



A MOBILE HEALTHCARE MONITORING AND RECOMMENDATION SYSTEM

An Integrated mHealth Platform Using Sensor-Enabled Data and Cloud
Deployment

SEPTEMBER 1, 2025
SUNIDHI ABHANGE
CM3050: Mobile Development

ABSTRACT



TrustCura+ is a mobile health application designed to support users in managing day-to-day health concerns through integrated monitoring and personalized wellness guidance. Unlike many existing solutions that focus on narrow functions such as fitness tracking or symptom checking, **TrustCura+** combines structured symptom logging, weather-aware recommendations, secure communication with healthcare professionals, and GPS-based access to nearby facilities into a single platform. Sensor integration, including an accelerometer for activity tracking, camera for symptom capture and document scanning, and GPS for location services, further enhances functionality. The system was implemented using React Native (Expo) for the mobile interface, Node.js for backend services, and SQLite3 for data management, with deployment on AWS EC2 to ensure scalability, security, and availability. This integrated design demonstrates how computer science methods can be applied to create an innovative, lightweight, and practical mHealth solution.

Table of Contents

| | |
|---|----|
| 1. Introduction | 6 |
| 1.1 Background | 6 |
| 1.2 Problem Statement | 6 |
| 1.3 Solution | 6 |
| 1.4 Objectives | 7 |
| 1.5 Scope | 7 |
| 2. Literature Review / Research Background | 8 |
| 2.1 Overview of mHealth Trends | 8 |
| 2.2 Gaps in Current mHealth Solutions | 8 |
| 2.3 Research Findings in mHealth | 8 |
| 2.3.1 Sensor-Driven Health Monitoring | 8 |
| 2.3.2 Contextual Nudges and Behavioural Triggers | 9 |
| 2.3.3 Health Outcomes from Sensor-Based mHealth Platforms | 9 |
| 2.4 Influence on <i>TrustCura+</i> Design | 9 |
| 2.5 Research Insights to <i>TrustCura+</i> Feature Mapping | 10 |
| 2.6 Summary and Link to Evaluation | 10 |
| 3. Concept Development | 11 |
| 3.1 Idea Formation | 11 |
| 3.2 User Personas | 11 |
| 3.3 Requirements: Epics and User Stories | 12 |
| 3.4 Main Concept | 13 |
| 3.5 System Framework | 14 |
| 3.6 Key Metrics and calculations | 15 |
| 3.7 Benefits of <i>TrustCura+</i> | 15 |
| 3.8 Linking Requirements to Conceptual Design | 15 |
| 4. System Design | 16 |
| 4.1 Introduction to Design | 16 |
| 4.2 System Architecture | 17 |
| 4.3 Database Design | 18 |
| 4.3.1 Overview Entity relationship | 18 |
| 4.3.2 User Identity, Authentication and Doctor Verification | 19 |
| 4.3.3 Health Logging and Daily Well-being Management Entities | 20 |
| 4.3.4 Communication and Appointment management Entity | 21 |

| | | |
|-----------|---|-----------|
| 4.3.5 | Personalised Recommendation and Alert Management entity | 22 |
| 4.4 | Information Architecture (IA) | 23 |
| 4.5 | Application Flow | 24 |
| 4.5.1 | Application Flow Users | 24 |
| 4.5.2 | Application Flow Doctors | 25 |
| 4.6 | Security and Privacy Design | 26 |
| 4.7 | Design Summary | 26 |
| 5. | Prototyping | 27 |
| 5.1 | Introduction (Definition and Purpose)..... | 27 |
| 5.2 | Objectives and Critical Features Selected | 27 |
| 5.3 | Technical Setup for Prototype | 27 |
| 5.4 | User Interface (UI/UX Design) | 27 |
| 5.5 | Screenshots of the Developed Prototype..... | 36 |
| 5.6 | Prototype Testing (Method and Participants) | 37 |
| 5.7 | Outcomes (Changes proposed for Development)..... | 38 |
| 5.8 | Limitations | 38 |
| 6. | User Feedback | 39 |
| 6.1 | Feedback on Wireframes | 39 |
| 6.2 | Prototype Feedback Collection | 40 |
| 6.3 | Target Users..... | 40 |
| 6.4 | Feedback by Epic and User Story..... | 40 |
| 6.5 | Findings | 41 |
| 6.6 | Summary..... | 41 |
| 7. | Development | 42 |
| 7.1 | Introduction | 42 |
| 7.2 | Project Structure | 43 |
| 7.3 | Technology Stack..... | 44 |
| 7.4 | Core Features and Implementation..... | 45 |
| 7.5 | Theme Switching (Accessibility – Light/Dark Mode) | 52 |
| 7.6 | Security: Encryption of API Requests | 52 |
| 7.7 | Deployment..... | 52 |
| 7.8 | Summary..... | 52 |
| 8. | Unit Testing | 53 |
| 8.1 | Unit Testing Approach | 53 |
| 8.2 | Unit Test Framework | 53 |

| | | |
|-------------|--|-----------|
| 8.3 | Unit Testing Execution..... | 54 |
| 8.4 | Representative Unit Test Cases | 54 |
| 8.5 | Unit Testing Results | 54 |
| 8.6 | Unit Test Coverage and Limitations..... | 54 |
| 9. | Integration Process and Testing..... | 55 |
| 9.1 | Approach..... | 55 |
| 9.2 | Test Scenarios | 55 |
| 9.3 | Interpretation..... | 55 |
| 9.4 | Limitations..... | 55 |
| 9.5 | Summary of Results..... | 55 |
| 10. | Evaluation..... | 56 |
| 10.1 | Evaluation Criteria | 56 |
| 10.2 | Comparison with Existing Solutions..... | 56 |
| 10.3 | Innovation & Differentiation..... | 56 |
| 10.4 | Limitations of Current System..... | 57 |
| 10.5 | Future Enhancements | 57 |
| 10.6 | Critical Evaluation | 57 |
| 11. | Conclusion | 58 |
| 12. | References..... | 59 |
| 13. | Appendices..... | 61 |
| 13.1 | Appendix A – Research Insights to Feature Mapping | 61 |
| 13.2 | Appendix B – Detailed User Personas | 62 |
| B.1 | Primary Persona: “Student Aisha” | 62 |
| B.2 | Secondary Persona: “Office Worker Daniel” | 63 |
| B.3 | Tertiary Persona: “Doctor Emily” | 64 |
| 13.3 | Appendix C – Detailed User Stories | 65 |
| 13.4 | Appendix D – Key Metrics and calculations | 67 |
| 13.5 | Appendix E – Traceability Matrix..... | 68 |
| 13.6 | Appendix F – Development – Extended Code Snippets..... | 69 |
| F.1 | – Prototype Technical Configuration..... | 69 |
| F.2 | OTP Authentication | 70 |
| F.3 | Doctor Credentials Upload + OCR..... | 71 |
| F.4 | Daily well-being Logging | 72 |
| F.5 | Symptom Tracking | 72 |
| F.6 | Messaging (Socket.io Extended) | 73 |

| | |
|---|-----------|
| F.7 Utility Functions..... | 73 |
| 13.7 Appendix G – Screen shots | 74 |
| G.1 Onboarding Screens | 74 |
| G.2 Authentication and Doctor Verification Screens | 74 |
| G.3 Add symptoms | 75 |
| G.4 View symptoms and upcoming appointment..... | 75 |
| G.5 Mark recovered and view recovery Plan | 75 |
| G.6 Step Tracker using Accelerometer | 75 |
| G.7 Trends..... | 76 |
| 13.8 Appendix H – User Feedback | 77 |
| H.1 Survey Questions | 77 |
| H.1.1 Key Findings | 77 |
| H.1.2 Adjustments Based on Feedback..... | 77 |
| H.2 Feedback Limitations | 78 |
| H.3 Prototype Adjustments: Before vs After Feedback | 79 |
| 13.9 Appendix I – Detailed Unit testing Evidence | 80 |
| I.1 Cycle -by-Cycle Test Log (Red-Green-Refactor)..... | 80 |
| I.2 Expandable Module Test Tales | 82 |
| I.3 Complete Test command references | 83 |
| I.4 Extended Library Reference | 83 |
| I.5 Test Coverage..... | 84 |
| 13.10 Appendix J – Integration Test results | 85 |
| J.1 Integration Test Results..... | 85 |
| J.2 Key Bugs Identified and Fixed | 85 |
| 13.11 Appendix K – Evaluation..... | 86 |
| K.1 Comparison with Existing Apps: | 86 |
| K.2 Innovation & Differentiation | 87 |
| K.3 Proposed developments include: | 88 |
| K.4 Critical Evaluation..... | 89 |

Total Word Count: 6957

(Excluding Abstract, TOC, References and Appendix)

1. Introduction

Word count: 501

1.1 Background

Mobile healthcare applications have become essential tools for individuals aiming to monitor health, manage wellness routines, locate medical resources, and communicate with healthcare providers. These tools are particularly useful for tracking everyday health concerns such as fever, flu, headache, or fatigue, while offering users convenient access to advice.

Despite their growing use, many existing apps provide only basic functionality such as symptom logging or fitness tracking and often operate in isolation. They rarely integrate environmental data, personalized recommendations, or secure communication with healthcare professionals (Ventola, 2014). This lack of integration reduces their ability to support timely interventions and informed decision-making.

TrustCura+ is developed to address these limitations. It combines symptom tracking, real-time insights, encrypted doctor–user communication and appointment scheduling in a unified platform. By leveraging smartphone sensors—accelerometer, camera, and GPS—the app provides enhanced personalization and user experience (Piras & Murgia, 2017).

1.2 Problem Statement

Most health apps fail to connect personal health data with environmental context or professional input, limiting their usefulness for proactive care (Boulos et al., 2014). Users may face incomplete monitoring, delays in recognizing health changes, and difficulty interpreting their records. Accessing professional guidance through such apps is also often unreliable (Martinez-Perez, de la Torre-Díez & López-Coronado, 2013).

This project addresses the need for a secure, unified, and scalable platform that integrates tracking, recommendations, and verified professional support.

1.3 Solution

TrustCura+ provides a single, user-friendly mobile platform that enables logging of symptoms with timestamps and severity, real-time personalized suggestions based on symptoms, environment, and location, and secure doctor–user communication through encrypted text, images, and calendar sharing. The app employs device sensors to monitor activity (accelerometer), capture images (camera), and provide location-aware services such as clinic locators and weather-informed advice (GPS). Doctors are verified using OCR-based credential checks, ensuring professionalism.

The innovation lies in unifying structured tracking, adaptive recommendations, sensor-based personalization, and secure communication into one lightweight tool. Unlike many fragmented apps, TrustCura+ provides a coherent, multi-functional solution for everyday health management.

1.4 Objectives

- To design and develop a mobile healthcare app that integrates symptom tracking, health recommendations, and secure communication.
- To utilize device sensors (accelerometer, camera, GPS) for enhanced personalization.
- To deliver real-time health tips informed by symptoms, environment, and nearby healthcare resources.
- To ensure strong security and privacy through encryption.
- To deploy the app on a scalable cloud infrastructure (AWS EC2).

1.5 Scope

Inclusions

- Role-based authentication (user/doctor).
- OCR-based verification of doctor credentials.
- Symptom logging with severity and timestamps.
- Personalized health plans and recommendations.
- Encrypted communication (text, image).
- Appointment scheduling with calendar sharing.
- GPS-based facility locator with ratings and navigation.
- Integration of accelerometer, camera, and GPS.
- Deployment on AWS EC2 for scalability.

Exclusions

- Chronic or critical disease management (e.g., diabetes, cancer).
- Emergency medical response.
- Formal medical diagnosis beyond recommendations.
- Integration with wearable devices (future enhancement).
- In-app payments.

The boundaries outlined define the intended scope of TrustCura+. The following section reviews existing mHealth research and trends that informed the system's design and highlights the gaps this project addresses.

2. Literature Review / Research Background

Word count: 791

2.1 Overview of mHealth Trends

Mobile health (mHealth) applications are increasingly adopted for supporting everyday wellness, encouraging physical activity, improving sleep, managing minor symptoms, and promoting healthier routines. These applications typically combine self-monitoring features, educational resources, and behavioural prompts to help users stay proactive in managing their health (Digital Health Interventions, 2022). Research also highlights that widespread adoption is driven by smartphone accessibility, low cost, and convenience (Deniz-Garcia et al., 2023).

This context establishes the significance of mHealth and demonstrates the potential value of an integrated mobile platform such as **TrustCura+**.

2.2 Gaps in Current mHealth Solutions

Despite growing popularity, many mHealth solutions remain limited in scope. Applications often concentrate on narrow domains—such as step counting, symptom logging, or medication reminders—without offering a fully integrated experience. This separation of functions results in fragmented data, lower user engagement, and restricted insights (Grundy et al., 2022).

For example:

- MyChart provides record-keeping and messaging but lacks self-care functions.
- Ada Health offers AI triage but limited long-term tracking or communication.
- Google Fit focuses on fitness but does not address illness recovery.
- Existing apps lack seamless appointment scheduling within chat/calendar integration

These examples illustrate the fragmented nature of current tools and highlight an opportunity for a unified system that combines tracking, contextual recommendations, and secure communication. **TrustCura+** is designed to fill this gap.

2.3 Research Findings in mHealth

While gaps highlight opportunities, research also provides evidence of what works effectively in mHealth. These findings form the foundation of **TrustCura+**'s design.

2.3.1 Sensor-Driven Health Monitoring

Studies demonstrate that smartphone sensors extend the functionality of mHealth platforms:

- Accelerometers monitor movement, physical activity, sleep, and fall detection (German Journal of Sports Medicine, 2024).
- GPS supports location-based services such as finding healthcare facilities and mapping outdoor activity.
- Cameras enable image capture for remote consultations, including dermatology and ophthalmology (PMC, 2019).

Emerging work on passive sensing shows potential for continuous data collection without user input, supporting long-term monitoring of sleep disorders and mental health (Cornet & Holden, 2018).

These findings confirm that using accelerometers, GPS, and cameras is technically feasible and supports TrustCura+ features in Epics 3, 4, and 7.

2.3.2 Contextual Nudges and Behavioural Triggers

Research shows that integrating environmental data increases user engagement. Weather-based prompts, for instance, significantly improve exercise participation when matched with real-time conditions (University of Minnesota, 2023). Mobile tools that incorporate air quality, temperature, or pollution levels are gaining importance in public health promotion (Climate-focused mobile tools, 2024).

This evidence supports **TrustCura+** Epic 9 (Personalised Recommendations and Alerts), where weather and environmental data are integrated to provide more relevant, context-aware advice

2.3.3 Health Outcomes from Sensor-Based mHealth Platforms

Evidence suggests that when apps combine tracking with personalised, timely, and context-sensitive feedback, users are more likely to follow recommendations and remain engaged (Ghose et al., 2021). These systems also demonstrate potential for reducing unnecessary in-person visits by enabling remote advice, monitoring and structures appointment booking.

This directly justifies the **TrustCura+** integrated design (Epics 3–6), where symptom logging, task management, and secure doctor communication are provided in one coherent platform.

2.4 Influence on *TrustCura+* Design

The reviewed evidence shaped the guiding principles of TrustCura+:

- Integrating Features – Combining symptom tracking, recommendations, and communication (addressing fragmentation).
- Leveraging Sensors – Using accelerometer, GPS, and camera for richer, context-aware health data.
- Adding Contextual Recommendations – Providing weather-based and location-aware guidance to increase adherence.
- Applying Evidence-Based Methods – Designing the system to align with proven approaches to wellness monitoring.
- Enabling Appointments – In-chat booking with calendar view.
- Preparing for Passive Data Collection – Ensuring the platform can later integrate low-effort, continuous monitoring.

These principles demonstrate how findings from Sections 2.3.1–2.3.3 informed TrustCura+ design decisions.

2.5 Research Insights to *TrustCura+* Feature Mapping

TrustCura+ translates insights from existing research into specific features. Prior studies demonstrate that mHealth platforms improve health behaviours, engagement increases when context-aware feedback is included, and sensors enhance accuracy and usability.

In response, TrustCura+ consolidates these findings into one system by integrating symptom tracking, wellness recommendations, activity monitoring, facility locators, doctor–user communication and integrated appointment scheduling.

[A full mapping of research insights to feature implementations is provided in Appendix A.](#)

2.6 Summary and Link to Evaluation

The literature highlights both opportunities and shortcomings in existing mHealth solutions. While individual features such as tracking and environmental nudges have proven effective, their fragmented implementation limits outcomes and engagement.

“TrustCura+ builds directly on these findings by integrating evidence-based features into a unified, scalable platform. This integrated approach, rarely seen in existing apps, is further strengthened by appointment integration, which enhances overall completeness and underpins its innovation.

The impact of these design choices is further examined in Section 10 (Evaluation), where TrustCura+ is compared against leading alternatives to demonstrate its effectiveness and distinctiveness within the mHealth landscape.

3. Concept Development

Word count: 769

3.1 Idea Formation

The development of TrustCura+ is motivated by gaps identified in research and current mHealth applications. Many existing platforms support only narrow functions such as activity tracking or symptom diaries (Boulos et al., 2014), forcing users to rely on multiple apps.

Although sensors such as accelerometers, GPS, and cameras are common in smartphones, their use in health apps remains limited (Grundy et al., 2022). At the same time, contextual nudges such as weather-based prompts can improve adherence to healthy behaviours (University of Minnesota, 2023).

TrustCura+ addresses these gaps by providing a unified healthcare platform for common conditions such as flu, fatigue, or mild fever. It combines symptom tracking, personalised recommendations, doctor verification, appointment scheduling and secure communication within a single system.

3.2 User Personas

To validate requirements and align features with real user needs, three user personas were developed representing the primary stakeholders of TrustCura+.

Student Aisha (Primary User): A 21-year-old undergraduate in Manchester. She experiences stress and fatigue, particularly during exams, and wants an easy way to log wellbeing, receive reminders, and consult doctors securely.

Office Worker Daniel (Secondary User): A 32-year-old marketing executive in Birmingham. He wants to monitor symptoms like headaches or fatigue and receive simple, actionable advice. He values clear summaries and lightweight reminders.

Doctor Emily (Tertiary User): A 42-year-old GP in Leeds. She requires credential verification, access to structured patient logs, and secure communication. She prefers concise dashboards that summarise patient trends.

Detailed persona descriptions are included in Appendix B.

3.3 Requirements: Epics and User Stories

The system requirements were structured into ten epics, each supported by user stories to capture user needs and technical expectations:

- Authentication – OTP-based registration, login, and logout.
- Doctor Verification – Credential upload, OCR checks, profile editing, availability.
- Well-being Logging – Mood, energy, and sleep recording.
- Symptom Tracking – Structured logging with severity and notes.
- Task Management – View, complete, and add tasks.
- Communication – Secure text and image messaging.
- Geo-Location – GPS-based facility finder.
- Trend Analysis – Graphs for symptoms, wellbeing, and tasks.
- Recommendations – Adaptive suggestions based on symptoms and weather.
- Security & Privacy – End-to-end encryption and safe storage.
- Appointments – Calendar sharing and doctor slot booking

These epics formed the bridge between research findings (Section 2) and the conceptual design that follows.

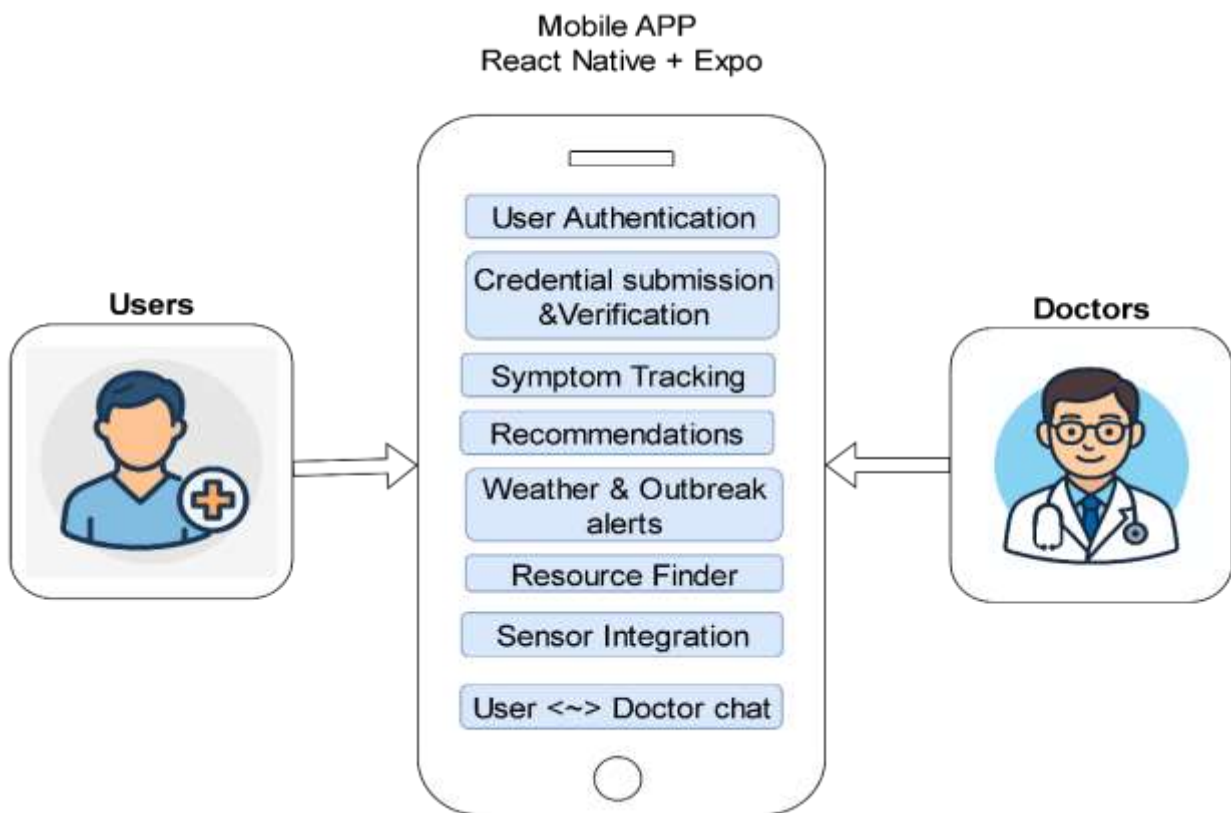
[Detailed user stories with acceptance criteria are provided in Appendix C.](#)

3.4 Main Concept

The main concept of TrustCura+ links users and doctors through symptom logging, secure communication, adaptive recommendations, and sensor integration. Together, these features enable real-time health monitoring and professional guidance (Figure 3.1).

- Users log symptoms and access adaptive advice.
- Doctors review structured logs and provide consultation.
- Sensor integration supports activity, location, and image-based data collection.

Figure 3.1 visualizes these interactions within the **TrustCura+** mobile healthcare ecosystem.

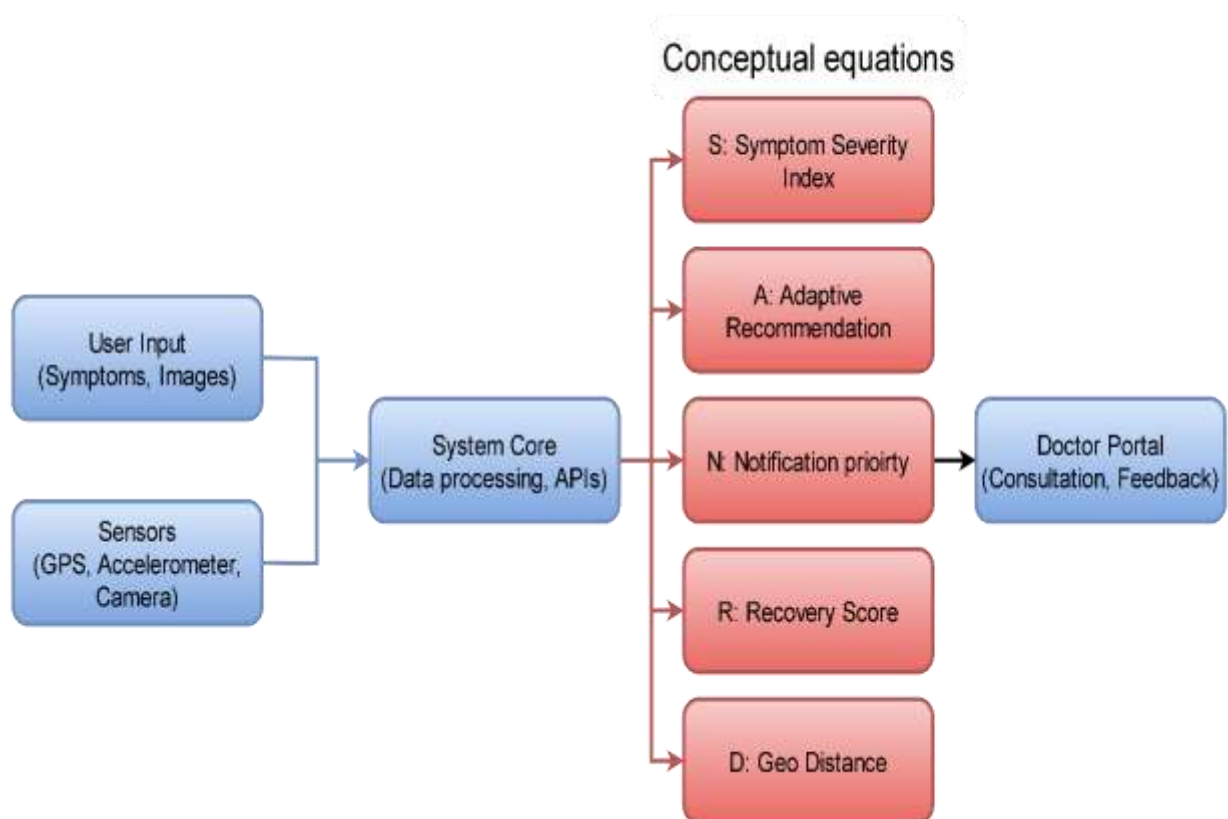


3.5 System Framework

The framework positions TrustCura+ as a user-centered ecosystem (Figure 3.2).

- User Input: Symptoms and images.
- Sensors: GPS, accelerometer, and camera data.
- System Core: Combines user data with external inputs (weather, alerts).
- Key Metrics: Symptom Severity Index (S), Adaptive Recommendation (A), Notification Priority (P), Recovery Score (R), Geo-Distance (d).
- Doctor Portal: Provides access to structured insights, enabling remote consultations.

Figure 3.2: System framework showing how TrustCura+ integrates user inputs, sensor data, environmental context, and professional feedback into a unified mHealth system.



3.6 Key Metrics and calculations

TrustCura+ uses core metrics to support analysis and recommendations:

- Symptom Severity Index (S): Weighted score of logged symptoms.
- Adaptive Recommendation (A): Adjusts baseline plan with environmental data.
- Notification Priority (P): Ranks alerts by severity, outbreak risk, and environment.
- Recovery Score (R): Tracks progress using task adherence and symptom reduction.

Sensors enable these functions: accelerometer (activity), GPS (facility location), and camera (image capture). [*Full derivations are in Appendix D.*](#)

3.7 Benefits of TrustCura+

For Users:

- One platform to log health issues, receive recommendations, and connect with doctors, book an appointment directly from the chat.
- Example: A user logs a headache (S), receives hydration reminders adapted to local temperature (A), and shares a medication photo for doctor feedback.

For Doctors:

- Access to structured logs and trends improves consultation quality compared to informal messaging tools (Martinez-Perez et al., 2013).

For Computing:

- Combines client–server design, sensor integration, and cloud scalability.
- AWS EC2 supports deployment and reliability, while React Native ensures cross-platform compatibility.

By integrating symptom tracking, recommendations, communication, and sensor insights into one lightweight system, TrustCura+ addresses fragmentation in mHealth and provides a clear contribution aligned with its epics.

3.8 Linking Requirements to Conceptual Design

A traceability matrix ensures that requirements map to design. It links epics and user stories with implemented features and conceptual elements, confirming alignment between user needs, system implementation, and modelling. [*The full matrix is in Appendix E.*](#)

4. System Design

Word count: 962

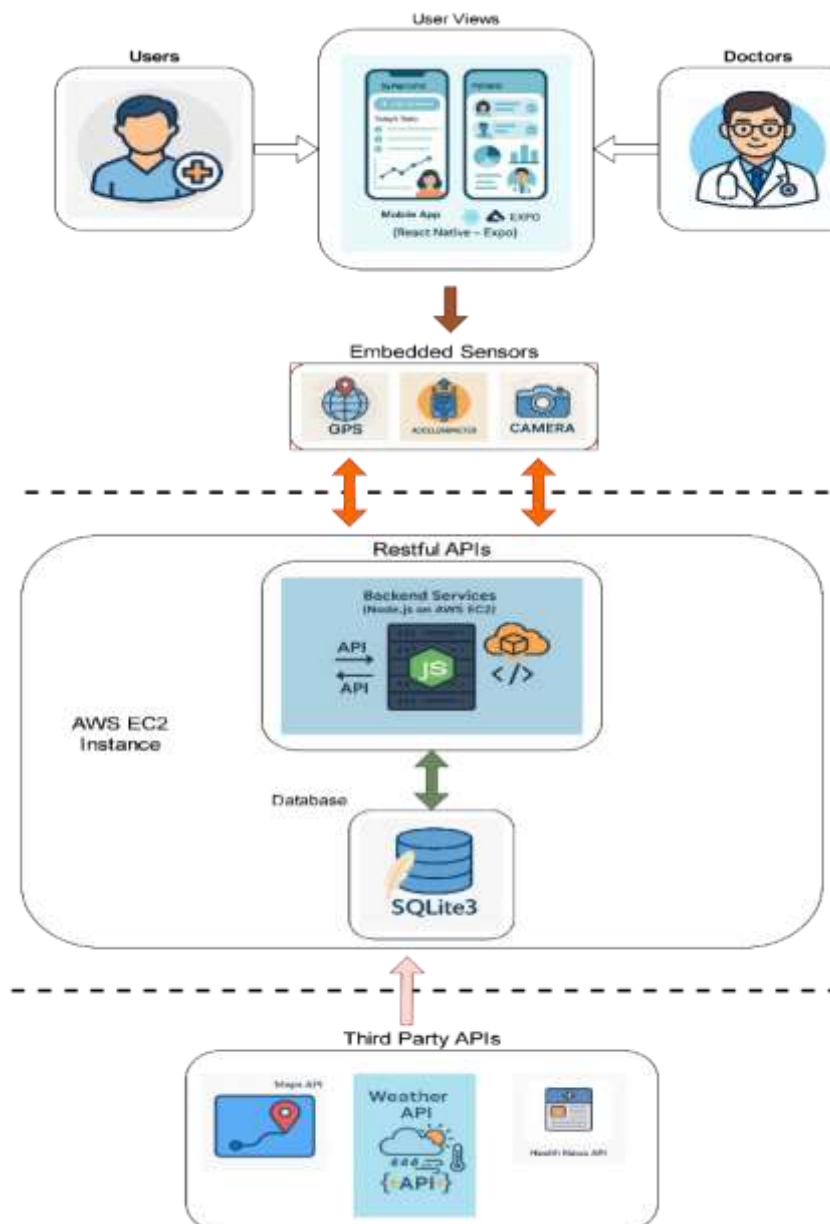
4.1 Introduction to Design

The design phase of *TrustCura+* transforms functional requirements into technical and user-centred artefacts. It ensures every epic and user story is implemented through consistent architecture, database schema, workflows, and security measures. The design is structured across six layers: (1) System Architecture, outlining the client–server model with React Native, Node.js, SQLite3, and AWS EC2; (2) Database Design, structuring user, doctor, health, and communication entities; (3) Application Flow, modeling end-to-end processes such as authentication, verification, and facility search; (4) Information Architecture, defining navigation for users and doctors; (5) User Interface, presenting wireframes; and (6) Security, ensuring compliance and privacy.

4.2 System Architecture

TrustCura+ adopts a client–server architecture integrating mobile applications, backend services, databases, and third-party APIs (Figure 4.1). The mobile client, developed using React Native (Expo), supports Android and iOS, delivering OTP login, health logging, task tracking, chat, and facility search. Device sensors—GPS for location, accelerometer for activity monitoring, and camera for document upload—are embedded to enhance functionality.

Figure 4.1: System Architecture - *TrustCura+*



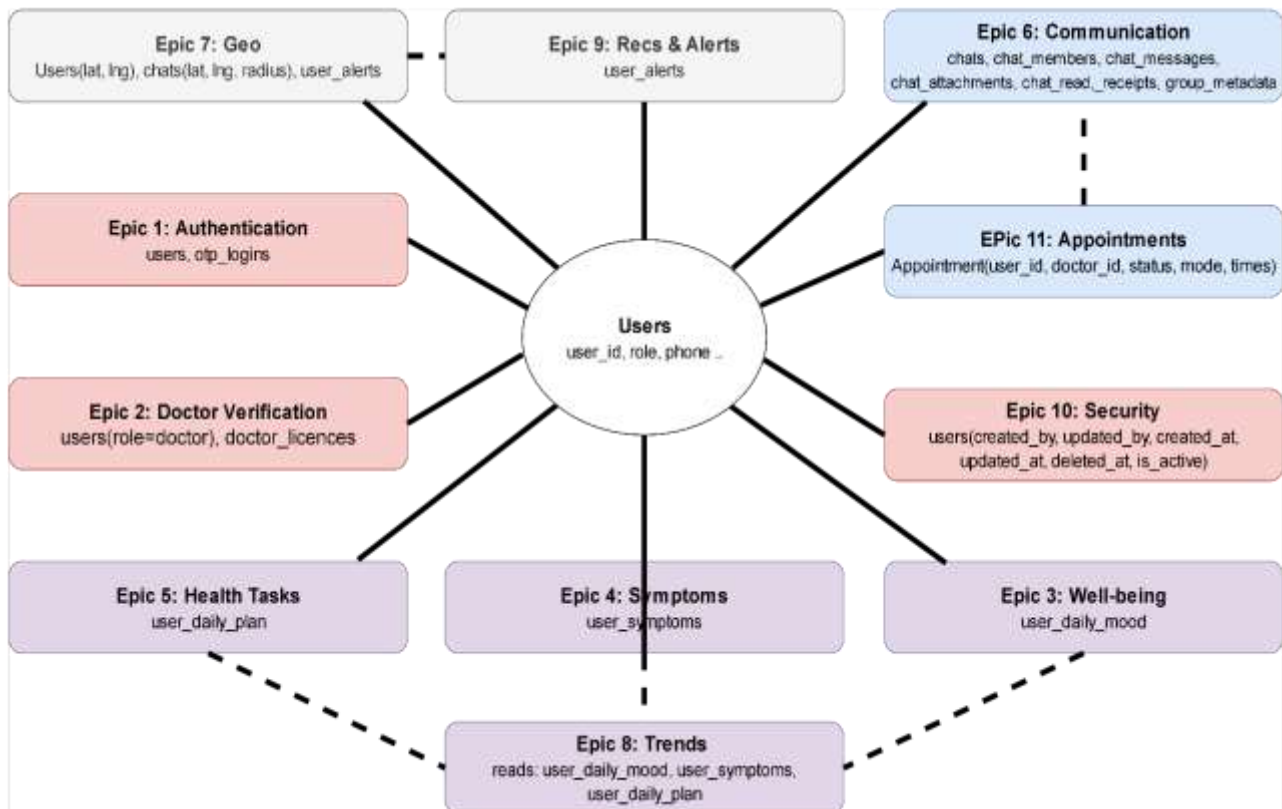
The backend, built on Node.js and hosted on AWS EC2, provides REST APIs managing authentication, doctor verification through OCR, health records, task recommendations, and secure communication. Data is maintained in SQLite3, a lightweight relational database optimized for mobile environments. Integration with external APIs (OTP, Maps, Weather, OCR) ensures dynamic, context-aware services. All client–server interactions are secured via TLS, with session tokens and role-based access preserving privacy. This architecture guarantees scalability, efficiency, and compliance with healthcare standards.

4.3 Database Design

4.3.1 Overview Entity relationship

The ERD (Figure 4.3.1) presents *Users* as the central entity connecting all modules. It links OTP authentication, doctor verification, appointments, health logs, tasks, communication, and alerts. This relational schema ensures modularity, consistency, and secure integration across authentication, healthcare management, daily logging, and user communication.

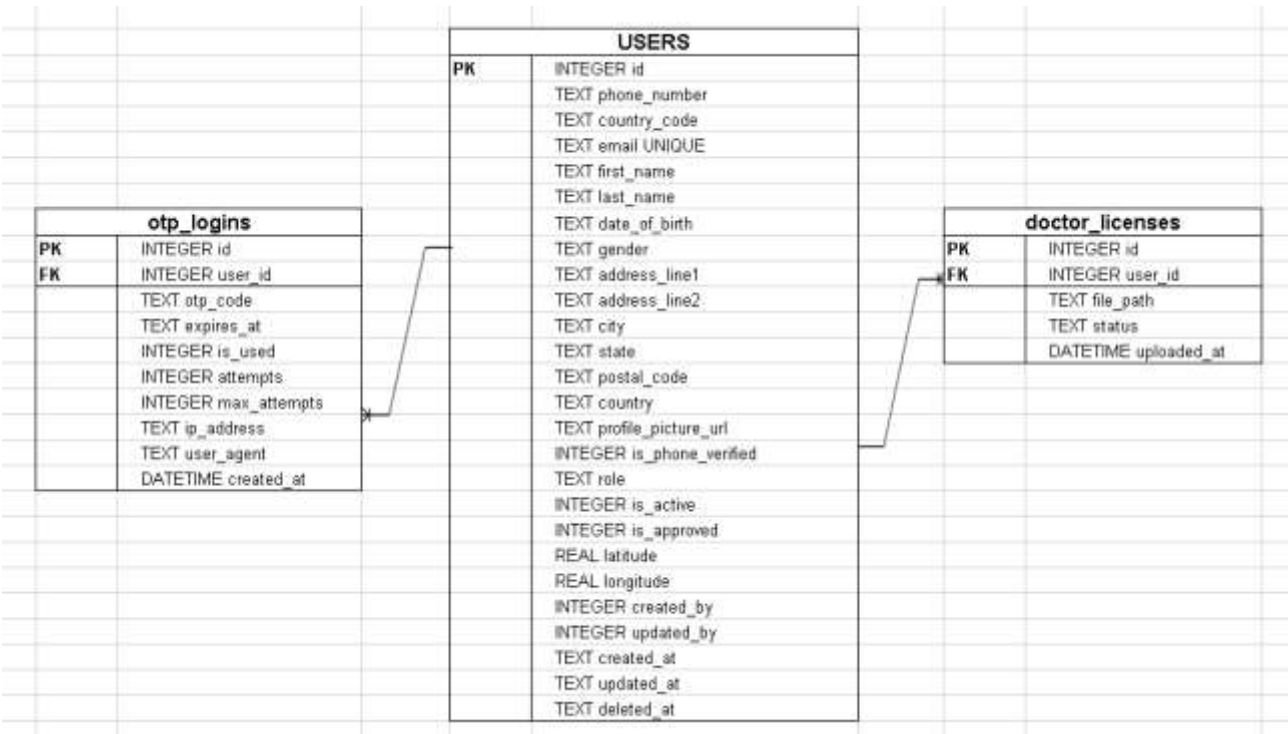
Figure 4.3.1: Overall ERD for TrustCura+ showing relationships across authentication, doctor verification, appointments, health logging, communication, and alerts.



4.3.2 User Identity, Authentication and Doctor Verification

The *Users* entity underpins identity management. OTP_Logins securely records authentication attempts, while Doctor_Licenses links professional credentials to accounts. Verification status (pending, approved, rejected) ensures credibility and aligns with Epics 1–2. This schema guarantees secure registration, login, and doctor validation.

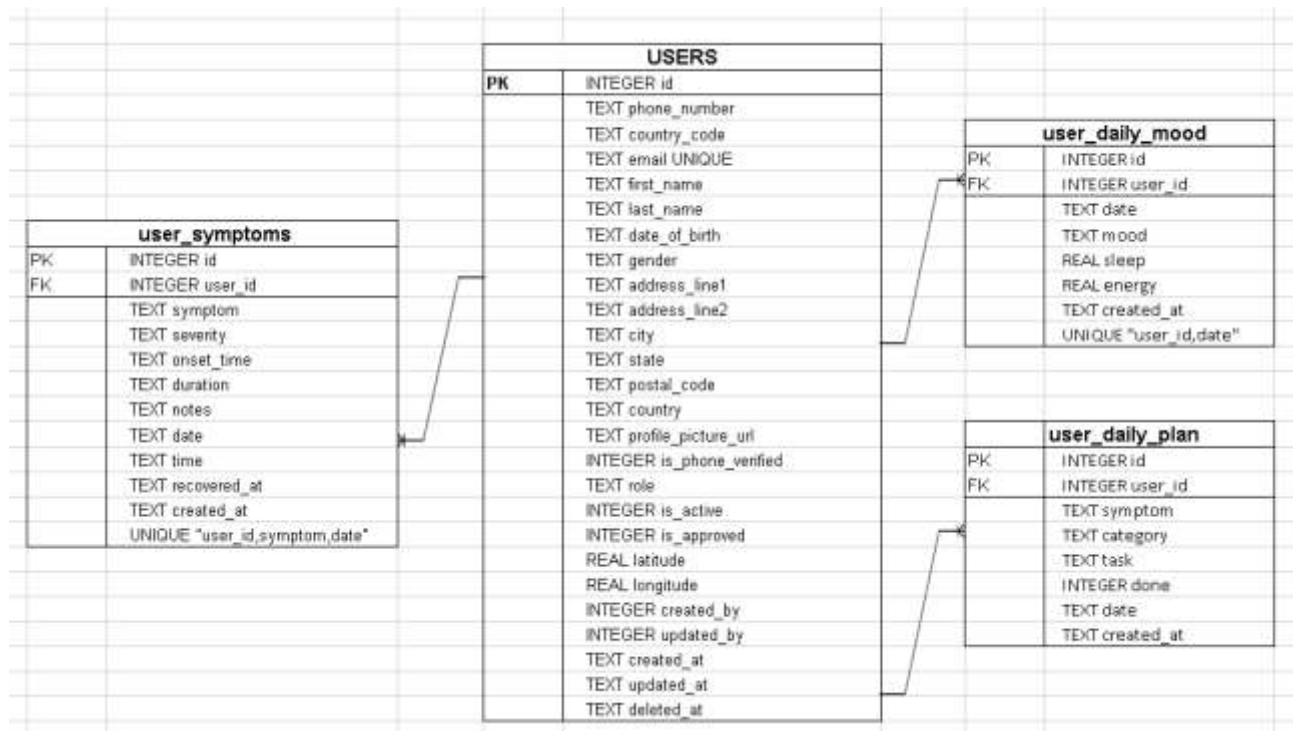
Figure 4.3.2: ER Diagram showing *Users*, *OTP_Logins*, and *Doctor_Licenses* entities supporting authentication and verification.



4.3.3 Health Logging and Daily Well-being Management Entities

Users connect directly to *User_Symptoms*, *User_Daily_Mood*, and *User_Daily_Plan*. These entities log symptoms, daily mood, sleep, energy, and health tasks. Together, they support consistent tracking, adherence, and trend analysis, implementing Epics 3–5 for structured well-being monitoring and daily engagement.

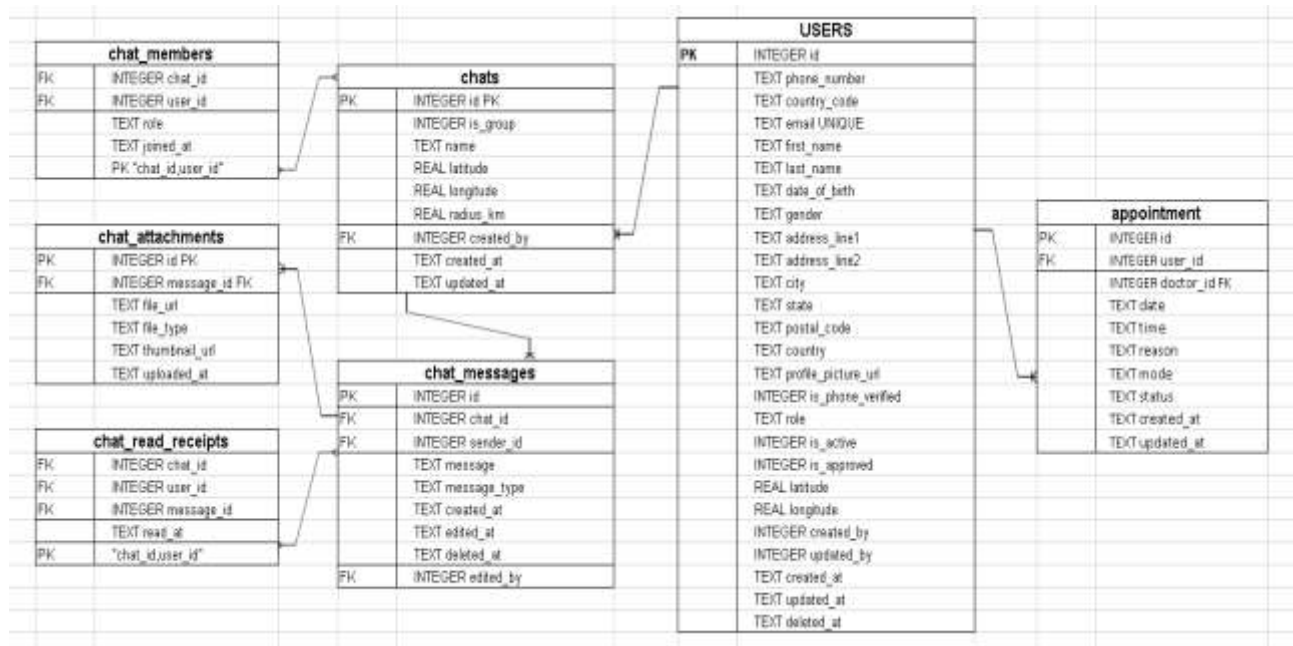
Figure 4.3.3: ER Diagram showing relationships between *Users*, *User_Symptoms*, *User_Daily_Mood*, and *User_Daily_Plan* for well-being management.



4.3.4 Communication and Appointment management Entity

The communication schema includes Chats, Members, Messages, Attachments, and Read_Receipts, enabling secure multimedia messaging. Appointments link patients and doctors with time and status attributes. These entities implement Epic 6, supporting doctor–user interactions and healthcare scheduling through structured, reliable communication and appointment management.

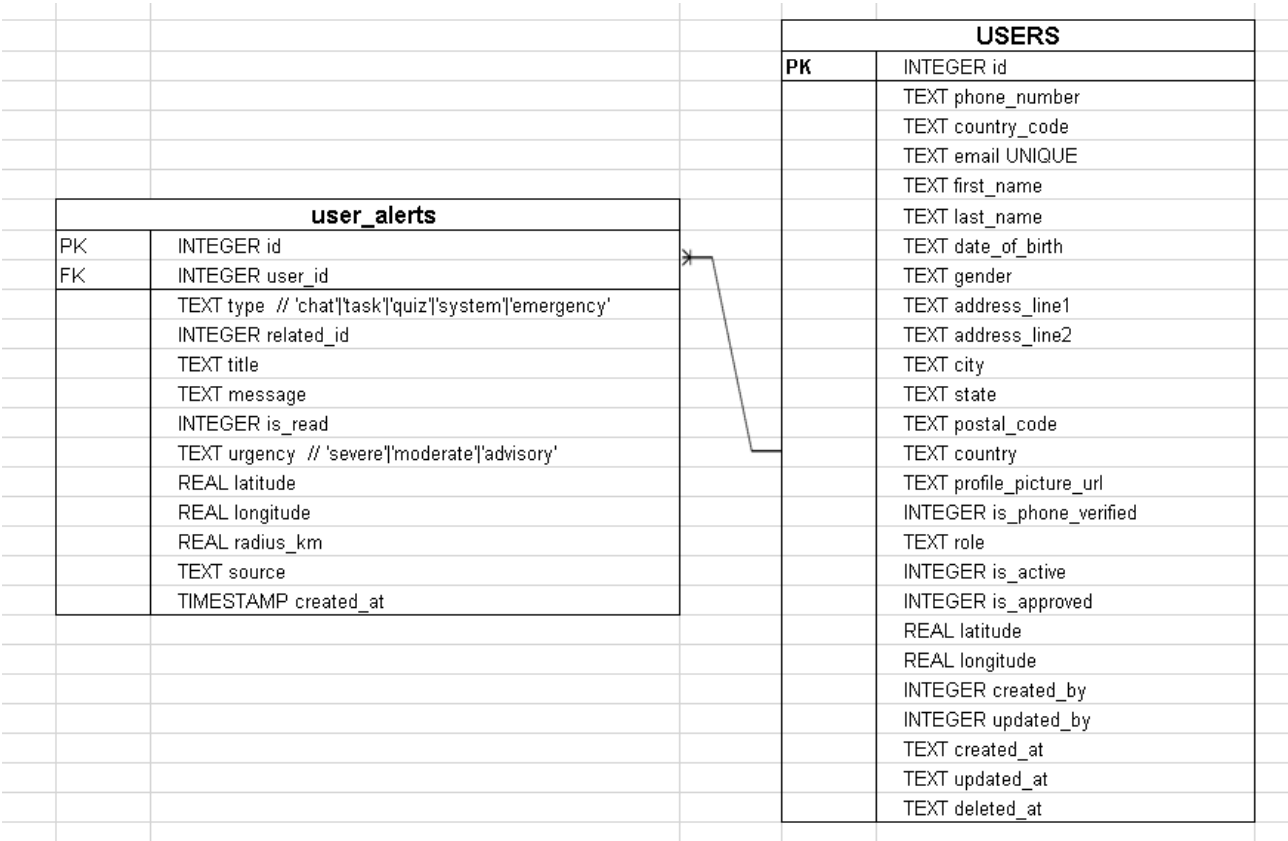
Figure 4.3.4: ER Diagram showing Users, Chats, Members, Messages, Attachments, Read_Receipts, and Appointments for communication and scheduling



4.3.5 Personalised Recommendation and Alert Management entity

User_Alerts delivers personalized recommendations, reminders, and system notices. Each alert links to a single user, supporting one-to-many communication. Attributes such as type, urgency, and geo-scope adapt messages contextually, fulfilling Epic 9 requirements for dynamic, timely, and user-specific guidance.

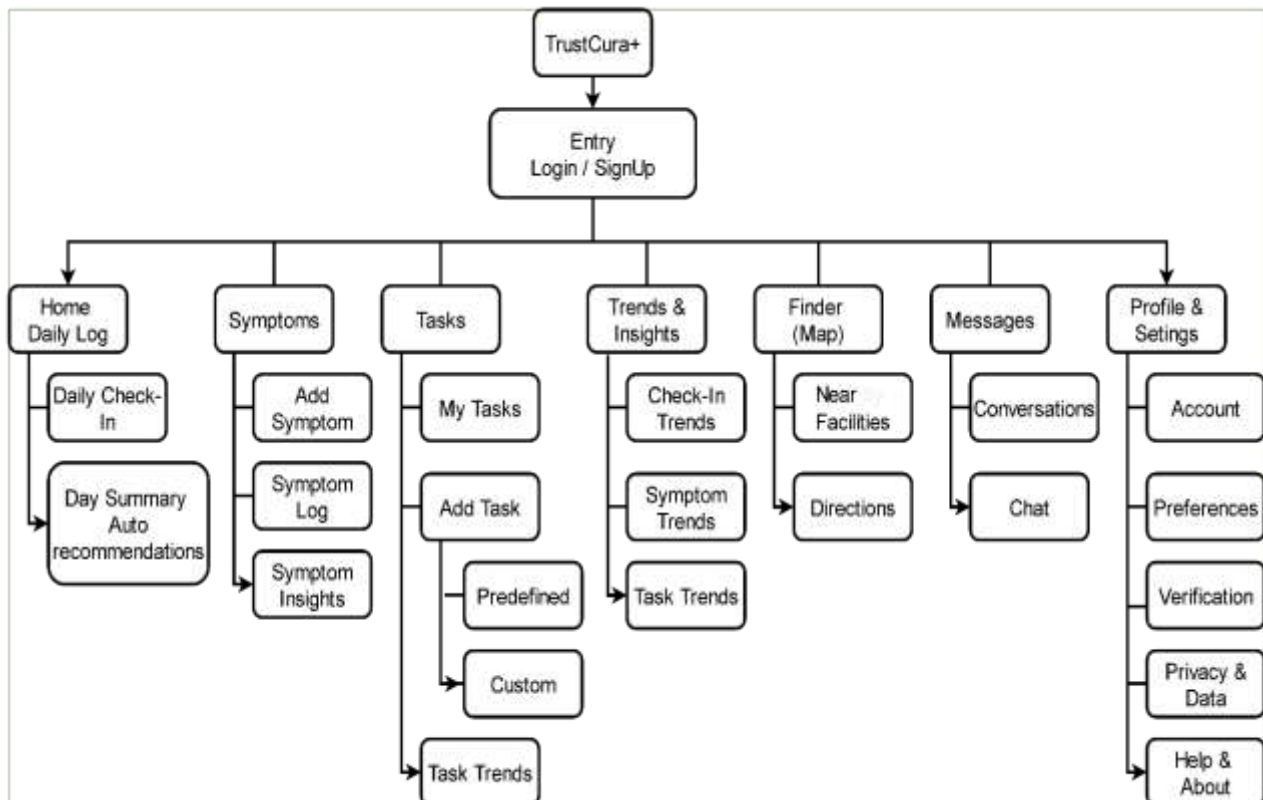
Figure 4.3.5: ER Diagram showing one-to-many relationship between *Users* and User_Alerts for recommendations and notifications.



4.4 Information Architecture (IA)

The Information Architecture (IA) of *TrustCura+* supports two primary roles: users and doctors. Both share a foundation of secure authentication, communication, and health tracking, but differ in access rights. User authentication manages sign-up, login, and verification, leading to role-specific dashboards. The **User Dashboard** enables symptom logging, daily check-ins, recommendations, task management, and facility search, supported by embedded sensors (accelerometer, camera, GPS). The **Doctor Dashboard** allows record review, recommendations, consultations, and care plan updates. Core modules include symptom tracking, recommendation engine, secure communication, task management, and resource finder. External APIs (weather, maps, health news) enhance context-aware guidance.

Figure 4.4.1: Information Architecture of *TrustCura+* showing relationships between user/doctor roles, dashboards, and supporting modules.

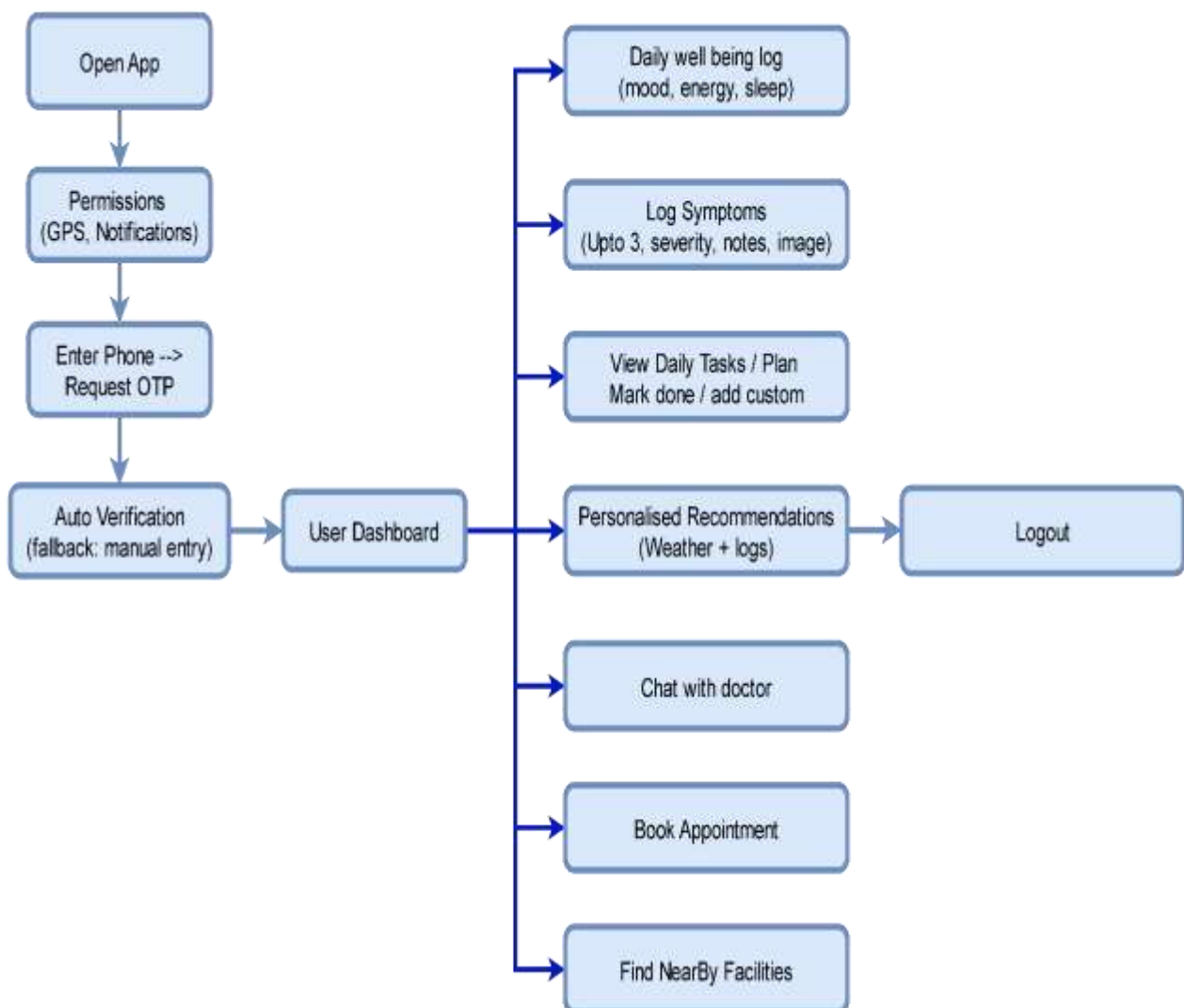


4.5 Application Flow

4.5.1 Application Flow Users

TrustCura+ begins with OTP-based authentication, where users enter a phone number and receive auto or SMS fallback verification. After login, the User Dashboard provides access to daily well-being logging, symptom entries (≤ 3), task management, personalized recommendations, secure chat, appointments, and facility search. This streamlined flow ensures smooth navigation and consistent engagement.

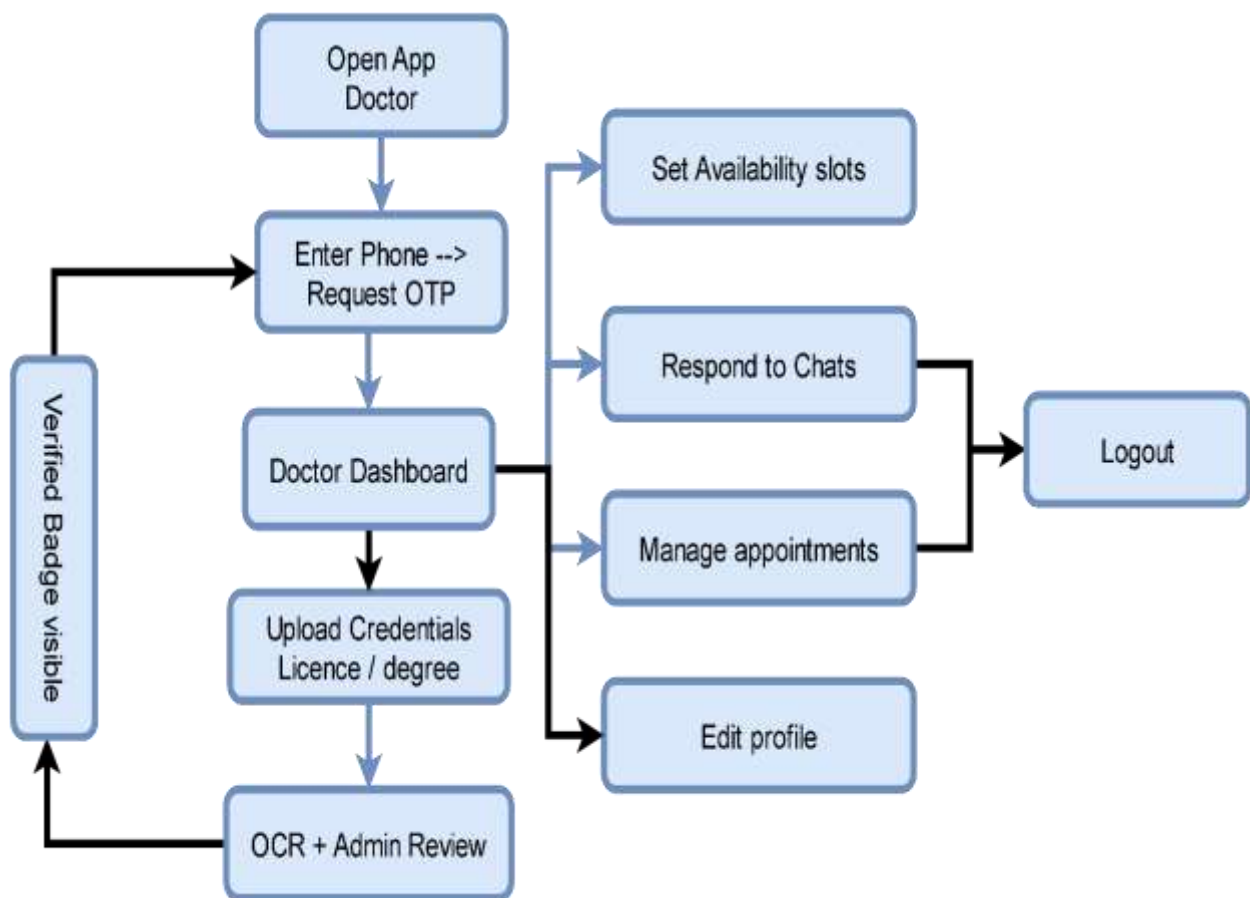
Figure 4.5.1: User Application Flow showing OTP login leading to dashboards for logging, recommendations, communication, appointments, and facility discovery.



4.5.2 Application Flow Doctors

For doctors, the flow begins with OTP login, credential submission, and OCR-based verification, followed by profile setup. Once validated, the Doctor Dashboard provides tools to manage availability, review patient records, respond to messages, and schedule appointments. This structured process ensures secure onboarding, authenticity of professionals, and efficient clinical interactions.

Figure 4.5.2: Doctor Application Flow showing OTP login, OCR verification, and dashboard access for availability management, patient communication, appointment handling, and profile updates.



4.6 Security and Privacy Design

TrustCura+ applies multi-layered security to protect health data. Authentication is OTP-based, with sessions tracked and invalidated on logout. JSON communication between client and server is secured through encryption and decryption logic. Role-based access ensures users and doctors only view relevant information. Sensitive data, including personal identifiers and health records, is encrypted at rest in SQLite. OCR-based doctor verification further validates professional credentials, ensuring authenticity, protecting patients, and strengthening trust in the platform's secure healthcare environment.

4.7 Design Summary

The design of *TrustCura+* translates requirements into system architecture, database schema, workflows, interfaces, and security. Centered on the Users entity, it integrates OTP authentication, doctor verification, health logging, communication, and recommendations, ensuring scalability, reliability, and compliance with healthcare privacy standards.

5. Prototyping

Word count: 885

5.1 Introduction (Definition and Purpose)

The prototype for *TrustCura+* validated core features before full-scale implementation, including OTP authentication, doctor verification, health logging, symptom recording, and communication. It focused on essential workflows to confirm integration across frontend, backend, and database rather than covering all modules. This working model demonstrated feasibility of earlier design assumptions and generated preliminary usability insights, later analysed in Section 6: User Feedback.

5.2 Objectives and Critical Features Selected

The prototype was developed to confirm technical feasibility and user interaction for selected features. Implemented modules included OTP-based authentication with country code detection, doctor verification via OCR credential upload, daily logging of mood, energy, and sleep, symptom tracking of up to three entries with severity and notes, and basic messaging between user and doctor. Collectively, these formed the system backbone, covering identity management, trust, daily input, symptom monitoring, and communication.

5.3 Technical Setup for Prototype

The prototype was implemented using the planned technology stack to validate workflows before deployment. The frontend was built with React Native (Expo Go), while a mocked backend in Node.js ran temporarily on AWS EC2. Data was managed locally in SQLite3. External services were simulated: SMS/OTP providers were mocked, and OCR verification was tested with Tesseract. This lightweight setup enabled rapid iteration while remaining consistent with the overall system architecture described in Section 4.2.

5.4 User Interface (UI/UX Design)

The TrustCura+ interface ensures simplicity and consistency for users and doctors. Medium-fidelity wireframes defined layout, navigation, and features, aligning with functional requirements across Epics 1–9.

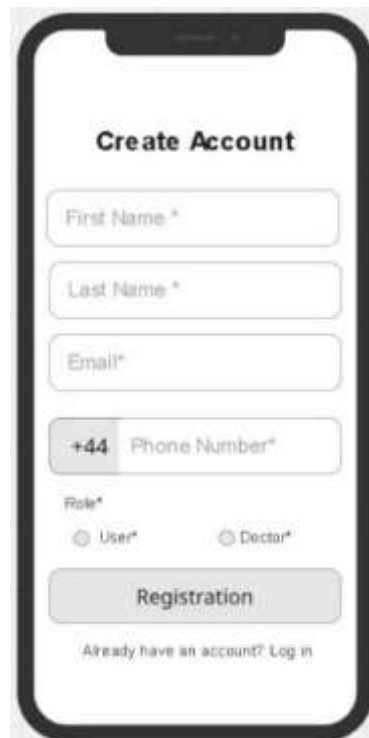
5.4.1 Onboarding

Onboarding introduces core features across three guided screens: personalized health recommendations, secure doctor connectivity, and well-being tracking. These help users understand the app's purpose quickly and encourage confident engagement.



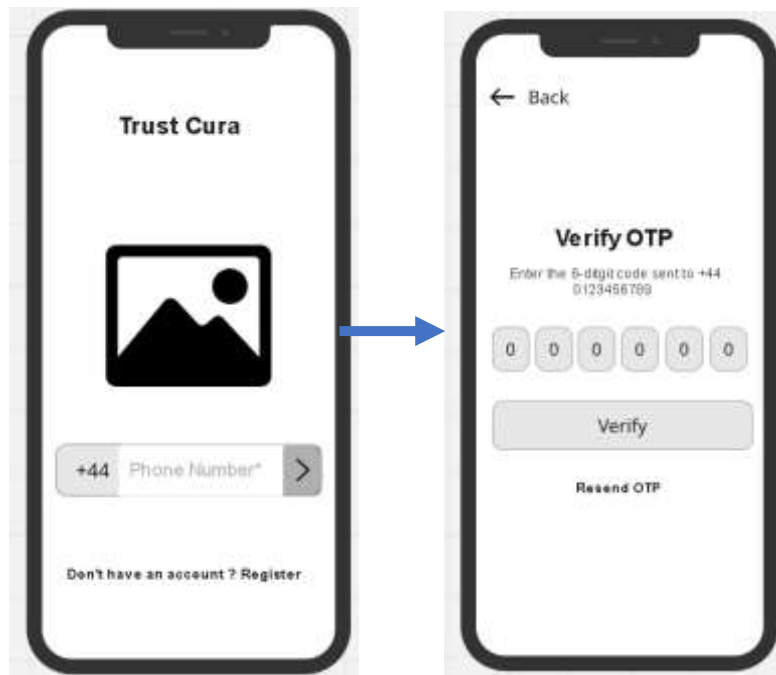
5.4.2 Create Account

The Create Account screen enables registration with name, email, phone, and role. Country code is auto-detected via GPS or selected manually, ensuring accuracy, authentication, and tailored access.

A mobile app 'Create Account' screen. The screen has a white background with a black border. At the top, the text 'Create Account' is centered. Below it are four input fields: 'First Name *', 'Last Name *', 'Email*', and a phone number field with a grey dropdown menu showing '+44' and the text 'Phone Number*'. Below the phone number field is a 'Role*' section with two radio buttons: 'User*' and 'Doctor*'. At the bottom is a grey 'Registration' button. Below the button is the text 'Already have an account? Log in'.

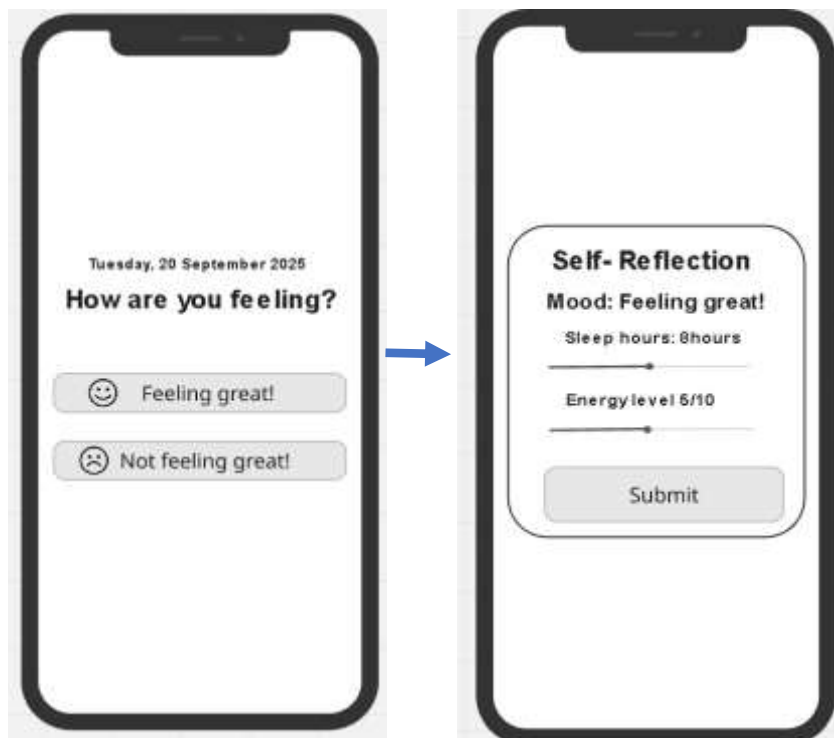
5.4.3 User Authentication Process

Login involves phone number entry with auto/manual country code, followed by OTP verification. This ensures secure access and prevents unauthorized entry..



5.4.4 Daily Health Check-In (Mood, Energy, Sleep)

Users record mood, energy, and sleep once daily using simple inputs. Summaries and trends support insights, monitoring, and well-being tracking with minimal effort.



5.4.5 Symptom Tracking

Users log up to three symptoms daily with severity, onset, and notes. This structured input supports accurate monitoring and personalized insights.

The image illustrates a symptom tracking interface on a mobile device, showing two screens connected by a blue arrow indicating a flow from selection to detail entry.

Screen 1: Select Your Symptoms

Header: Select Your Symptoms
Sub-header: Add your symptoms (max 3 per day)

Available symptoms (each with a picture icon):

- Sore throat
- Stomach ache
- Fever
- Fatigue
- Joint Pain
- Eye Irritation
- Shortness of breath
- Headache
- Gums Bleeding
- Tooth Pain

Screen 2: Sore Throat

Severity

Mid Moderate Severe

Onset Time

00:21:24

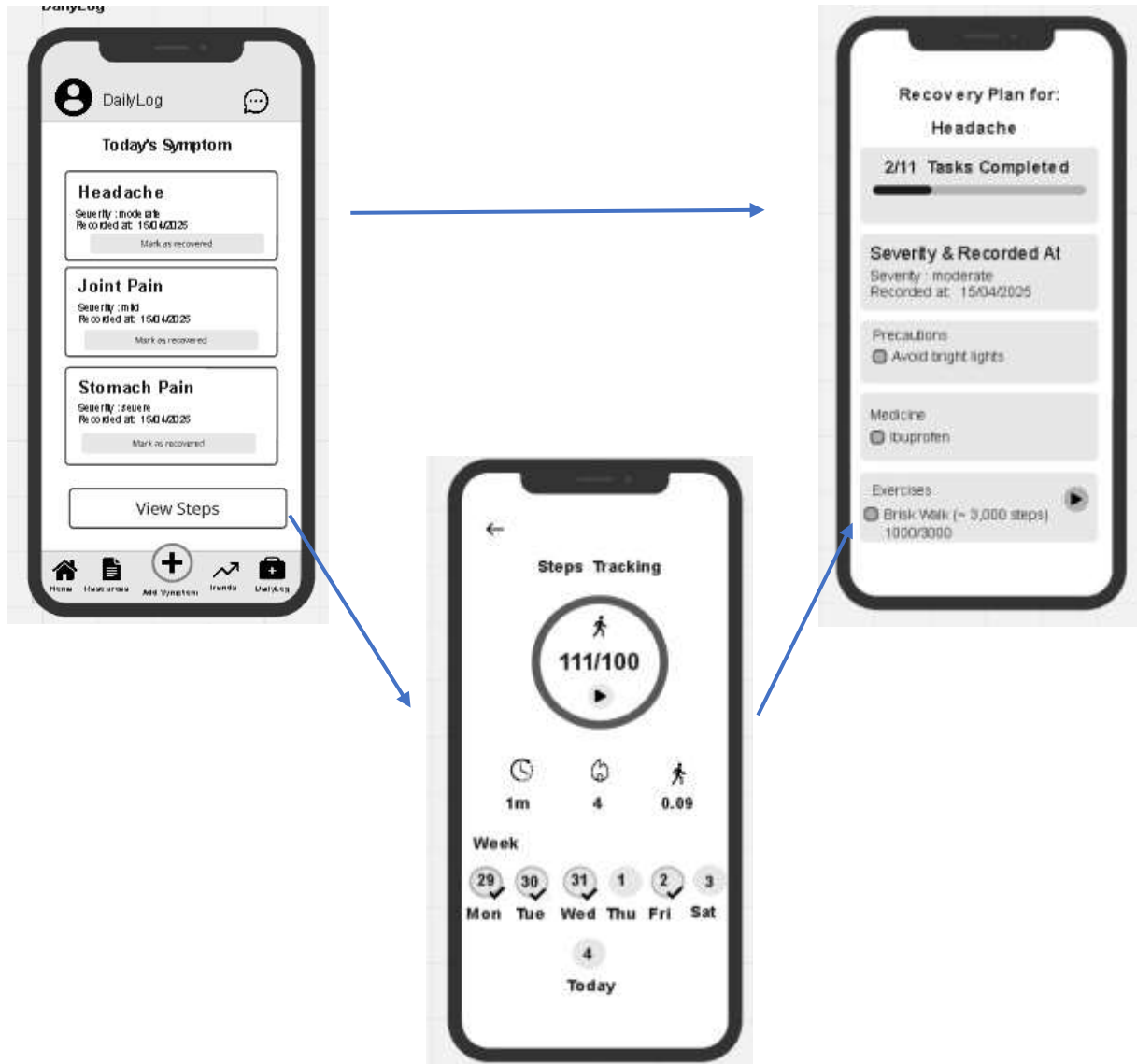
Duration (optional)

Notes(optional)

Save to Calendar

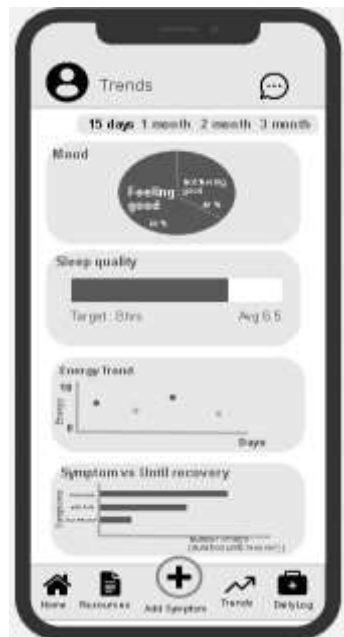
5.4.6 Symptom Log, Recovery Plan and Steps Tracking

The Symptom Log displays daily entries with severity and recovery status. The Recovery Plan adds tasks, medicines, and exercises with auto-updating completion indicators. Steps Tracking uses the accelerometer to monitor steps, calories, and progress, promoting accountability and holistic care.



5.4.7 Health Trends

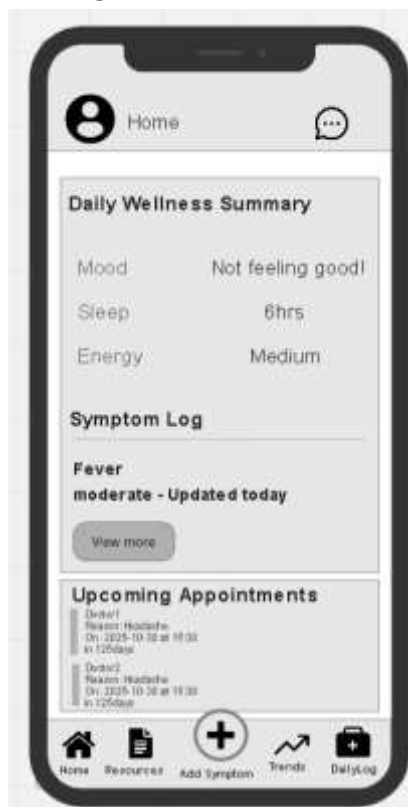
The Trends screen visualizes mood, sleep, energy, and symptom recovery across 3–15 days, supporting reflection, pattern recognition, and informed health decisions.



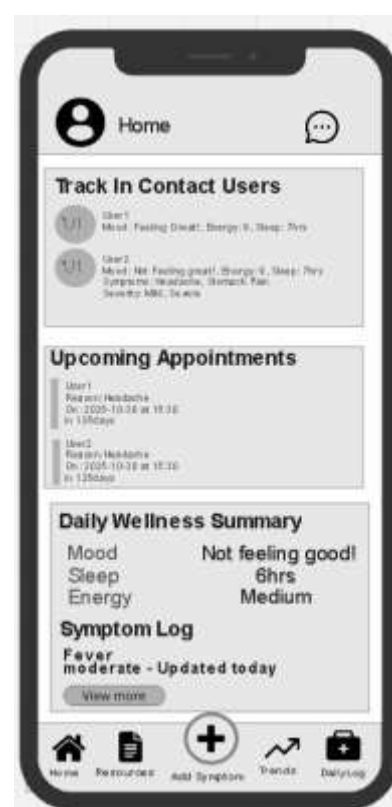
5.4.8 My Daily Wellness Hub

The Home screen centralizes summaries, logs, and appointments. The extended version allows caregiver oversight by displaying connected users' indicators.

Users Dashboard



Doctors Dashboard



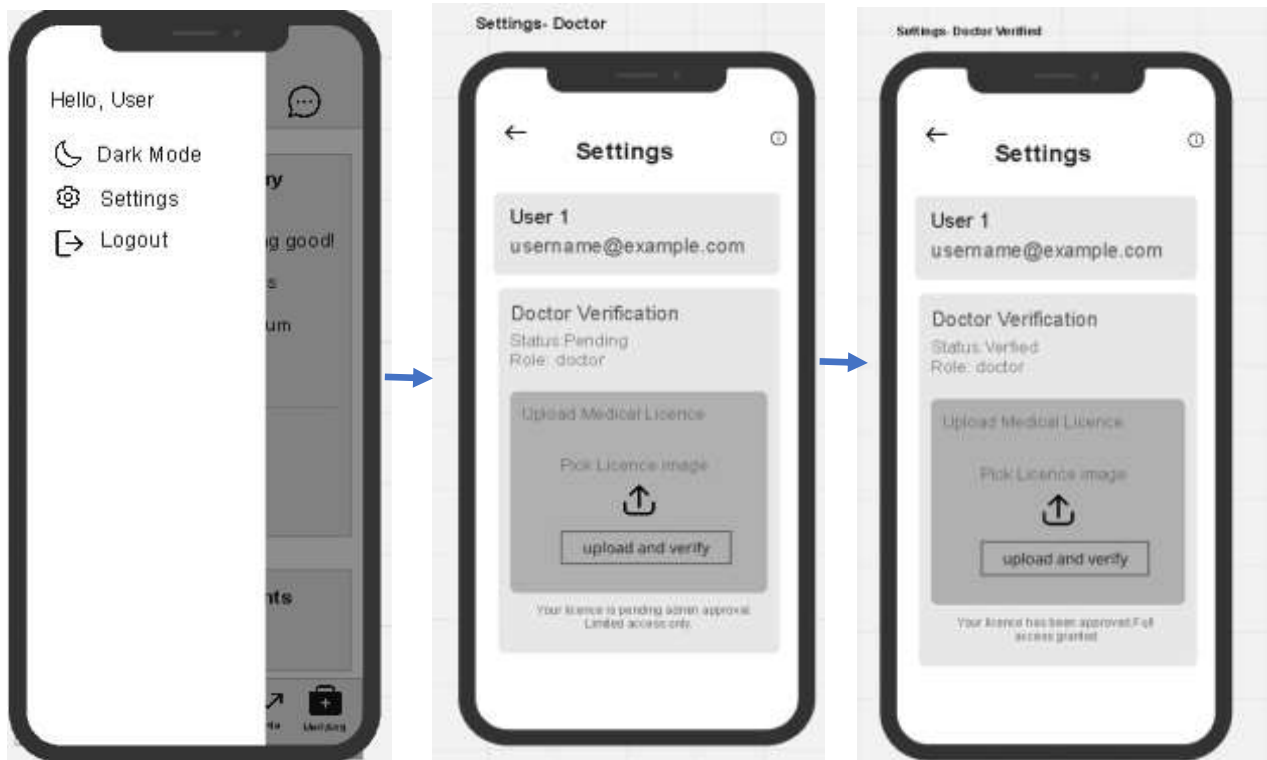
5.4.9 Resource Hub

The Resources screen shows weather-based safety prompts and nearby emergency services via GPS. Users can call, chat, or share location with next-of-kin.



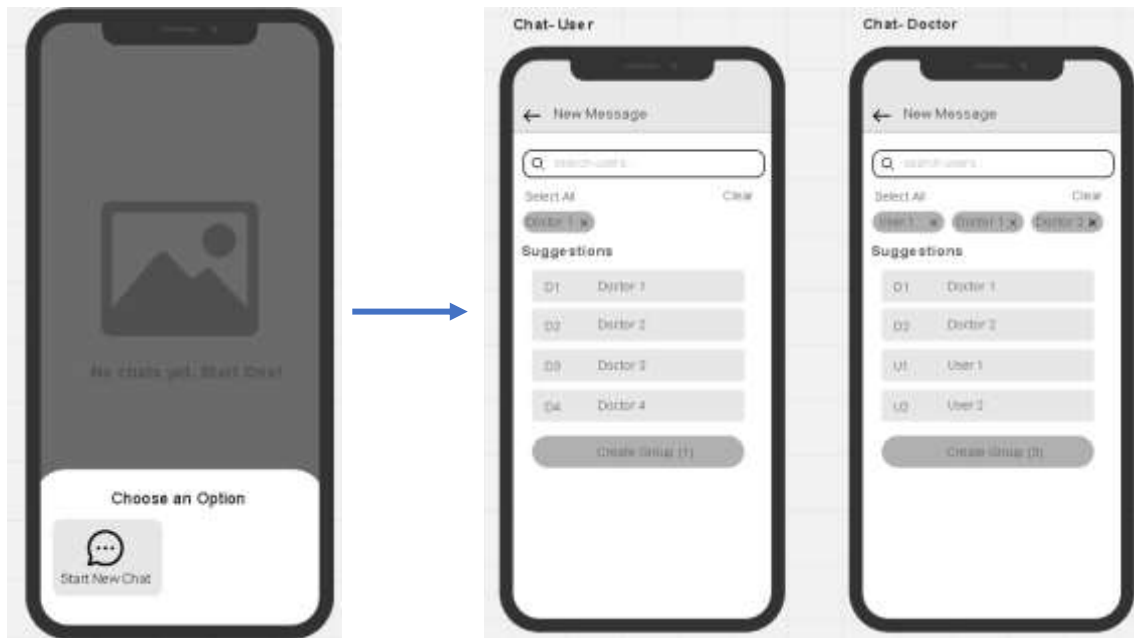
5.4.10 Side Navigation Bar and Doctor Verification

The Side Menu provides quick access to Dark Mode, Settings, and Logout, ensuring usability and personalization. Doctors upload licenses for OCR-based verification. Once validated, access is granted, ensuring authenticity and security



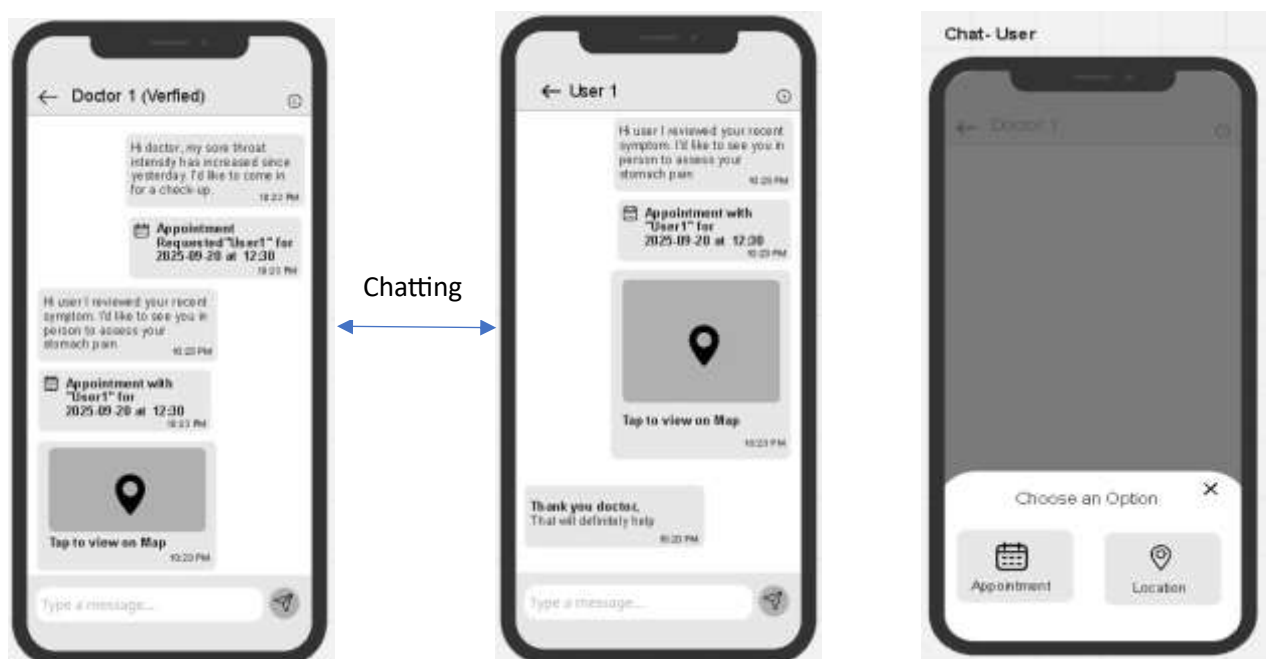
5.4.11 Start New Chat and Add Users

Patients may add only doctors, while doctors can add peers or patients. This ensures secure and structured communication.



5.4.12 Consultation Collaboration

Chats support messages, treatment suggestions, appointments, and location sharing. Features streamline healthcare coordination and reduce miscommunication.



5.5 Screenshots of the Developed Prototype

The screens illustrate OTP authentication, doctor verification, daily logging, symptom tracking, communication, and health monitoring. They validate alignment with design requirements and demonstrate feasibility of technical implementation and user interaction. Comprehensive supporting screenshots for all features are compiled in [Appendix G](#) to provide visual confirmation of the prototype's functionality.

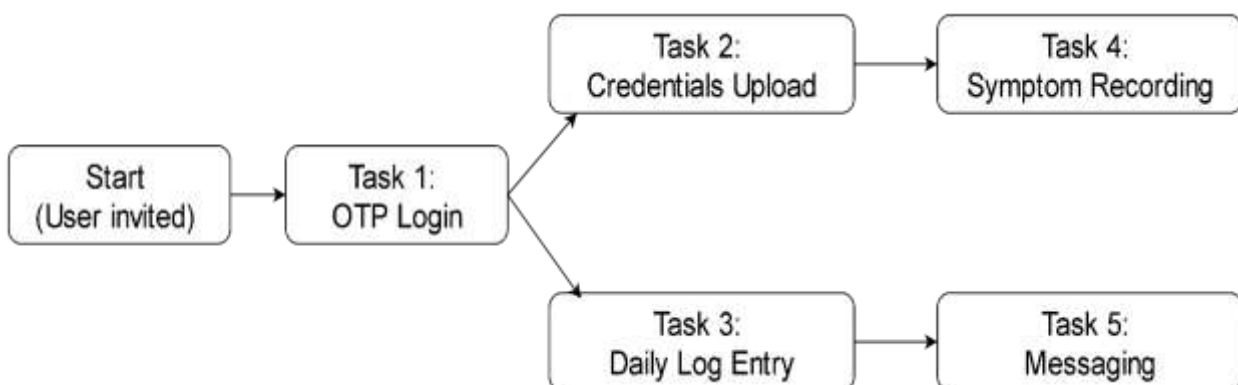
5.6 Prototype Testing (Method and Participants)

Testing of the prototype was carried out with four participants (three peers and one family member). Each was asked to complete tasks including OTP login, credential upload, daily log entry, symptom recording, and messaging.

Observations focused on navigation, task completion, and error handling. A short questionnaire captured impressions after task completion.

Only a summary of findings is included here; a detailed discussion of user feedback and its implications for the system is provided in Section 6: User Feedback.

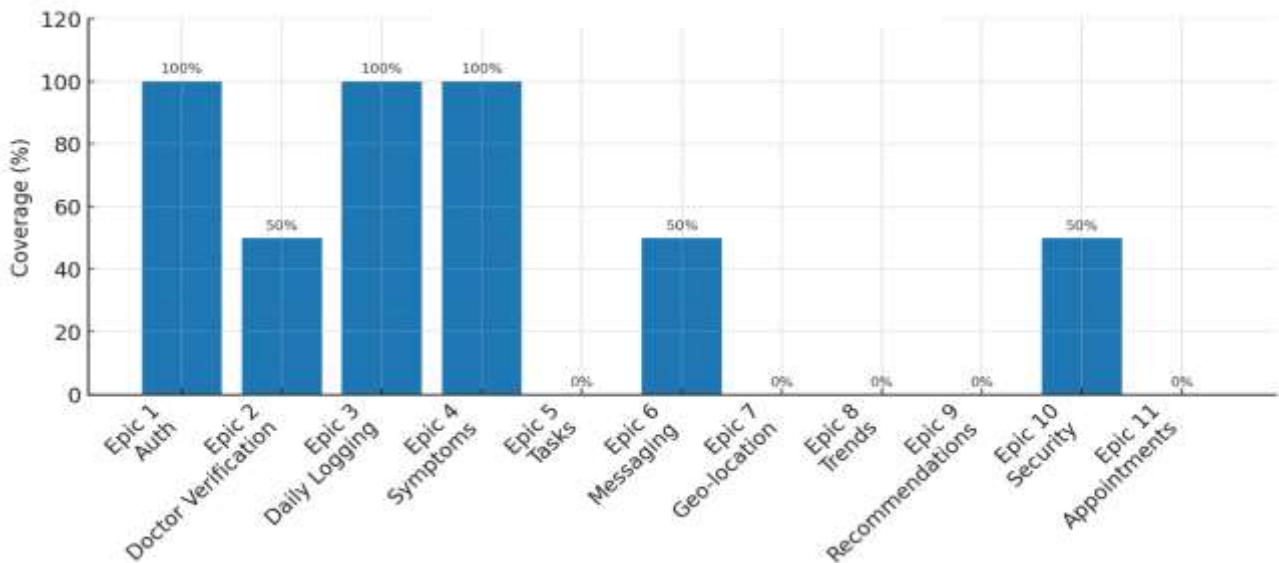
Figure 5.5: Prototype Testing Workflow



5.7 Outcomes (Changes proposed for Development)

Feedback from four participants suggested refinements: replace text input with sliders for daily logs, support PDF uploads for credentials, retain manual OTP fallback, and add delivery/read indicators in chat. Despite the small sample, responses were consistent. Prototype coverage was 41% across 11 epics, with full implementation of Epics 1, 3, and 4, and partial coverage of 2, 6, and 10.

Figure 5.6: Prototype Feature Coverage



5.8 Limitations

The prototype implemented only 41% of planned features, excluding task management, geo-location, recommendations, and appointments. Security was limited to OTP verification, and messaging supported text only. OCR integration failed with poor-quality scans, while testing with four participants restricted feedback diversity. SQLite3 and AWS EC2 were adequate for prototyping but unsuitable for scalability. iOS testing was omitted due to the need for an Apple development environment. Detailed participant feedback is presented in Section 6: User Feedback.

6. User Feedback

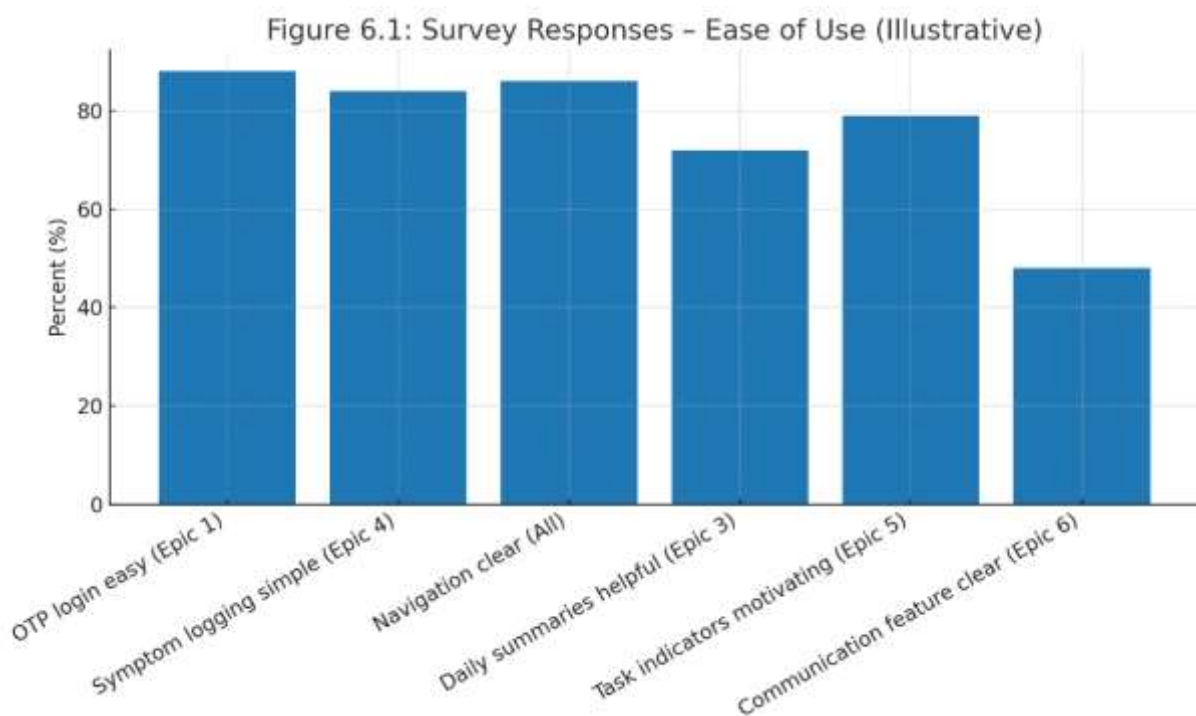
Word count: 457

User feedback was central to evaluating *TrustCura+*. Input was collected iteratively, starting with low-fidelity wireframes and extending through interactive prototypes. This process validated early assumptions and later confirmed alignment of features with the defined epics and user stories. [Full survey instruments and comparative results are provided in Appendix H.](#)

6.1 Feedback on Wireframes

Preliminary wireframes were shared with peers to test layout and navigation assumptions. Feedback emphasized the need for severity scales in symptom entry, reduced effort for daily tasks, and clearer navigation flows. Participants recommended sliders, dropdowns, quick-add buttons, progress indicators, and consistent navigation bars. They also requested summaries of progress, clearer facility access points, and more prominent communication tools. These refinements informed higher-fidelity prototypes. Figure 6.1 illustrates participant ratings of authentication, symptom logging, and navigation, confirming overall ease while highlighting issues with summaries and communication.

[Details of survey questions are provided in Appendix H.1.](#)



6.2 Prototype Feedback Collection

Interactive prototypes were tested with students, chosen for accessibility within project constraints, a method consistent with formative mHealth studies (Grundy et al., 2022). Feedback focused on usability and clarity rather than medical accuracy. Short surveys, combining multiple-choice and open-ended questions, captured impressions of navigation, summaries, logging, and communication.

[Limitations of this student-only sample are discussed in Appendix H.2.](#)

6.3 Target Users

TrustCura+ is designed for general users wishing to track symptoms, follow recommendations, and manage tasks, including students, professionals, and wellness-focused individuals. For this project, students served as the feedback group. While providing practical insights into usability, this limited demographic did not reflect the diversity of the intended audience, such as older adults or frequent healthcare users.

6.4 Feedback by Epic and User Story

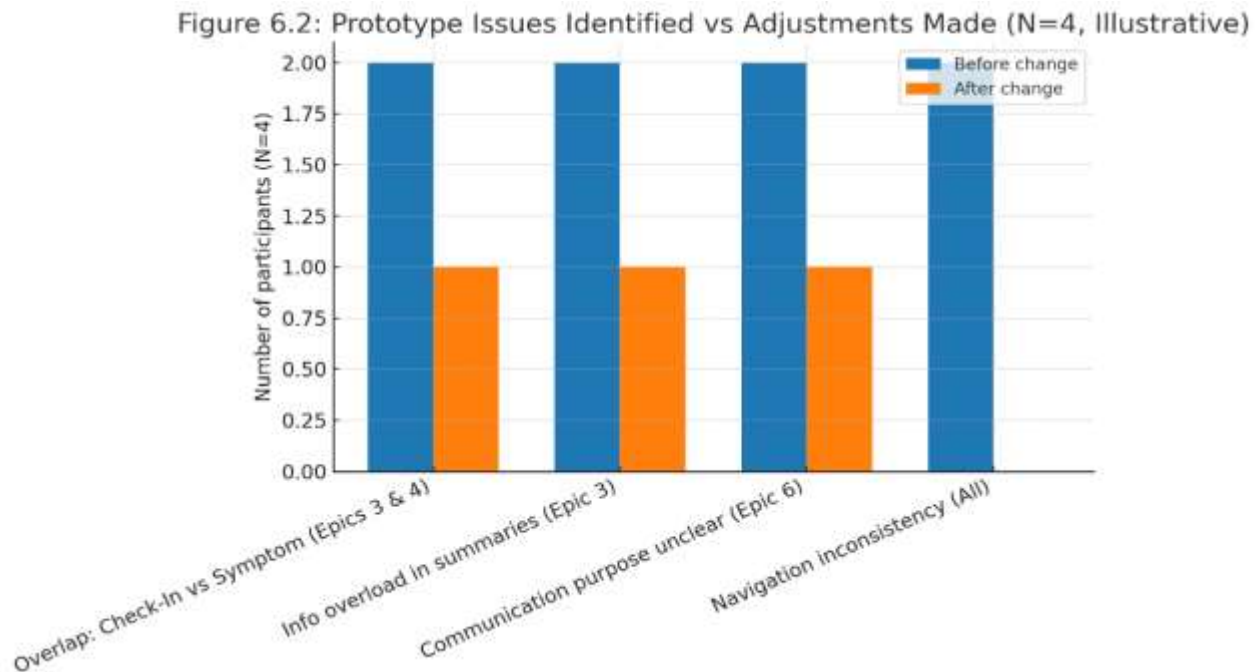
Feedback was mapped to system epics to ensure traceability between design decisions and requirements.

- Epic 1 – Authentication: Auto-verification was valued, manual fallback retained, logout kept clear.
- Epic 3 – Daily Logging: Sliders/icons preferred, summaries useful but simplified.
- Epic 4 – Symptom Tracking: Logging clear, overlap with daily check-in resolved by merging flows.
- Epic 5 – Tasks: Indicators motivating, task visibility increased on home screen.
- Epic 6 – Communication: Messaging reframed as general health communication and made more prominent.

[Detailed before/after changes are presented in Appendix H.3.](#)

6.5 Findings

Common themes emerged: symptom logging was simple, navigation clear, and task indicators motivating. Issues included overlapping features, cluttered summaries, and unclear communication purpose. Adjustments introduced unified flows, simplified summaries, clarified messaging, and enhanced visuals. Figure 6.2 shows the decline in reported issues after refinements.



6.6 Summary

The feedback process confirmed that *TrustCura+* met acceptance criteria while identifying areas for refinement. Iterative changes reduced overlap, improved clarity, and strengthened usability. Although limited to student participants, this evaluation provided essential validation of core functionality. [Broader testing across age groups and lifestyles remains necessary \(Appendix H.2\).](#)

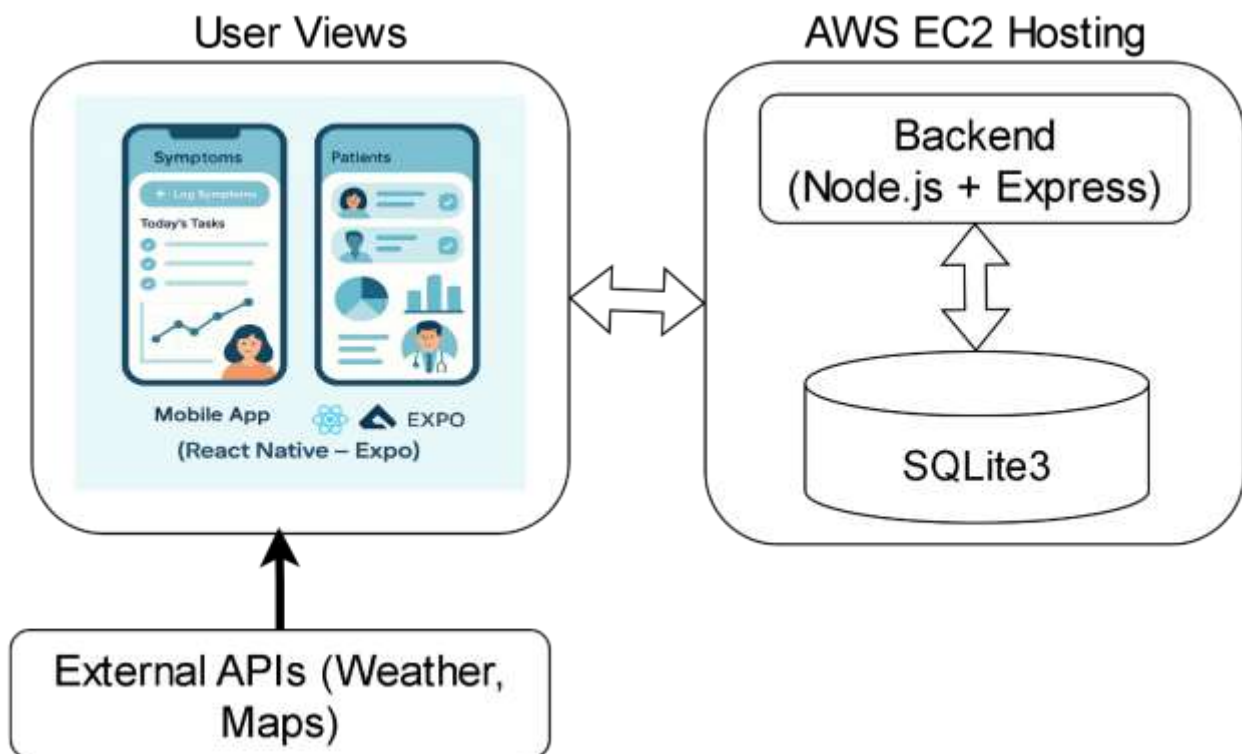
7. Development

Word count: 944

7.1 Introduction

The development of TrustCura+ adopted a mobile-first approach, prioritising the client built in React Native (Expo) for cross-platform deployment. Backend services were developed in Node.js/Express, with SQLite3 serving as the lightweight database. Redux Toolkit managed state, while Socket.IO enabled real-time communication. The system's integration of mobile, backend, and external APIs is depicted in Figure 7.1 – System Architecture. [This chapter explains module implementations, supported by diagrams and extended listings in Appendix F.](#)

Fig 7.1 – Simplified System



7.2 Project Structure

The project was divided into two codebases. The separation ensures modularity: the client manages UX and sensors, while the server handles persistence, security, and communication.

Client (React Native / Expo):

- *screens/* – UI views (Registration, OTP, Health Logs, Recovery Plan, Calendar, Resources, Settings).
- *store/* – Redux slices for authentication, chat, appointments, and health logs.
- *components/* – reusable widgets such as license upload and progress indicators.
- *modals/* – feature-specific inputs (symptoms, mood details).
- *utils/* – helpers for API calls, token handling, and sensor access.

```
Client/
├── src/
│   ├── screens/      # UI screens (Registration, OTP, HealthLog, Calendar, Chat, Settings)
│   ├── components/   # Reusable UI components (Footer, RadioButton, DoctorLicenseUpload)
│   ├── modals/       # Modal dialogs (SymptomsModal, SymptomDetailModal, MoodDetailsModal)
│   ├── store/        # Redux slices and async actions
│   │   ├── actions/  # API interaction logic (loginActions, chatActions, etc.)
│   │   └── reducers/ # State slices (auth, healthlog, chat, settings, theme, weather)
│   ├── utils/        # Helper modules (api.js for encryption, config.js for modes & URLs,
│   └── assets/       # Fonts, icons, static resources
```

Server (Node.js / Express):

- *routes/* – REST APIs for authentication, chat, health logs, and appointments.
- *migrations/* – schema initialization for SQLite3.
- *sockets/* – Socket.IO handlers for messaging.
- *config/* – database setup.
- *server.js* – entry point linking routes, middleware, and WebSockets.

```
Server/src/
├── server.js          # Entry point, Express + Socket.IO setup, CORS
├── config/db.js       # SQLite3 connection
├── migrations/        # Database schema (initSchema.js)
├── routes/            # REST API routes (users.js, auth.js, license.js, chat.js, healthlog
├── sockets/index.js   # Real-time messaging handlers with Socket.IO
└── utils/crypto.js    # Request/response encryption/decryption
```

7.3 Technology Stack

The *TrustCura+* technology stack was carefully selected to ensure scalability, cost-effectiveness, rapid prototyping, and reliable mobile healthcare deployment.

- Frontend: React Native with Expo for cross-platform deployment and sensor access.
- Backend: Node.js with Express for REST APIs.
- Database: SQLite3 for lightweight, offline-first storage.
- Cloud: AWS EC2 for hosting APIs.
- Sensors: accelerometer (steps, activity), GPS (location), camera (credentials, vitals).
- External APIs: weather (recommendations), maps (facility search).

[For details refer to Appendix F.1 Technology Stack](#)

7.4 Core Features and Implementation

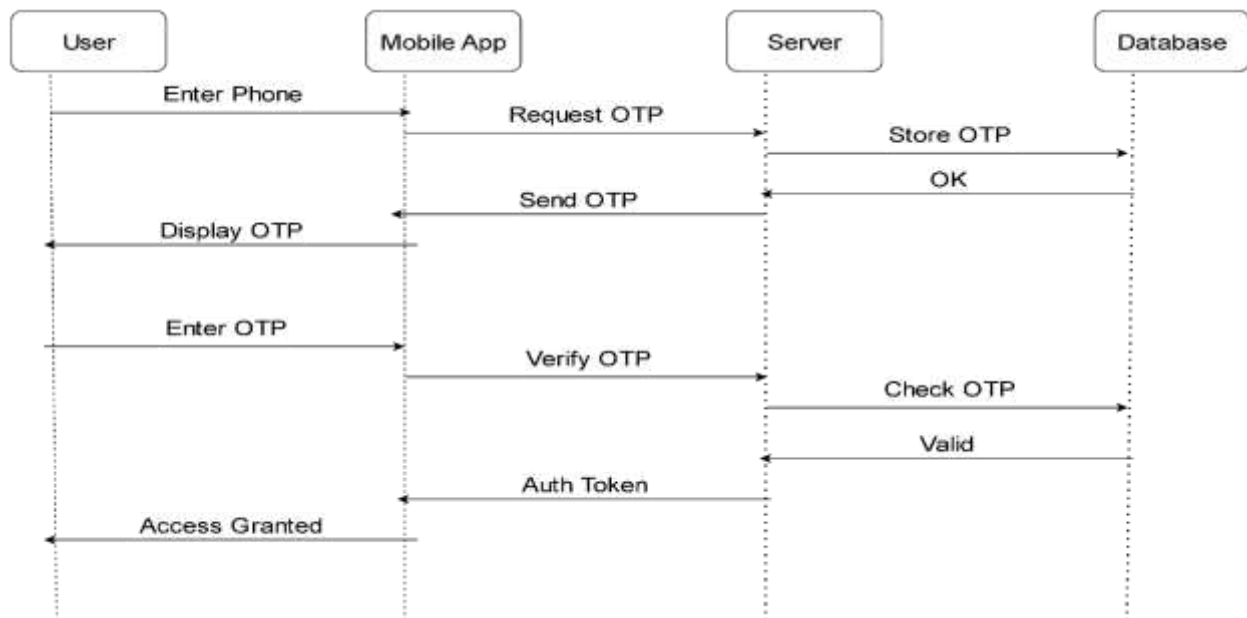
This section explains how each functional module was implemented, with critical code excerpts included for illustration.

- [Supporting screenshots for all the features are provided separately in Appendix G](#)
- [Extended code listings are provided in Appendix F.](#)

7.4.1 User Registration and OTP Authentication

User registration captures profile details, role, and location. The backend issues an OTP, verified via secure API. OTP may auto-fill or be manually entered. **Figure 7.2** illustrates the complete OTP authentication sequence.

Figure 7.2 shows the sequence diagram for the OTP login flow. [See Appendix F.2](#)



Snippet: OTP Verification (Client)

```
// OTPVerificationScreen.js
const handleOtpChange = (text, index) => {
  const newOtp = [...otp];
  newOtp[index] = text;
  setOtp(newOtp);

  // Auto-submit when all digits are filled
  if (newOtp.every((d) => d !== '')) {
    const finalOtp = newOtp.join('');
    dispatch(verifyOtp({ user_id: user?.id || userId, otp_code: finalOtp }));
  }
};
```

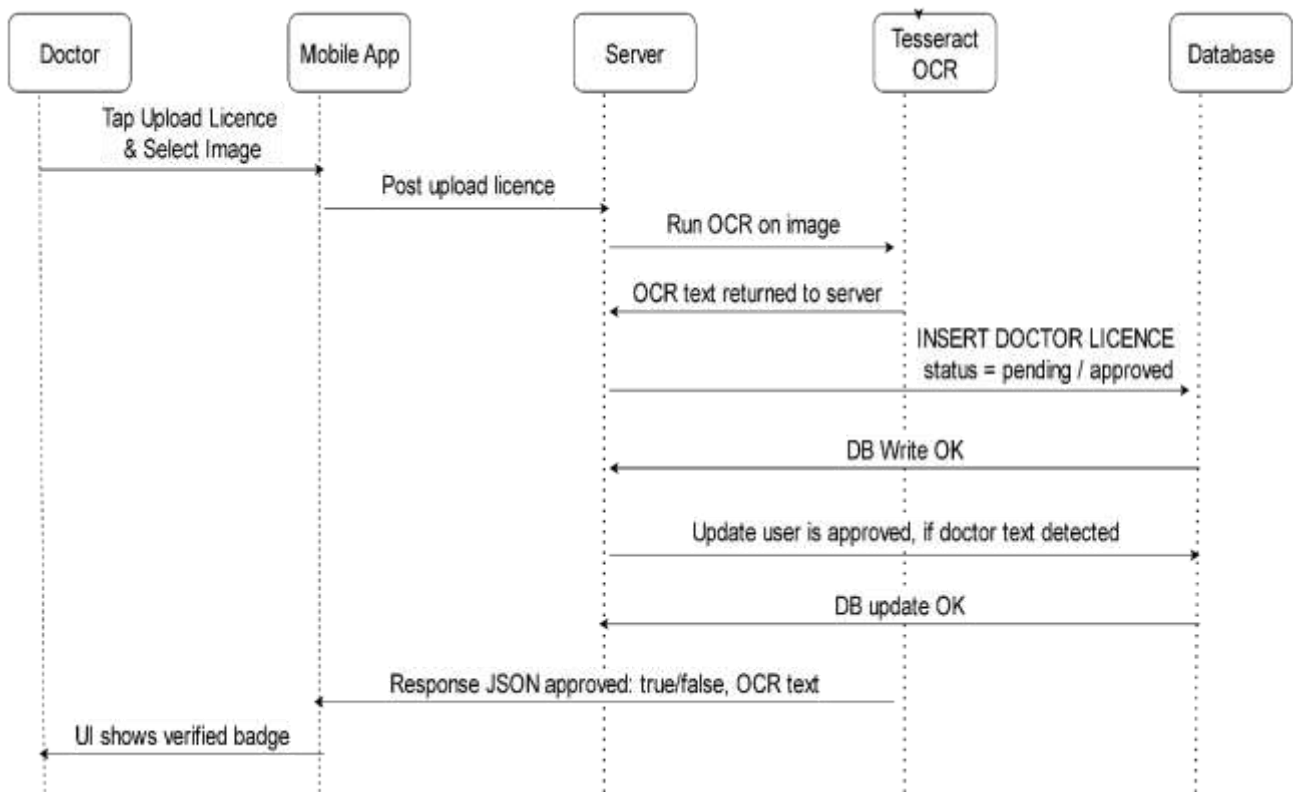
Snippet: OTP Login (Server)

```
db.get(  
  `SELECT * FROM users WHERE phone_number = ? AND country_code = ?`,  
  [phone_number, country_code],  
  (err, user) => {  
    if (!user) return res.status(401).json({ success: false });  
    const token = jwt.sign({ id: user.id, role: user.role }, SECRET, { expiresIn: '7d' });  
    res.json({ success: true, token });  
  }  
);
```

7.4.2 Doctor Licence Upload and OCR Verification

Doctors upload a licence document, which is processed using Tesseract.js OCR. The server extracts text and validates whether it contains expected patterns such as “Dr.” or licence numbers. If valid, the doctor’s status is updated to approved. This ensures only verified professionals can provide medical guidance. [\(See Appendix F.3 for additional snippets\)](#)

Figure 7.3 depicts the sequence diagram of the OCR verification workflow.



Snippet (Server Licence Route):

```
// license.js
const result = await Tesseract.recognize(imagePath, 'eng');
const text = result?.data?.text || '';
const isDoctor = /dr[\s\.\:~]|doctor/i.test(text) || /DOC-\d{4}-[A-Z]{3}/.test(text);

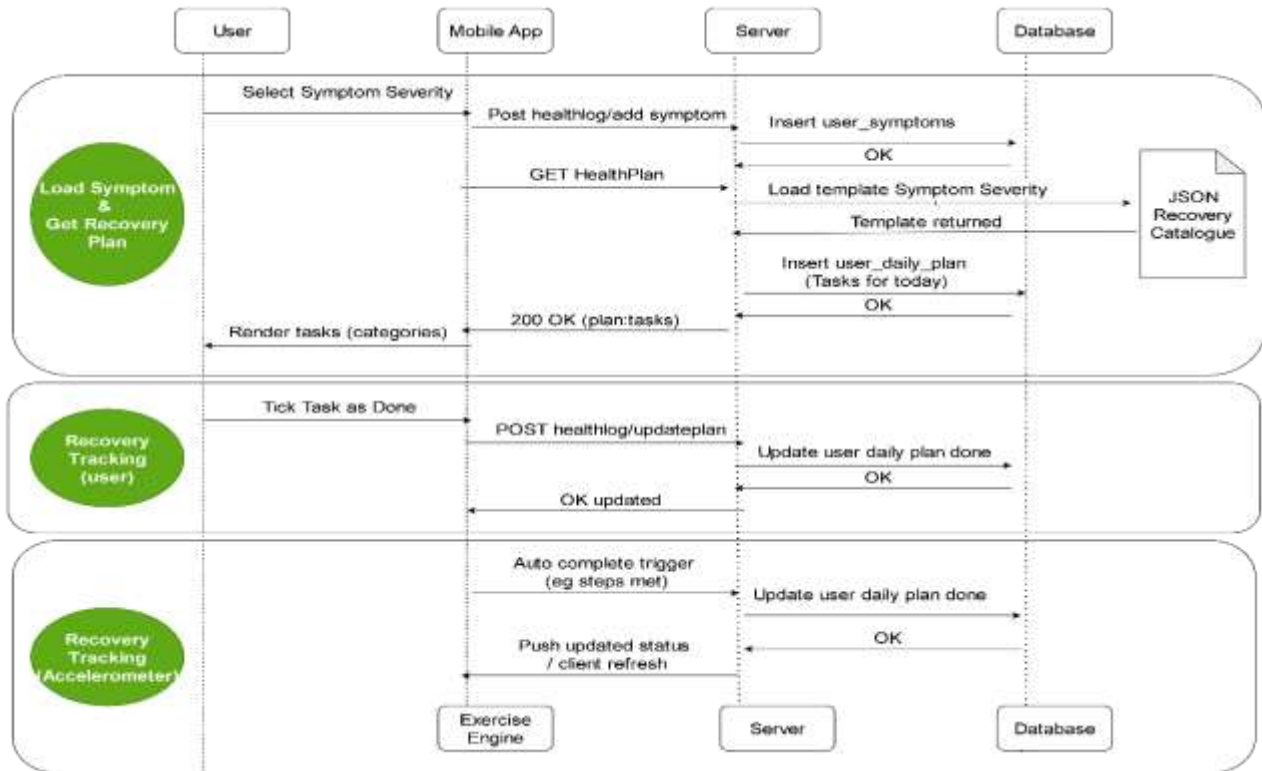
if (isDoctor) {
  db.run(
    'INSERT INTO doctor_licenses (user_id, file_path, status) VALUES (?, ?, ?)',
    [user_id, relativePath, 'approved']
  );
  db.run('UPDATE users SET is_approved = 1 WHERE id = ?', [user_id]);
}
```


7.4.3 Symptom Logging and Recovery Tracking

Patients record daily mood, symptoms, and health metrics through the HealthLogScreen and DailySymptomTrackerScreen, with entries stored locally in SQLite and synced to the backend. The system supports:

- Mood logging with details like energy level and sleep hours.
- Symptom recording (up to three per day).
- Recovery tracking, where users can mark symptoms as resolved.

Figure 7.4 – Flowchart: Symptom Logging and Recovery Workflow



Code Excerpt – Mark Symptom as Recovered (Client):

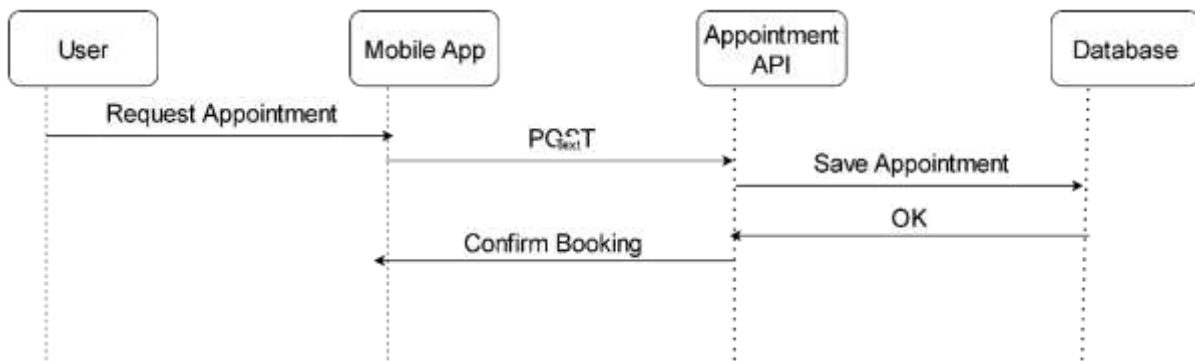
```
// DailySymptomTrackerScreen.js
const handleMarkRecovered = async (symptom) => {
  const recoveredAt = new Date().toISOString();
  const updated = symptoms.map(s =>
    s.symptom === symptom.symptom ? { ...s, recovered_at: recoveredAt } : s
  );
  dispatch(setTodaySymptoms(updated));
  await post(`${API_URL_HEALTHLOG}/recoverSymptom`, {
    user_id: userId, symptom: symptom.symptom, date: symptom.date
  });
};
```

Backend APIs support symptom submission, daily log retrieval, and recovery updates ([see Appendix F.4](#)).

7.4.4 Appointment Booking

The system enables patients to book appointments directly with healthcare providers. The frontend uses `appointmentActions.js` with `createAsyncThunk` to post appointment requests. The backend (`appointment.js`) handles insertion into the appointments table.

Figure 7.5 illustrates the appointment booking process flow.



Code Excerpt – Appointment Booking (Backend):

```
// appointment.js
router.post('/ai-book', async (req, res) => {
  const { date, time, reason, createdBy, chatId } = req.body;
  const stmt = `
    INSERT INTO appointment (user_id, date, time, reason, created_at)
    VALUES (?, ?, ?, ?, datetime('now'))
  `;
  const result = await runQuery(stmt, [createdBy, date, time, reason]);
  res.json({ success: true, appointmentId: result.lastID });
});
```

7.4.5 Real-Time Chat with Encryption

The chat module enables secure real-time communication using **Socket.IO**, supporting both direct and group chats. Messages are encrypted before sending and decrypted upon receipt. Features include group management, read receipts, and queued messages with automatic retry on reconnection.

Figure 7.6 – Sequence Diagram: Real-Time Chat with Encryption



Code Excerpt – Sending a Message (Client):

```
// chatActions.js
export const sendMessage = createAsyncThunk(
  'chat/sendMessage',
  async ({ chatId, senderId, message }, { rejectWithValue }) => {
    const payload = { sender_id: senderId, message, message_type: 'text' };
    const response = await post(`${API_URL_CHAT}/${chatId}/messages`, payload);
    return {
      chatId,
      message: {
        id: response.message_id,
        chat_id: chatId,
        sender: { id: senderId },
        content: message,
        message_type: 'text',
        timestamp: new Date().toISOString(),
      },
    };
  }
);
```

Server: Socket.IO is used to broadcast new messages to all participants ([see Appendix F.6 for additional snippets](#)).

7.4.6 Steps Tracking (Accelerometer Sensor)

Step tracking was implemented using the accelerometer to estimate steps, distance, and calories.

Critical Snippet (StepsTrackerScreen):

```
// StepsTrackerScreen.js
useEffect(() => {
  const unsubscribe = SensorTracker.startStepTracking((stepData) => {
    setData(stepData);
  });
  return () => { if (unsubscribe) unsubscribe(); };
}, []);
```

7.4.7 GPS and Location-Aware Features

TrustCura+ leverages **GPS integration** to capture user location at registration, storing latitude and longitude in the **users** table of SQLite. This geospatial data supports emergency resource mapping, local healthcare recommendations, and location-aware chat groups.

Critical Snippet (Client – RegistrationScreen.js):

```
let { status } = await Location.requestForegroundPermissionsAsync();
if (status === 'granted') {
  let location = await Location.getCurrentPositionAsync({});
  latitude = location.coords.latitude;
  longitude = location.coords.longitude;
}
dispatch(registerUser({ ...form, latitude, longitude }));
```

7.4.8 Radius-Based Location Queries

TrustCura+ computes **proximity searches** between users and healthcare facilities using the **Haversine formula** inside SQL queries. The formula calculates distances based on latitude/longitude, filtering results within a configurable radius (default 5 km). This enables services like “nearest hospital lookup” directly from the mobile client.

Critical Snippet (Server – Example SQL query):

```
SELECT id, name, latitude, longitude,
       (6371 * acos(
         cos(radians(?)) * cos(radians(latitude)) *
         cos(radians(longitude) - radians(?)) +
         sin(radians(?)) * sin(radians(latitude))
       )) AS distance
FROM healthcare_resources
HAVING distance < ?
ORDER BY distance ASC;
```

7.5 Theme Switching (Accessibility – Light/Dark Mode)

Theme switching in TrustCura+ allows users to toggle between light, dark, or system modes. Preferences are saved in AsyncStorage and applied dynamically using the React Native Appearance API. Snippet

```
export const toggleTheme = () => (dispatch, getState) => {  
  const { mode } = getState().theme;  
  const nextMode = mode === 'light' ? 'dark' : mode === 'dark' ? 'system' : 'light';  
  dispatch(applyThemeMode(nextMode));  
};
```

7.6 Security: Encryption of API Requests

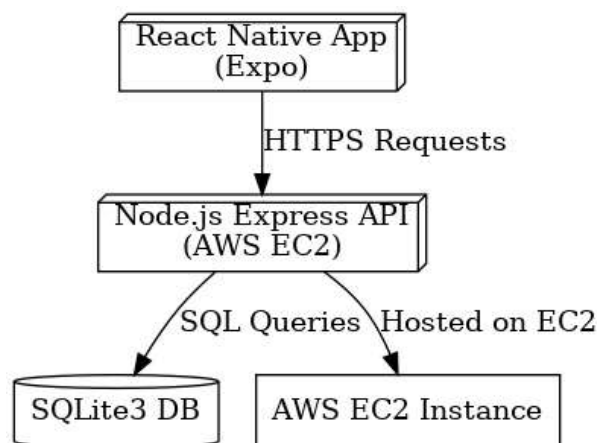
API request and response bodies are encrypted to ensure secure data exchange between client and server. Encryption ensures that sensitive health and communication data is not exposed during transmission.

7.7 Deployment

Deployment was carried out in two parts:

- Mobile App: distributed through Expo, enabling QR-based installation for testers.
- Backend: hosted on AWS EC2, serving REST APIs and handling database access.

A configuration file toggles between development and production, enabling use of mock or live APIs.



7.8 Summary

TrustCura+ was developed as a cross-platform mobile application with React Native, Node.js, and SQLite. Core features included OTP authentication, OCR-based doctor verification, encrypted chat, appointment scheduling, daily health logging, sensor integration, and location-aware resources. Diagrams clarified workflows, [while extended code is documented in Appendix F](#). This implementation demonstrated how modern mobile and cloud technologies can support a secure, scalable, and user-friendly healthcare management platform.

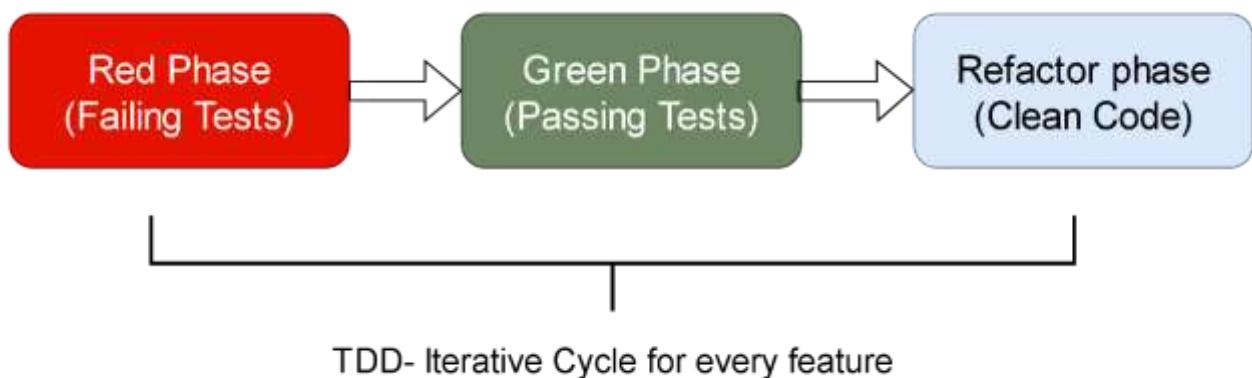
8. Unit Testing

Word count: 447

8.1 Unit Testing Approach

TrustCura+ followed a **Test-Driven Development (TDD)** strategy to ensure systematic feature validation. Using **Jest**, expected behaviours were first defined in test cases (red phase), followed by minimal code to pass them (green phase), and subsequent refactoring for efficiency while retaining correctness. This approach encouraged incremental development and supported robust validation.

For example, Daily Health Check-In was preceded by tests validating mandatory fields, input ranges, and data persistence. Similarly, backend APIs for symptom submission and weather-based recommendations were implemented incrementally using TDD. This process reduced defects, addressed edge cases, and improved confidence during refactoring by maintaining a reliable automated suite.



8.2 Unit Test Framework

Unit testing was conducted for both **mobile (React Native)** and **backend (Node.js)** components using Jest. A single runner simplified cross-platform testing, offering fast execution, watch mode, built-in mocks, snapshot support, and coverage reporting.

Key libraries:

- *Mobile (React Native – Expo)*: jest-expo (preset), @testing-library/react-native (component tests), @testing-library/jest-native (matchers).
- *Backend (Node.js)*: ts-jest or Babel (for TS/ESM), supertest (HTTP testing),nock/msw (API mocking).
- *Utilities*: jest-fetch-mock (network calls), whatwg-fetch polyfill, mockdate (time), and Expo sensor stubs.

The complete dependency list is provided in **Appendix I.4**.

8.3 Unit Testing Execution

Tests were executed continuously during development using Jest watch mode for rapid feedback. Coverage reports tracked function and branch coverage. Example commands:

```
npm test  
npm test -- --coverage
```

This ensured code quality and identified regressions early [Comprehensive list at Appendix I.3](#)

8.4 Representative Unit Test Cases

A broad range of cases was developed, with representative examples including:

1. Symptom Logging: Empty entries initially passed; validation added for non-empty input and severity range (1–5).
2. Task Completion: Database update bug corrected with explicit persistence logic.
3. GPS Facility Locator: Incorrect parameters fixed by correcting API query and coordinate handling.
4. Messaging Encryption: Messages initially stored in plain text; AES-based encryption and decryption implemented to secure exchanges.

These examples illustrate the red–green–refactor cycle, where failing tests drove targeted fixes. [Extended logs are included in Appendix I.1](#)

8.5 Unit Testing Results

The Jest suite achieved comprehensive coverage across modules, running 26 representative tests: 17 passed initially, while nine failures revealed issues in GPS integration, symptom validation, and messaging security. All were resolved through iterative fixes. Additional problems included missing mandatory fields and inconsistent timestamp storage, which were corrected with validation and standardized ISO formats. [Full results appear in Appendix I.2.](#)

8.6 Unit Test Coverage and Limitations

Automated tests achieved approximately

- Client side: 75% function coverage and 72% branch coverage
- Server side: 72% function coverage and 50% branch coverage

validating core modules including symptom logging, task management, recommendations, authentication, and messaging across both normal and edge cases. Unit testing confirmed module reliability, though it cannot guarantee complete correctness. Advanced integration and UI automation (e.g., Cypress, Detox) remain future priorities to reduce reliance on manual verification. Detailed coverage statistics are presented in [Appendix I.5 – Test Coverage.](#)

9. Integration Process and Testing.

Word count: 357

9.1 Approach

While unit tests validated modules in isolation, integration testing is carried out to ensure correct interaction between components of **TrustCura+**. Since automated integration frameworks were not within scope, testing is performed manually on staging builds (v0.9.3-stg) across Android 2 and iOS 2 devices. Representative workflows were simulated, including authentication, symptom logging, GPS resource lookup, recommendations, chat communication, and progress tracking.

9.2 Test Scenarios

The following scenarios were validated:

- Authentication → Symptom Logging – Verifying token-based access to log entries.
- Symptom Logging → Recommendations – Ensuring logged symptoms generated correct, environment-aware plans.
- GPS → Resource Finder → Recommendations – Confirming accurate healthcare resource mapping and context-aware recommendations.
- Authentication → Communication (Chat) – Securing doctor–patient messaging via encryption and token validation.
- Task Management → Progress Dashboard – Checking completion ratios reflected correctly in analytics.
- Offline Logging → Sync on Reconnect – Ensuring logs created offline synced without duplication.
- Media Upload → Chat → Doctor View – Verifying patient-uploaded photos displayed correctly to doctors.

9.3 Interpretation

Integration testing confirmed that once unit-level bugs were resolved, modules interacted correctly across end-to-end workflows. The most critical issues—such as missing input validation, GPS parameter mismatches, and lack of chat encryption—were uncovered and fixed early, significantly strengthening system reliability. Final retesting showed all scenarios passed, with the platform demonstrating stable interactions between authentication, data logging, recommendation generation, and doctor communication.

9.4 Limitations

Testing is performed manually, which ensured real workflows were validated but left potential gaps in repeatability and regression coverage. Future work should include automated integration and end-to-end UI tests (e.g., Cypress, Detox) to complement manual testing and provide continuous verification across environments. A key limitation of the project is the lack of iOS testing. While the app has been developed and tested on Android Studio, testing on iOS devices has not been conducted due to resource constraints and the need for an Apple development environment.

9.5 Summary of Results

Overall, all scenarios passed after fixes, with improvements in validation, API compatibility, encryption, and offline data handling. A detailed breakdown of test cases, bugs, fixes, and outcomes is provided in [Appendix J \(Integration Test Results\) for reference](#).

10. Evaluation

Word count: 620

10.1 Evaluation Criteria

Evaluation of *TrustCura+* focused on usability and accuracy, while formal performance testing was deferred.

Usability: Informal testing with peers and family (n=4) covered sign-up, symptom logging, recommendations, and doctor messaging. Navigation was clear, daily logs were valued for simplicity, and doctor role testers appreciated structured histories. However, participants requested stronger visual cues for reminders and simplified summaries, aligning with evidence that clear design and timely prompts support adoption (Grundy et al., 2022).

Performance: No formal load or stress testing was completed due to project scope. Screens loaded promptly during ad hoc checks on mid-range Android devices, but latency and throughput metrics were not systematically measured. Future work should implement automated benchmarks to evaluate scalability (WHO, 2019).

Accuracy: Evaluation assessed whether recommendations and validation operated as intended. Outputs matched common self-care practices (e.g., hydration for fever, rest for fatigue) and consistently reflected expected guidance. Validation rules successfully blocked invalid entries such as empty fields or symptom severity outside the 1–5 scale, maintaining data quality.

10.2 Comparison with Existing Solutions.

TrustCura+ was benchmarked against MyChart, Ada Health, and Google Fit. While competitors address specialised domains, none combine everyday illness management with integrated communication and context-aware support.

- MyChart: Hospital-focused, centred on medical records and appointments; *TrustCura+* is lighter and independent of provider systems.
- Ada Health: Provides AI symptom triage but lacks continuous logging, contextual recommendations, or direct doctor communication.
- Google Fit: Strong in fitness tracking and wearables, but not designed for illness recovery or healthcare guidance.

This positioning highlights *TrustCura+* as a differentiated tool that bridges features fragmented across existing solutions ([see Appendix K.1](#)).

10.3 Innovation & Differentiation

Innovation lies in **integration** rather than novelty. *TrustCura+* consolidates symptom logging, adaptive recommendations, encrypted doctor–patient chat, and GPS-based resources within a single lightweight app. Informal testing indicated this coherence improved clarity, usability, and perceived value compared to siloed alternatives. Further innovations, such as passive sensing and predictive analytics, are discussed in Section 10.5 ([Appendix K.2](#)).

10.4 Limitations of Current System.

The main limitations identified were:

- No performance/load evaluation – scalability is untested.
- Manual integration testing – limited repeatability (Ammar et al., 2021).
- External API dependence – GPS and weather service outages could disrupt functionality.
- Limited personalisation – rule-based recommendations do not adjust to long-term histories.
- No wearable integration – excludes continuous tracking (e.g., heart rate, sleep cycles).
- Scope boundaries – chronic and emergency care are deliberately excluded.
- Connectivity reliance – many features require stable internet access.
- Limited cross-platform testing – iOS testing remains unperformed, limiting full compatibility evaluation.

10.5 Future Enhancements

TrustCura+ has met its initial goal of integrating symptom tracking, recommendations, and doctor–patient communication in a single mobile app, but several improvements remain. Future work should expand validation beyond student testers to include patients and clinicians, ensuring accuracy and trust. Integration with wearable devices would enable continuous monitoring of vital signs such as heart rate and sleep. Passive sensing could reduce manual input by collecting background data on activity and environment. Additional modules for chronic disease management and the use of machine learning for predictive analytics would further enhance personalisation and scalability.

Planned directions include broader user validation, wearable device support, passive sensing, chronic disease modules, and AI-driven analytics.

[Detailed proposals are in Appendix K.3.](#)

10.6 Critical Evaluation

Test-Driven Development improved reliability and supported refactoring, though reliance on manual integration testing reduced consistency; future work should introduce automated suites (Ammar et al., 2021). Focusing on everyday recovery made scope manageable but limited broader applicability, while rule-based recommendations remain less adaptive than machine-learning models. Usability insights came from a small, homogenous sample, and accuracy checks lacked clinical validation. Despite these limitations, TrustCura+ successfully demonstrated feasibility by integrating logging, adaptive advice, secure communication, and local resource access. [See Appendix K.4 for extended critique.](#)

11. Conclusion

Word count: 283

The development of **TrustCura+** demonstrates how mobile health applications can provide practical support for users managing day-to-day health concerns. Unlike many existing solutions that focus on single functions such as fitness tracking or medication reminders, **TrustCura+** integrates symptom logging, wellness recommendations, secure communication, and sensor-enabled monitoring into a single platform. By doing so, it addresses the issue of fragmented user experiences in mHealth and offers a more holistic tool for everyday illness recovery.

The project followed a structured process, beginning with research into existing gaps, concept development, wireframing, prototyping, and feedback analysis. A clear system architecture was implemented using React Native (Expo) for the mobile interface, Node.js for backend services, SQLite3 for data persistence, and AWS EC2 for deployment. Embedded sensors such as the accelerometer, GPS, and camera enhanced functionality by enabling activity monitoring, location-based services, and image-based symptom logging.

Testing through Jest and TDD ensured reliability and revealed practical issues—such as improper validation and incorrect API parameters—that were resolved before integration. User feedback, while limited to student groups, provided useful insights into usability and design improvements. The inability to test directly with healthcare professionals and patients remains a limitation but also presents an opportunity for future validation studies.

In its current form, **TrustCura+** supports common, non-chronic health issues such as fever, mild fatigue, or seasonal flu. Future work could expand the system by integrating wearable devices, passive sensing, and professional validation to extend its scope.

Overall, the project highlights both the practical application of computer science methods to healthcare and the innovative potential of integrating structured symptom tracking, adaptive recommendations, and secure communication into one lightweight tool. This positions **TrustCura+** as a differentiated contribution within the mHealth landscape.

12. References

- Ammar, T., Ayed, A. B., & Rzig, S. (2021). Continuous integration and continuous deployment in mobile app development. *Journal of Software Engineering and Applications*, 14(3), 101–114.
- Boulos, M. N. K., Brewer, A. C., Karimkhani, C., Buller, D. B., & Dellavalle, R. P. (2014). Mobile medical and health apps: State of the art, concerns, regulatory control, and certification. *Online Journal of Public Health Informatics*, 5(3), 229. <https://doi.org/10.5210/ojphi.v5i3.4814>
- Cornet, V. P., & Holden, R. J. (2018). Systematic review of smartphone-based passive sensing for health and wellbeing. *Journal of Biomedical Informatics*, 77, 120–132. <https://doi.org/10.1016/j.jbi.2017.12.008>
- Deniz-Garcia, A., et al. (2023). Quality, usability, and effectiveness of mHealth apps and wearables. *Journal of Medical Internet Research*. Retrieved from <https://pmc.ncbi.nlm.nih.gov/articles/PMC/>
- Digital Health Interventions. (2022). *Wikipedia*. Retrieved from https://en.wikipedia.org/wiki/Digital_health_interventions
- El Morr, C., Ritvo, P., Ahmad, F., & Moineddin, R. (2020). Effectiveness of mobile apps for managing stress in university students: Systematic review and meta-analysis. *JMIR Mental Health*, 7(6), e16949. <https://doi.org/10.2196/16949>
- Epic Systems. (2021). *MyChart: Connecting patients to healthcare providers*. Epic.
- European Climate Adaptation Platform. (2024). *Climate tools for mobile health*. Retrieved from <https://climate-adapt.eea.europa.eu/>
- German Journal of Sports Medicine. (2024). *Accelerometry in sports medicine and rehabilitation*. Retrieved from <https://www.germanjournalsportsmedicine.com>
- Ghose, A., Li, B., & Liu, Y. (2021). Empowering users using smart mobile health platforms: Evidence from randomized field experiments. *BMC Public Health*, 21(1), 1–12. <https://doi.org/10.1186/s12889-021-10484-6>
- Google. (2023). *Google Fit app overview*. Retrieved from <https://www.google.com/fit>
- Grundy, M., Abdelrazek, M., & Curumsing, M. K. (2022). Data collection mechanisms in health and wellness apps: Review. *JMIR mHealth and uHealth*, 10(8), e38056. <https://doi.org/10.2196/38056>
- Grundy, Q., Held, F. P., & Bero, L. A. (2022). Apps for health behaviour change: A content analysis. *JMIR mHealth and uHealth*, 7(2), e11959. <https://doi.org/10.2196/11959>

Johnson, D., Deterding, S., Kuhn, K. A., Staneva, A., Stoyanov, S., & Hides, L. (2016). Gamification for health and wellbeing: A systematic review of the literature. *Internet Interventions*, 6, 89–106. <https://doi.org/10.1016/j.invent.2016.10.002>

Martinez-Perez, B., de la Torre-Díez, I., & López-Coronado, M. (2013). Privacy and security in mobile health apps: A review and recommendations. *Journal of Medical Systems*, 39(1), 181. <https://doi.org/10.1007/s10916-014-0181-3>

Piras, E. M., & Murgia, V. (2017). The use of personal health data: A scoping review of the literature. *BMC Medical Informatics and Decision Making*, 17, 83. <https://doi.org/10.1186/s12911-017-0478-1>

PMC. (2019). *Smartphone sensors for medical imaging*. National Library of Medicine. Retrieved from <https://pmc.ncbi.nlm.nih.gov/articles/PMC/>

ResearchGate. (2023). *Smartphone geolocation for behavior tracking*. Retrieved from <https://www.researchgate.net/>

University of Minnesota. (2023). *Digital nudges based on weather increase exercise*. University of Minnesota Twin Cities. Retrieved from <https://twin-cities.umn.edu/>

Ventola, C. L. (2014). Mobile devices and apps for health care professionals: Uses and benefits. *P & T: A Peer-Reviewed Journal for Formulary Management*, 39(5), 356–364.

World Health Organization (WHO). (2019). *WHO guideline: Recommendations on digital interventions for health system strengthening*. WHO. <https://apps.who.int/iris/handle/10665/311941>

13. Appendices

The following appendices provide supporting material that complements the main body of this dissertation. Together, these appendices ensure transparency, traceability, and completeness of the project documentation while keeping the main body concise and focused.

13.1 Appendix A – Research Insights to Feature Mapping

| Research Insights | TrustCura+ Feature Implementation | Related Epics |
|---|---|--------------------|
| mHealth improves behaviours and outcomes | Symptom tracking, wellness recommendations, doctor-user communication | Epics 3, 4, 6 |
| Current apps are fragmented | Unified platform combining tracking, recommendations, communication | All epics |
| Accelerometers support movement & sleep | Activity tracking via accelerometer | Epics 3, 5 |
| GPS enables healthcare services | GPS-based facility finder & outdoor activity mapping | Epic 7 |
| Cameras enable remote symptom documentation | Camera-enabled symptom capture for teleconsultations | Epics 4, 6 |
| Weather-based nudges increase engagement | Weather-integrated recommendations | Epic 9 |
| Passive sensing reduces user burden | Future-ready for continuous monitoring | Epic 9 (extension) |

13.2 Appendix B – Detailed User Personas

B.1 Primary Persona: “Student Aisha”

Demographics: Name: Aisha Khan.

- Age: 21
- Occupation: Undergraduate Computer Science Student
- Location: Manchester, UK
- Living Situation: Shared student accommodation
- Tech Savviness: High

Goals

- Track daily wellbeing indicators such as mood, sleep, and energy.
- Receive simple, context-based recommendations (e.g., hydration, exercise reminders).
- Maintain balance during exam periods by monitoring fatigue and stress.

Motivations

- Health awareness and productivity.
- Convenience compared to manual journaling.
- Self-improvement through feedback and trends.

Frustrations

- Limited access to quick medical advice.
- Overcomplicated apps with non-relevant features.
- Forgetfulness in logging without prompts.

Behaviour Patterns

- Uses mobile reminders daily.
- Tracks sleep and energy around exams.
- Relies on notifications for consistency.

Justification

Primary Persona (Student Aisha): University students are frequent users of wellbeing apps, especially for stress and sleep management (El Morr et al., 2020). Engagement increases when apps are simple and provide reminders, while complexity reduces continued use (Grundy et al., 2022).

B.2 Secondary Persona: “Office Worker Daniel”

Demographics

- Name: Daniel Wright
- Age: 32
- Occupation: Marketing Executive
- Location: Birmingham, UK
- Living Situation: Lives alone in a rented flat
- Tech Savviness: Moderate

Goals

- Monitor symptoms like headaches or fatigue linked to workload.
- Receive short, clear suggestions for balance between work and wellbeing.
- Access summaries without heavy medical data.

Motivations

- Time-saving, especially at work.
- Preventive care to reduce sick days.
- Preference for clear, actionable feedback.

Frustrations

- Generic, non-personalised advice.
- Too many metrics creating overload.
- Inconsistency in logging after long workdays.

Behaviour Patterns

- Uses apps during commutes and work breaks.
- Logs symptoms when reminders are timely.
- Reviews weekly summaries for patterns.

Justification

Secondary Persona (Office Worker Daniel): Professionals often adopt lightweight apps to track stress and work-related fatigue (Ghose et al., 2021). However, discontinuation is common when apps demand too much data entry or provide vague recommendations (Cornet & Holden, 2018).

B.3 Tertiary Persona: “Doctor Emily”

Demographics

- Name: Dr. Emily Carter
- Age: 42
- Occupation: General Practitioner
- Location: Leeds, UK
- Living Situation: Lives with family in suburban area
- Tech Savviness: Moderate–High (uses medical systems daily, mobile apps occasionally)

Goals

- Verify professional credentials easily on the platform.
- Access structured patient logs (symptoms, wellbeing, tasks) instead of unstructured text messages.
- Provide quick, context-based advice via secure messaging (text, image).
- Set and manage consultation availability for patients.

Motivations

- Improve efficiency by reducing unnecessary in-person visits.
- Offer reliable support to patients in between appointments.
- Enhance professional reputation by using a secure, verified platform.

Frustrations

- Receiving incomplete or unclear symptom descriptions from patients.
- Time wasted on fragmented tools (e.g., WhatsApp, email) without medical structure.
- Difficulty managing availability across multiple systems.

Behaviour Patterns

- Reviews patient logs during free slots between appointments.
- Prefers structured dashboards with symptom trends over raw text.
- Responds to patients using concise text.

Justification

Tertiary Persona (Doctor Emily): Verified doctors need structured, time-efficient tools to provide quality care. Research shows that lack of secure and organized communication channels leads to inefficiencies and risks in patient management (Martinez-Perez et al., 2013; Piras & Murgia, 2017). By integrating verification, structured logs, and secure messaging, TrustCura+ addresses these challenges directly.

13.3 Appendix C – Detailed User Stories

| Epic | User Story ID | User Story | Acceptance Criteria |
|--|---------------|---|---|
| Epic 1: User Authentication & Authorization | US1.1 | As a user, I want to register using my phone number with auto country code. | GPS auto-detects country; OTP sent to phone; OTP auto-verified; user role selected. |
| | US1.2 | As a user, I want to log in with OTP so I can access my account easily. | OTP generated and sent; silent auto-verification; manual entry fallback; successful login redirects to dashboard. |
| | US1.3 | As a user, I want to log out securely. | Logout button available; session cleared; return to login screen. |
| Epic 2: Doctor Verification & Profile Management | US2.1 | As a doctor, I want to upload my medical certifications for verification. | Upload licenses/degrees; OCR validation; admin approval; verified badge displayed. |
| | US2.2 | As a doctor, I want to manage my profile details. | Editable fields: name, photo, specialization, bio; changes updated on public profile. |
| | US2.3 | As a doctor, I want to set my availability slots. | Add/edit/remove slots; visible to users for booking. |
| Epic 3: Daily Well-being Logging | US3.1 | As a user, I want to record my daily mood, energy, and sleep quality. | One entry per day; intuitive input (sliders, icons); timestamped and stored. |
| Epic 4: Symptom Tracking | US4.1 | As a user, I want to log up to three symptoms per entry. | Fields for symptom name, severity, optional notes; entries timestamped. |
| Epic 5: Health Task Management | US5.1 | As a user, I want to view and complete daily health tasks. | Task list visible; each task can be marked complete. |
| | US5.2 | As a user, I want to add custom health tasks. | Search predefined tasks; option to create custom; set name, type, and quantity. |
| Epic 6: Doctor–User Communication | US6.1 | As a user, I want to send text, image messages to my doctor. | Secure chat interface; supports text, image; real-time notifications; history saved. |
| | US6.2 | As a doctor, I want to reply to messages and review user logs. | Secure reply; access to user well-being, symptoms, tasks; ability to mark tasks complete. |
| | US6.3 | As a user, I want to send my calendar in chat so users | Users send cards in chat that display and allow |

| | | | |
|---|---------|---|---|
| | | (patients/doctors) can book an appointment | action to schedule a new appointment |
| Epic 7: Geo-Location & Resource Finder | US7.1 | As a user, I want to see nearby healthcare facilities. | GPS detects location; facilities shown with name, distance, contact. |
| | US7.2 | As a user, I want to filter providers by category and reviews. | Filters available (hospital, pharmacy, dentist); ratings and reviews shown. |
| Epic 8: Trend Analysis & Visualization | US8.1 | As a user, I want to view mood, sleep, and energy trends. | Graphs available for selected timeframe (7D, 30D, custom). |
| | US8.2 | As a user, I want to monitor symptom progress. | Graphs show symptom severity over time. |
| | US8.3 | As a user, I want to track task completion trends. | Graphs show completion percentage; filters for timeframes. |
| Epic 9: Personalized Recommendations & Alerts | US9.1 | As a user, I want daily health recommendations based on my symptoms and weather. | Dynamic recommendations generated; context-sensitive (e.g., hydration in hot weather). |
| | US9.2 | As a user, I want real-time alerts based on local conditions. | Notifications triggered by poor air quality, heavy rain, or similar. |
| Epic 10: Data Security & Privacy | US10.1 | As a user, I want my data stored securely. | End-to-end encryption; privacy settings available. |
| | US10.2 | As a user, I want assurance of data privacy compliance. | Sessions protected; secure hosting; no unauthorized access. |
| Epic 11: Schedule an appoint with doctor | US 11.1 | As a user, I want to create an appointment with a doctor | view doctor's available slots, selects a slot and provides a reason, appointment created, appointment appears in user's calendar. |
| | US 11.2 | As a user/doctor, I want to view calendar of appointment so I can manage my time and prepare for consultations. | Calendar shows appointments, filters for days/week/month |

13.4 Appendix D – Key Metrics and calculations

1. Symptom Tracking – Users log symptoms with severity and duration. Structured logs with timestamps enable trend analysis and support better doctor consultations (WHO, 2019).

To combine multiple symptoms into a daily score, a Symptom Severity Index is calculated as:

$$S = \frac{\sum_{i=1}^n w_i * S_i}{\sum_{i=1}^n w_i}$$

where s_i is the severity of symptom i and w_i is its assigned weight.

2. Personalized Recommendations – Advice adapts based on user data and environmental inputs. For example, hydration reminders may appear during hot weather, supported by findings that digital health interventions improve behavioural adherence (Deniz-Garcia et al., 2023).

$$A = H + \gamma W$$

where H is the base health plan, W is an environmental adjustment factor (e.g., temperature, air quality), and γ controls sensitivity.

3. Communication – Encrypted channels allow text, images exchange between users and doctors, replacing informal tools like messaging apps that lack medical context (Piras & Murgia, 2017).

To ensure priority handling, alerts and messages are ranked by a Notification Priority Score:

$$P = \delta_1 S + \delta_2 O + \delta_3 E$$

where S is symptom severity, O is outbreak risk, and E is environmental stressors.

4. Sensor Utilization

Accelerometer: activity and sedentary behaviour tracking (German Journal of Sports Medicine, 2024).

GPS: location-based healthcare resource finder and mapping outdoor activities (PMC, 2019). Distance to facilities is calculated via the Haversine formula:

$$d = 2r \arcsin \left(\sqrt{\sin^2 \left(\frac{\Delta\varphi}{2} \right) + \cos(\varphi_1) \cos(\varphi_2) \sin^2 \left(\frac{\Delta\lambda}{2} \right)} \right)$$

where φ and λ denote latitudes and longitudes of user and facility, and r is Earth's radius.

5. Camera: image capture for dermatological or eye-related consultations (Cornet & Holden, 2018).

13.5 Appendix E – Traceability Matrix

| Epic | Feature Implementation | Conceptual Framework / Equations |
|--|---|--|
| Epic 1: Authentication & Authorization | OTP registration, login, logout | System Core – secure session management |
| Epic 2: Doctor Verification & Profile | Credential upload with OCR, profile editing, availability slots | System Core – credential verification module |
| Epic 3: Daily Well-being Logging | Mood, energy, sleep entry | Recovery Score (R) integrates adherence with wellness logs |
| Epic 4: Symptom Tracking | Log up to 3 symptoms with severity and notes | Symptom Severity Index (S) = weighted score of symptoms |
| Epic 5: Health Task Management | View, complete, add custom tasks | Recovery Score (R) includes task adherence (T_c/T_t) |
| Epic 6: Doctor–User Communication | Encrypted chat (text, image), task completion review | Notification Priority (P) ranks alerts for doctor review |
| Epic 7: Geo-Location & Resource Finder | GPS-based map of clinics, hospitals, pharmacies | Geo-Distance (d) via Haversine formula |
| Epic 8: Trend Analysis & Visualization | Graphs for mood, sleep, symptoms, tasks | System Core – data visualization module |
| Epic 9: Recommendations & Alerts | Dynamic advice based on symptoms + weather | Adaptive Recommendation (A) = $H + \gamma W$ |
| Epic 10: Data Security & Privacy | End-to-end encryption, secure storage | System Core – encryption and privacy enforcement |
| Epic 11: Appointment Scheduling | Appointment lifecycle: patient books slot, doctor confirms/declines, shared calendars update for both roles | System Core – Appointment module (Appointments table, status flow) Calendar views derived from Appointments |

13.6 Appendix F – Development – Extended Code Snippets

This appendix presents key TrustCura+ code excerpts, highlighting essential logic and main feature implementation. Concise listings with explanations provide context, avoiding redundancy, while the complete source remains accessible in the repository.

F.1 – Prototype Technical Configuration

The **TrustCura+** application is developed using a combination of mobile, backend, database, and cloud technologies. The selection of each tool is based on suitability for mobile healthcare applications, ease of development, scalability, and cost-effectiveness.

- **Frontend: React Native (Expo Framework)**
React Native is chosen for mobile application development because it supports cross-platform deployment on Android and iOS with a single codebase. The Expo framework simplifies access to device features such as camera, GPS, and accelerometer, which are core requirements for TrustCura+. It also provides a rapid testing and debugging environment suitable for iterative development.
- **Backend: Node.js**
Node.js is used to implement backend services due to its event-driven, non-blocking architecture, which enables efficient handling of multiple user requests. Express.js, a lightweight web framework, is used for building RESTful APIs to connect the mobile app with backend services.
- **Database: SQLite3**
SQLite3 is selected as the database for its lightweight nature, local storage capabilities, and offline-first support. It is embedded within the application and is well-suited for handling structured data such as symptom logs, user accounts, and communication records.
- **Cloud Hosting: AWS EC2**
Amazon Web Services Elastic Compute Cloud (AWS EC2) is chosen to host backend services. EC2 offers scalability, cost management through pay-as-you-go billing, and reliability. By hosting the backend on EC2, the system ensures consistent access to APIs and maintains secure communication between the app and the database.
- **External APIs**
 - **Weather API:** Provides weather-based health recommendations.
 - **Maps API:** Enables the healthcare facility locator and navigation services.
- **Device Sensors**
 - **Accelerometer:** Tracks activity levels and sedentary behaviour.
 - **GPS:** Provides location-aware services.
 - **Camera:** Captures pulse rate
- App has been developed and tested on Android Studio

F.2 OTP Authentication

2.1 Send OTP route (Node.js express)

```
// routes/auth.js
router.post("/send-otp", async (req, res) => {
  const { phone } = req.body;
  if (!phone) return res.status(400).json({ error: "phone required" });

  const otp = generateOTP();
  const ts = Date.now();

  await db.run(
    "INSERT INTO otp_logins (phone, otp_code, ts) VALUES (?, ?, ?)",
    [phone, otp, ts]
  );

  console.log(`[SMS to ${phone}] ${otp}`); // SMS gateway disabled to avoid costs
  res.json({ ok: true });
});
```

Generates an OTP, stores it, and simulates SMS delivery (gateway disabled for cost reasons).

2.2 Verify OTP route (with 5-minute expiry)

```
// routes/auth.js
router.post("/verify-otp", async (req, res) => {
  const { phone, code } = req.body;

  const row = await db.get(
    "SELECT otp_code, ts FROM otp_logins WHERE phone = ? ORDER BY [phone]"
  );

  if (!row) return res.status(400).json({ error: "no otp" });
  const expired = Date.now() - row.ts > 5 * 60 * 1000;

  if (expired || row.otp_code !== code) {
    return res.status(401).json({ error: "invalid or expired" });
  }

  await db.run(
    "INSERT INTO sessions (phone, created_at) VALUES (?, ?)",
    [phone, Date.now()]
  );

  res.json({ ok: true });
});
```

Validates the most recent OTP within 5 minutes and creates a session record.

2.3 Manual OTP fallback (React Native client)

```
const OtpFallback = ({ phone }) => {
  const [code, setCode] = useState("");
  const verify = async () => {
    await fetch(`${API}/auth/verify-otp`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ phone, code })
    });
  };
  return (
    <View>
      <TextInput value={code} onChangeText={setCode} keyboardType="number-pad" />
      <Button title="Verify" onPress={verify} />
    </View>
  );
};
```

Provides manual entry when auto-verification fails

F.3 Doctor Credentials Upload + OCR

3.1 Upload handler with OCR

```
import multer from "multer";
import { extractText } from "../services/ocr.js";

const upload = multer({ dest: "uploads/" });

router.post("/upload-credential", upload.single("doc"), async (req, res) => {
  if (!req.file) return res.status(400).json({ error: "file required" });

  const text = await extractText(req.file.path);
  const status = text.length > 20 ? "under_review" : "rejected";

  await db.run(
    "INSERT INTO doctor_licenses (phone, file_path, status, ocr_text) VALUES (?, ?, ?, ?)",
    [req.body.phone, req.file.path, status, text]
  );

  res.json({ ok: true, status });
});
```

Handles credential uploads and stores OCR results for verification

F.4 Daily well-being Logging

4.1 Add daily log endpoint

```
router.post("/add-daily", async (req, res) => {
  const { phone, mood, sleep_quality, energy, date } = req.body;
  await db.run(
    `INSERT INTO daily_logs (phone, mood, sleep_quality, energy, date)
    VALUES (?, ?, ?, ?, ?)`,
    [phone, mood, sleep_quality, energy, date || new Date().toISOString().slice(0,10)]
  );
  res.json({ ok: true });
});
```

Stores one daily log entry per user per day.

4.2 React Native sliders UI

```
<Slider value={mood} onValueChange={setMood} minimumValue={0} maximumValue={10} />
<Slider value={sleep} onValueChange={setSleep} minimumValue={0} maximumValue={10} />
<Slider value={energy} onValueChange={setEnergy} minimumValue={0} maximumValue={10} />
<Button title="Save" onPress={submitLog} />
```

Compact UI for recording well-being values.

F.5 Symptom Tracking

5.1 Add symptom endpoint

```
router.post("/add", async (req, res) => {
  const { phone, name, severity, notes } = req.body;
  await db.run(
    `INSERT INTO symptoms (phone, symptom_name, severity, notes, date)
    VALUES (?, ?, ?, ?, ?)`,
    [phone, name, severity, notes || "", new Date().toISOString()]
  );
  res.json({ ok: true });
});
```

Allows entry of symptoms with severity and optional notes.

F.6 Messaging (Socket.io Extended)

6.1 Socket.io server with persistence

```
import { Server } from "socket.io";
const io = new Server(httpServer, { cors: { origin: "*" } });

io.on("connection", (socket) => {
  socket.on("join", (room) => socket.join(room));
  socket.on("message", async (msg) => {
    await db.run(
      "INSERT INTO messages (room, sender, content, ts) VALUES (?, ?, ?, ?)",
      [msg.room, msg.sender, msg.content, Date.now()]
    );
    io.to(msg.room).emit("message", msg);
  });
});
```

Supports real-time chat and stores messages in SQLite.

6.2 React Native client hook

```
import { io } from "socket.io-client";
const socket = io(API_BASE);

export const useChat = (room) => {
  useEffect(() => { socket.emit("join", room); }, [room]);
  const send = (content, sender) => socket.emit("message", { room, content, sender });
  const subscribe = (cb) => { socket.on("message", cb); return () => socket.off("message", cb) };
  return { send, subscribe };
};
```

Handles joining rooms, sending, and receiving messages on the client.

F.7 Utility Functions

7.1 Auto country code detection

```
import * as Localization from "expo-localization";
export const guessDialCode = () => {
  const c = Localization.region; // e.g., "GB"
  const map = { GB: "+44", IN: "+91", US: "+1" };
  return map[c] || "+44";
};
```

Prefills phone input with a country code depending on location

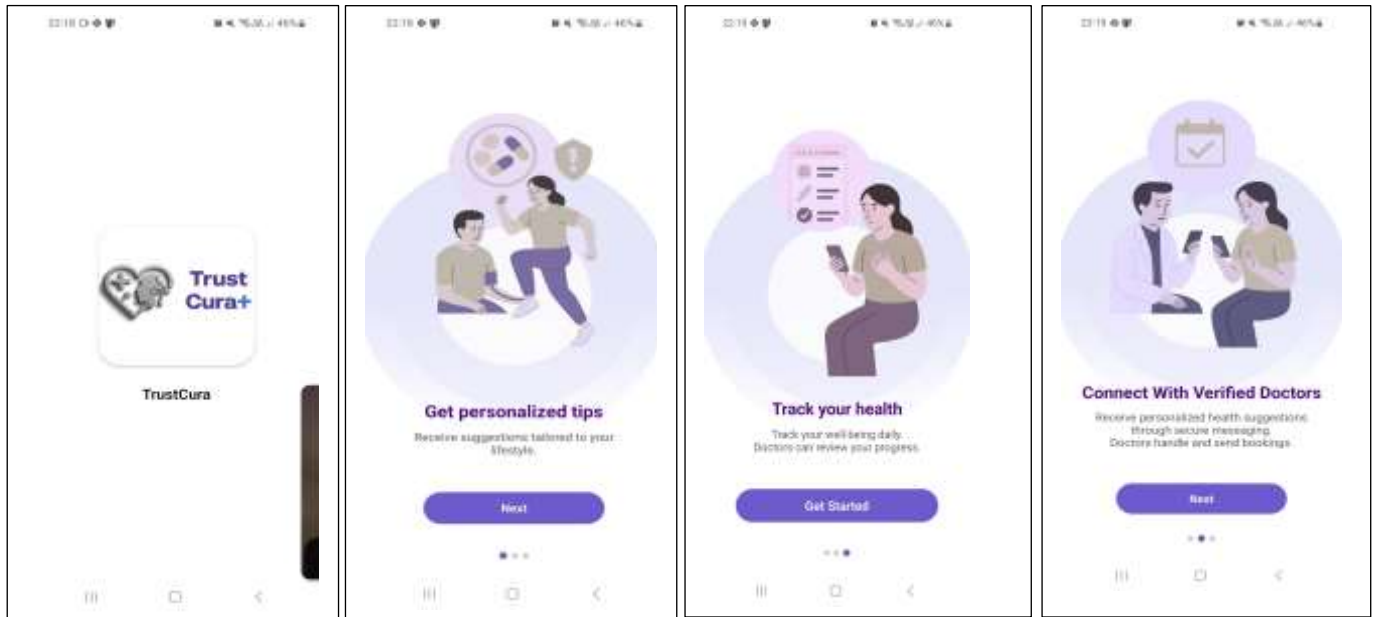
7.2 JSON request validation middleware

```
export const requireJson = (req, res, next) => {
  if (!req.is("application/json"))
    return res.status(415).json({ error: "JSON required" });
  next();
};
```

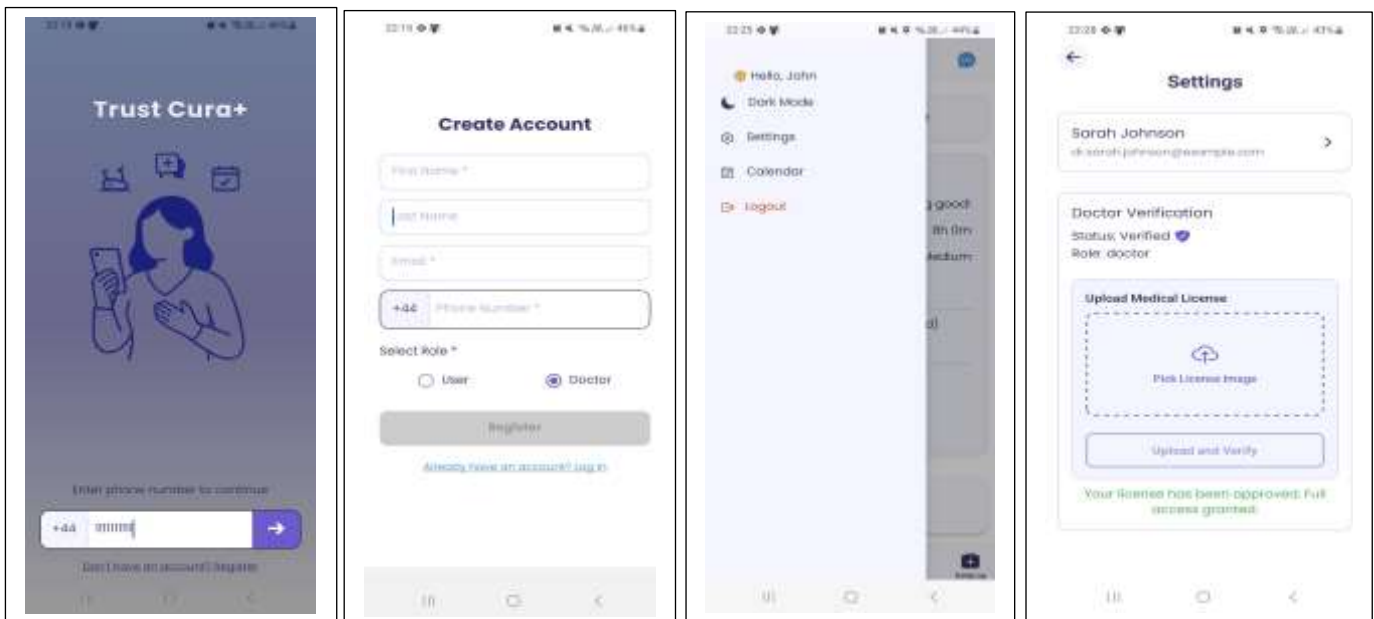
Ensures consistent request format.

13.7 Appendix G – Screen shots

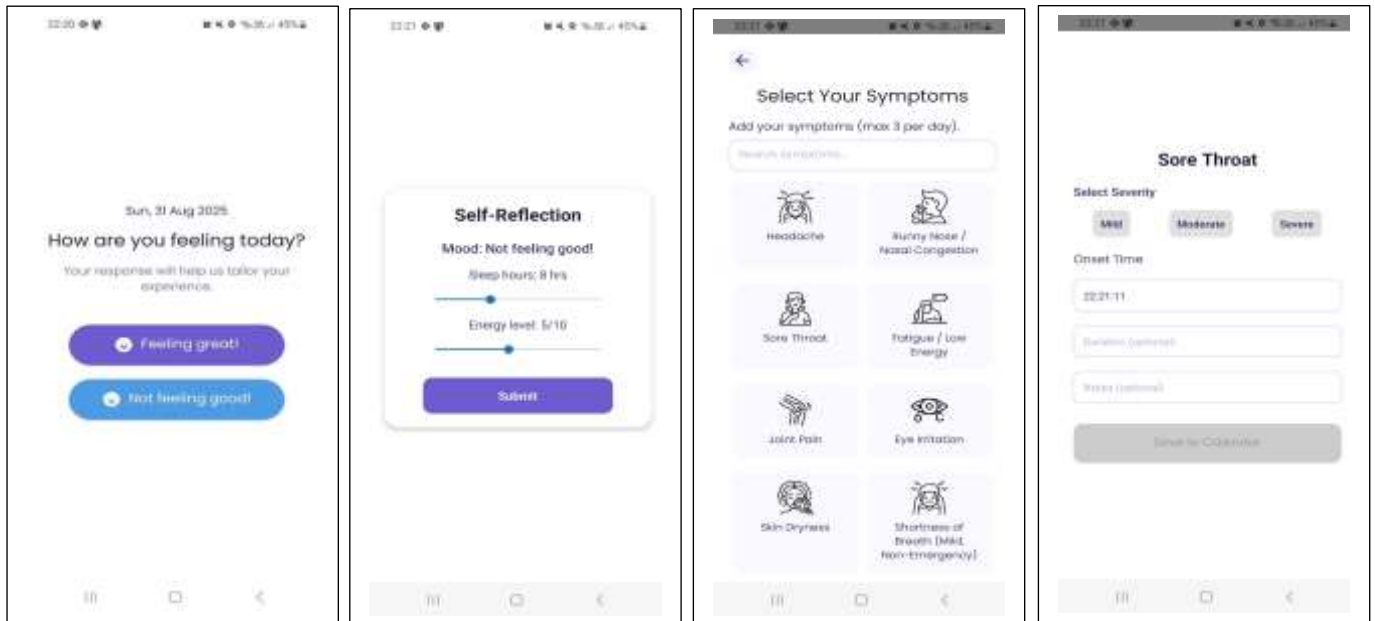
G.1 Onboarding Screens



G.2 Authentication and Doctor Verification Screens



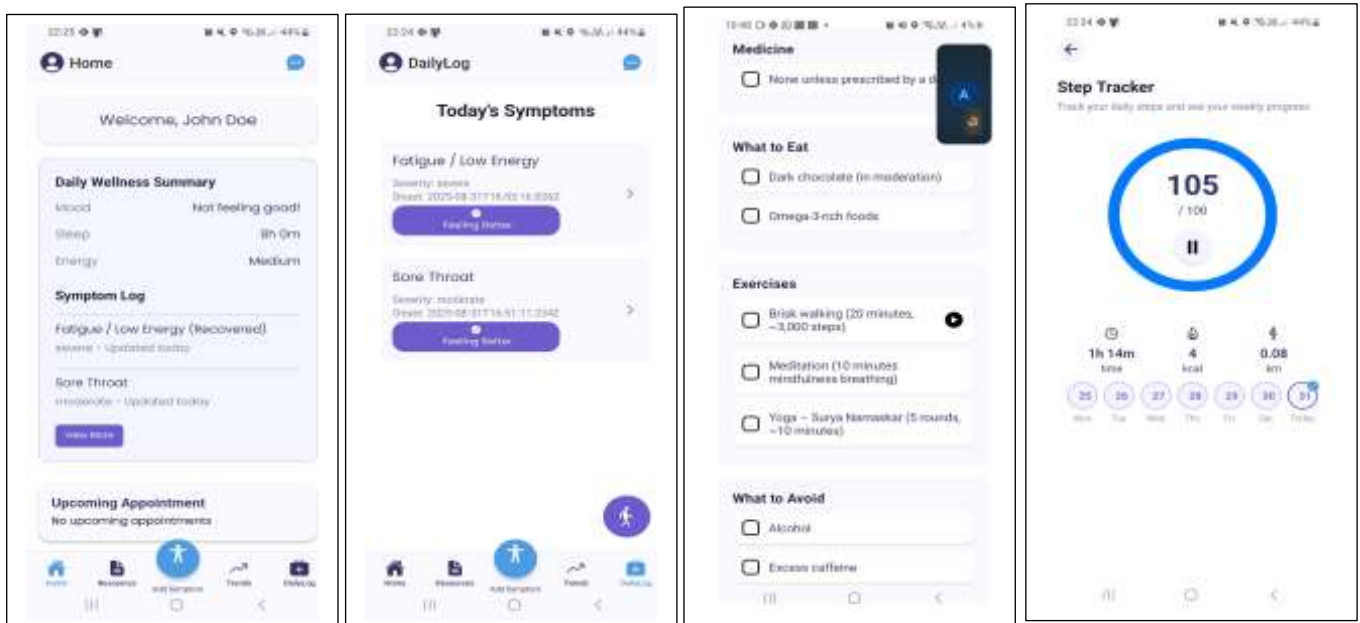
G.3 Add symptoms



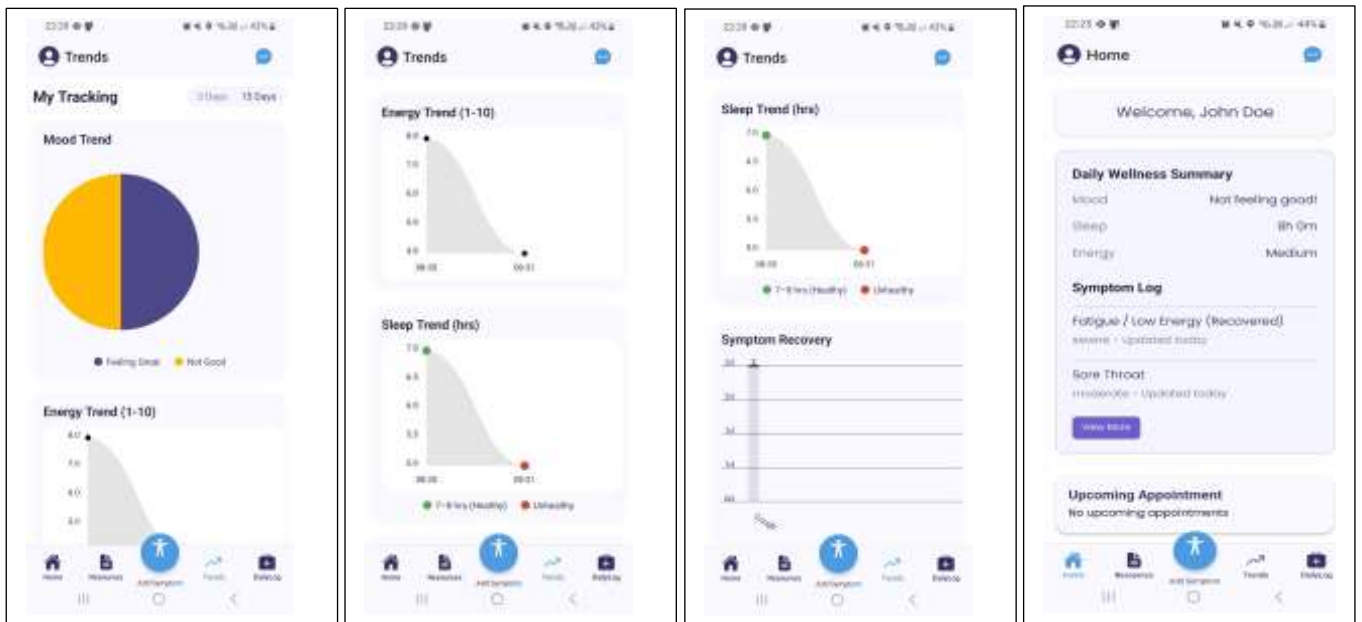
G.4 View symptoms and upcoming appointment

G.5 Mark recovered and view recovery Plan

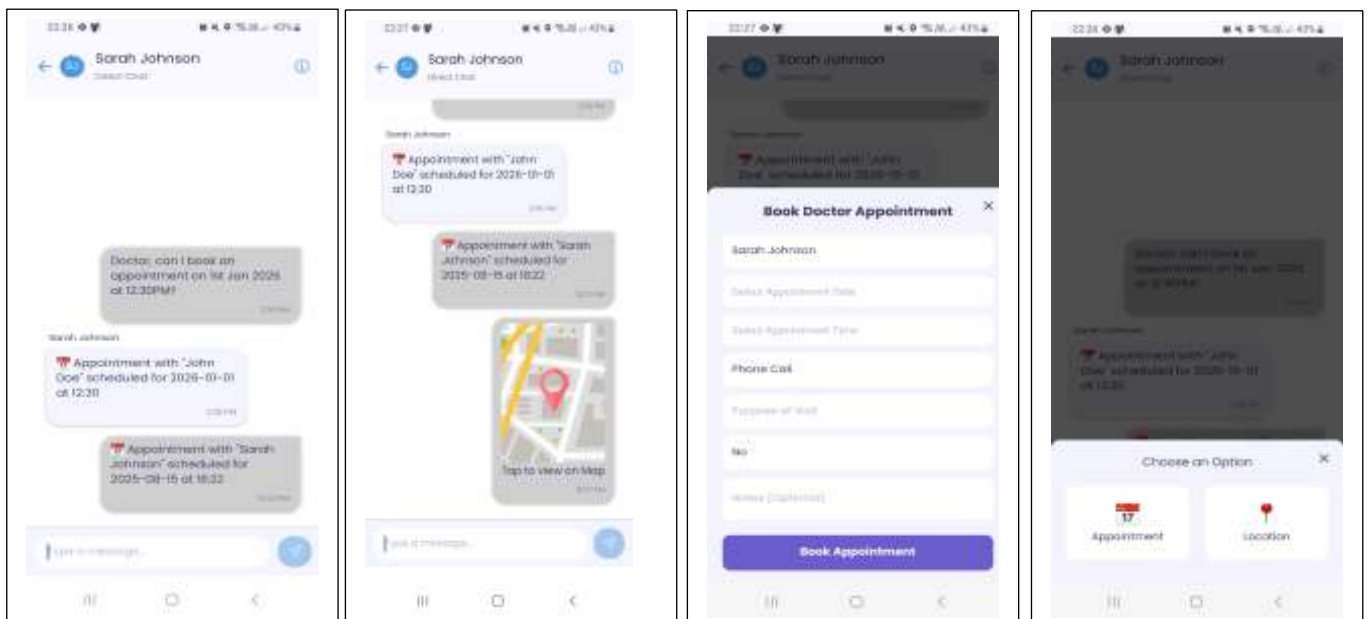
G.6 Step Tracker using Accelerometer



G.7 Trends



G.8 Communication and Appointment



13.8 Appendix H – User Feedback

H.1 Survey Questions

- How easy was it to sign up or log in to the app?
- Was symptom logging (adding severity, details) simple to use?
- Did the daily summaries help you understand your health information?
- Were navigation paths clear when moving between features?
- How useful were the daily health recommendations?
- Did you understand the purpose of the communication feature?
- Were task reminders and progress indicators motivating?
- Did the interface feel overloaded with information?

H.1.1 Key Findings

Positive Observations

- **Symptom Logging Simplicity:** Users found it straightforward to log symptoms using sliders and predefined categories.
- **Daily Summaries:** Summaries showing progress and trends were considered helpful for quick overviews.
- **Navigation Clarity:** Bottom navigation tabs were rated as intuitive and easy to follow.
- **Task Completion Tracking:** Visual task indicators (progress bars, check marks) were appreciated for helping users stay consistent.

Issues Identified

- **Overlap Between Features:** Some users were uncertain about the distinction between “Daily Check-In” and “Symptom Logging.”
- **Information Overload:** Daily summaries sometimes displayed too much information at once.
- **Unclear Communication Feature:** Since most users were not expecting doctor–user style messaging, the communication feature is confusing without clear context.

Suggestions for Improvement

- Merge daily check-in and symptom logging into one seamless flow.
- Use icons and visuals to make symptom categories more accessible.
- Prioritize tasks and reminders on the home screen.

H.1.2 Adjustments Based on Feedback

Several improvements were made:

1. **Streamlined Logging:** Daily check-ins and symptom logging were merged into a single-entry flow.
2. **Simplified Summaries:** Summaries were redesigned to highlight only key data, with options to expand for more details.
3. **Consistent Navigation:** A bottom navigation bar is adopted for easier movement between features.
4. **Clarified Communication:** Messaging is reframed as a general health communication tool instead of being tied only to doctors.

5. Improved Visuals: Icons and labels were added to help non-medical users understand categories quickly.

H.2 Feedback Limitations

While student feedback improved usability, the project did not include testing with a broader range of general users (e.g., older adults, non-student populations).

Reasons Broader User Testing Is Not Done

- Access Constraints: Coordinating with non-student groups required resources and permissions beyond the project scope.
- Time Limitations: The academic timeline restricted extended testing phases.
- Ethical Considerations: Broader community testing would require additional approvals for responsible data handling.

Advantages Missed

- Age Diversity: Older adults may have offered insights into usability challenges such as readability or navigation difficulties.
- Lifestyle Variations: Working professionals or parents could have provided feedback on how the app fits into busy schedules.
- Cultural Contexts: Broader feedback could reveal how different groups interpret health recommendations.

What Is Missed

- Validation of usability for different age groups.
- Insights into how different lifestyles affect app usage.
- Wider perspectives on the usefulness of recommendations

H.3 Prototype Adjustments: Before vs After Feedback

Evidence of Prototype Changes Based on Feedback

| Feature | Before Feedback (Initial Prototype) | After Feedback (Revised Prototype) | Reason for Change |
|-----------------------------------|--|---|---|
| Symptom Logging vs Daily Check-In | Separate screens for "Daily Check-In" and "Symptom Logging," which confused users. | Combined into one unified flow where users log daily health status and symptoms together. | Feedback showed overlap caused confusion (students unsure which to use). |
| Daily Summaries | Summaries displayed all data at once, leading to clutter. | Redesigned to show key highlights first, with expandable sections for details. | Feedback highlighted "information overload." |
| Communication Feature | Presented as "Doctor Messaging" only. Students found this irrelevant or unclear. | Reframed as General Health Communication, positioned more prominently in navigation. | Feedback showed students expected peer/support-style chat, not doctor-only. |
| Visual Design (Icons & Progress) | Text-heavy task lists and recommendations. | Added icons, progress bars, and checkmarks for clarity and motivation. | Feedback indicated preference for visual feedback over plain text. |
| Navigation | Some screens used mixed top/bottom navigation. | Standardized with bottom navigation bar across all screens. | Feedback suggested users wanted consistent navigation to avoid confusion. |

13.9 Appendix I – Detailed Unit testing Evidence

This appendix contains the extended results of unit testing performed by the single developer during the TrustCura+ project. The purpose of this appendix is to provide transparency and technical depth without overloading the main report.

I.1 Cycle -by-Cycle Test Log (Red-Green-Refactor)

The following table presents the results of the unit tests conducted under the Test-Driven Development (TDD) approach for the TrustCura+ system. Each test case was first written and executed in iterative red–green–refactor cycles, where initial failures revealed issues such as missing input validation, incorrect API integration, and incomplete encryption handling. After code refinements, all modules behaved as expected. The table records input conditions, expected outputs, actual results, and final status for each cycle. This structured process helped verify correctness, reduce regressions, and ensure system reliability. A comprehensive Jest-based test suite was implemented, covering critical modules such as user authentication, OTP verification, symptom tracking, daily health task management, recovery plan generation, real-time chat, weather services, and data persistence.

Table I.1 – Summary of Unit Testing under TDD Approach

| Cycle | Test Written (Fail First) | Implementation | Result | Refactor / Final Code |
|-------------------------------|--|--|--------------------------------------|---|
| 1 – Symptom Logging | Jest test: it("should not save symptom if description is empty") expecting validation error. | Initial code allowed saving empty symptoms. | Test failed (empty input accepted). | Added validation in backend to reject empty input. Test passed. |
| 2 – Symptom Severity | Test: it("should accept severity values between 1–5 only"). | Initial code accepted any value. | Test failed for out-of-range values. | Added range check, ensured error message returned. Passed. |
| 3 – Task Completion | Test: it("should mark task as completed") checking DB update. | Function markTaskComplete() added but didn't persist status. | Test failed (status unchanged). | Fixed DB update logic, added confirmation response. Passed. |
| 4 – Task Progress Calculation | Test: it("should return % | Initial logic gave wrong calculation (off by 1). | Test failed (mismatched output). | Corrected formula and rounded |

| | | | | |
|-------------------------------|---|---|---------------------------------------|---|
| | completion correctly"). | | | output. Passed. |
| 5 – Weather Recommendation | Test: it("should return advice for cold weather"). | API parsing error, wrong object key used. | Test failed (undefined response). | Corrected JSON parsing and key mapping. Passed. |
| 6 – Health News API | Test: it("should fetch latest 5 news items"). | Returned full payload without filtering. | Test failed (too many items). | Added array slicing to limit output. Passed. |
| 7 – GPS Facility Locator | Test: it("should return nearest facility given valid coordinates"). | First version returned empty list. | Test failed (no results). | Corrected API call and query parameters. Passed. |
| 8 – Invalid GPS Input | Test: it("should throw error for invalid coordinates"). | Function crashed with NaN values. | Test failed. | Added input validation to reject invalid coordinates. Passed. |
| 9 – Camera Upload | Test: it("should store uploaded image with correct filename"). | Stored with random name (not linked to user). | Test failed (wrong filename mapping). | Implemented userID-based naming. Passed. |
| 10 – Doctor–Patient Messaging | Test: it("should send encrypted message and retrieve same"). | Message saved as plain text. | Test failed (encryption missing). | Added AES encryption + decryption on retrieval. Passed. |
| 11 – Authentication | Test: it("should not login with wrong password"). | Hash check missing, login succeeded with wrong input. | Test failed. | Integrated bcrypt password hashing check. Passed. |
| 12 – Database Storage | Test: it("should persist user data in SQLite"). | Insert worked but retrieval returned null. | Test failed. | Fixed async handling (await missing). Passed. |

1.2 Expandable Module Test Tales

The following table summarises results by module. While hundreds of tests were written, this table shows the categories of issues identified and fixed.

| Module / Test Case | No. of Tests | Passed Initially | Failed Initially | Bug Identified | Fix Applied | Final Status |
|--------------------------------|--------------|------------------|------------------|--|--|--------------|
| Symptom Logging | 5 | 3 | 2 | Blank entries allowed; severity out of range | Added mandatory field validation; enforced 1–5 range | Passed |
| Daily Health Task Management | 4 | 3 | 1 | Missing mandatory fields | Implemented stricter schema validation | Passed |
| Recommendation Generation | 4 | 3 | 1 | Incorrect weather API parameters | Corrected API query structure, added error handling | Passed |
| GPS & Resource Finder | 3 | 2 | 1 | Wrong coordinate arguments in API calls | Fixed parameters, added coordinate validity checks | Passed |
| Communication (Chat) | 5 | 3 | 2 | Messages unencrypted; file formats unchecked | Applied AES encryption; added file type validation | Passed |
| Authentication & Authorization | 3 | 2 | 1 | Invalid/expired tokens accepted | Improved JWT verification in middleware | Passed |
| Database Persistence | 2 | 1 | 1 | Inconsistent timestamp storage | Standardized ISO timestamp format | Passed |

Interpretation The TDD process allowed early detection and resolution of critical defects. After iterative refinement, all 26 tests passed, demonstrating reliability across core features. The results confirm fewer defects, stable modules, and improved confidence in refactoring, with reduced risk of regression.

I.3 Complete Test command references

```
# Run all tests once
npm test

# Generate coverage report
npm test -- --coverage

# Run tests in watch mode (reruns automatically on file changes)
npm test -- --watch

# Run only a specific test file
npx jest tests/symptom.test.js

# Run tests matching a keyword (e.g., "login")
npx jest -t login

# Run with detailed output for debugging
npx jest --verbose
```

I.4 Extended Library Reference

Mobile (React Native – Expo):

- `jest-expo` – Jest preset for Expo apps
- `@testing-library/react-native` – UI rendering and behaviour testing
- `@testing-library/jest-native` – Custom matchers for React Native

Backend (Node.js):

- `supertest` – HTTP request and response validation
- `nock` – Mocking external HTTP requests
- `ts-jest` / `Babel` – For ES module and TypeScript support

Utilities:

- `jest-fetch-mock` – Mocking fetch calls
- `whatwg-fetch` – Polyfill for fetch API
- `mockdate` – Time manipulation in tests
- Expo sensor mocks – For camera and GPS simulation

I.5 Test Coverage

Client-side code:

The test coverage report shows strong overall results, with many core areas such as assets, store, selectors, and theme reaching 100%. This indicates these foundational pieces are well tested and reliable. Components, modals, and utilities also have healthy coverage around 80–90%, though some branches and functions remain untested. The weaker spots lie in navigation, reducers, and especially screens, which sit near 60%. These highlights missed code paths, particularly conditional logic. While functional reliability is strong in shared modules, end-user flows in screens need more robust testing. Focusing on edge cases, error states, and navigation logic will significantly improve confidence.

All files

74.37% Statements 1608/2162 58.48% Branches 793/1356 72.56% Functions 347/616 74.96% Lines 1512/2017

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

| File | Statements | Branches | Functions | Lines |
|-----------------|------------|----------|-----------|---------|
| assets | 100% | 5/5 | 100% | 0/0 |
| assets/fonts | 100% | 2/2 | 100% | 0/0 |
| store | 100% | 2/2 | 100% | 0/0 |
| store/selectors | 100% | 8/8 | 100% | 6/6 |
| theme | 100% | 2/2 | 100% | 84/84 |
| context | 94.44% | 34/36 | 50% | 9/18 |
| components/Chat | 87.83% | 65/74 | 71.79% | 28/39 |
| modals | 85.99% | 221/257 | 68.61% | 129/188 |
| components | 85.79% | 151/176 | 68.91% | 133/193 |
| utils | 84.23% | 203/241 | 66.66% | 92/138 |
| module | 79.78% | 150/188 | 66.33% | 67/101 |
| screens | 73.07% | 285/390 | 49.22% | 95/193 |
| store/reducers | 68.3% | 209/306 | 53.33% | 64/120 |
| navigation | 65.55% | 59/90 | 40.54% | 15/37 |
| store/actions | 55.06% | 212/385 | 30.29% | 73/241 |

Server-Side code:

The coverage results show excellent performance in data and migrations, both achieving 100%, confirming complete test reliability in schema and configuration logic. middleware and sockets are also very strong, above 90%, indicating most logic paths and interactions are well tested. However, routes stand out as the weakest area, with only 54% statements and particularly low branch coverage at 42%. This means many conditional flows and alternative request scenarios remain untested. Since routes often handle critical validation, authentication, and error handling, focusing additional tests here would greatly strengthen confidence. Expanding edge case, error, and alternate-path tests should be prioritized.

| File | Statements | Branches | Functions | Lines |
|------------|------------|----------|-----------|---------|
| data | 100% | 2/2 | 100% | 0/0 |
| migrations | 100% | 30/30 | 100% | 0/0 |
| middleware | 95.45% | 63/66 | 87.17% | 34/39 |
| sockets | 91.17% | 31/34 | 85.71% | 12/14 |
| routes | 54.23% | 384/708 | 42.63% | 191/448 |

13.10 Appendix J – Integration Test results

J.1 Integration Test Results

| ID | Scenario | Expected Outcome | Initial Result (Bug) | Fix Applied | Final Status |
|-------|---|---|--------------------------------------|-------------------------------------|--------------|
| IT-01 | Auth → Symptom Logging | Token required; valid symptom stored | Empty symptom accepted | Mandatory input + schema validation | Pass |
| IT-02 | Symptom Logging → Recommendations | Plan includes rest + hydration tip (if hot) | Hydration tip missing (API mismatch) | Fixed API field mapping | Pass |
| IT-03 | GPS → Resource Finder → Recommendations | Coordinates valid, facilities listed, plan adjusted | API 400 error (long vs lng) | Corrected parameter & checks | Pass |
| IT-04 | Auth → Chat (Doctor) | Messages stored encrypted | Messages in plain text | AES-GCM encryption at rest | Pass |
| IT-05 | Task Completion → Progress Dashboard | 3/4 = 75% completion | Shown as 50% | Fixed float division | Pass |
| IT-06 | Token Expiry → Re-auth | Expired token rejected (401) | Correct as expected | — | Pass |
| IT-07 | Offline Log → Sync on Reconnect | Single record synced | Duplicate records created | Idempotency key + server upsert | Pass |
| IT-08 | Media Upload → Chat → Doctor View | Photo visible on doctor portal | iOS HEIC not supported | MIME sniff + JPEG conversion | Pass |

J.2 Key Bugs Identified and Fixed

1. BUG-001: Empty symptom entries allowed → fixed with validation.
2. BUG-002: Weather API mismatch (temp_c vs temperature) → corrected mapping.
3. BUG-003: GPS API param typo (long instead of lng) → fixed, backward compatibility added.
4. BUG-004: Chat unencrypted at rest → AES-GCM implemented.
5. BUG-005: Incorrect completion percentage → fixed float division.
6. BUG-006: Offline logs duplicated → idempotency keys introduced.
7. BUG-007: HEIC images unsupported → converted to JPEG server-side.

13.11 Appendix K – Evaluation

K.1 Comparison with Existing Apps:

This comparison highlights how **TrustCura+** differs from established tools:

- Compared to MyChart, which is heavily tied to provider networks, **TrustCura+** is lightweight and accessible to the general public, with no dependency on hospital systems.
- Compared to Ada Health, which primarily focuses on triage through AI symptom checking, **TrustCura+** provides ongoing structured logging, contextual recommendations, and doctor communication.
- Compared to Google Fit, which is focused on fitness tracking and wearable data, **TrustCura+** targets everyday illness recovery with features such as symptom severity tracking and access to nearby healthcare resources.

| Feature / App | TrustCura+ | MyChart | Ada Health | Google Fit |
|-----------------------------|--|--|---------------------------------|--|
| Primary Focus | Everyday illness recovery (mild fever, flu, fatigue) | Patient records and hospital integration | Symptom assessment and triage | Fitness and activity monitoring |
| Symptom Logging | Structured logs with severity & timestamps | Provider-dependent | Symptom checker only | Not supported |
| Recommendations | Context-aware (weather, news, activity) | Provider-dependent | AI-based diagnostic suggestions | Fitness targets only |
| Doctor Communication | Encrypted in-app chat | Secure provider messaging | Not available | Not available |
| Sensor Integration | GPS, accelerometer | Minimal | None | Wearable integration (steps, heart rate) |
| Healthcare Resources | Nearby hospitals, pharmacies via GPS | Limited | Not available | Not available |
| Scope | General public; lightweight tool | Tied to provider networks | Global consumer health app | General fitness enthusiasts |

Taken together, these differences demonstrate that **TrustCura+** occupies a distinct space by combining structured symptom logging, adaptive recommendations, and patient–doctor communication in one tool. This creates a differentiated solution that addresses gaps left by current apps. Section 10.3 expands on this by highlighting the innovative aspects of this integration.

K.2 Innovation & Differentiation

As shown in the comparison (Appendix K.1), **TrustCura+** differentiates itself by combining functionality that is usually spread across multiple existing solutions. While MyChart emphasises hospital integration, Ada Health prioritises diagnostic triage, and Google Fit focuses on fitness tracking, none provide an integrated approach to everyday illness recovery.

TrustCura+ introduces several innovative elements:

- Structured symptom logging with severity and timestamps, offering more consistent records than free-form notes or checker-style inputs.
- Context-aware recommendations that adapt to local conditions (e.g., weather or activity levels), extending beyond the static tips or fitness-only targets of other tools.
- Encrypted in-app doctor communication, bridging a gap left by consumer apps like Ada Health and Google Fit that lack direct provider channels.
- Integrated healthcare resources, such as GPS-based access to nearby hospitals and pharmacies, a feature generally absent from comparator apps.

The innovation lies not in creating entirely new individual features, but in integrating symptom tracking, adaptive self-care, and communication into a single lightweight tool designed for the general public. Informal user testing further highlighted that participants valued this clarity and adaptability, reinforcing that **TrustCura+** offers both novel functionality and an improved user experience compared to established alternatives.

K.3 Proposed developments include:

While **TrustCura+** has achieved its primary goal of integrating symptom tracking, recommendations, and doctor–patient communication in a single mobile application, several areas remain open for further development and improvement.

- **Broader User Validation:** The current feedback is collected mainly from student groups. Future work should involve patients, healthcare providers, and clinicians to validate the system in real-world healthcare settings. Their insights would ensure medical accuracy, improve usability in clinical contexts, and strengthen trust in the platform.
- **Wearable Device Integration:** **TrustCura+** currently relies on embedded smartphone sensors (accelerometer, GPS, and camera). Expanding integration with wearables (e.g., smartwatches, fitness bands) would allow continuous data collection on vital signs such as heart rate, sleep cycles, and blood oxygen levels. This would enrich health monitoring and extend the app's usefulness for long-term tracking.
- **Passive Sensing and Automation:** Research shows that passive data collection can reduce user burden while maintaining accuracy. Future iterations of the app could implement background monitoring for step counts, activity levels, and even environmental conditions (air quality, noise) without requiring frequent manual input.
- **Chronic Disease Support:** Although the current focus is on simple health issues such as fever, fatigue, and seasonal flu, the platform could gradually be extended to support chronic disease management (e.g., diabetes, hypertension) in collaboration with healthcare professionals. This would involve developing modules for medication adherence, specialist communication, and long-term health analytics.
- **Advanced Analytics and AI:** Finally, machine learning could be incorporated to generate predictive health insights, flagging patterns that indicate worsening conditions or risks. This would support early intervention and more personalized care.

In summary, these directions would allow **TrustCura+** to evolve from a basic, supportive health app into a scalable digital health platform capable of serving a wider population with more advanced needs.

K.4 Critical Evaluation

- **Methodology.** Test-Driven Development (TDD) improved module-level reliability and facilitated safe refactoring. Manual integration testing, however, limited systematic coverage and repeatability. Automated test suites are recommended (Ammar et al., 2021).
- **Design Trade-offs.** Focusing on everyday recovery (not chronic or emergency care) kept scope feasible but reduced applicability. Rule-based recommendations are transparent but less adaptive than machine-learning alternatives, suggesting a staged path to AI with safeguards.
- **Validity Threats.** Usability findings stem from a small, convenience sample (peers and family), limiting demographic diversity. Accuracy checks referenced WHO guidelines, but lacked formal clinical validation. Performance metrics were not benchmarked.
- **Value Realised.** Despite these limits, TrustCura+ demonstrated feasibility in integrating symptom logging, adaptive advice, secure communication, and local healthcare resources. The evaluation identifies a roadmap for improvement while validating the core concept.