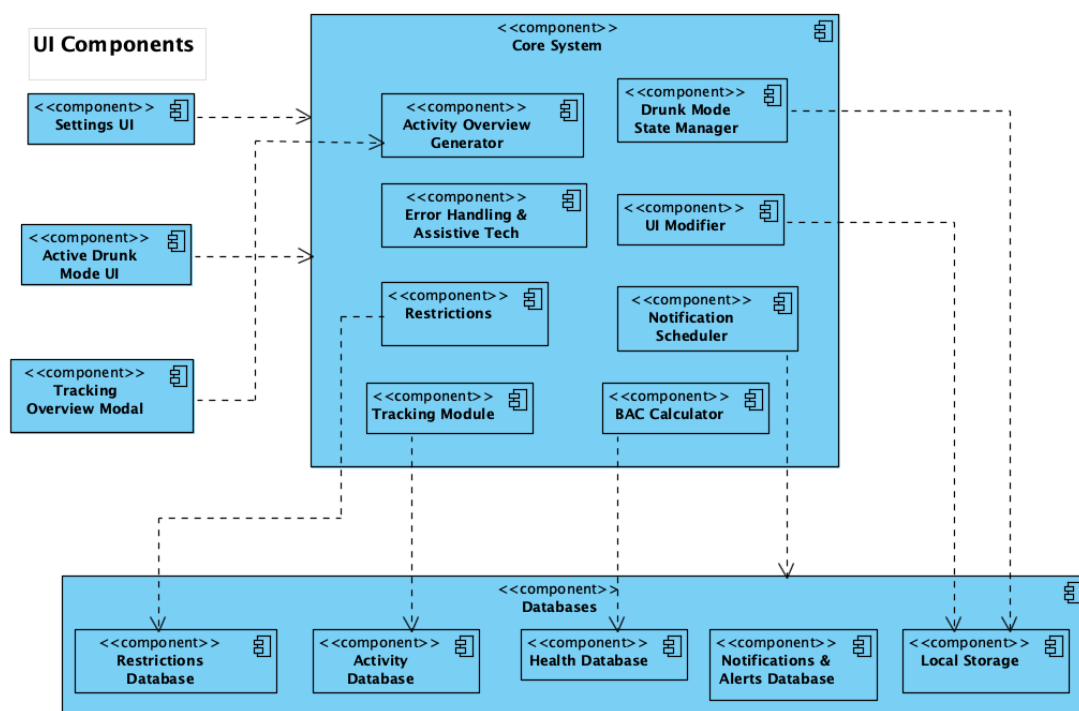


Figure 5: Component Diagram

### 4.3 State Diagram

Figure 6 shows how the system transitions between key states based on user actions. It starts in an idle state, awaiting input. When the user chooses to set up Drunk Mode, the system enters a configuration state where restrictions, safety contacts, alerts and health data are defined. After saving the system returns to idle until Drunk Mode is activated.

Once activated, the system enters the composite *Drunk Mode Active* state, where all core features run simultaneously – including tracking, restrictions, UI changes, alerts, BAC estimation, quick-access safety tools, and assistive tech. This concurrency ensures a smooth and responsive experience while reducing cognitive strain (Norman, 2013).

Exiting Drunk Mode triggers a summary state that displays the activity overview. After the user closes it, the system returns to idle, completing the loop and ready for future use.

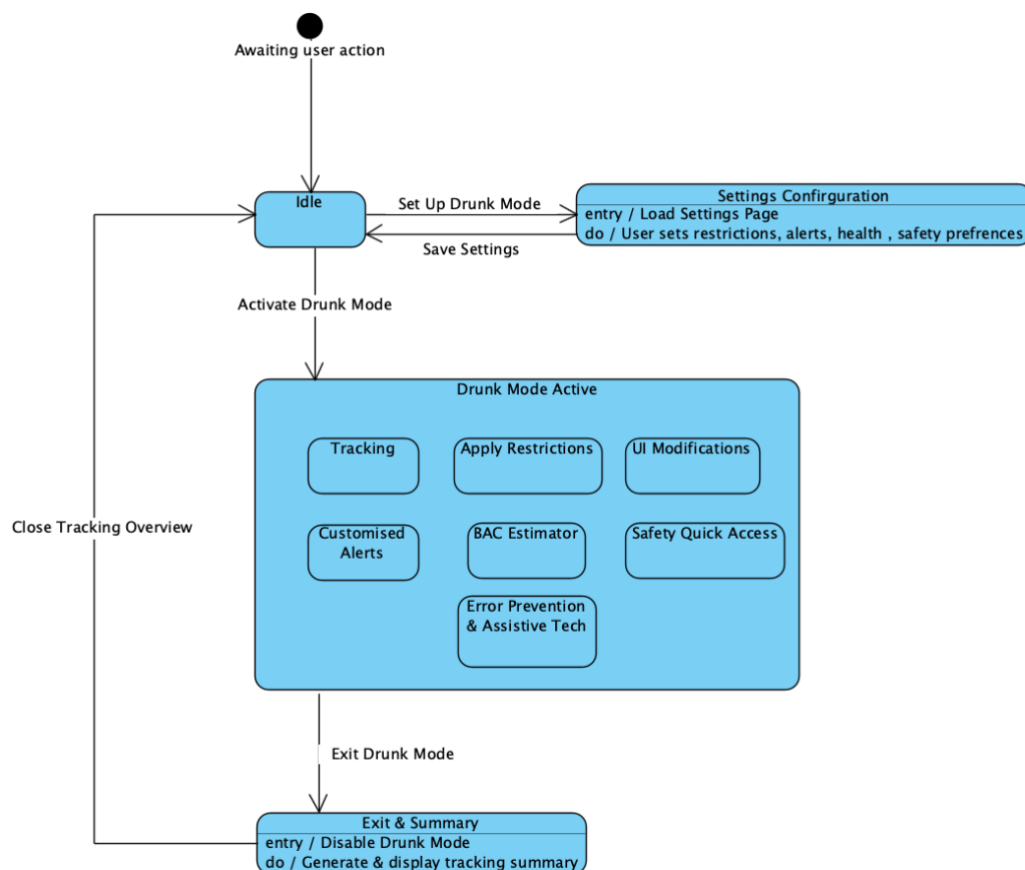


Figure 6: Drunk Mode State Diagram

## 4.4 System Architecture

The diagram below demonstrates how the system components work together to support Drunk Mode. The user sets preferences and activates Drunk Mode through the UI, which interacts with both local storage and the backend. The active state of Drunk Mode is primarily managed locally via AsyncStorage, to allow for fast, offline toggling without the need for network access. However, for tracking purposes, the Drunk Mode status is also stored in the Supabase. When active, tracking and health data handled by service functions, which generates summaries and recommendations. These are stored and retrieved when needed. The system keeps the UI separate from logic to improve maintainability and scalability.

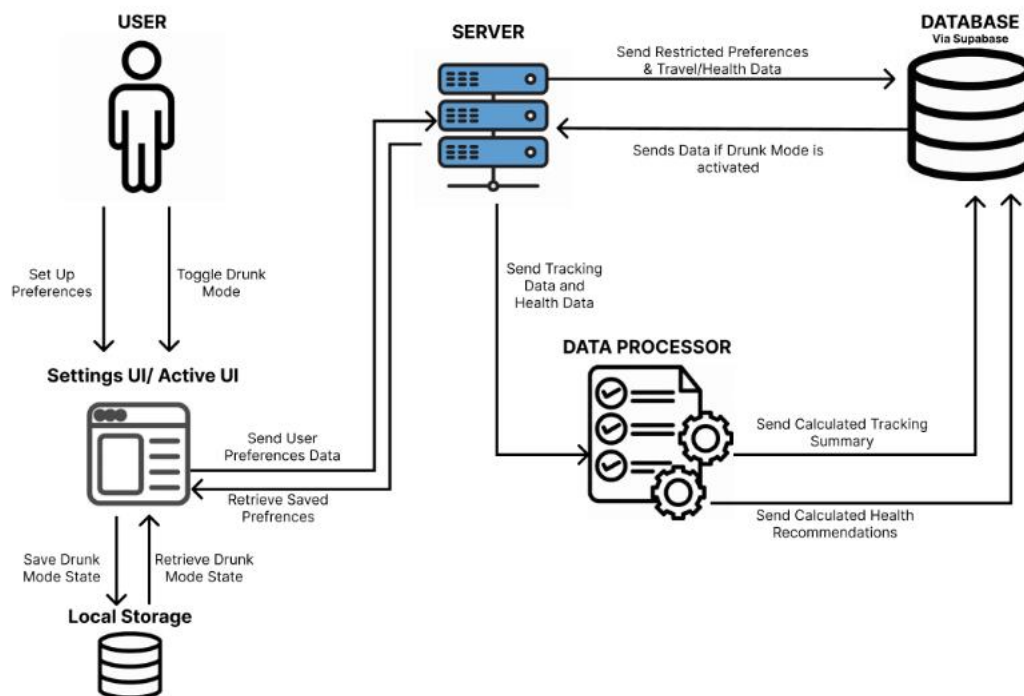


Figure 7: System Architecture



## 4.5 Design Pattern

Model-View-ViewModel (MVVM) was chosen as the design pattern for this project due to its compatibility with React Native and emphasis on separation of concerns. The View contains the UI components and captures user actions such as toggling Drunk Mode. These actions are passed to the ViewModel which contains the business logic like managing the global state (via the DrunkModeContext). The ViewModel then interacts with the Model, which includes data sources like Supabase, AsyncStorage, and services logic. The Model returns processed data, which the ViewModel formats and sends to the view for UI updates. MVVM simplifies state management and makes the code easier to maintain, test, and scale.

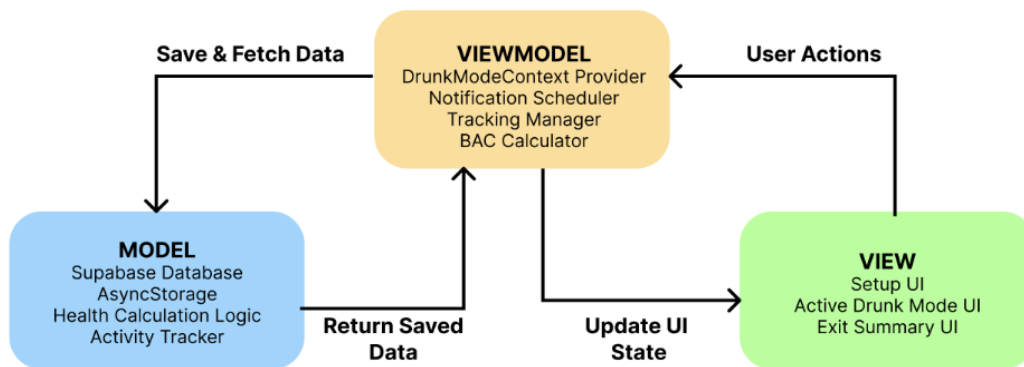


Figure 8: Model-ViewModel-Model Diagram

## 4.6 Design Principles

The following design principles were shaped by McKay's (2013) communicative interface principles, focusing on clear feedback and direct user guidance. Inclusive design principles (Clarkson et al., 2003) further shaped the below principles to ensure support for users with temporary impairments such as intoxication.

### 1. *Simplicity and clarity*

A minimalist layout reduces overload, using large tap targets and reduced visual clutter to support impaired vision and shaky motor control.

### 2. *Error Prevention*

Confirmation prompts are used for sensitive actions like making emergency calls or disabling Drunk Mode, reducing the chance of accidental errors – a common challenge for intoxicated users.

### **3. *Accessibility***

The interface accounts for temporary cognitive changes and alcohol myopia (Harvey et al., 2013) placing key actions in the centre and simplifying navigation.

### **4. *Trust and Reassurance***

Emergency access, location sharing, and gentle prompts help build trust and reduce stress. This aligns with affective computing principles aimed at calming and supporting users (Picard, 2000).

### **5. *Customisation***

Personalised alerts, contact settings, safety settings and restrictions let users shape the system to their needs, encouraging long-term use and relevance.

### **6. *Familiar Interaction***

The UI mirrors system-level IOS features like Focus or Do Not Disturb. This familiarity reduces learning time and helps users adopt it more quickly, even when impaired. These features align with good mobile UI principles and user experience goals (Yang et al., 2018).

## **4.7 Justification for System-Level Integration**

Instead of a third-party app, the system is designed to prototype as a native phone feature, similar to the existing Focus Mode's on IOS systems (Apple n.d). This approach increases accessibility, safety, and trust.

### **1. *System Access***

A built-in feature can manage app restrictions, safety tools, and UI overlays – capabilities third-party apps don't have.

### **2. *Privacy and Trust***

Users are more likely to trust sensitive settings like health data when handled natively within their phone.

### **3. *Ease of Use***

A downloadable app creates friction. A built-in option is easier to discover and use in the moment, ensuring system accessibility.

### **4. *Stability and Performance***

Native tools are more reliable and efficient – this is essential when users may be frustrated or distracted.

## 5. Maintenance

Native features can be updates through system updates, reducing user effort and improving long-term support.

## 4.8 Wireframes

The following wireframes illustrate early design concepts for the system's key interfaces, helping to visualise the structure and intended user flow before implementation. Figure 10 shows the Settings UI, where users can set up Drunk Mode and configure their preferences. Figure 9 presents the Standard Home Screen UI, which mimics the normal phone layout before Drunk Mode is enabled. Figure 11 displays the Drunk Mode UI, a simplified layout with high-contrast buttons and quick access to emergency actions. Last, Figure 12 shows the Activity Overview Modal, summarising the user's interactions during a Drunk Mode session. Although these wireframes are simple, they still capture the system's layout intentions and helped inform iterative design decisions.

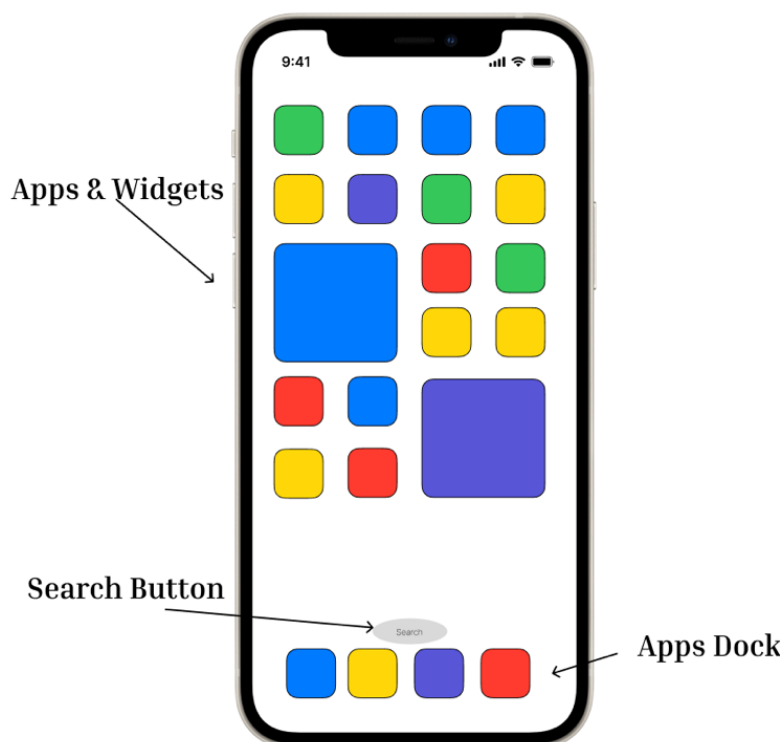


Figure 10: Normal Home Screen UI

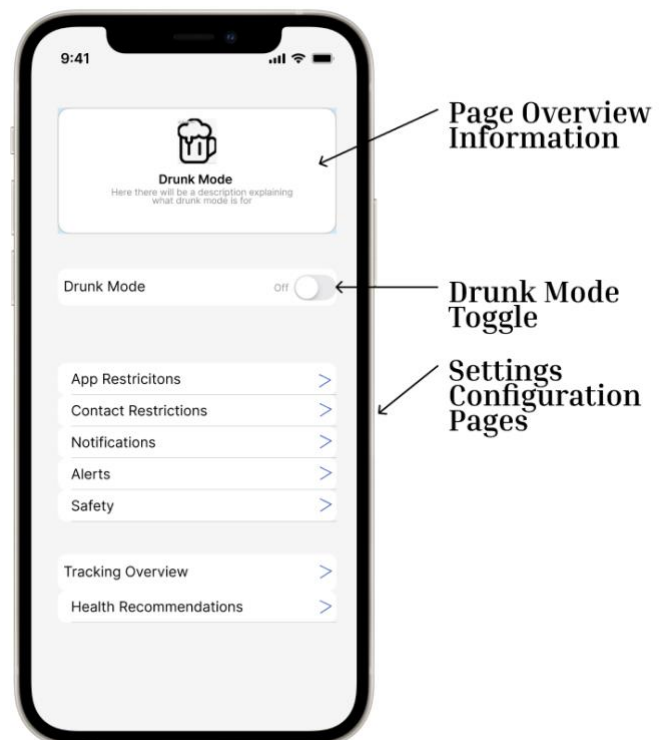


Figure 9: Drunk Mode Settings Page

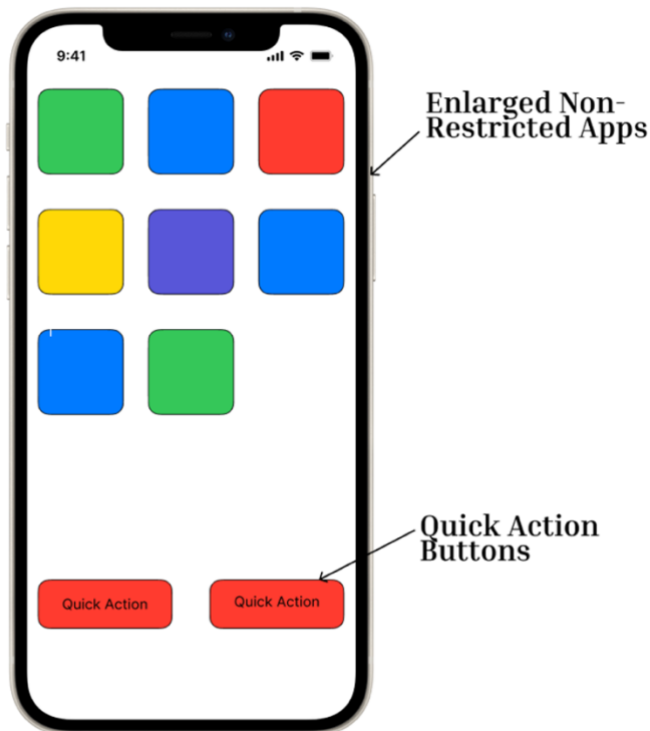


Figure 11: Drunk Mode Home Screen UI



Figure 12: Activity Overview Modal

## 4.9 Summary of Key Design Objectives

The design process focused on creating an interface that supports users in a vulnerable and often chaotic state – when intoxicated. By prioritising simplicity, error prevention, and trust, the system aims to reduce cognitive load and guide users toward safer choices. Each design choice was made with the user’s emotional, mental, and physical state in mind, ensuring the interface feels familiar, clear, and responsive in important moments. Overall, the design serves not just functionality, but empathy.

## Chapter: 5

### Implementation

---

This chapter provides a thorough overview of the technologies and tools used to implement the interface prototype; alongside how key functionalities were developed. It also outlines the structural decisions, architecture, and steps taken to ensure the system works effectively as a proof of concept. Additionally, before implementation began, a risk analysis was conducted to inform development decisions, covering technical, academic, and ethical concerns (see Appendix E).

#### 5.1 Development Environment

Below details the reasoning behind the choice of technologies, tools, and frameworks used when implementing the application. Each technology was selected based on suitability for mobile interface development, alignment with project goals, and developer familiarity.

##### 5.1.1 React Native with Expo

React Native is an open-source framework for building mobile interfaces using JavaScript and React. It enables cross-platform development with native performance, making it ideal for developing the Aware Phone Interface. The use of Expo further simplified development as it provided tools for building, testing, and debugging the app without native code compilation. Expo's built-in modules (e.g., Notifications and Sensors) allowed for fast prototyping of key features such as activity tracking and modal interactions.

##### 5.1.2 JavaScript and TypeScript

The project was primarily developed using JavaScript and TypeScript within the React Native environment. TypeScript was used for certain components and services due to its static type-checking benefits, which helped improved reliability in core logic such as alert scheduling and data fetching. However, JavaScript was used in several UI screens where rapid iteration and flexibility were prioritized. This hybrid approach was used to avoid TypeScript-related configuration and type issues during development, while still benefitting from its structure in key areas of the app. The use of both languages allowed the project to balance safety and speed throughout implementation.

### 5.1.3 Supabase

Supabase served as the backend-as-a-service (BaaS) solution, replacing the need for a traditional backend. It provided secure data storage and real-time subscriptions using a PostgreSQL database. Supabase was used to store alerts, health data, safety data, restrictions, and tracking data. Its simplicity and strong querying made it a good fit for mobile-first development, especially when used with client-side logic to schedule notifications.

### 5.1.4 AsyncStorage

AsyncStorage was used to store data locally in the app. It saved things like whether Drunk Mode was on and cached health info, even when the interface was closed or offline. This made it faster and more reliable, since the interface didn't need internet to check important settings.

### 5.1.5 Device State Management

This interface was designed as a system-level feature, similar to existing tools like Focus Mode or Do Not Disturb on iOS. Therefore, it does not require traditional user authentication or login. Instead, it operates based on the device's local configuration and state. This approach simplifies the user experience, reduces barriers to access during intoxication, and aligns with the intended integration as a built-in smartphone feature rather than a standalone application.

## 5.2 Architecture and MVVM Design Pattern

This section provides an overview of the system architecture used to develop the Aware Phone Interface and describes how the MVVM pattern was applied to ensure clear separation of concerns and maintainability.

### 5.2.1 System Architecture Overview

The implemented system closely follows the conceptual design outlined in Section 4.4, which separates concerns across distinct layers to improve maintainability and scalability. The interface adopts the MVVM design pattern. This ensures a clean separation between the UI components (views), business logic (view models), and backend communication or data processing (services/models).

**The structure is broken down as follows:**

## View (UI Layer)

All user-facing components are implemented inside the `screens` and `components` folders using JavaScript. These files are responsible for presenting the UI and capturing user interactions, such as toggling Drunk Mode, setting alerts, or entering health information. By keeping business logic out of these files, the UI remains clean and focused solely on presentation. This clear separation ensures that views remain modular and easy to maintain, especially as features evolve. Each screen is reactive and updates automatically in response to changes in the `ViewModel`, ensuring a smooth and dynamic user experience.

Example files include:

```
screens/SettingScreen.js
screens/ActivityOverview.js
components/TrackingOverviewModal.tsx
```

## ViewModel (State & Logic Layer)

The `viewmodels/DrunkModeViewModel.ts` file contains shared state and core business logic (e.g. how and when Drunk Mode is toggled, managing state transitions). It connects the user interface with the service layer and allows the views to reactively update based on state changes.

The following code is the function to toggle Drunk Mode:

```
const toggleDrunkMode = async () => {
  drunkMode.toggleDrunkMode();
  setDrunkMode (new DrunkModeModel ());
  await AsyncStorage.setItem("drunkMode",
JSON.stringify(drunkMode.getStatus()));
};
```

## Model / Service Layer

This layer, found in the `services` and `models` folders, includes reusable logic for interacting with Supabase, AsyncStorage, notification scheduling, health estimation, and more. The services are written in TypeScript to improve safety and reduce runtime bugs, particularly for backend tasks. While TypeScript is used across these backend services, some screens and UI components were implemented in JavaScript due to compatibility issues and challenges encountered with certain TypeScript configurations during development.

Example Services:

`NotificationService.ts`: manages notification restrictions

`ActivityService.ts`: fetches and organises activity overview data

`SafetyService.ts`: fetches and stores safety data

A key example is `saveSafetySettings`, which handles storing user-defined safety data in Supabase (see Appendix C1)

## Local Storage

Local state persisted via `AsyncStorage` to allow synchronous access and offline functionality.

This is used for checking whether Drunk Mode is currently enabled, storing health data locally, and remembering user settings between sessions.

```
await AsyncStorage.setItem("drunkMode", JSON.stringify(true));
```

## 5.3 Feature Implementation

This section outlines how key functionalities of the interface were implemented. It describes the logic behind Drunk Mode activation, restrictions, notifications, user support mechanisms, and real-time data handling. Each feature was designed to enhance user safety, reduce cognitive effort, and maintain usability during intoxication, in line with inclusive design principles.

### 5.3.1 Drunk Mode Activation

Drunk Mode can be enabled manually via a toggle or suggested automatically through a context-aware algorithm. Once activated, the interface applies all relevant changes – including UI adjustments, restrictions, notifications, alerts, and tracking – and locks the Drunk Mode settings to prevent changes mid-session.



### Activation Methods:

1. Manual toggle – Users can enable/disable Drunk Mode through the interface (see Figure 13)
2. Context-aware prompt (see Figure 15)– A heuristic scoring system runs in the background when the user types. It calculates a total score based on:
  - a. Time of day and day of week (e.g. Friday 11pm = high score)
  - b. Accelerometer data (device shaking),
  - c. Typo frequency (from the drunk auto-correction service),

Each factor contributes to a cumulative score, and if the total exceeds a threshold ( $\geq 1.5$ ), the system triggers a non-intrusive prompt suggesting the user activates Drunk mode.

Importantly, users don't need to meet all criteria – Drunk Mode flexibly triggers based on combined risk signals.

Once Drunk Mode is active, all restrictions remain in effect until the user manually deactivates it. If a user attempts to turn off Drunk Mode, a confirmation prompt appears (see Figure 14) to prevent accidental toggling. This includes two options: *Cancel* or *Turn Off*, encouraging intentional decision-making while impaired.

The Drunk Mode state is primarily managed locally using AsyncStorage for fast, offline access. However, the activation status is also stored in Supabase (`drunk_mode_status`) to support tracking features.

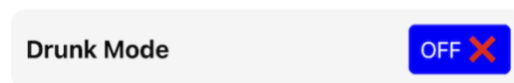


Figure 13: Drunk Mode on and Off Toggle

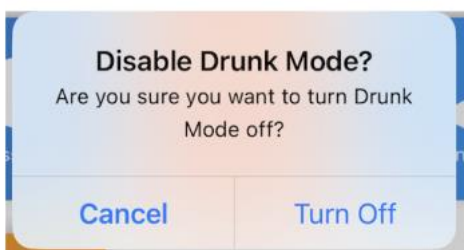


Figure 15: Drunk Mode Deactivation Prompt

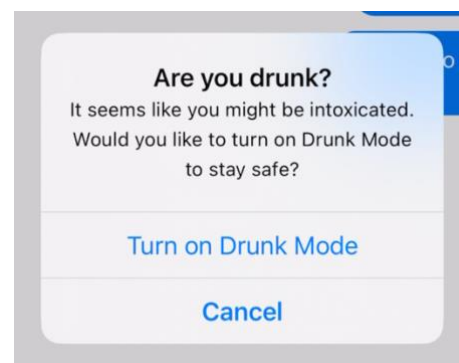


Figure 14: Context-Aware Prompt

**Implementation details:**

The detection logic is defined in `services/DrunkDetectionService.ts`, which listens to sensor data for 5 seconds and calculates a weighted `totalScore` from multiple signals:

```
totalScore = (typoScore * 0.6) + (shakeScore * 0.3) +  
(timeScore * 0.1)
```

- `timeScore`: Adds up to 2 points based on late-night hours and weekends.
- `shakeScore`: Increases when device shaking exceeds magnitude  $\geq 1.1$  (iterative testing was used to identify magnitude sensitivity for best detection).
- `typoScore`: Passed in from the auto-correction service when erratic typing is detected.

If both `typoScore`  $\geq 2$  and `shakeScore`  $\geq 2$ , a bonus of +0.5 is added to reflect stronger combined evidence of intoxication. When the final score reaches 1.5 or higher, a Drunk Mode prompt is triggered (Figure 15).

This weighted approach prioritises behavioural signals over weaker context cues, making detection more balanced and accurate. This approach builds on the alcohol myopia concept (Harvey et al., 2013), offering just-in-time support when users may not realise how intoxicated they are.

### 5.3.2 Restrictions System

The restrictions system helps prevent risky or unintended actions during Drunk Mode by enforcing limits users set in advance.

#### Contact Restrictions

Users can define trusted contacts they can call or message while Drunk Mode is active (see figure 16) (`ContactRestrictions.js`, `ContactService.ts`). All of the restricted contacts are greyed out and users are unable to click on them to contact them.

#### App Restrictions

Users can select which apps remain visible during Drunk mode. All the restricted apps are hidden from the home screen (see figure 17). (`AppRestrictions.js`, `AppService.ts`).

#### Notification Restrictions

Users can choose to mark apps which they do not want to receive notifications during Drunk Mode. While this currently functions as a preference-setting page, it lays groundwork for potential deeper integrations with third-part apps (see figure 18).

(`NotificationRestrictions.js`, `NotificationService.ts`)

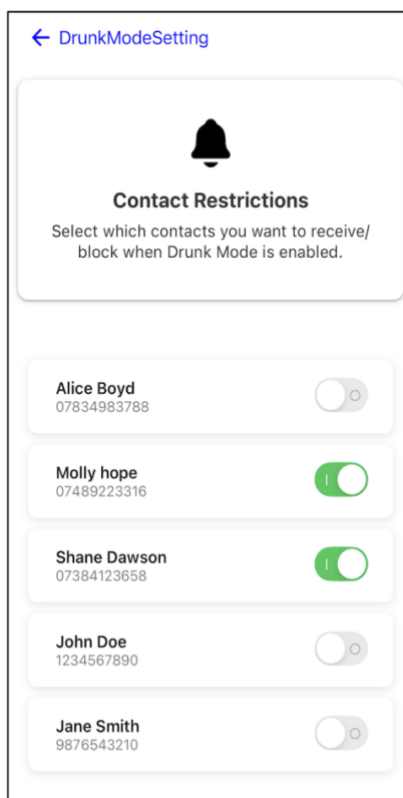


Figure 17: Interface for Setting Contact Restrictions

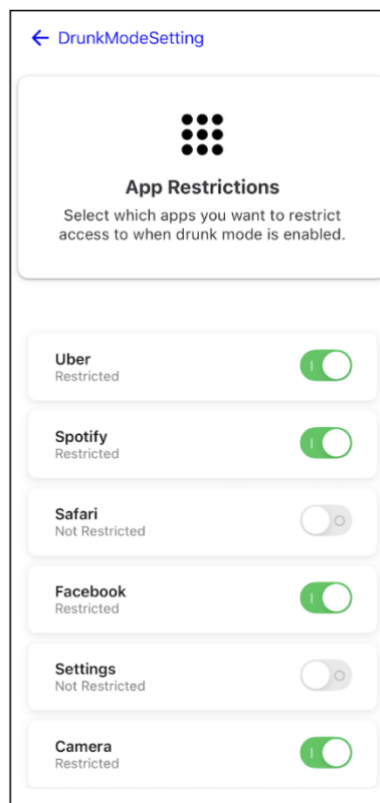


Figure 16: Interface for setting App Restrictions

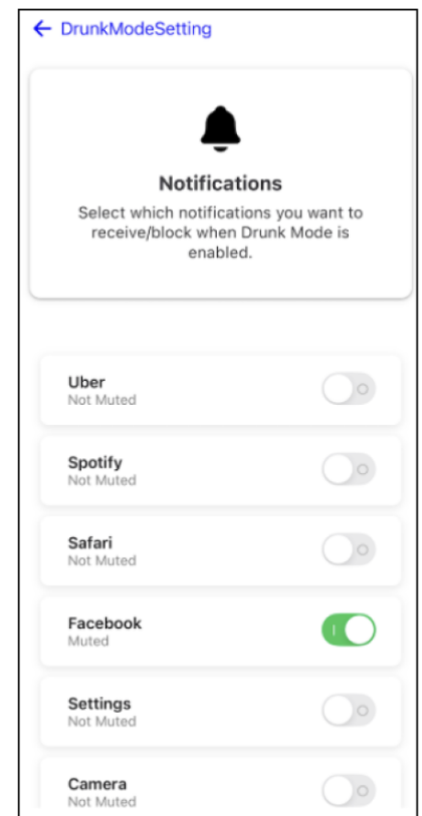


Figure 18: Interface for Setting Notification Restrictions

## Settings Lock

The Drunk Mode settings menu is locked during Drunk Mode to prevent users from altering safety options mid-session. Attempting to access Drunk Mode settings displays an alert explaining that access is denied, and that the mode has to be turned off first.

(SettingScreen.js)

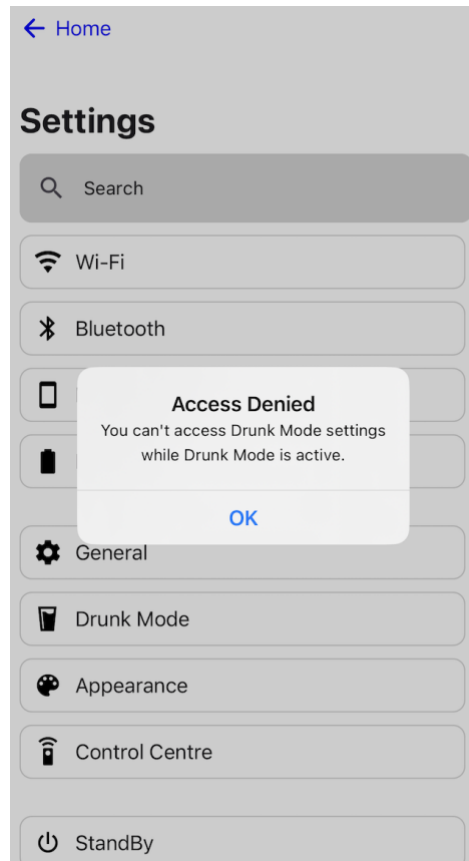


Figure 19: Locked Drunk Mode Settings Prompt

### 5.3.3 Notifications and Alerts

The interface includes two types of prompts while Drunk Mode is active: system notifications and user-defined alerts, both designed to guide, reassure, and gently interrupt risky behaviour while Drunk Mode is active.

#### Pre-Set System Notifications

To support the user's wellbeing, timed notifications are triggered automatically when Drunk Mode is activated. These include hydration reminders, auto-correct awareness messages, and supportive nudges to check the user has their belongings.

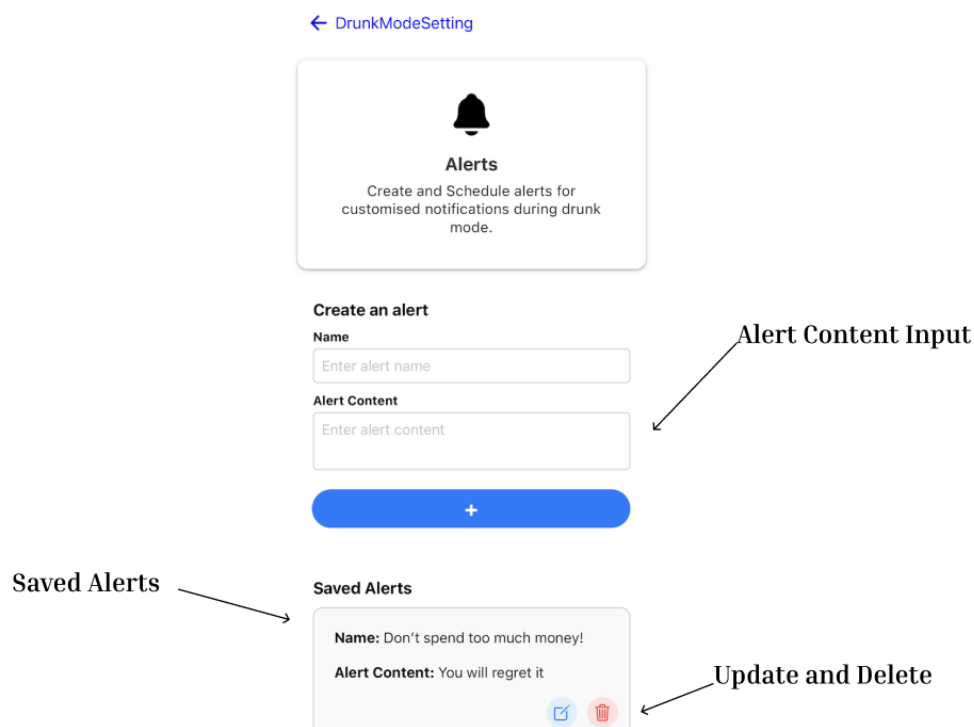
- These are managed in `NotificationService2.js`
- Scheduled using `expo-notifications`
- Appendix C2 displays examples of the system notifications

Activating Drunk Mode triggers the relevant alerts, though full dynamic rescheduling of notifications was limited by Expo Go.

### User-defined Alerts

Users can pre-create custom alerts (e.g. “Don’t spend too much money!”) that are triggered during Drunk Mode. Users can also update existing alerts and delete alerts. These alerts are stored in Supabase and fetched when Drunk Mode activates.

- Created via the `Alerts.js` settings screen (Figure 20).
- Stored in the `timed_alerts` table in Supabase.
- Retrieved and displayed using `NotificationService2.js` upon activation.



These alerts can’t be modified while Drunk Mode is active, ensuring that users receive the advice they wrote while sober. The logic for fetching and scheduling these notifications is implemented in `NotificationService2.js`, with implementation details available in Appendix C3.

### 5.3.4 Health Recommendations & BAC Estimation

The health features are designed to promote safer drinking by helping users understand their personalised limited based on personal health data.

#### Health Settings

Users can input their height and weight in the Health Settings screen, which calculates their BMI (Figure 21). Based on the user's BMI, the system provides guidance on how much alcohol is safe to drink. This data is saved to Supabase and cached locally with AsyncStorage for offline access and faster lookup. (`HealthRecommendations.js`).

```
const calculateAlcoholUnits = (bmiValue) => {
  let recommendedUnits = bmiValue < 18.5 ? 2 : bmiValue > 25? 4 : 3;
  setAlcoholUnits(recommendedUnits);
};
```

This function recommends a safe alcohol limit (in alcohol units) based on the user's BMI: lower BMI gets 2 units, higher BMI gets 4, and average BMI gets 3.

This logic, inspired by NHS guidelines on alcohol limits (NHS, 2023), uses a simplified BMI-based rule-of-thumb to offer quick, personalised safety feedback, but is not intended as medical advice.

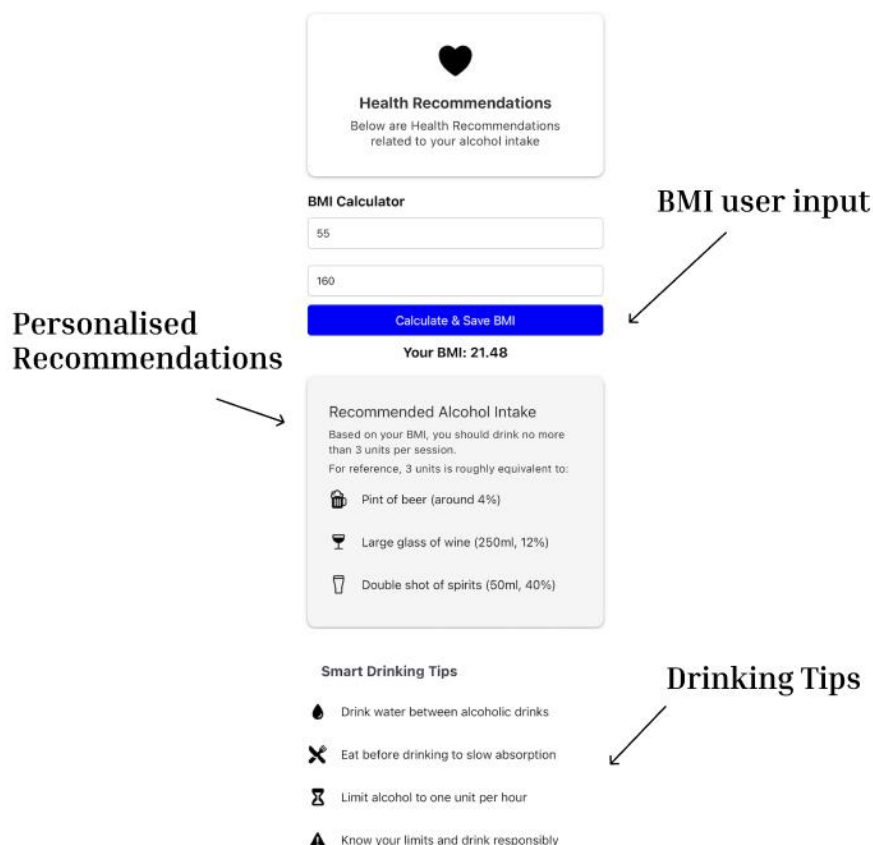


Figure 21: Health Recommendation Screen

## BAC Estimation Tool

A quick-access BAC Estimator tool is available during Drunk Mode via one of the quick access buttons on the home screen (Figure 22). This tool allows users to input how many drinks they've had and get an estimate of their Blood Alcohol Concentration using a simplified Widmark formula. This helps users self-monitor and avoid exceeding safe limits (Widmark, 1932).

The estimator uses the user's previously saved weight (entered via the health settings) to approximate BAC, applying a simplified version of the Widmark formula. This approach personalises the result using individual health data. A sample of the core logic is shown below:

```
const estimated = ((drinks * 10) / (weight * 0.58) - 0.015).toFixed(3);  
setBac(Math.max(estimated, 0));
```

This equation assumes 10g of alcohol per drink, a distribution ratio of 0.58, and subtracts a fixed metabolic rate (0.015%) to simulate breakdown over time. (Brick, 2006)

BAC Estimator Modal

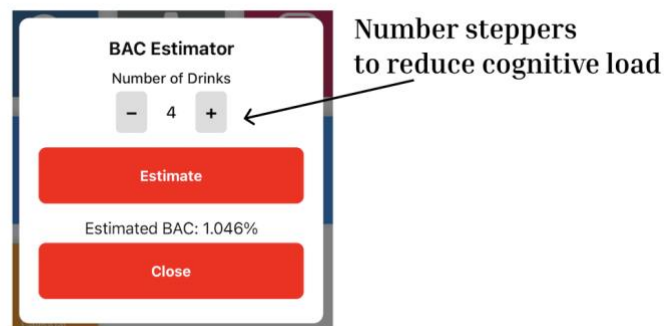


Figure 22: BAC Estimator Modal

### 5.3.5 Auto-Correction System

The auto-correction system reduces the risk of unclear or embarrassing messages while intoxicated by correcting common drunk spelling mistakes in real time. It combines a drunk typo dictionary (e.g., “dfrunk” = “drunk”) with fuzzy matching using Levenshtein Distance (Levenshtein, 1966), a method which checks how many changes it would take to turn one word into another to find the closest match.

Each word is processed individually. First the system removes repeated letters (e.g., “dfunkkkk” = “dfunkk”) to handle excessive character repetition. Repeated letters are trimmed down to a maximum of two, so real words like “coffee” aren’t affected. If the processed word matches the typo dictionary list, it is corrected immediately. Otherwise, fuzzy matching applies a correction only if the replacement is clearly closer.

A whitelist of short words (e.g., “no”, “me”) prevents overcorrection. Users then receive a notification when texts are corrected (Appendix C4). The correction process is applied as follows:

```
const correctedMessage = message
  .split("")
  .map(word => fuzzyCorrectedWord(word))
  .join("");
```

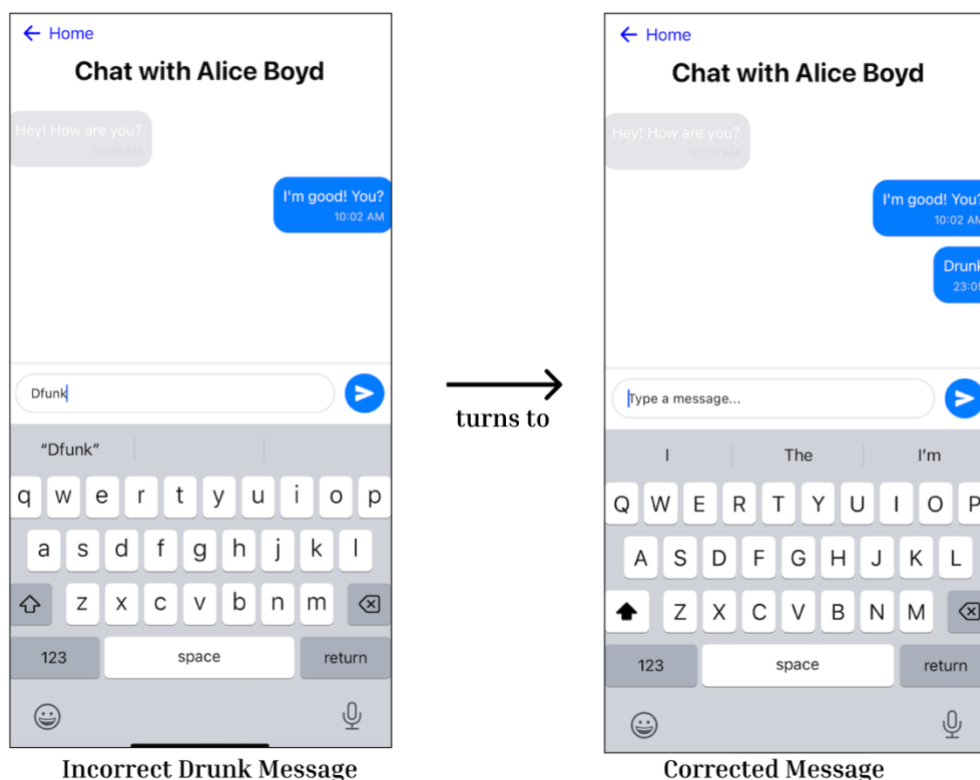


Figure 23: Drunk Mode Autocorrect



### 5.3.6 User Interface Adaptation

#### Simplified Drunk Mode Home Screen UI

When Drunk Mode is enabled, the interface adjusts to support reduced cognitive and motor control. The home screen is simplified: only allowed apps are shown, with enlarged icons for easier selection. Restricted apps are hidden entirely, and the standard app dock is removed to reduce complexity. It is replaced by clearly labelled quick access buttons (described below).

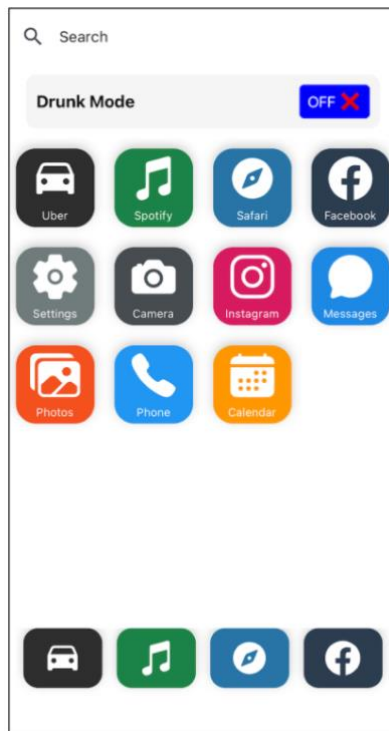


Figure 25: Normal Home Screen

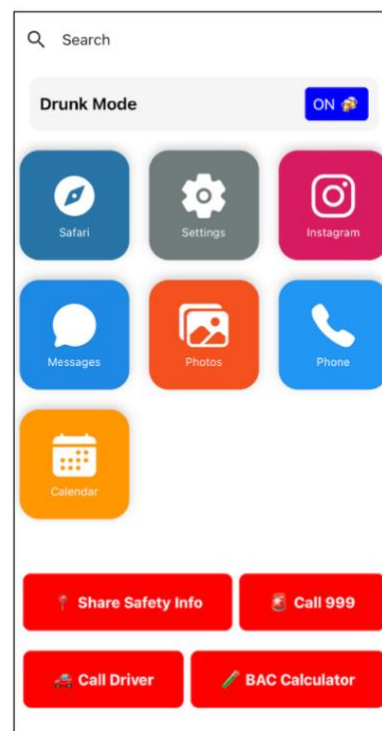


Figure 24: Drunk Mode Activated Home Screen

#### Quick Access Buttons

The Drunk Mode UI includes four quick-access buttons presented in a 2x2 grid fixed at the bottom of the screen to enhance visibility and reduce the need for scrolling. These include calling emergency services, sharing safety info (location and pre-set travel details), calling users pre-set designated driver, and opening the BAC Estimator. The high-risk actions are protected with confirmation prompts.

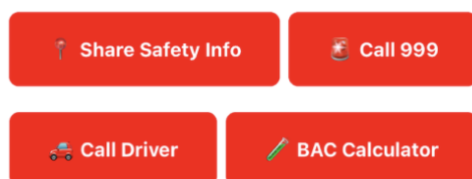


Figure 27: Quick Access Buttons

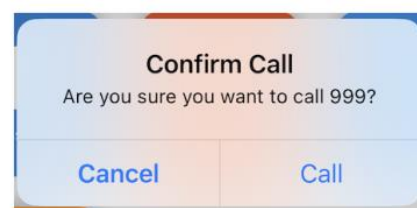


Figure 26: Confirmation Prompt

## Accessibility Rationale

These changes were guided by research from the literature review (Harvey et al. 2013).

Intoxicated users may struggle with precision, memory, and inhibition – so the UI priorities clarity, minimalism, and direct access to essential features. Using larger icons, reduced visual clutter, and confirmation prompts helps avoid errors while reinforcing safety.


### 5.3.7 Pre-Set Safety settings

To support safer use during Drunk Mode, users can pre-set key safety options including (Figure 28):

- An emergency contact
- A designated driver
- A preferred transport method (e.g. private or public)

These are configured in the Safety Settings Screen and used during Drunk Mode to power quick access buttons like “Call designated driver” or “Share Safety info”. This reduces decision-making while intoxicated and ensures help is just one tap away.

[← DrunkModeSetting](#)



### Safety

Please input the details below to improve your safety.

Select Emergency Contact

Alice Boyd

Select Designated Driver

Shane Dawson

Select Transport Mode

Public Transport

Save Settings

Figure 28: Safety Settings Screen

### 5.3.8 Activity Tracking Overview

To support reflection and accountability, the interface automatically tracks key user activities while Drunk Mode is active.

#### Live session tracking

While Drunk Mode is enabled, the system logs core actions such as sent messages, calls, and notifications. Each event is timestamped and categorised in the background using the `logTrackingEvent()` service (e.g. `event_type :q 'message'`, `event_detail: 'Message to John'`). This data is saved in Supabase for later retrieval.

#### Post-Session Summary

Once Drunk Mode is turned off, users can view their full history via the Activity overview screen (Figure 29), which lists sessions by date and shows a bar chart summary. Detailed session data is shown in Figure 30, and Figure 31 shows the modal summary that appears after Drunk Mode is turned off.

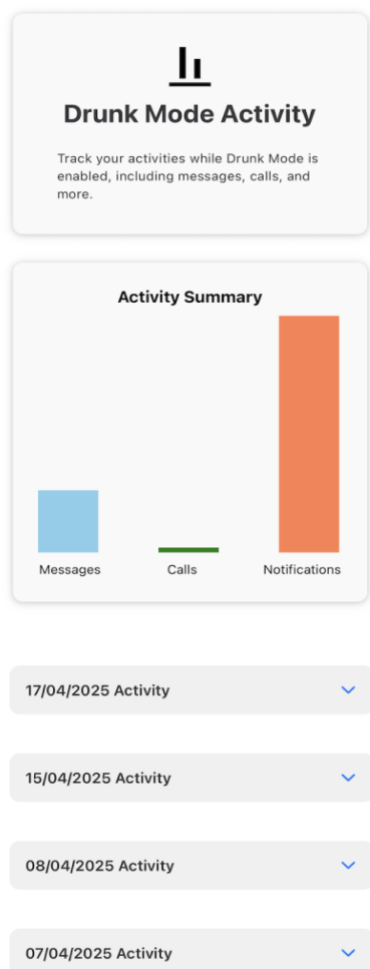


Figure 31: Activity Overview Screen

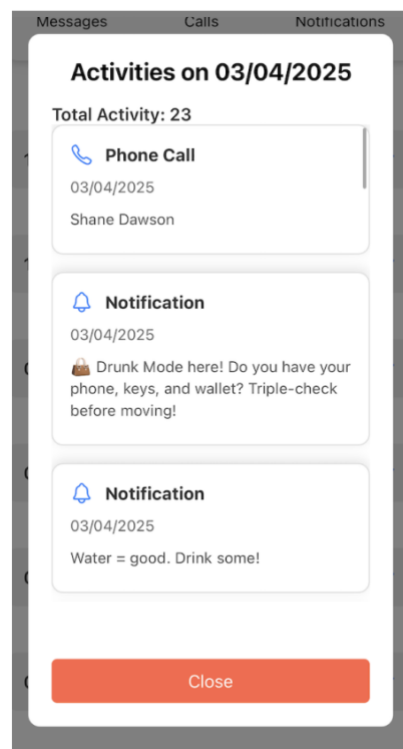


Figure 30: Past-Session Activity Data

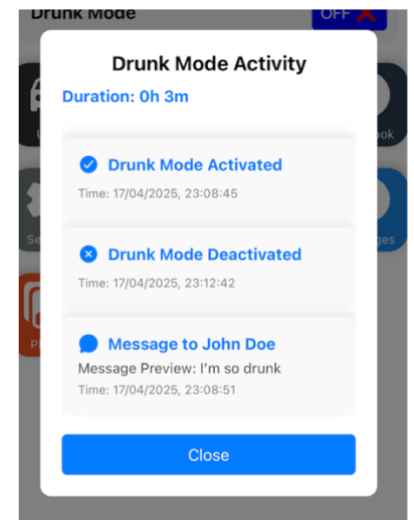


Figure 29: Activity Overview Modal

This tracking system encourages mindful use and provides users with helpful insight into their behaviour while intoxicated – without requiring any manual input.

## 5.4 Supabase Integration

The interface uses Supabase as the backend for storing user preferences, alerts, health data, and activity logs. The diagram in Figure 32 shows the schema structure, with tables including `user_health_data`, `timed_alerts`, `drunk_mode_status`, and more.

The interface uses `Supabase.channel()` to update things like app restrictions instantly, without needing to reload all the data as the feature listens for changes in the database and updates the interface live. This approach helps keep the interface responsive by only reacting to relevant changes, rather than repeatedly reloading data.

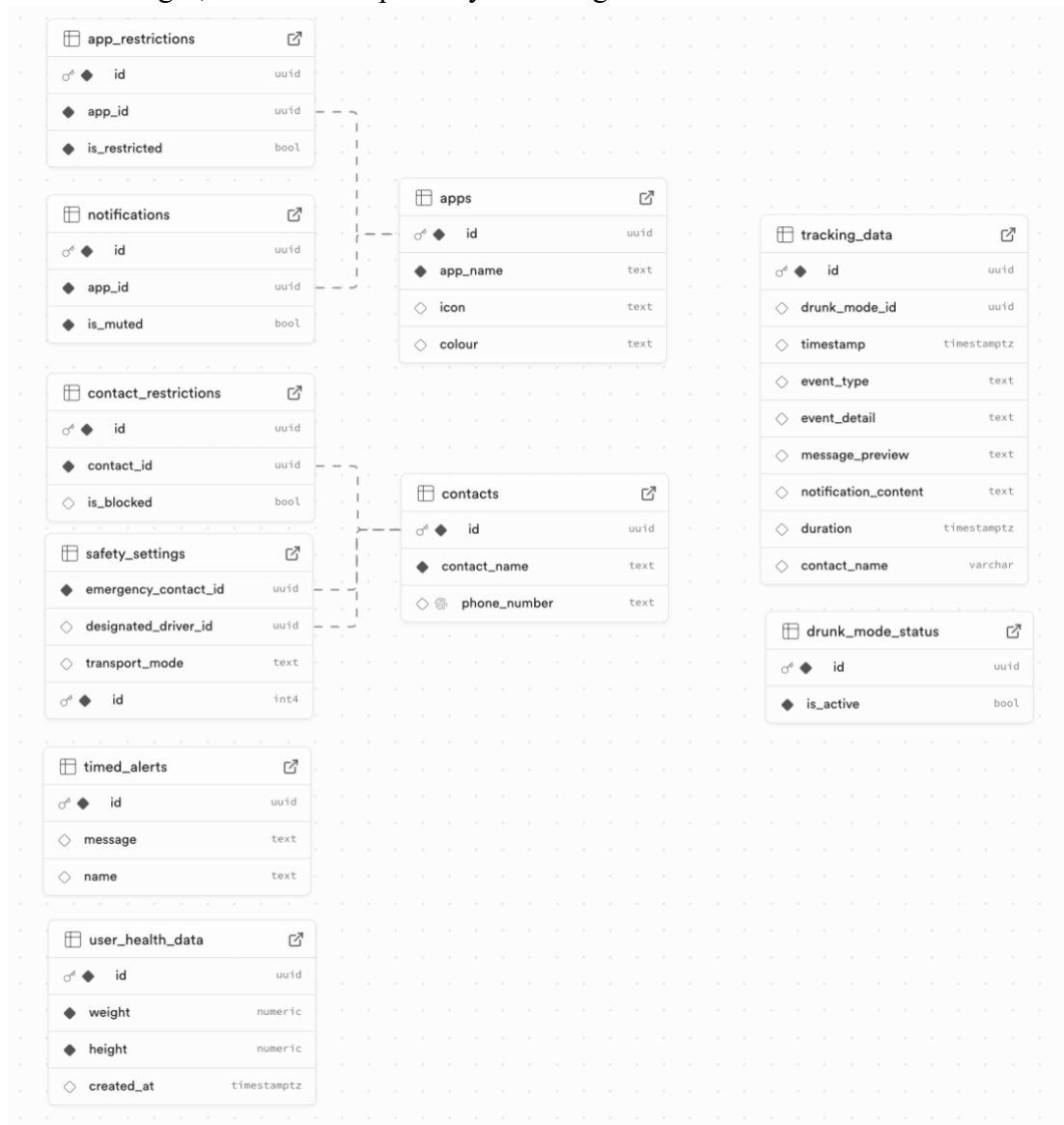


Figure 32: Supabase Database Schema

## 5.5 Theoretical Grounding in Design

This section explains how theoretical frameworks from cognitive science informed design decisions of the Aware Interface. Two key concepts – Distributed Cognition and Phenomenology – were used to guide interaction design, error prevention, and the structure of the supportive interface elements during intoxicated states.

### 5.5.1 Distributed Cognition

Distributed cognition is a theory that views cognitive processes as shared across people, tools, and environments, rather than confined within the individual mind. For this projects interface, the system acts as an external cognitive aid – helping users make safer decisions while intoxicated by distributing some of the mental workload to the interface instead of the users' mind (Perry 2003).

The following design decisions were directly informed by this concept:

- **Confirmation prompts before risk actions** (e.g., disabling Drunk Mode, calling emergency services) provide a moment of reflection and prevent impulsive errors by offloading judgement to the system.
- **Activity tracking logs and session overviews** act as an external memory aid as it helps users recall what actions were taken during impaired states.
- **Drunk mode scoring system and prompts** help users notice when their behaviour might be risk (e.g. if they are shaking their phone a lot), therefore offloading the need for self-assessment.
- **Simplified interfaces and quick access buttons** reduce cognitive load by reducing the need for complex navigation, giving users only what they really need in the moment.

By shifting some of the mental effort onto the interface, especially under intoxicated conditions, the system supports safer behaviour, better memory retention, and reduced frustration (Perry, 2003)

### 5.5.2 Phenomenology

Phenomenology focuses on how people actually experience the world – how things feel, appear, and make sense in the moment. It's less about logic and more about the lived, embodied experience. This perspective is especially useful when designing for intoxicated

users due to their perception, attention, and physical responses are often altered in subtle but important ways (Dourish, 1999).

**Key ways this influenced the design:**

- **Larger touch targets and fewer visual distractions** – the interface adapts to the user’s altered motor control and attention span. For example, when Drunk Mode is active, non-essential elements are hidden and app icons become larger and easier to tap.
- **Emergency and safety features placed centrally** – this supports alcohol myopia theory (Harvey et al., 2013), which shows that intoxicated user focuses narrowly on what’s in front of them and ignore peripheral context.
- **Simpler navigation** – users don’t need to think ahead or remember complex steps – most actions, like calling a contact or using the BAC estimator, can be completed in just one or two taps.
- **Confirmation prompts** – create a moment of reflection that breaks through the user’s automatic behaviour, therefore helping them feel a sense of control without needing to think too hard about consequences.
- **Feedback on actions** – (e.g. auto-correct alerts, tracking overviews) reinforces that the system is watching out for them – subtly building trust in the interface even when the user’s judgement is impaired.

By designing around the user’s embodied state rather than assuming a “normal” level of awareness, the interface becomes more humane and usable under stress.

## Chapter: 6

# Testing

---

### 6.1 Testing Approach

Testing focused on ensuring core features functioned as expected. Manual functional testing was carried out throughout development, checking toggles, alerts, and UI behaviour.

Functional testing was conducted on a physical device (iPhone via Expo Go) to verify expected behaviours. Additionally, informal feedback from 3 peers provided insight into usability and helped validate the clarity of the interface. No formal user testing was conducted due to ethical and time constraints.

### 6.2 Informal Usability Feedback

To complement technical testing, informal usability feedback was gathered from a small number of peers ( $n=3$ ) interacting with the prototype. Participants were asked open-ended, non-identifying questions about their experience. This feedback revealed useful insights into UI clarity and feature functionality. While it was not a substitute for formal usability testing, this step helped validate initial design assumptions and identify minor points of confusion, which were noted for future refinement.

#### 6.2.1 Method

Participants were asked to explore the interface and complete a small set of tasks (e.g., enabling Drunk Mode, setting restrictions, using the BAC estimator, and viewing the activity summary). They were then asked five open-ended questions focused on usability and clarity. All responses were anonymous and non-identifiable, following ethical guidance for informal peer testing. The questions and summarised results are available in Appendix D.

#### 6.2.3 Results & Discussion

Feedback from the three participants was broadly positive, with all testers describing the navigation as clear, simple, and easy to follow. Features like emergency quick-access buttons and BAC estimator were seen as reassuring and helpful for promoting safer decisions.

Some confusion was noted around the meaning of “BAC”, suggesting terminology could be made clearer and explanation of BAC results could be more user friendly. One participant also recommended a brief onboarding explanation for first-time users of the feature.

Overall, the feedback suggests the interface achieves its usability goals while highlighting areas for improvement, mainly around user guidance. Full responses are summarised in Appendix D.

### 6.2.3 Ethical Considerations

All user testing was conducted anonymously and voluntarily. No personal data was collected, and participants were informed that their feedback would be used solely for academic purposes. As no sensitive information was gathered and participation posed no risk, formal consent was not required in accordance with standard ethical guidelines for low-risk studies.

## 6.3 Feature Testing

To ensure that all core functionalities performed as intended, each key feature of the interface was manually tested against its expected behaviour. This process involved checking whether user actions triggered the correct responses, whether the different interface states update appropriately, and whether data was logged or retrieved as supposed to. Feature testing checked that everything worked as expected based on the requirements listed in Chapter 3.

All tests were run using Expo Go on a single iOS device, which may limit generalisability across platforms. The table below summarises the features tested and whether they passed based on the observed outcomes.

Feature	Description	Status
Drunk Mode Toggle	Manual toggle activates/deactivates mode, locks settings	Pass
Context-Aware Prompt	Triggers correctly when score $\geq 3$	Partial Pass: <i>Prompt appears as expected but device occasionally freezes due to accelerometer feature load.</i>
Deactivation confirmation	Appears with proper message when user clicks deactivation button	Pass



App restrictions	Correct Restricted apps disappear from the home screen	Pass
Contact restrictions	Greyed-out contacts can't be clicked	Pass
System notifications	Auto-correct + hydration reminders appear at correct times	Pass
User-defined alerts	Custom alerts stored in Supabase display on Drunk Mode activation with correct content	Pass
Real-time correction	Common drunk typos are auto corrected when sent to user	Pass
Correction accuracy	Short common words (e.g. "no") are ignored and correction isn't overly aggressive	Pass
BMI unit suggestion	Suggestion displays after entering height/weight and is accurate	Pass
Safety settings save	Safety settings save correctly to the DB	Pass
Saved preferences	Preferences persist across sessions	Pass
Tracking events are logged	Messages, calls, notifications are logged in the DB	Pass
Modal accuracy	Tracking modal shows correct activities for previous session	Partial Pass: <i>only the user set alerts display in the modal</i>

Activity Overview page	Accurately lists past sessions with date and details	Pass
Drunk Mode UI	When Drunk Mode is on: app dock is hidden, restricted apps disappear, larger buttons, quick access buttons.	Pass
Quick access buttons	Each button works correctly	Pass
Real-time updates	Real-time updates work correctly and everything is dynamic	Pass
No full refresh	Settings preferences appear without app reload.	Pass

Table 3: Feature Testing Results Table

## 6.4 Testing Tools, Environments & Bug Handling

Testing and debugging were carried out continuously throughout development using Expo Go on a physical iPhone device. This setup enabled live reloading and quick iteration, which was essential for testing features like Drunk Mode activation, notifications, and more.

Bugs were handling using a combination of trial-and-error, console logging to track data flow and errors, and targeted testing of individual components. Common issues included incorrect state updates, notifications scheduling issues, and UI glitches. These were typically resolved by console logging, commenting out sections to isolate the issue, and reworking the logic or file structure. When needed, I also searched online or consulted documentation for solutions specific to Expo or React Native. Additionally, GitHub Copilot was useful to speed up development or suggest logic, though code was manually reviewed and adapted to suit the interface and system's needs.

The iterative approach made it easier to spot bugs early and stopped problems in one area from breaking the rest of the system. More complex bugs, like occasional interface freezing, were identified through user testing and flagged for refinement in future iterations.

## Chapter: 7

# Evaluation

This chapter evaluates the effectiveness of the final interface in meeting the project's goals of improving safety, reducing errors, and supporting decision-making while impaired. The evaluation draws on both heuristic analysis and informal peer feedback to assess usability, clarity, and feature success. While formal user testing was outside the project scope, the use of Nielsen's Usability Heuristics (Wong, 2024) provided a structured method for identifying strengths and areas of improvement early in development.

## 7.1 Heuristic Evaluation

To assess the usability of the Drunk Mode interface, a heuristic evaluation was carried out using Jakob Nielsen's 10 Usability Heuristics – a widely accepted set of principles in Human-Computer Interaction for evaluating user interfaces (Wong, 2024). This method was chosen for its effectiveness in identifying usability problems early, particularly in resource-constrained projects without full-scale user testing.

### 7.1.1 Evaluation Method

The evaluation was carried out by the developer using basic HCI principles and considering the needs of the target user – someone who may be impaired due to alcohol. Each heuristic (e.g. error prevention, visibility of system status) was assessed based on how well the interface follows that rule when someone's actually using it. A 1-5 scale was used to indicate the level of compliance.

- 1 – Poor: Violates the heuristic significantly
- 2 – Weak: Minor adherence; some noticeable issues
- 3 – Fair: Average adherence with room for improvement
- 4 – Good: Mostly meets expectations
- 5 – Excellent: Strongly alignment with the heuristic

### 7.1.2 Scoring Table and Observations

Heuristic	Score	Justification
Visibility of system status	5	The interface clearly shows when the Drunk Mode is on or off through visual toggles and UI changes. Feedback is immediate and visible.
Match between system and real world	4	System uses familiar terms and metaphors, e.g. “Designated Driver”, “Alerts”, “Restrictions”. However, some phrases may be slightly unclear if taken out of context (e.g., “Activity Overview”). Additionally, system is similar to functions such as Do Not Disturb which most users will be familiar with.
User control and freedom	4	Emergency options are always accessible, but once Drunk Mode is activated, settings are intentionally locked, and user set restrictions are put in place. This is by design but may limit user freedom in some cases. However, the user is always able to deactivate Drunk Mode at any time if they wish to do so.
Consistency and standards	5	UI components, layouts, and icons are consistent across screens. The Drunk Mode setting screens all follow the same standard design and is therefore easy for the user to navigate and get used to.
Error prevention	4	The interface actively helps users avoid mistakes through confirmation prompts, restricted access to apps and contact, and automatic typo corrections. Features like the BAC estimator and Drunk Mode suggestions also promote safer behaviour. While not all errors (mainly technical bugs) can be prevented, the system includes several built-in safeguards to suit the user’s impaired state.

Recognition rather than recall	4	The interface minimizes memory load by making key actions visible and accessible. Quick access buttons, large icons, and labels helps users recognise options without needed to remember them. While most elements follow this principle, a few features (like the activity summary or health data) could be explained more thoroughly for the user.
Flexibility and efficiency of use	3	The interface is efficient for its intended user as it supports quick actions (e.g., BAC estimator, emergency contacts) for all users, but lacks advanced shortcuts or personalisation options that could improve efficiency for frequent users.
Aesthetic and minimalist design	5	Visual clutter is removed in Drunk Mode. Only essential apps and buttons remain visible, aligned with cognitive limitations when intoxicated.
Helps users recognise and recover from errors	3	Mistake prevention is strong (e.g., auto-correct), but limited support exists for understanding or correcting an error once it occurs (e.g., no undo feature).
Help and documentation	2	There is no formal help system or tutorial, but the interface is intuitive enough for most users to navigate without confusion.

Table 4: Heuristic Scoring and Observations Table

### 7.1.3 Overall Summary

Overall, the interface demonstrates strong alignment with key usability heuristics, particularly in areas such as system visibility, consistency, and minimalist design. Features like confirmation prompts, simplified navigation, and real-time feedback supports the needs of impaired users. While limitations remain, especially help documentation, recovering from errors, and efficiency, the evaluation suggests that the interface is well-suited for its intended context. Future iterations could address these areas to further strengthen usability under real-world conditions.

## 7.2 Strengths and Achievements

The evaluation findings highlight several strengths of the interface, particularly in supporting users during impaired cognitive states. The system achieved high scores in areas such as system visibility, consistency, and aesthetic design, all of which contribute to a smooth user experience. Features like auto-correction, confirmation prompts, and simplified UI flows help users avoid mistakes and stay safe while using the interface.

Additionally, the system's modular architecture and use of Supabase for real-time updates enabled a responsive interface without compromising the performance of the system. The flexibility of manual and context-aware Drunk Mode activation further reflects the system's adaptability to user needs.

These outcomes collectively suggest that the interface meets its core goal: to provide subtle, intuitive support during moments of reduced judgement without overwhelming the user.

## 7.3 Limitations

While the system was tested regularly throughout development, there were some clear limitations. All testing was done using Expo Go on one iOS device, so it was not able to check how the interface would behave across different devices or platforms. Therefore, layout bugs or platform-specific behaviour issues may not have been caught.

Due to time and ethical constraints, usability testing was kept informal and limited to a few peers. Although the feedback was useful, it didn't fully reflect how real users (especially those who are intoxicated) might use the interface in the moment.

The heuristic evaluation was also done by the developer alone, which has a risk of bias. Some features like typo correction and context-aware prompts are difficult to evaluate thoroughly since they depend on unpredictable intoxicated user behaviour.

There were also a few technical limitations. The system was built as a prototype rather than a fully integrated system-level feature, so things like app restrictions were simulated rather than done through the iOS operating system. There were also some issues with Expo Go's notification scheduling – particularly around background behaviour and consistency. This was due to Expo Go's limitations with background processes and scheduled notifications, which are not fully supported without a custom development build.

## **7.4 Ethic, Legal, Social, and Sustainability Considerations**

### **7.4.1 Ethical**

This interface was designed with user independence and privacy in mind. It does not use any personally identifying information, it also doesn't require users to create an account or login as its pre-built into the user's phones interface, not a third-party app. Any behavioural suggestions (e.g., prompting Drunk Mode activation) are optional and non-intrusive, avoiding any feeling of control. Additionally, the customisation of the system allows users to determine the level of restriction and preferences that they feel comfortable with. The goal is to support users in making safer decisions without judgement or pressure.

### **7.4.2 Legal**

While the interfaces include a BMI-based alcohol recommendation and BAC estimator, it does not claim to provide medical advice. These features are based on general guidelines and are positioned as estimations and not a professional diagnosis. In a real-world OS-level implementation, more rigorous testing and legal compliance (particularly the emergency call functionality and data protections) would be necessary.

### **7.4.3 Social**

The interface was built with accessibility and impaired use in mind, simplifying navigation and decision-making for users who may be under the influence. It avoids judgemental language surrounding intoxication and focuses instead on safety and individual empowerment. Drunk Mode is intended to feel like a helpful tool, not a restriction, and the system is flexible enough to support a wide range of user behaviours and comfort levels.

### **7.4.4 Sustainability**

The system was built with adaptability in mind. Its modular structure and use of the MVVM pattern make it easy to maintain, update, or expand in the future. Although Supabase was used for the prototype, the design could be adapted to work with other databases if needed. Much of the system runs directly on the phone, so it doesn't always need to be connected to the server. This makes it faster and more efficient.

## Chapter: 8

### Conclusion

---

This conclusion reflects on the outcomes of the project, highlighting key achievements, the challenges faced during development, and potential areas for future work. It is an honest assessment of the system's effectiveness, and the learning process involved.

#### 8.1 Learning and Achievements

The aim of this project was to design and develop an aware phone interface that supports safer smartphone use during intoxicated states, whilst focusing on restrictions, and error prevention instead of simply blocking or punishing user behaviour. Through the features such as context-aware prompts, UI interface changes, real-time typo detection, and personalised safety features (e.g., BAC estimator), the system therefore successfully achieves the intended functionality outlined in the objectives of the project.

The functional and non-functional requirements identified in the requirements analysis were largely met, with key features implemented and tested across the system, including Drunk Mode activation, app and contact restrictions, user-customised alerts, safety settings, health recommendations, and activity tracking. The use of MVVM architecture and modular separation of concerns helped ensure the system was maintainable and scalable. Additionally real-time updates were supported through Supabase channels.

The project demonstrated creative problem-solving alongside technical capability across several areas (e.g., global state management, notification scheduling, auto-correction algorithm, and a detection algorithm that combines heuristic factors). Integrating design theory – including distributed cognition and phenomenology – into the design process helped to ensure the interface was not only technically functional but also contextually useful and empathetic to the impaired user experience.

Throughout the process, I developed a much deeper understanding of mobile interface development, particularly with React Native, TypeScript and Supabase. Most importantly, I learned how to manage a project from concept to delivery – ensuring the implementation remained aligned with the initial research, requirements, and user needs. This experience



helped me see how design theory can directly influence user experience decisions, rather than just being theoretical. I also gained valuable skills in translating research insights into actionable features, and using iterative testing and problem-solving to ensure the progress of the project remained on track and functionality worked as expected.

The outcome is not just a working prototype, but an interface shaped by intentional design decisions, with a real consideration for how it might support users in vulnerable states. Additionally, the work contributes to the wider conversation about what inclusive design really means – pushing the idea that we need to think beyond typical use cases and consider how different people, in different situations, interact with technology. It reflects an approach that goes beyond just making something usable, towards making something genuinely supportive, thoughtful, and human-centred.

## 8.2 Challenges Faced

There were several technical challenges encountered throughout development. Managing notification scheduling reliably through Expo Go was a key difficulty, as certain background behaviours like customised alert delivery and persistent notifications were harder to implement without moving to a full development build.

Another key challenge was making sure the user journey flowed logically and didn't allow users to bypass important safeguards unless intended. Ongoing testing revealed small but important issues, like users still being able to change Drunk Mode settings while it was active – which went against the system's safety goals. Fixing these issues required several rounds of refinement, ensuring certain actions were properly locked or restricted at the right moments. This highlighted how important it was not only to build features, but to carefully consider how they interact across different states of use, particularly when user safety and trust are at stake.

Beyond the technical challenges was making sure the design intentions were fully realised in the implementations. Turning theoretical research into functional, reliable features required ongoing adjustment and testing. Overall, these difficulties reinforced the need for iterative testing, flexible problem-solving, and responsiveness to technical constraints.

## 8.3 Future Work

While the current system delivers a functional prototype and proof of concept, there are several areas where the project could be extended or improved with additional time and resources.

- **OS-Level Integration:**

The prototype currently operates as a standalone interface within Expo Go, but the concept is intended as an integrated phone feature. Future work could explore how to actually implement these features directly at the system level, allowing for true app blocking, call restrictions, and deeper access controls that are not possible in the current setup.

- **Expanded Notification Logic & Scheduling:**

Some limitations were encountered around notification handling in Expo Go. A development build (outside of Expo Go) could enable more reliable background scheduling, remote notifications, and smarter alert timing, such as reminders based on previous user actions.

- **Improved Drunk Detection Algorithm:**

The current heuristic scoring system (based on time, shake detection, and typos) could be enhanced with machine learning approaches or user customisation. Deeper user testing could also help fine-tune scoring thresholds for better accuracy.

- **Formal Usability Testing:**

Future iterations should include structured user testing with a wider and more diverse sample. Ethic approval for such testing could allow for the collection of more robust feedback on usability, effectiveness, and emotional impact.

- **Integrating AI and Machine Learning**

The system could benefit from the integration of AI and machine learning across multiple features. For example, it could be used to personalise notifications based on individual usage patterns, improve the accuracy of auto-correction through adaptive learning, or recommend safety features based on previous user behaviour. This could make the interface more intelligent, user-specific, and responsive to changing contexts.

## 8.4 Summary

The project set out to explore how smartphone interfaces could better support users in vulnerable states, specifically during alcohol intoxication. Through research-informed design,

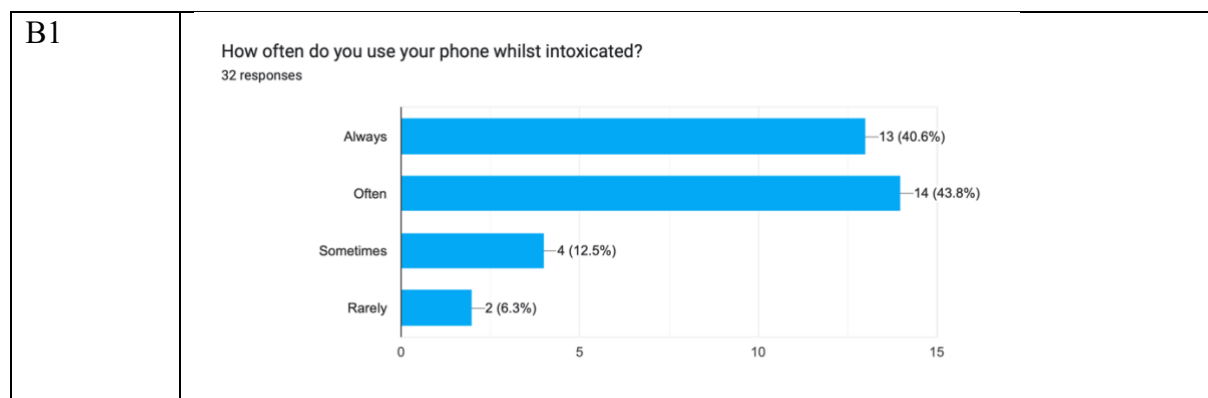
iterative development, and testing, the resulting system delivers a functional prototype that prioritises user safety, error prevention, and cognitive support. The project not only demonstrates technical achievement through its feature set but also looks at the broader questions of inclusive and responsible design. While there remain areas for further development, the work represents a strong foundation for how interfaces might adapt to better serve users across a range of real-life contexts.

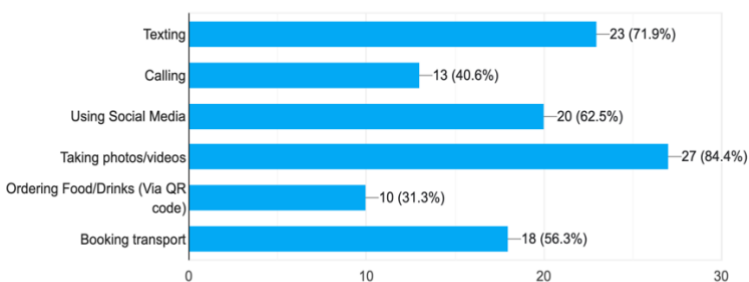
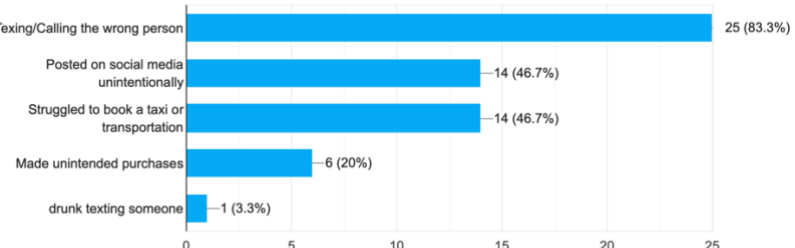
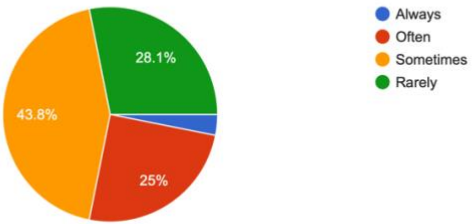
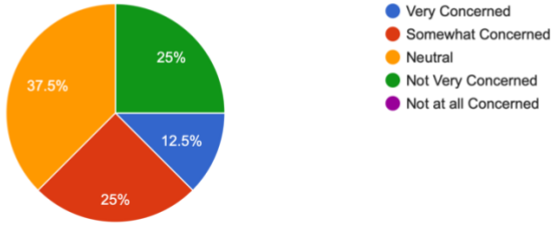
## Appendix

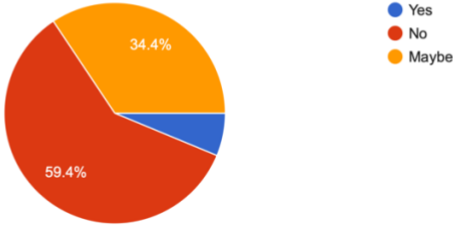
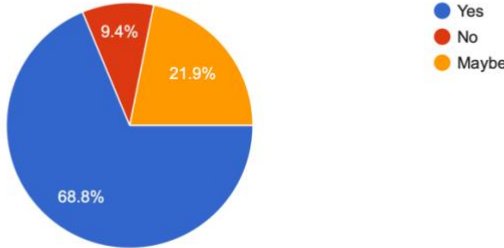
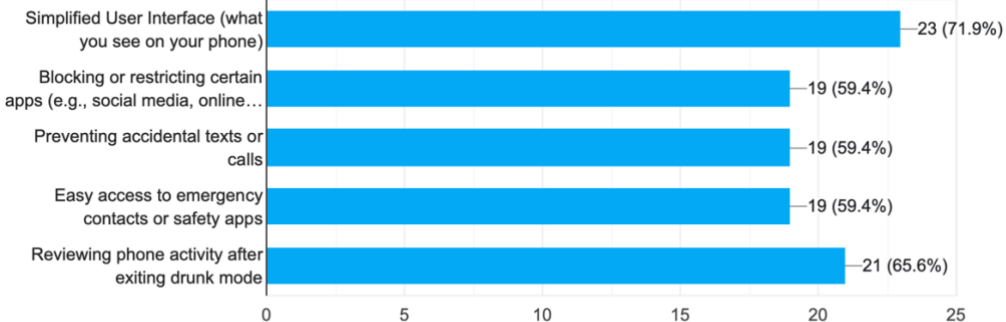
### Appendix A – Existing Systems Comparison Table:

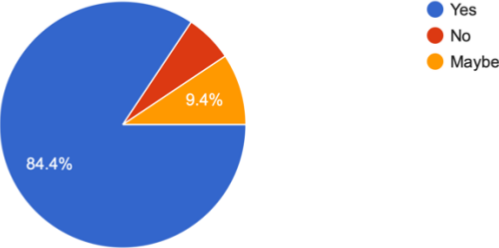
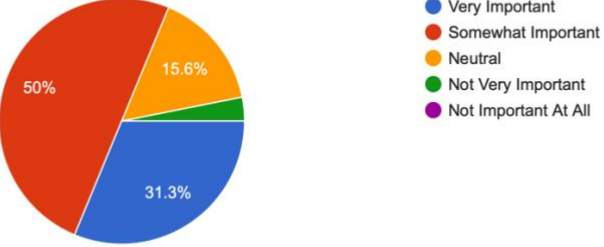

Existing System	Features	Limitations
<b>Design and Assessment of a Personal Breathalyser for Intervention (Srivastava et al., 2021)</b>	Breathalyser linked to smartphone app to track BAC, Colour-coded BAC indicators, Contact notifications.	Small study sample (n=24), Requires external breathalyser, Users must remember to use the device, UI not optimised for intoxicated users.
<b>Drunk Locker: Mobile App (Free Apps for Me – Drunk Locker 2025)</b>	Blocks apps during scheduled times, Pre-set lock windows, Access requires answering challenges, Multilingual support	User must schedule in advance, Easy to bypass (e.g. uninstall), No core system function control
<b>Drunk Mode App (Drunk Mode App 2025)</b>	Pre-select apps to block, Timer + puzzle-lock, Community engagement and events, auto-deactivation	Puzzles can be annoying and overcontrolling for users, can be bypassed, Limited assistive features
<b>Drunk Mode Keyboard (Free Apps for Me – Drunk Mode Keyboard 2025)</b>	Disables texting when active, Customisable keyboard features, swipe gestures and drawing	Only affects messaging, Users can easily switch keyboards, Poor maintenance and buggy according to reviews.

### Appendix B – User Survey and Results:



B2	<p>What are the most common activities you perform on your phone while intoxicated? (Select all that apply)</p> <p>32 responses</p>  <table><thead><tr><th>Activity</th><th>Count</th><th>Percentage</th></tr></thead><tbody><tr><td>Texting</td><td>23</td><td>71.9%</td></tr><tr><td>Calling</td><td>13</td><td>40.6%</td></tr><tr><td>Using Social Media</td><td>20</td><td>62.5%</td></tr><tr><td>Taking photos/videos</td><td>27</td><td>84.4%</td></tr><tr><td>Ordering Food/Drinks (Via QR code)</td><td>10</td><td>31.3%</td></tr><tr><td>Booking transport</td><td>18</td><td>56.3%</td></tr></tbody></table>	Activity	Count	Percentage	Texting	23	71.9%	Calling	13	40.6%	Using Social Media	20	62.5%	Taking photos/videos	27	84.4%	Ordering Food/Drinks (Via QR code)	10	31.3%	Booking transport	18	56.3%
Activity	Count	Percentage																				
Texting	23	71.9%																				
Calling	13	40.6%																				
Using Social Media	20	62.5%																				
Taking photos/videos	27	84.4%																				
Ordering Food/Drinks (Via QR code)	10	31.3%																				
Booking transport	18	56.3%																				
B3	<p>Have you ever encountered any of the following issues when using your phone while intoxicated? (Select all that apply)</p> <p>30 responses</p>  <table><thead><tr><th>Issue</th><th>Count</th><th>Percentage</th></tr></thead><tbody><tr><td>Texting/Calling the wrong person</td><td>25</td><td>83.3%</td></tr><tr><td>Posted on social media unintentionally</td><td>14</td><td>46.7%</td></tr><tr><td>Struggled to book a taxi or transportation</td><td>14</td><td>46.7%</td></tr><tr><td>Made unintended purchases</td><td>6</td><td>20%</td></tr><tr><td>drunk texting someone</td><td>1</td><td>3.3%</td></tr></tbody></table>	Issue	Count	Percentage	Texting/Calling the wrong person	25	83.3%	Posted on social media unintentionally	14	46.7%	Struggled to book a taxi or transportation	14	46.7%	Made unintended purchases	6	20%	drunk texting someone	1	3.3%			
Issue	Count	Percentage																				
Texting/Calling the wrong person	25	83.3%																				
Posted on social media unintentionally	14	46.7%																				
Struggled to book a taxi or transportation	14	46.7%																				
Made unintended purchases	6	20%																				
drunk texting someone	1	3.3%																				
B4	<p>How often do you feel frustrated or confused when trying to use your phone while intoxicated?</p> <p>32 responses</p>  <table><thead><tr><th>Frequency</th><th>Percentage</th></tr></thead><tbody><tr><td>Always</td><td>3.1%</td></tr><tr><td>Often</td><td>25%</td></tr><tr><td>Sometimes</td><td>43.8%</td></tr><tr><td>Rarely</td><td>28.1%</td></tr></tbody></table>	Frequency	Percentage	Always	3.1%	Often	25%	Sometimes	43.8%	Rarely	28.1%											
Frequency	Percentage																					
Always	3.1%																					
Often	25%																					
Sometimes	43.8%																					
Rarely	28.1%																					
B5	<p>How concerned are you about the potential consequences of using your phone while intoxicated (e.g., safety risks, social embarrassment, financial losses)?</p> <p>32 responses</p>  <table><thead><tr><th>Concern Level</th><th>Percentage</th></tr></thead><tbody><tr><td>Very Concerned</td><td>12.5%</td></tr><tr><td>Somewhat Concerned</td><td>25%</td></tr><tr><td>Neutral</td><td>37.5%</td></tr><tr><td>Not Very Concerned</td><td>25%</td></tr><tr><td>Not at all Concerned</td><td>0%</td></tr></tbody></table>	Concern Level	Percentage	Very Concerned	12.5%	Somewhat Concerned	25%	Neutral	37.5%	Not Very Concerned	25%	Not at all Concerned	0%									
Concern Level	Percentage																					
Very Concerned	12.5%																					
Somewhat Concerned	25%																					
Neutral	37.5%																					
Not Very Concerned	25%																					
Not at all Concerned	0%																					

B6	<p>Have you ever felt unsafe or unable to complete an essential task (e.g., contacting someone, booking a ride) due to your phones layout or design while intoxicated?</p> <p>32 responses</p>  <table><thead><tr><th>Response</th><th>Percentage</th></tr></thead><tbody><tr><td>Yes</td><td>6.2%</td></tr><tr><td>No</td><td>59.4%</td></tr><tr><td>Maybe</td><td>34.4%</td></tr></tbody></table>	Response	Percentage	Yes	6.2%	No	59.4%	Maybe	34.4%										
Response	Percentage																		
Yes	6.2%																		
No	59.4%																		
Maybe	34.4%																		
B7	<p>If answer is yes please explain...</p> <p>1 response</p> <div>I have to close one eye to be able to see what i'm texting</div>																		
B8	<p>Would you be interested in a customisable "drunk mode" focus setting for your phone?</p> <p>32 responses</p>  <table><thead><tr><th>Response</th><th>Percentage</th></tr></thead><tbody><tr><td>Yes</td><td>68.8%</td></tr><tr><td>No</td><td>9.4%</td></tr><tr><td>Maybe</td><td>21.9%</td></tr></tbody></table>	Response	Percentage	Yes	68.8%	No	9.4%	Maybe	21.9%										
Response	Percentage																		
Yes	68.8%																		
No	9.4%																		
Maybe	21.9%																		
B9	<p>Which features would you find most helpful in a drunk mode setting? (Select all that apply)</p> <p>32 responses</p>  <table><thead><tr><th>Feature</th><th>Count</th><th>Percentage</th></tr></thead><tbody><tr><td>Simplified User Interface (what you see on your phone)</td><td>23</td><td>71.9%</td></tr><tr><td>Blocking or restricting certain apps (e.g., social media, online...)</td><td>19</td><td>59.4%</td></tr><tr><td>Preventing accidental texts or calls</td><td>19</td><td>59.4%</td></tr><tr><td>Easy access to emergency contacts or safety apps</td><td>19</td><td>59.4%</td></tr><tr><td>Reviewing phone activity after exiting drunk mode</td><td>21</td><td>65.6%</td></tr></tbody></table>	Feature	Count	Percentage	Simplified User Interface (what you see on your phone)	23	71.9%	Blocking or restricting certain apps (e.g., social media, online...)	19	59.4%	Preventing accidental texts or calls	19	59.4%	Easy access to emergency contacts or safety apps	19	59.4%	Reviewing phone activity after exiting drunk mode	21	65.6%
Feature	Count	Percentage																	
Simplified User Interface (what you see on your phone)	23	71.9%																	
Blocking or restricting certain apps (e.g., social media, online...)	19	59.4%																	
Preventing accidental texts or calls	19	59.4%																	
Easy access to emergency contacts or safety apps	19	59.4%																	
Reviewing phone activity after exiting drunk mode	21	65.6%																	

B10	<p>Would you like to receive notifications or reminders while using drunk mode (e.g., reminders to drink water, check transportation, how much money you have spent)?</p> <p>32 responses</p>  <table border="1"> <thead> <tr> <th>Response</th> <th>Percentage</th> </tr> </thead> <tbody> <tr> <td>Yes</td> <td>84.4%</td> </tr> <tr> <td>No</td> <td>6.2%</td> </tr> <tr> <td>Maybe</td> <td>9.4%</td> </tr> </tbody> </table>	Response	Percentage	Yes	84.4%	No	6.2%	Maybe	9.4%				
Response	Percentage												
Yes	84.4%												
No	6.2%												
Maybe	9.4%												
B11	<p>How important is it for you to be able to customise the settings of drunk mode to suit your needs?</p> <p>32 responses</p>  <table border="1"> <thead> <tr> <th>Response</th> <th>Percentage</th> </tr> </thead> <tbody> <tr> <td>Very Important</td> <td>31.3%</td> </tr> <tr> <td>Somewhat Important</td> <td>50%</td> </tr> <tr> <td>Neutral</td> <td>15.6%</td> </tr> <tr> <td>Not Very Important</td> <td>2.1%</td> </tr> <tr> <td>Not Important At All</td> <td>1.0%</td> </tr> </tbody> </table>	Response	Percentage	Very Important	31.3%	Somewhat Important	50%	Neutral	15.6%	Not Very Important	2.1%	Not Important At All	1.0%
Response	Percentage												
Very Important	31.3%												
Somewhat Important	50%												
Neutral	15.6%												
Not Very Important	2.1%												
Not Important At All	1.0%												
B12	<p>How would you prefer to activate drunk mode on your phone?</p> <p>32 responses</p>  <table border="1"> <thead> <tr> <th>Response</th> <th>Percentage</th> </tr> </thead> <tbody> <tr> <td>Manual toggle in settings</td> <td>68.8%</td> </tr> <tr> <td>Pre-scheduled activation (e.g., before an event or night out)</td> <td>28.1%</td> </tr> <tr> <td>both options</td> <td>3.1%</td> </tr> </tbody> </table>	Response	Percentage	Manual toggle in settings	68.8%	Pre-scheduled activation (e.g., before an event or night out)	28.1%	both options	3.1%				
Response	Percentage												
Manual toggle in settings	68.8%												
Pre-scheduled activation (e.g., before an event or night out)	28.1%												
both options	3.1%												
B13	<p>Do you have any additional suggestions or ideas for features that a drunk mode should include?</p> <p>5 responses</p> <p>An "Are you sure you want to..." button pop-up before confirming a specific function on your phone while in drunk mode or during a specific event or timeframe. To allow users to experience an interference in general practices to interrupt their thinking in hopes of helping them realise.</p> <p>feature to block off sending messages to certain people</p> <p>No</p> <p>Friends (who you select) can see your location when in drunk mode and vice versa</p>												

## Appendix C – Implementation – Code and Screenshots:

C1	<pre> export const saveSafetySettings = async (   emergencyContactId: string null,   driverId: string null,   transportMode: string null, ): Promise&lt;Boolean&gt; =&gt; {   const {error} = await supabase.from("safety_settings").upsert ([   {     emergency_contact_id:emergencyContactId,     designated_driver_id:driverId,     transport_mode:transportmode,   }, ]) ; </pre>
C2	
C3	<pre> data.forEach ((alert) =&gt; {   Notifications.scheduleNotificationsAsync({     content: {title: alert.name, body: alert.message },     trigger: null,   });   logTrackingEvent({     event_type: "notification_recieved",     event_detail: alert.name,     notification_content: alert.message,     timestamp: new Date(). toISOString(),   }); }); </pre>
C4	



## Appendix D – Anonymous User Testing Results

Participant	Q1: How did you find navigation?	Q2: Any confusing features?	Q3: How did you feel using the system?	Q4: Were there any surprises?	Q5: Do you have any suggestions?
P1	Logical and Clear to follow, nice simple layout and intuitive.	Nothing that wouldn't have been eventually figured out.	Reassured, Helpful, Protected	The BAC being based on your previously inputted BMI, can call emergency contact/driver easily.	Make the BAC wording not abbreviated to users know what it means.
P2	Navigation was simple	Didn't know what BAC stands for but person who owns the phone would understand their own phone.	Worry me in a good way, to keep me informed of my alcohol levels and if things were going too far.	Calling your driver and emergency contact buttons are really useful.	No suggestions.
P3	Found it clear where everything was. Simple flow.	Unsure at first how restrictions worked, but after triggering Drunk Mode it was made very clear.	Safe, and Feelings considered.	The confirmation prompt for disabling drunk mode and calling 999, good for preventing mistakes.	Maybe a quick explanation pop-up for first time users of the feature – however there were explanations on each settings page which is good.

## Appendix E – Risk Assessment Analysis

Risk Description	Impact	Likelihood	Impact Rating	Prevention Strategy
Core functionality difficulties	Delay in development or inability to deliver working prototype	Medium	High	Follow agile development, break tasks down into smaller chunks

Prototype bugs or instability	Reduced usability/ feature failure, affecting testing and demoing	Medium	High	Conduct iterative testing and debugging; use version control.
Time management issues	Missing internal deadlines or rushing implementations	Medium	High	Use Notion to plan ahead and track weekly goals and progress.
Misalignment between theory and implementation	Features may not reflect design goals or literature findings	Low	Medium	Revisit theoretical framework during implementation phases/ ask for supervisor input.
Platform Limitations	Some planned features may be constrained by Expo Go limitations or third-party library issues (e.g., notification scheduling, real-time sync)	Medium	Medium	Research limitations and use workarounds or adjust features accordingly.
Ethical concerns around user testing	Risk of unintentionally gathering identifiable or sensitive feedback despite aiming for anonymous responses	Low	High	Use fully anonymous surveys and information feedback, avoid personal or sensitive questions, don't store any personal information.

## References

Apple (n.d.). Use Focus on your iPhone or iPad. Available at: <https://support.apple.com/en-gb/guide/iphone/iphd6288a67f/ios> (Accessed: 26 April 2025).

AsyncStorage (2024). AsyncStorage Documentation (React Native Community). Available at: <https://react-native-async-storage.github.io/async-storage/> (Accessed: 25 February 2025)

Callstack (2024). React Native Paper Documentation. Available at: <https://reactnativepaper.com/> (Accessed: 10 December 2024)

Clarkson, J. et al. (eds.) (2003) Inclusive Design, SpringerLink. Available at: <https://link.springer.com/book/10.1007/978-1-4471-0001-0> (Accessed: 27 October 2024).

Dourish, P., 1999. Embodied interaction: Exploring the foundations of a new approach to HCI. *Work*, 1(1), pp.1-16. (Accessed: 26 March 2025)

Drunk Mode Company (2025) Drunk Mode Locker Mobile App, Drunk mode. Available at: <https://www.drunkmode.app/?ref=producthunt> (Accessed: 2 November 2024).

Expo (2024). Expo Documentation. Available at: <https://docs.expo.dev/> (Accessed: 10 December 2024)

Expo Notifications (2024). Using Notifications with Expo. Available at: <https://docs.expo.dev/versions/latest/sdk/notifications/> (Accessed: 15 February 2025)

Expo Sensors (2024). Expo Sensors Documentation. Available at: <https://docs.expo.dev/versions/latest/sdk/accelerometer/> (Accessed: 20 March 2025)

Fast-Levenshtein (2024). Levenshtein Distance Package Documentation. Available at: <https://www.npmjs.com/package/fast-levenshtein> (Accessed: 17 March 2025)

Fillmore, M.T. (2003). Alcohol-induced impairment of behavioural control: Effects on inhibition, error detection, and correction. *Neuropsychology Review*, 13(2), 75–96. <https://www.jsad.com/doi/abs/10.15288/jsa.2003.64.687> (Accessed: 25 October 2024)

Free Apps for Me – Drunk Locker (2025) 9 best drunk mode apps for Android and iPhone in 2025: Freeappsforme - free apps for Android and IOS, Freeappsforme. Available at: [https://freeappsforme.com/drunken-mode-apps/#Drunk\\_Locker](https://freeappsforme.com/drunken-mode-apps/#Drunk_Locker) (Accessed: 2 November 2024).

Free Apps for Me – Drunk Mode Keyboard (2025) 9 best drunk mode apps for Android and iPhone in 2025: Freeappsforme - free apps for Android and IOS, Freeappsforme. Available at: [https://freeappsforme.com/drunken-mode-apps/#Drunk\\_Locker](https://freeappsforme.com/drunken-mode-apps/#Drunk_Locker) (Accessed: 2 November 2024).

GitHub Copilot (2024). GitHub Copilot Documentation. Available at: <https://docs.github.com/en/copilot> (Accessed: 20 November 2024).

Harvey, A.J., Kneller, W. and Campbell, A.C. (2013) 'The effects of alcohol intoxication on attention and memory for visual scenes', *Memory*, 21(8), pp. 969–980. doi:10.1080/09658211.2013.770033. (Accessed: 20 October 2024)

McKay, E.N. (2013) *UI is Communication: How to Design Intuitive, User Centered Interfaces by Focusing on Effective Communication*, Google Books. Available at: [https://books.google.co.uk/books?hl=en&lr=&id=wNozxtKuOKcC&oi=fnd&pg=PP1&dq=How%2Bto%2BDesign%2BIntuitive%2C%2BUser%2BCentered%2BInterfaces%2Bby%2BFocusing%2Bon%2BEffective%2BCommunication.&ots=va5cm9CejY&sig=uJtQ9K\\_jz-1ljTWKUSTHQcX0Jvo&redir\\_esc=y#v=onepage&q=How%20to%20Design%20Intuitive%2C%20User%20Centered%20Interfaces%20by%20Focusing%20on%20Effective%20Communication.&f=false](https://books.google.co.uk/books?hl=en&lr=&id=wNozxtKuOKcC&oi=fnd&pg=PP1&dq=How%2Bto%2BDesign%2BIntuitive%2C%2BUser%2BCentered%2BInterfaces%2Bby%2BFocusing%2Bon%2BEffective%2BCommunication.&ots=va5cm9CejY&sig=uJtQ9K_jz-1ljTWKUSTHQcX0Jvo&redir_esc=y#v=onepage&q=How%20to%20Design%20Intuitive%2C%20User%20Centered%20Interfaces%20by%20Focusing%20on%20Effective%20Communication.&f=false) (Accessed: 15 November 2024).

Meta (2024). *React Native Documentation*. Available at: <https://reactnative.dev/docs/environment-setup> (Accessed: 10 December 2024)

Norman, D.A. (2013). *The Design of Everyday Things: Revised and Expanded Edition*. New York: Basic Books. (Accessed: 15 October 2024)

Perry, M. (2003). Distributed cognition. In: Carroll, J.M. (ed.) *HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science*. San Francisco: Morgan Kaufmann, pp. 193–223. Available at: <https://doi.org/10.1016/B978-155860808-5/50008-3> (Accessed: 25 March 2025).

Picard, R. W. (2000). *Affective computing*. Cambridge, MA: MIT Press. (Accessed: 04 November 2024)

Pokinko, T. (2015) *Designing mobile applications for older adults with cognitive decline: Inclusive design considerations for User Experience Designers*, *Designing Mobile Applications for older adults with cognitive decline: inclusive design considerations for user experience designers - Open Research Repository*. Available at: <https://openresearch.ocadu.ca/id/eprint/261/> (Accessed: 29 October 2024).

React (2024). *React Documentation*. Available at: <https://react.dev/> (Accessed: 10 December 2024)

Srivastava, S. et al. (2021) 'Wingman: Your Digital Drinking Companion, India HCI 2021, pp. 84–89. doi:10.1145/3506469.3506482. (Accessed: 28 October 2024)

Supabase (2024). *Supabase Documentation*. Available at: <https://supabase.com/docs> (Accessed: 10 January 2024)

Trub, L. and Starks, T.J., 2017. Textual healing: Proof of concept study examining the impact of a mindfulness intervention on smartphone behavior. *Mindfulness*, 8(5), pp.1225–1235. Available at: <https://doi.org/10.1007/s12671-017-0697-y>. (Accessed: 25 October 2024)

Trub, L. et al. (2021) 'Drunk Texting: When the Phone Becomes a Vehicle for Emotional Dysregulation and Problematic Alcohol Use', *Substance Use & Misuse*, 56(12), pp. 1815–1824. doi: 10.1080/10826084.2021.1954027. (Accessed: 25 October 2024)

Wong, E. (2024). How to conduct a heuristic evaluation for usability in HCI and information visualization. The Interaction Design Foundation. Available at: <https://www.interaction-design.org/literature/article/heuristic-evaluation-how-to-conduct-a-heuristic-evaluation> (Accessed: 04 December 2024).

Yang, S.-C., Lee, T.-L. and Feng, T.-T. (2018). User experience of mobile application's interface: Measurement development. Proceedings of the 5th Multidisciplinary International Social Networks Conference (MISNC '18), Article 2, pp. 1–5. New York, NY: Association for Computing Machinery. Available at: <https://doi.org/10.1145/3227696.3227698> (Accessed: 05 December 2024)

