Míra: A User-Centered Approach to Designing an Inclusive Cycle Tracking Application

By

Chloe Dwyer

N00220849

BSc [Hons] Creative Computing

Institute of Art, Design & Technology

2026

John Montayne & John Dempsey

Declaration of Ownership

The incorporation of material without formal and proper acknowledgment (even with no deliberate intent to cheat) can constitute plagiarism.

If you have received significant help with a solution from one or more colleagues, you should document this in your submitted work and if you have any doubt as to what level of discussion/collaboration is acceptable, you should consult your lecturer or the Course Director.

WARNING: Take care when discarding program listings lest they be copied by someone else, which may well bring you under suspicion. Do not to leave copies of your own files on a hard disk where they can be accessed by other. Be aware that removable media, used to transfer work, may also be removed and/or copied by others if left unattended.

Plagiarism is considered to be an act of fraudulence and an offence against Institute discipline. Alleged plagiarism will be investigated and dealt with appropriately by the Institute. Please refer to the Institute Handbook for further details of penalties.

The following is an extract from the B.Sc. in Creative Computing (Hons) course handbook. Please read carefully and sign the declaration below

Collusion may be defined as more than one person working on an individual assessment. This would include jointly developed solutions as well as one individual giving a solution to another who then makes some changes and hands it up as their own work.

DECLARATION:

I am aware of the Institute's policy on plagiarism and certify that this thesis is my own work.

Student: Chloe Dwyer

X 

Failure to complete and submit this form may lead to an investigation into your work.

Abstract

This project documents the design and development of Míra, a mobile cycle-tracking application developed using a user-centered approach. Existing applications were found to be feature-heavy, paywalled, and poorly tailored to the real and varied needs of all women. To address this, a survey and interviews were conducted to identify real users' needs, with the findings directly shaping design and development decisions throughout the project.

The front-end was built with React Native, TypeScript, and NativeWind, and the back-end with Laravel and MySQL. Core features include period logging, symptom tracking, a calendar view, cycle insights, a journal, and a PDF export. The application was evaluated through user testing and assessed against WCAG 2.1 AA accessibility standards, with results confirming that Míra is easy to use, visually clear, and accessible.

Acknowledgments

To my supervisor, John, thank you. Thank you for your patience, your guidance, and for always pointing me in the right direction when I felt lost. I am very grateful.

To Aoife and Julia, I honestly don't know what I would have done without you both. Thank you for always being there for me, for letting me rant, for listening without judgment, and for making the hard days much easier. You have been two of the most constant people in my life throughout all of this, and I am forever grateful for you both.

To my Mam, Dad, and Sister, you are my everything, and I could not have done any of this without you. Thank you for always being there, for listening, and for supporting me every step of the way. It meant more than you will ever know.

To my boyfriend, Darren, thank you for your endless patience and support throughout all of this. For putting up with everything I threw at you, and the moments where I just needed a hug and a shoulder to cry on. It meant everything, and I am very grateful to have you by my side.

And finally, to myself, this project was completed during a difficult period of my life, navigating health challenges and a lot of uncertainty. There were days when showing up felt very hard, but I kept going through the blood, sweat, and tears. I am genuinely very proud of myself, and this will be proof for my future self that if I can get through this, I will be able to get through anything life throws at me.

"Bíonn gach tosú lag"

Table of Contents

1.0 Introduction & Project Context

1.1 Project Overview

This project focuses on the design and development of a cycle tracking application that helps users better understand and manage their menstrual cycle health. Many existing applications primarily focus on predicting periods and do not address tracking mood, pain levels, condition-specific tracking, or how cycles affect the day-to-day lives of many women. This application aims to create a well-rounded, supportive cycle-tracking app for all women.

This application will allow users to track their menstrual cycle along with daily symptoms such as pain, energy levels, mood changes, general symptoms and overall well-being. Alongside the generic tracking features, this app will include optional tools to support those who experience menstrual health-related conditions, such as and not limited to, Polycystic Ovary Syndrome (PCOS) and Endometriosis. These features will be optional so that the application remains inclusive and useful to a wide range of users.

User research is the main part of this project, and users will be involved throughout the design and development process and guided by the Double Diamond methodology. During the first two stages, user research methods such as surveys, interviews, and background research will be used to gain insight into real users' experience, needs, and frustrations. These findings will clearly help define the problem and influence the direction of the application. In the final two stages, user feedback will continue to play an important role in the design decision-making process, and the ideas will be developed, tested and redefined through iterative design and usability testing. By taking this approach, it ensures that the design decisions are informed by the users at every stage, allowing the application to be shaped by real user needs rather than assumptions.

This project will place a strong focus on creating an interface that is accessible, intuitive, and easy to use and navigate. Clear data visualisation will help users to recognise patterns in a cycle over time. Overall, this project aims to show how user-centered design and development can create an empowering tracking tool.

1.2 Context

The project falls within the areas of Mobile Application Development and Interaction Design / User Experience. This application will be designed specifically for mobile devices, with a strong focus on everyday mobile use. React Native will be used to build the front-end, enabling a smooth, responsive mobile user experience. Laravel will be used to build the back-end for data handling, authentication, and API integration.

User Experience (UX) and User Interface (UI) design are the main components of this project, with the application developed using a user-centered design approach. Design decisions will focus on user needs to ensure the final application reflects what users want for their cycle-tracking app. Overall, this project combines mobile app development, interaction design, and data visualisation to create a practical, user-focused cycle-tracking application.

1.3 Project Goals & Objectives

The overall aim of this project is to design and develop a user-centered mobile cycle-tracking application that helps women better understand and manage their menstrual health, with optional support for women with menstrual health conditions.

- Apply the double diamond methodology to define a core problem and guide the design and development process.
- Conduct user research through surveys and interviews during the early stages of this project to identify user needs and pain points related to cycle tracking.
- Develop a secure back-end using Laravel to handle user authentication, data storage, and API integration.
- Develop the front-end using React Native, implementing core cycle tracking features such as logging, symptom tracking, and optional condition-specific features.

- Evaluate the final application through user testing and WCAG standards to assess usability, functionality, and overall user experience.

1.4 Success Criteria

1	All high-priority functional requirements are implemented and functional	Review against the requirements table
2	The application gets an average rating of 4 out of 5 in usability testing	Post-testing survey results
3	Users can successfully complete all tasks	Observed task completion in user testing
4	The back-end API handles authentication securely	Back-end testing with Insomnia
5	Users report that the application is visually clear and easy to navigate	Results from user testing

Table 1 - Success Criteria

2.0 Research & Background

This chapter covers the research and background that underpins the design and development of this project. It starts by looking at user-centered design and the methodology used to structure the design process, then moves on to the research methods used to gather user insights. Then it investigates the technologies to be used to create this project.

2.1 Literature Review

2.1.1 User-Centered Design

User-centered design (UCD) is a design approach where the users actively shape how a product is designed and developed. The concept was introduced in the 1980s through Donald Norman's research and became widely known after he and Draper published User Centered System Design: New Perspectives on Human-Computer Interaction in 1986. Norman outlined the four core principles that UCD is built on: Empathy, Iterative Process, User Empowerment, and Accessibility/Inclusion (Abras et al., 2004). These principles form the foundation of what became the Norman Design Principles, and products that follow them tend to be more usable, more satisfying, and more engaging for users.

UCD is an iterative process that keeps the user's needs at the centre of every design decision. The idea is to design with users, not just for them. This means involving real people throughout the project, observing how they interact with the product, and testing with them regularly. Research shows that products built this way tend to be easier to use, need less training, lead to fewer errors, and result in higher user satisfaction (*What Is User Centered Design (UCD)?,* 2016). This is especially important for health-related applications, where users often have very specific needs.

In practice, UCD means grounding every decision in what users actually need, rather than making assumptions based on technical or business goals. Users need to be involved early and kept involved throughout, with the process following an ongoing cycle of design, testing, refinement, and repetition. The UCD process is broken down into five stages (Justinmind, 2020):

- Specify the context of use: Identify who will use the product and in what circumstances.
- Gather requirements: Determine user goals and the conditions necessary to meet them.
- Design solutions: Develop prototypes, progressing from low-fidelity sketches to high-fidelity prototypes.
- Evaluate designs: Conduct usability testing and gather feedback to refine the application.
- Iterate and improve: Repeat the cycle based on user feedback until the application adequately meets user needs.

2.1.2 UX and UI Principles

Closely tied to UCD is an understanding of UX and UI principles, which guide how an application should work and feel. UX principles are focused on the overall experience of using an application and cover things like:

- User-Centricity: making design decisions based on real user problems, backed by research and testing.
- Consistency: keeping design patterns uniform so users do not have to relearn things.
- Hierarchy: organising content so what matters most will stand out.
- Context: considering the environment, device, and emotional state of the user.
- User Control: giving users the ability to undo, cancel, and recover from mistakes.
- Accessibility: making sure the application works for people with a range of different abilities and in different conditions.
- Usability, which breaks down into five components: Learnability, Efficiency, Memorability, Error Recovery, and Satisfaction (Stevens, 2024).

UI principles take those broader goals and translates them into actual interface decisions. The key principles include:

- Clarity: making it obvious what is interactive, how to navigate, and what to do next.
- Familiarity: using established patterns so users do not need to learn them from scratch.
- User control: giving users ways to navigate, undo, and exit so they always feel in control,
- Hierarchy: using size, colour, and spacing to draw attention to what matters.
- Flexibility: supporting both new and experienced users across different devices.
- Accessibility: ensuring enough contrast, readable text, and inclusive design choices (Chappal, 2021).

Together, UX and UI principles will be used throughout this project to evaluate and guide every design decision. They help ensure the application is not just functional, but genuinely easy and inclusive to use.

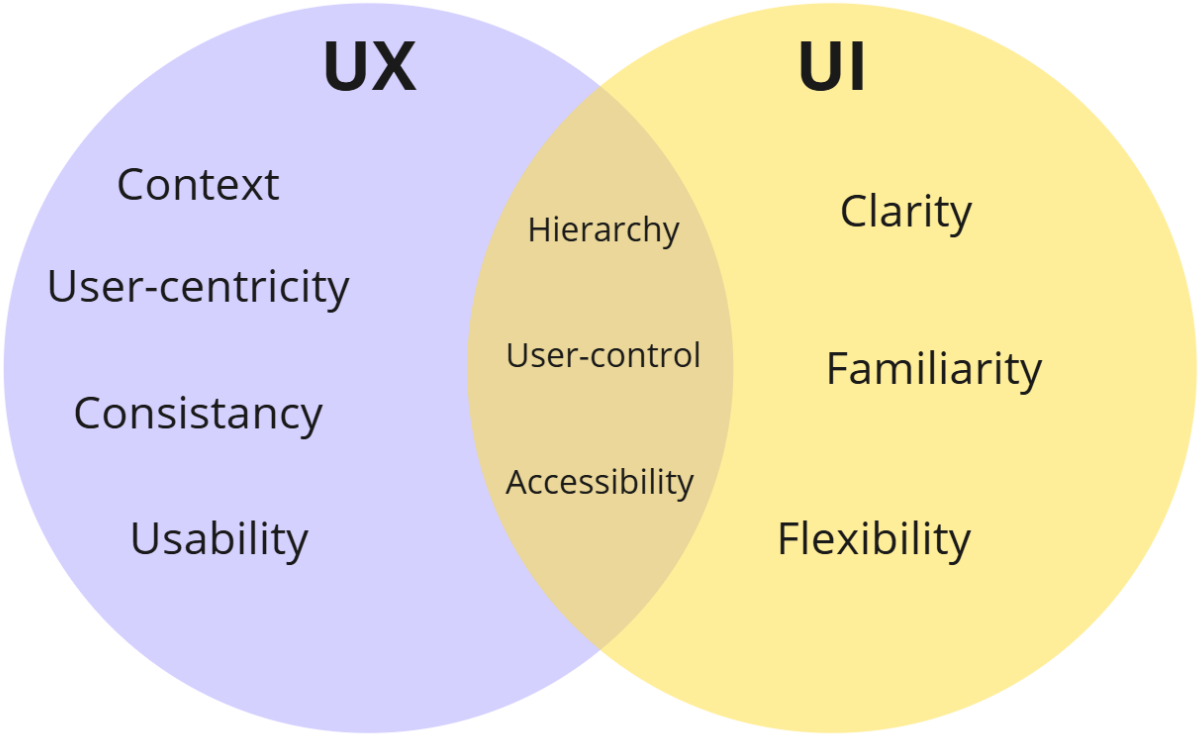


Figure 1 - Venn Diagram showing similarities and differences between UX and UI principles

2.1.3 Double Diamond Methodology

The Double Diamond methodology is a design framework that maps the design and innovation process in an easy-to-understand, easy-to-apply way. It was developed by the Design Council in 2005 and divides thinking into two diamonds: the first focuses on understanding the problem, and the second on developing the solution. The four phases in this methodology are: Discover, Define, Develop, and Deliver (Design Council, 2025).

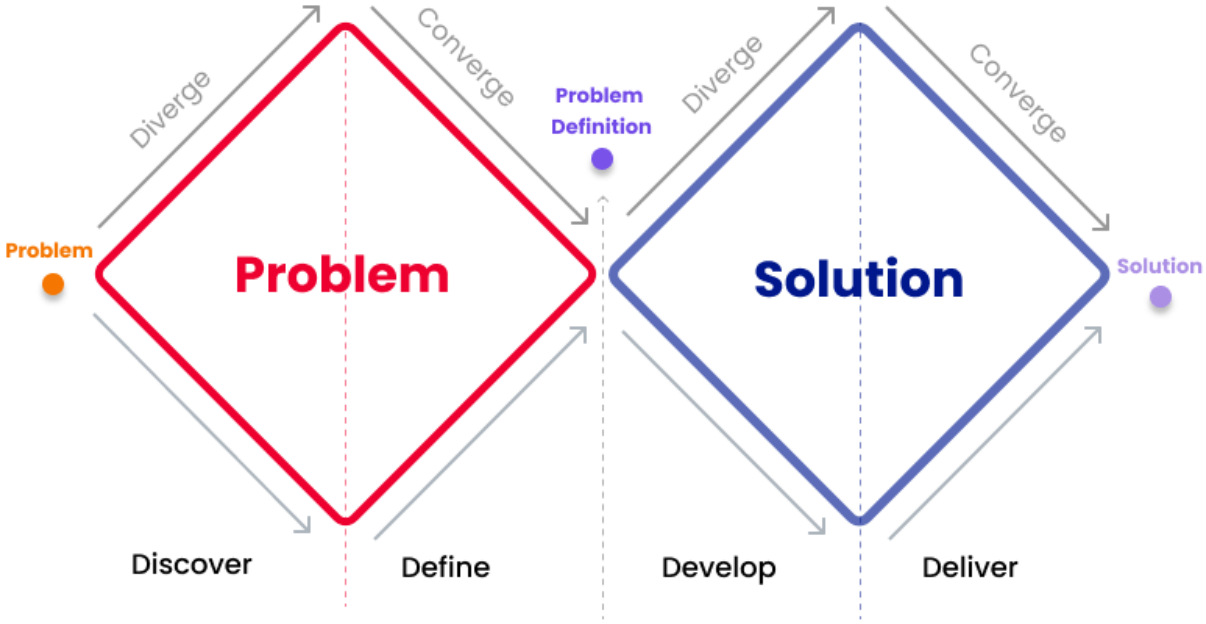


Figure 2 - Double Diamond Diagram (Sobrinho, 2023)

During the Discover phase, the focus is on understanding users and their needs without making assumptions. The Define phase is about taking what was learned during the discovery phase and narrowing it into a clear, well-defined challenge. In the Develop phase, some potential solutions are explored, prototyped, and tested. Finally, the Deliver phase involves testing the application with real users, incorporating their feedback, and refining it. As Schneider (2015) highlights, understanding how these phases connect is key to choosing the right methods at each stage.

This application will use the Double Diamond methodology. It fits naturally with the user-centered approach: the first two phases (Discover and Define) open up exploration of user needs and possible solutions, while the last two phases (Develop and Deliver) bring that thinking into focus.

2.1.4 User Surveys & Interviews

User interviews are a qualitative research method where a researcher asks participants questions, either structured or open-ended questions, listens to their response, and follows up to dig deeper to get a better understanding (Pernice & Rosala, 2023). The aim is to get a real understanding of what users want, need, and what frustrates them. User interviews are an important part of the Discover phase, giving the insight needed to define the problem accurately.

Conducting a good user interview involves six steps:

1. Setting a clear research goal.
2. Preparing an interview guide based on prior research.
3. Testing the guide with a friend to catch anything confusing.
4. Starting with simple questions to ease the participant in.
5. Building enough rapport that they feel comfortable about being honest.
6. Following up with probing questions beyond the prepared script (Maze, 2024).

Alongside interviews, user surveys are also used as a research method. Surveys are a quick, affordable, and scalable way to gather feedback and insights from large numbers of participants. Unlike interviews, surveys are unmoderated; they are sent out and completed by participants in their own time, without the researcher watching. They can include open questions, which allow participants to respond freely, or closed questions with predefined answers. This combination of qualitative and quantitative data makes surveys a powerful tool for validating assumptions, uncovering common pain points, and guiding the rest of the research process. The main limitation of surveys is that participants may not always give honest answers, which can affect data bias (Stevens, 2023).

2.1.5 User Testing

User testing is an important part of the Deliver phase and is central to UCD. It involves watching real users interact with the application to spot usability issues, check the design decisions, and gather feedback for improvements.

An important part of user testing in this project is ensuring the application is accessible and inclusive. Accessibility is about whether people with different abilities can perceive, understand, navigate, and interact with the application. Usability is about how effectively and effectively

users can achieve what they are trying to do within the interface. Inclusivity goes further and takes age, device, and cognitive load into account (W3C, 2016).

To put accessibility into practice, this project will follow the Web Content Accessibility Guidelines (WCAG), a set of internationally recognised rules for making applications accessible to people with different abilities (W3C, 2024). WCAG has four principles:

- Perceivable; content needs to be presented in ways users can actually perceive it.
- Operable; interface components need to be usable.
- Understandable; language and interactions need to be clear and consistent.
- Robust; content needs to work reliably across different technologies.

When development is complete, the application will be tested against these principles using axe DevTools and Lighthouse to ensure it meets these guidelines.

2.2 Technical Research

This section covers the key technical decisions made for this project. React Native was chosen to handle the client-side, whereas Laravel and MySQL were chosen for the server-side. The diagram below gives a full overview of the technology stack used in this project.

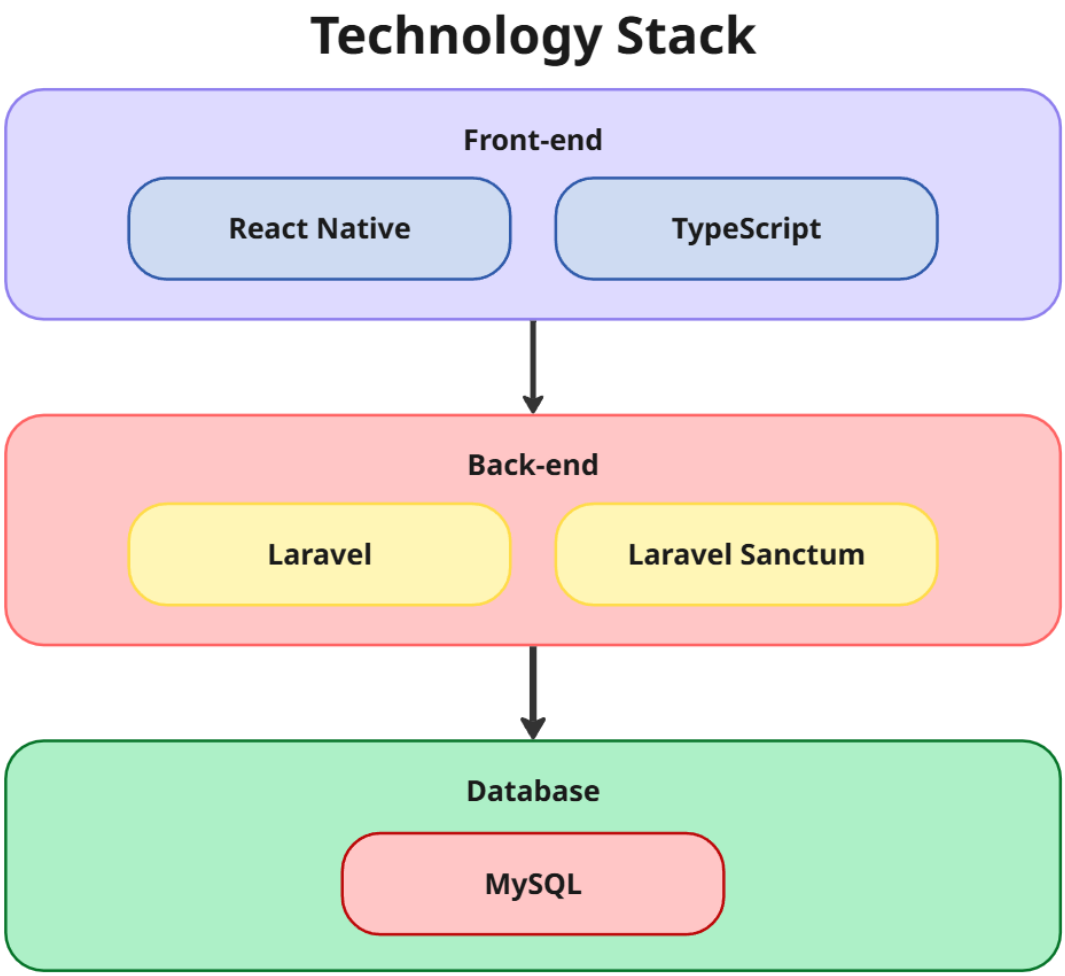


Figure 3 - Technology Stack

2.2.1 Front-end Technologies

React Native takes the React approach and applies it to native mobile development for iOS and Android. Instead of rendering HTML to the browser, React Native maps JSX or TSX components to native elements to produce a fully native interface (React Native, 2024). It runs in a background process that communicates directly with the native platform rather than going through the browser. TypeScript is most commonly used alongside React Native projects because of its enhanced safety and better development experience (React Native, 2020). React Native with TypeScript was chosen for this project as the end goal was to develop a mobile application. Having an existing foundation in React made the transition to React Native more

manageable, allowing more time and focus to be dedicated to the user research and design quality at the core of this project.

2.2.2 Back-end Technologies

A backend is needed to handle user authentication, data storage, and any server-side logic. For the backend of this application, Laravel was chosen as the main technology, with Laravel Sanctum handling user authentication and MySQL as the database. Laravel was selected for its easy integration with React Native, its straightforward setup, and its strong built-in security and data management tools. Laravel Sanctum is also a good fit, as it provides a simple but secure way to manage user authentication for mobile applications. MySQL was chosen for its reliability, widespread support, and compatibility with the Laravel ecosystem.

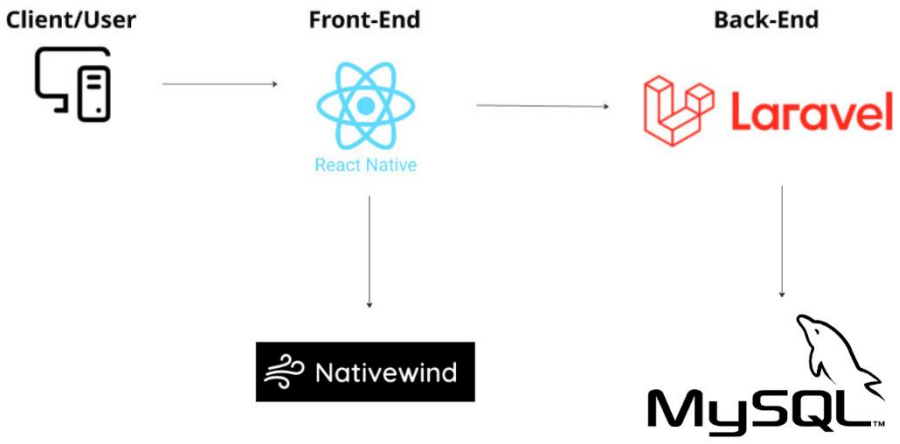


Figure 4 - Technologies used

3.0 Requirement Analysis

3.1 Existing Applications

3.1.1 Flo

Flo is one of the most widely used period and cycle-tracking applications, offering a range of features focused on menstrual health, fertility, and pregnancy. Users can log their period alongside a wide variety of symptoms, including mood, pain, and physical activity. Flo uses AI-powered algorithms to provide cycle and ovulation predictions. Flo also includes a pregnancy tracking mode with week-by-week updates and milestone tracking. The app operates on a freemium model, where the core tracking features are free, but the more advanced insights, detailed reports, and additional content is locked behind a pay wall (Flo, 2023).

Flo is a feature-rich application that offers highly powerful, personalised predictions and medical-style insights. However, there are some limitations: a heavy paywall restricts access to many of the app's key features, meaning users need to pay to get the most out of it. The interface can also feel overwhelming to some due to the volume of information displayed on the pages, and the overall experience is very data-driven.

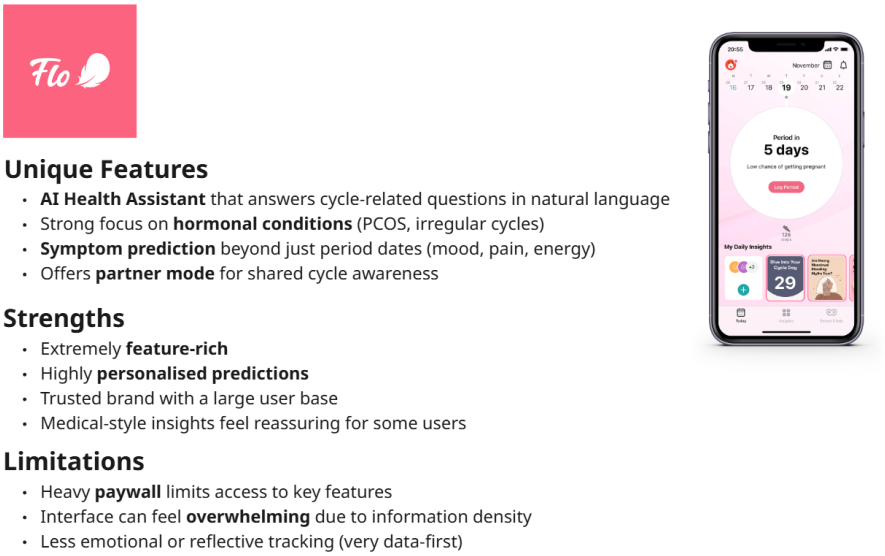


Figure 5 - Flo: Unique Features, Strengths, and Limitations

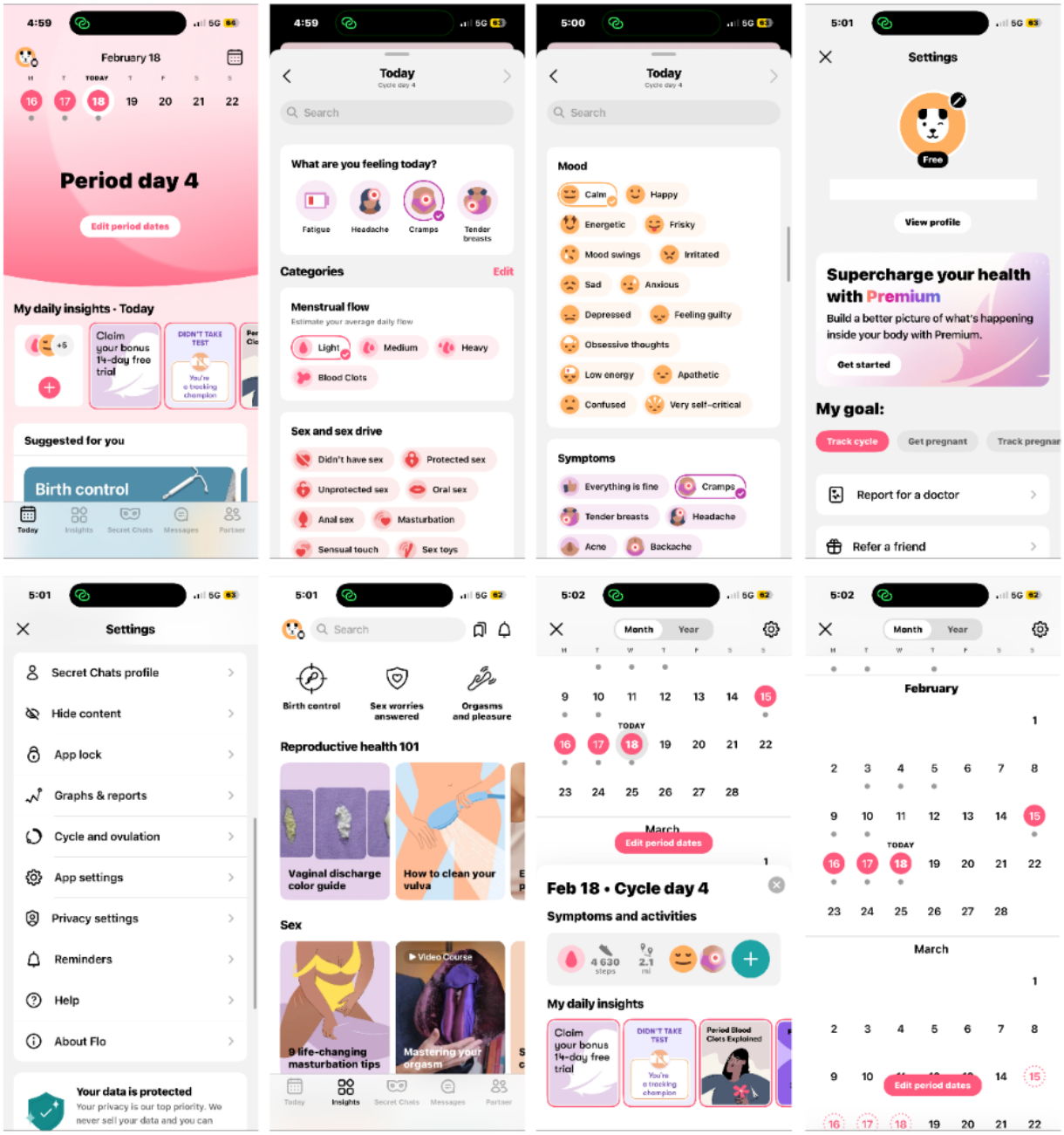


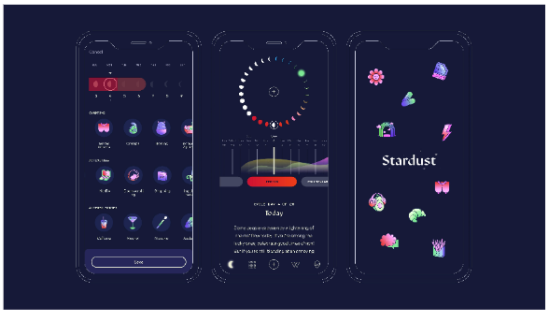
Figure 6 - Flo - App Screenshots (Flo, 2023)

3.1.2 Stardust

Stardust is a period, pregnancy, and hormone tracking app that sets itself apart from other tracking applications through its unique blend of science, AI, and astrology. It uses AI to predict upcoming periods and ovulation. Stardust integrates a lunar cycle; this is a core feature that syncs

user's cycle phases with the moon phases and offers daily insights that mix hormonal science and spirituality. The app is free to download, although some features require a premium subscription (Stardust, n.d.).

Stardust stands out as an emotionally engaging application that makes cycle tracking feel fun. It focuses on reflection over optimisation, which gives the applications some personality and sets them apart from other cycle tracking applications. However, this approach comes with significant limitations. The applications reliance on astrology and lunar phases may be fun, but this raises some questions about scientific credibility. The insights it provides tend to be more symbolic than actionable, which would leave users who need practical health information without the support they need.



Unique Features

- Uses **astrology metaphors** to explain cycle phases
- Strong emphasis on **emotions, relationships, and self-reflection**
- Partner/friend syncing framed as "cosmic connections"
- Highly expressive copywriting and visual storytelling

Strengths

- Very **emotionally engaging**
- Makes cycle tracking feel **fun and validating**
- Strong community appeal, especially for Gen Z users
- Encourages reflection instead of optimisation

Limitations

- Lack of **scientific credibility**
- Not suitable for users with medical conditions (PCOS, endometriosis)
- Insights are symbolic rather than actionable

Figure 7 - Stardust: Unique Features, Strengths, and Limitations

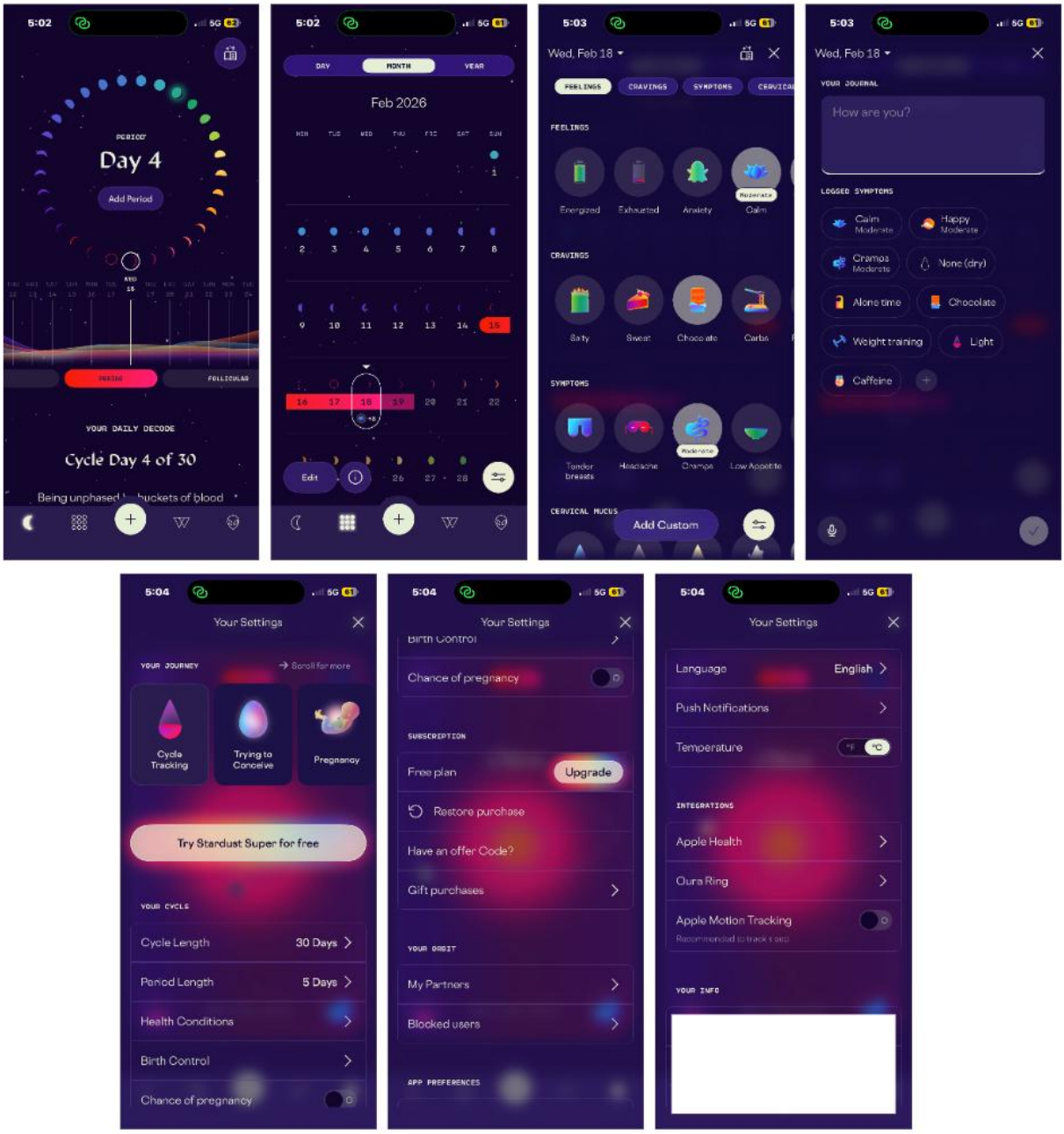


Figure 8 - Stardust - App Screenshots (Stardust, n.d.).

3.1.3 Clue

Clue is a science-based period and ovulation tracker application. It takes a more clinical and research-driven approach compared to the previous two. It uses a clinically tested algorithm to provide reliable predictions for upcoming periods, PMS and ovulation. Users can log over 200 different symptoms and experiences, making it one of the most detailed tracking applications

available. The app displays predictions and logged data on a calendar view and allows users to set custom reminders for important cycle events. The core features are a part of the free version, whereas the Clue Plus subscription offers extended predictions, additional tracking features, and access to premium health content (Clue, 2019).

Clue is well regarded for its clean, minimal interface and its science-first approach. It is particularly well suited to those who want full control over their data. However, the clinical approach can make the app feel emotionally distant, and may not be engaging enough for users who are looking for daily check-ins. The depth of tracking options available is a strong strength for experienced users, but it can cause a big learning curve for those who are new to tracking their cycles.

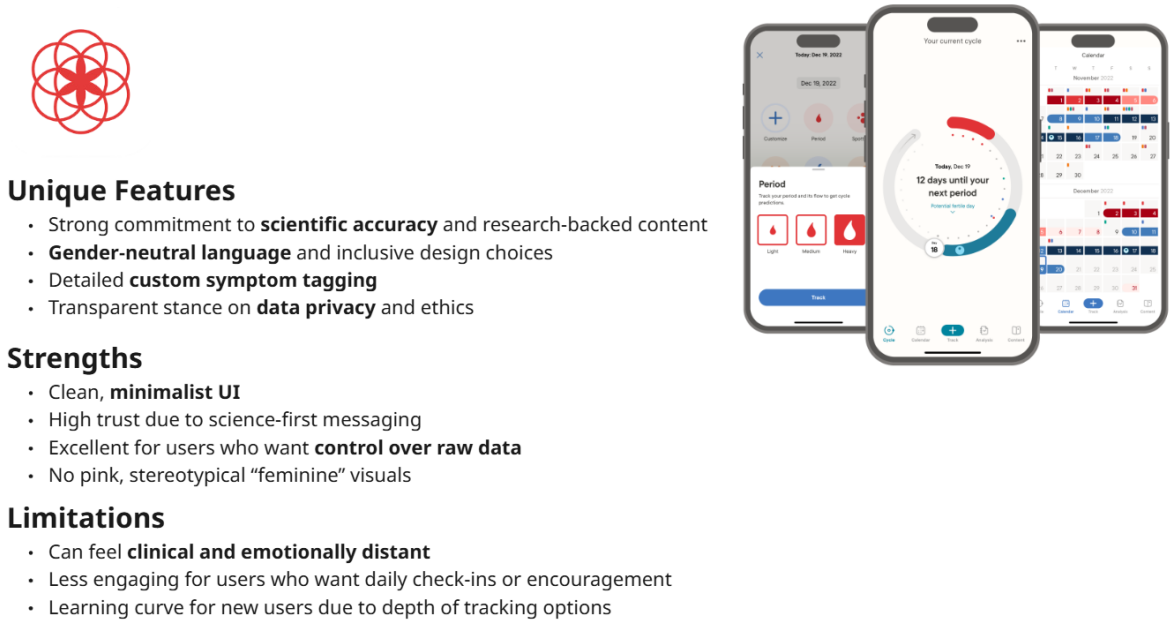


Figure 9 - Clue: Unique Features, Strengths, and Limitations

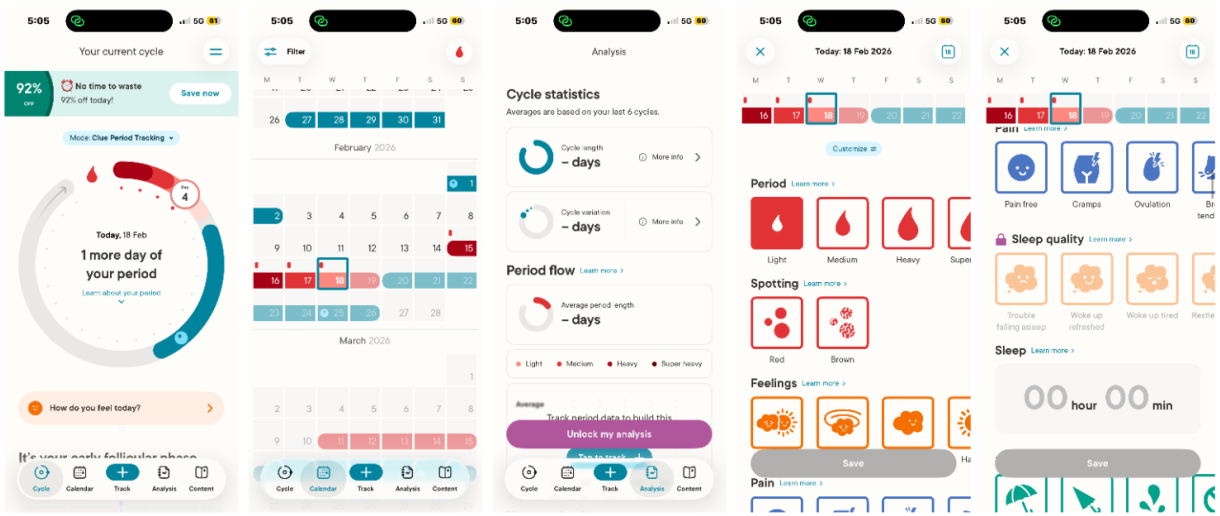


Figure 10 - Clue - App Screenshots (Clue, 2019)

3.1.4 Comparative Analysis

Each of the three application reviews takes a different approach to cycle tracking. While all three have clear strengths, a number of consistent limitations emerged across them that directly informed the design and scope of Mira.

The most significant shared limitation was paywalls, with the most useful feature locked behind a premium subscription in all three applications. Interface complexity is also a recurring issue, with Flo feeling overwhelming and Clue presenting a steep learning curve. None of the three applications offers meaningful support for users with menstrual health-related conditions, a gap identified as one of the most significant in the current market. Refer to Appendix A for comprehensive research on these existing applications.

Feature	Flo	Clue	Stardust
Period Prediction	TRUE	TRUE	TRUE
Mood and Pain tracking	TRUE	TRUE	TRUE
AI / smart insights	TRUE	FALSE	FALSE
Scientific credibility	Medium	High	Low
Visual personality	Polished	Minimal	Excessive

Table 2 - Comparison of Existing Cycle Tracking Applications

3.2 Project Scope

3.2.1 User Survey

As part of the initial user research phase, a survey was conducted to better understand real users' needs, frustrations, and experiences with menstrual health tracking. A total of 29 responses were collected from participants across a range of age groups, with equal representation from 18-24, 25-34, and 35-44 (8 respondents in each) and 5 respondents aged 45-54. This spread ensured that the findings were not limited to one demographic. Refer to Appendix A for full survey results.

When asked about cycle regularity, the majority of respondents described their cycles as very regular (34%) or somewhat regular (28%), whereas around 34% reported having irregular or very irregular cycles. This highlighted early that any cycle prediction (if any) would need to be flexible and considerate of users with irregular cycles.

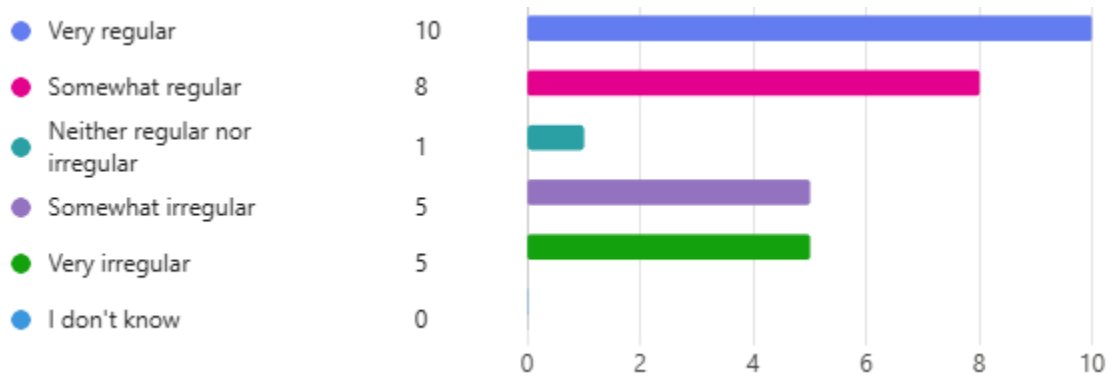


Figure 11 - Survey results showing respondents' cycle regularity

Over half of respondents (62%) reported experiencing medium to heavy periods, whereas only 10% described their flow as light. In terms of menstrual health conditions, 62% of respondents indicated they currently have or suspect they have conditions of PCOS, endometriosis, PMDD, or irregular cycles due to an unknown cause. Despite this, only 8 respondents have received a formal diagnosis, which suggests that many are trying to manage symptoms without any confirmed diagnosis. This reinforced the need for condition-specific tracking features.

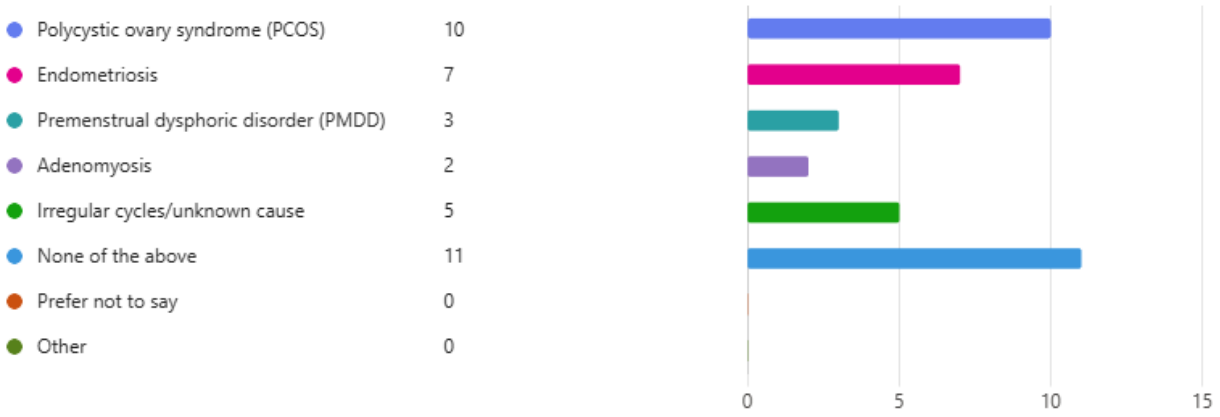


Figure 12 - Survey results showing self-reported menstrual health conditions among respondents

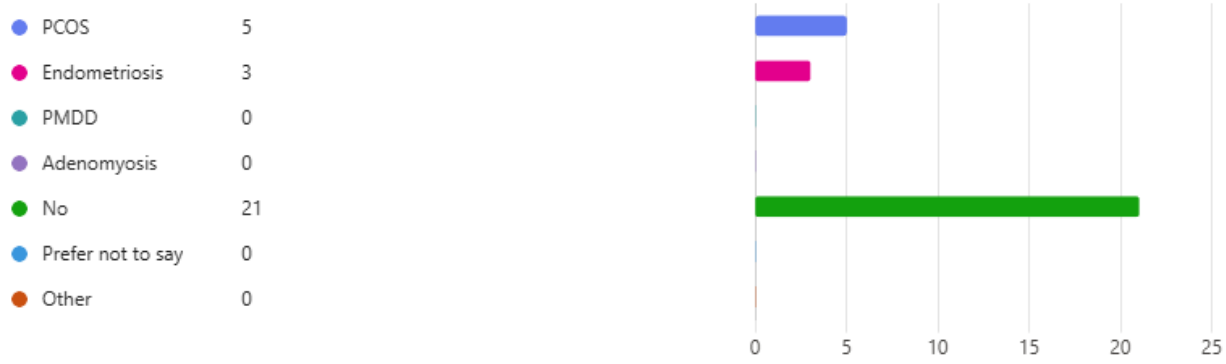


Figure 13 - Survey results showing formal diagnosis of menstrual health conditions among respondents

Logging habits revealed an important finding around engagement. The majority of respondents said they either don't log regularly (31%) or only log during their period (28%), with only a small percentage saying they log daily (3%). This suggested that current applications are not doing enough to encourage consistent engagement, and that any new application would need to make logging as quick and effortless as possible.

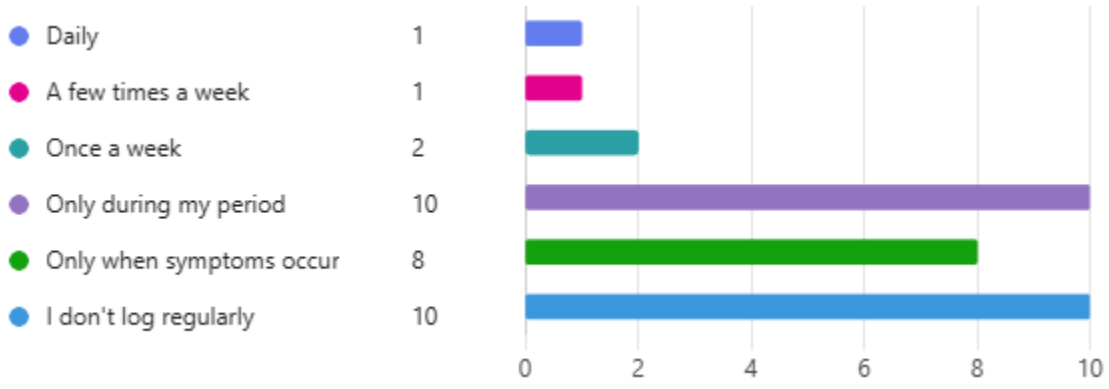


Figure 14 - Survey results showing how frequently respondents log information in their cycle tracking app

Period prediction was by far the more common reason (38%) people use tracking applications, but symptom logging, mood tracking, and condition management were also frequently mentioned. When asked to rate their experience with current applications, the results were mixed, with the most common ratings being between 3 and 5. This indicates that while existing applications are functional, there is clear room for improvement.

Figure 15 - Survey results showing respondents' satisfaction ratings with their current cycle tracking app(s)

The most common complaints were that applications contain too much unnecessary content (20%), that predictions weren't accurate (10%), and that applications aren't tailored to people with menstrual health conditions (20%). A recurring theme was that applications feel too "pink"

and gendered (12%), which some users found off-putting. These findings directly influenced the scope of this project, aiming for a clean, inclusive interface that avoids unnecessary clutter.

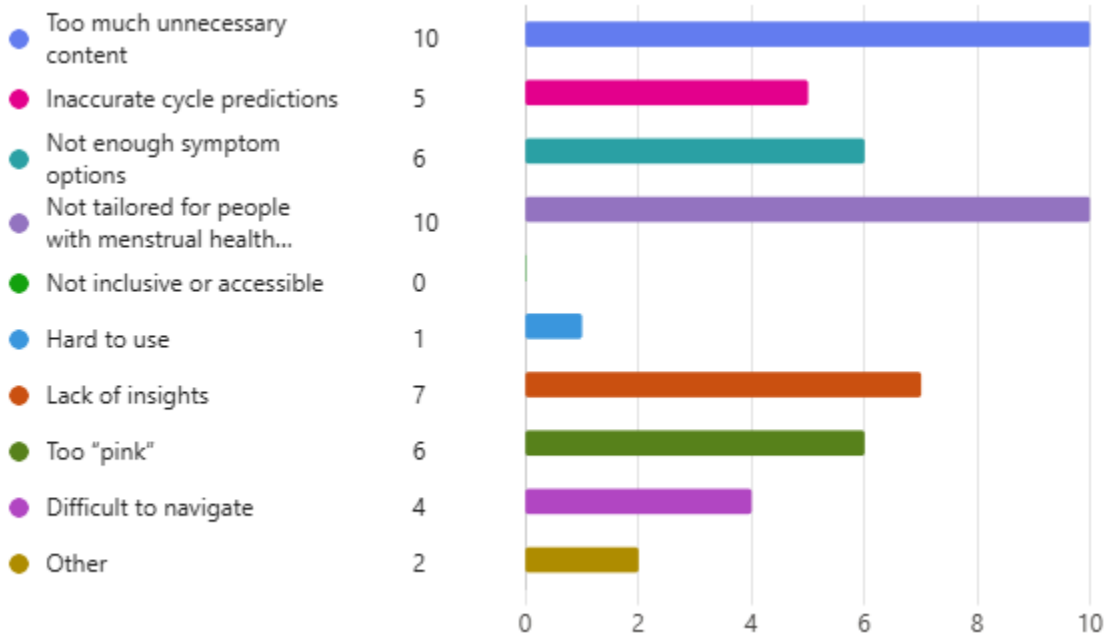


Figure 16 - Survey results showing respondents' biggest frustrations with current female health and cycle tracking applications

When asked what symptoms they most wanted to track, pain levels, cramping, mood swings, fatigue, and period flow came up most consistently. The most important features overall were symptom tracking, mood tracking and condition-specific tracking; these all became core requirements for this application.

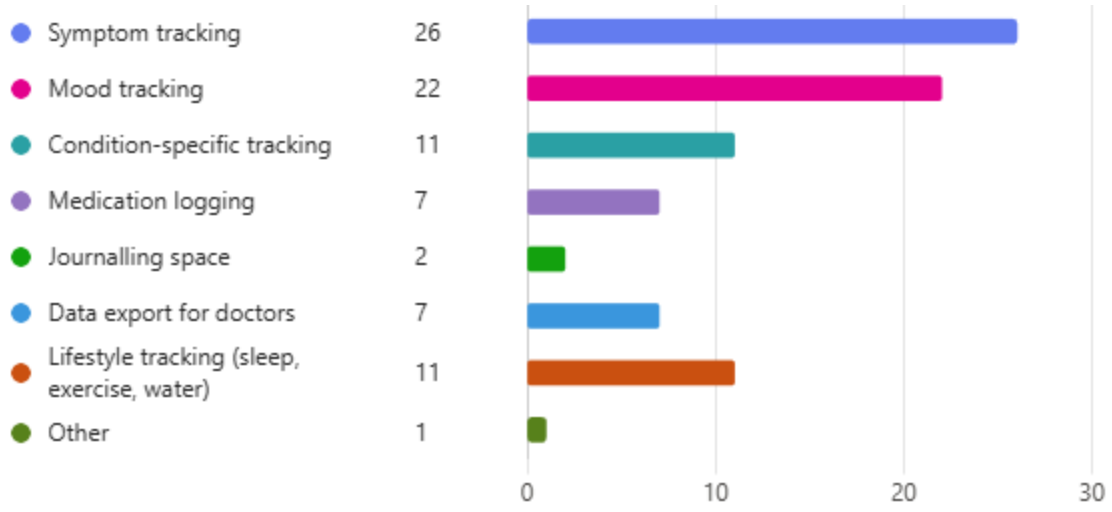


Figure 17 - Survey results showing the symptoms respondents most want to track in a cycle tracking app

Finally, when asked what would keep them engaged with an application, the most common answers were a clean, simple design and easy symptom logging (60% combined). These findings were consistent across almost all respondents, regardless of age or condition, so this became the thing that guided the project the most.

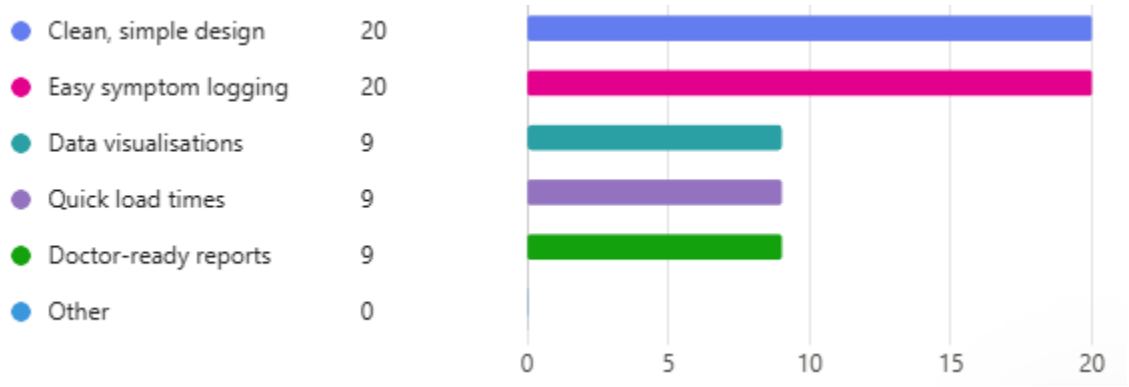


Figure 18 - Survey results showing what features would help keep respondents engaged with a cycle tracking app

Overall, the survey results backed up the initial concept for this project and helped define the scope. The finding confirmed that users want a more supportive, condition-aware, and easy-to-use experience. These insights directly shaped the application's features, design decisions, and priorities. For example, the frustration expressed around complex interfaces influenced the decision to keep the design clean and minimal, and the interface in structured symptom logging

informed how the symptom selection component was built. Rather than just validating the idea, the results guided the way the project was going to be designed and developed.

3.2.2 User Interviews

Following the survey, three one-to-one interviews were conducted to gather in-depth insights into users' experiences with cycle-tracking applications. The aim of the interviews was to go beyond the survey data and explore the real frustrations, habits, and needs of actual users.

All participants currently use or have used a cycle-tracking application, with Flo being the most mentioned. None of the participants logged consistently; all three said they only logged the start of their period or when symptoms occurred, rather than daily. These findings reinforced the need for the logging experience to be as quick and painless as possible, so that users are more likely to engage with it regularly.

Several common frustrations were encountered across all three interviews. Paywalls were the most commonly mentioned pain point; all three participants expressed frustration that most features are locked behind a subscription, with one participant describing Flo as "useless" because most of its features require payment. Two of the three participants also raised concerns about the interface being "too pink" or cluttered, and one participant raised a concern about data privacy, noting that their current application may have sold their data.

The most common features requested were simple and organised symptom logging, pain and cramp tracking, and mood tracking. One participant highlighted the importance of being able to track flow heaviness. Data visualisation was also mentioned as a preference over text-heavy content, with all participants wanting to see their information displayed clearly. All participants expressed a preference for a clean, simple interface that does not overwhelm the user with information. The recurring theme was that the homepage should only show what information is essential.

The interviews reinforced and expanded on the findings from the user survey, providing a clearer picture of what users actually need from a cycle tracking application. The consistent demand for simplicity, painless logging, and a clean, non-gendered interface became the guiding principles

for Míra. Together, the survey and interview findings provided a strong foundation for the decisions made throughout the project.

3.2.3 Personas

A persona is a fictional yet realistic description of a target user, designed to help teams empathise with the people they are designing for. Personas must be based on user research to accurately represent real users (Harley, 2015).

Four user personas were created based on the findings from the user surveys and interviews, each representing a different type of user that Míra is designed for. The first, Aoife, a 21-year-old student from Sligo who struggles with irregular periods and finds most existing applications overwhelming, wants a simple, straightforward way to feel more in control of her menstrual health. The second, Niamh, a 27-year-old UX designer from Dublin, who finds applications too cluttered and overly pink, needs quick daily logging and values privacy.

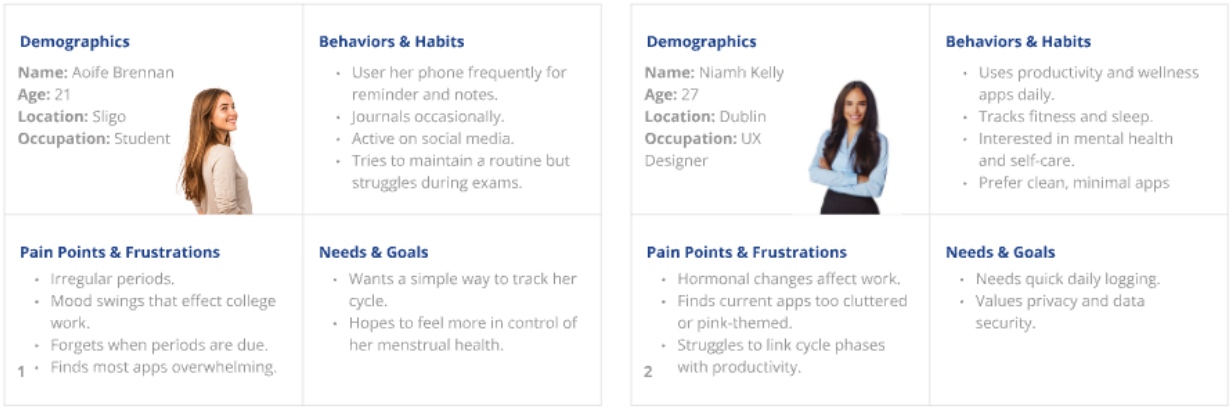


Figure 19 - First two personas

The third, Sarah, a 30-year-old retail manager from Limerick who experiences painful periods and feels dismissed by healthcare professionals, needs to track her symptoms clearly over time and wants data she can share with her doctor. Lastly, Chloe, a 24-year-old postgraduate student from Cork who has an active and organised lifestyle but struggles with painful periods and wants an application to help her manage her pain and symptoms better.

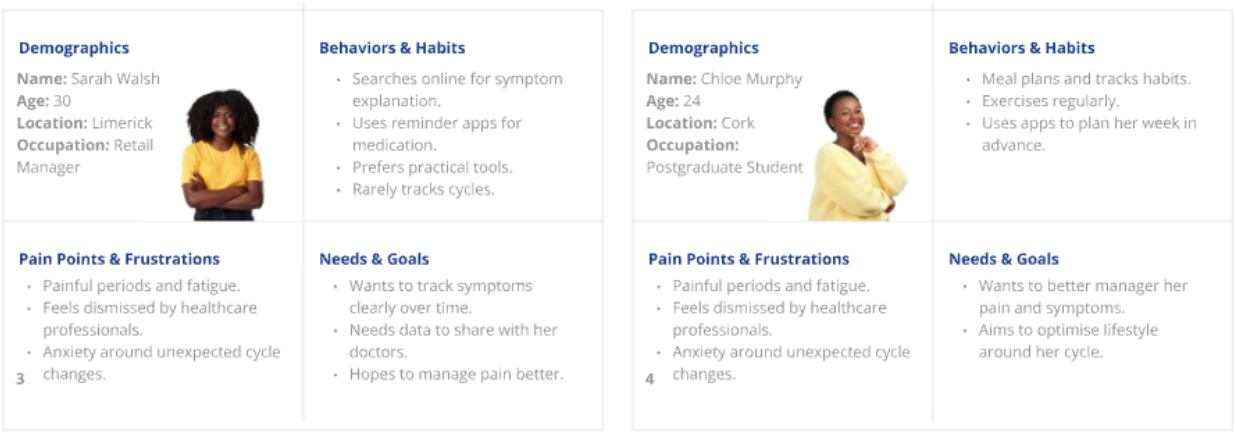


Figure 20 - Last two personas

Together, these personas helped ensure that the design and feature decisions made throughout this project were grounded in the real and varied needs of the application's target rather than assumptions.

3.2.4 Empathy Maps

An empathy map is a collaborative visualisation used to articulate what we know about a particular type of user, split into four sections: Says, Thinks, Does, and Feels, with the user in the middle. Empathy maps help categorise knowledge about users in one place, creating a shared understanding of their needs and guiding design decisions (Gibbons, 2018).

Two empathy maps were created for this project, based on the findings from the user surveys and interviews. The first, Julia, a 20-year-old university student living in Dublin with a busy academic and social life. She tracks her periods inconsistently, pushes through pain during lectures, and feels overwhelmed and anxious about the unpredictability of her cycle. Her goals are to understand her cycle, manage pain around exams, and feel in control of her menstrual health. Next, Zarah, a 25-year-old working professional from Cork, has been diagnosed with PCOS and suspects she might also have Endometriosis. She tracks her symptoms daily and saves menstrual health notes to share with her doctor, but feels frustrated and misunderstood because most applications don't support menstrual health conditions, and predictions don't apply to her cycle. Her goals are to identify symptom patterns and manage her conditions more effectively.

Figure 21 - Empathy Map for Julia Gorman

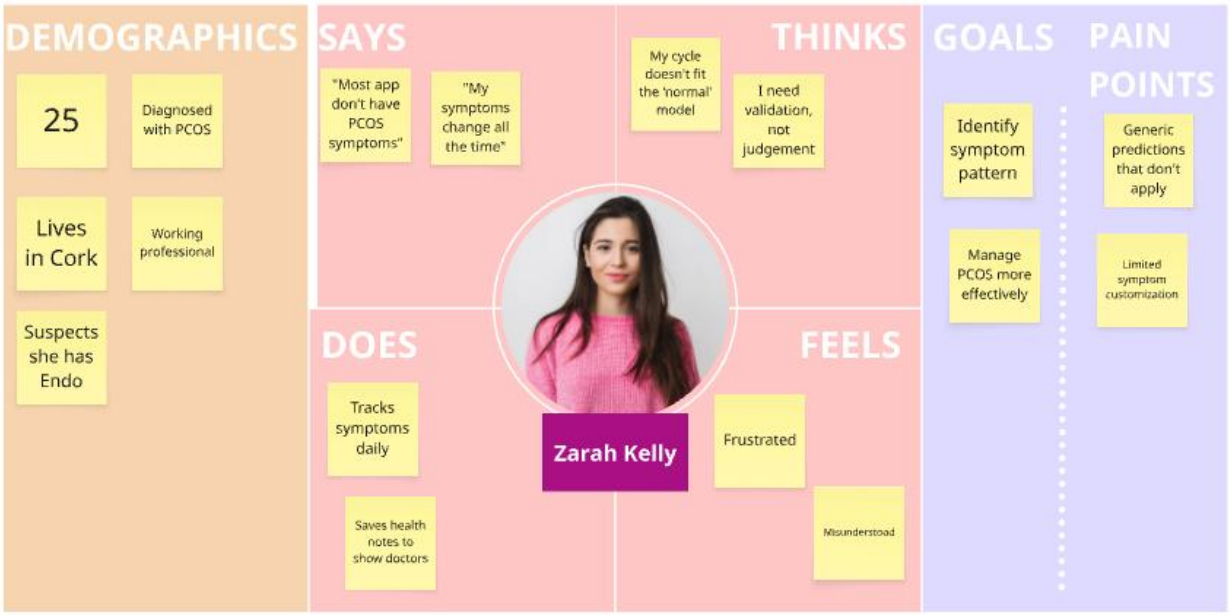


Figure 22 - Empathy Map for Zarah Kelly

3.3 Functional and Non-Functional Requirements

The requirements for this application are split into two categories: functional and non-functional requirements. Functional requirements define what the application should do, the specific

features and operations it must have to meet the user's needs. Whereas non-functional requirements define how the application should operate, covering qualities such as performance and usability (GeeksforGeeks, 2020).

3.3.1 Functional Requirements

The table below shows the functional requirements for this application, ordered from highest to lowest priority. The highest-priority requirements form the core functionality of the application: secure registration and login, period date logging, symptom tracking, and cycle history are fundamental to making this application work. The medium priority requirements build on these by adding condition-specific tracking and data visualisation. The ability to export health data for sharing with a healthcare professional was assigned low priority, as it is helpful but not critical for the core experience. These requirements were shaped by the findings from the user surveys/interviews during the Discover phase of Double Diamond. Refer to Appendix C to see the two iterations of functional requirements.

No.	Functional Requirements	Priority
1	Users must be able to create an account and log in securely	High
2	Users must be able to log the start and end dates of their period	High
3	Users must be able to log daily symptoms	High
4	Users must be able to view their cycle history on a calendar view	High
5	The application must store user data securely in a MySQL database	High
6	The application must support optional condition-specific tracking for users with PCOS and Endometriosis	Medium
7	The application must display data visualisations to help users identify patterns in their cycle over time	Medium
8	Users must be able to export their health data in a format suitable for sharing with a doctor	Low

Table 3 - Functional Requirements

3.3.2 Non-Functional Requirements

The table below shows the non-functional requirements for this application, organised by category. Security: all authentication must be handled securely using Laravel Sanctum with encrypted passwords. Performance: the application should load and respond within 3 seconds to ensure a smooth experience. The remaining requirements are centred on usability and accessibility: the application must conform to WCAG AA standards, be intuitive and easy to navigate for new users, use clear, inclusive language, display data visualisations clearly, and provide user-friendly error messages. These non-functional requirements reflect the project's core design values, ensuring the application is not only functional but also accessible and easy to use for a variety of users. Refer to Appendix C to see the two iterations of non-functional requirements.

1	User passwords must be encrypted, and authentication must be handled securely via Laravel Sanctum	Security
2	The application must load and respond to user interactions within 3 seconds	Performance
3	The application must conform to WCAG AA accessibility standards	Accessibility
4	The interface must be intuitive and easy to navigate for first-time users	Usability
5	The application must use clear and inclusive language throughout, avoiding overly clinical or gendered terminology	Usability
6	The application must display data visualisations clearly	Usability
7	The application should provide clear and user-friendly error messages	Usability

Table 4 - Non-Functional Requirements

3.4 Feasibility Study

3.4.1 Idea Exploration, Identification and Assessment

Application Concept

Mira is a cycle-tracking application designed to help users better understand and manage their menstrual health. The application will allow users to log their period dates, track daily symptoms, and view patterns over time. Unlike many existing applications, Mira will include

optional condition-specific tracking for users with menstrual health conditions, which makes it more inclusive and a supportive tool for a wider range of users.

The “problem” this application aims to solve

According to surveys and interviews conducted during the Discover phase of this project, many users are dissatisfied with the cycle-tracking applications currently on the market. The average satisfaction rating given to existing applications was just 2.85 out of 5. The most common frustrations included too much unnecessary content, a lack of condition-specific support, and interfaces that feel overly “pink” and gendered. Many participants also reported not tracking regularly. This suggests that current applications are not doing enough to make the experience simple and engaging. Mira aims to address these gaps by offering a clean, accessible, and condition-aware tracking experience, and actually puts the user's needs first.

Benefits of the application

- Provides a simple, focused space for menstrual health tracking without unnecessary clutter.
- Supports users with menstrual health conditions through optional condition-specific features.
- Help users identify patterns in their cycle through clear data visualisation.
- Gives users a greater sense of control and understanding over their own health.

3.4.2 Target Users

Míra is designed for anyone who menstruates and wants to better understand their cycle. It is particularly suited for those with menstrual health conditions, as well as for those who have found existing applications too complicated, too cluttered, or not tailored to their needs.

3.4.3 Pitfalls and Weaknesses

Weakness: Building a fully functional mobile application with a complete back-end within the project timeline.

Solution: Prioritise core functionality and add features only once the foundation is solid and stable.

Weakness: Designing an interface that feels inclusive and non-gendered without losing identity.

Solution: Avoid a stereotypical feminine colour palette and base all design decisions on user research findings rather than assumptions.

Weakness: Keeping users engaged with consistent logging over time.

Solution: Keep the logging process as quick as possible. Based on survey and interview findings, easy symptom logging was one of the top factors users cited as keeping them engaged with the application.

4.0 Project Management and Plan

4.1 Project Management

To support project planning, organisation, and development, four tools were used throughout this project: Miro, Notion, GitHub, and Artificial Intelligence (AI).

Miro is a cloud-based collaborative whiteboard platform that enables users to visually map ideas, create flowcharts, and organise workflows in an interactive and flexible environment (Miro, 2024). It was very useful during the early ideation and design phases of the project, where it was used to sketch out ideas, map user flows, create diagrams, and organise thinking before moving into a more detailed design work in Figma. Refer to Appendix A to see the full Miro board.

Notion is an all-in-one productivity and note-taking application that combines documents, databases, and task management into a single workspace (Notion, 2019). This tool was used consistently throughout the project to maintain structured notes, track weekly progress, store research materials, and record user interview summaries and testing findings. The example below shows one way Notion was used. Refer to Appendix D to see the Notion web page used.

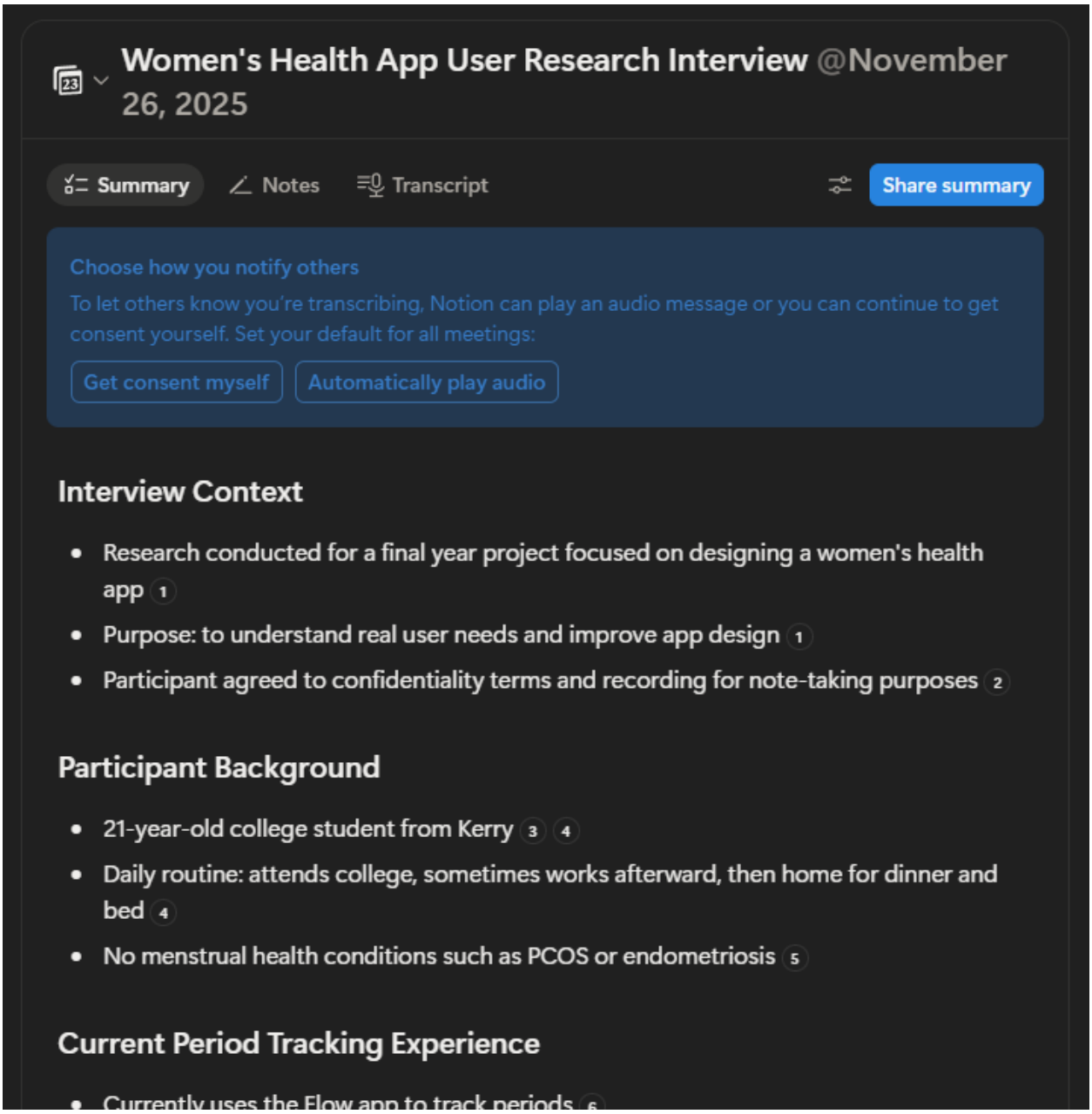


Figure 23 - User interview summary stored in Notion

GitHub was used for version control and code management throughout the development phase. All code was committed to separate GitHub repositories, with commit messages documenting progress and changes at each stage. Using GitHub meant that the full development history of this project was documented, making it easy to track how the application evolved over time. Refer to Appendix E to access the GitHub repository.

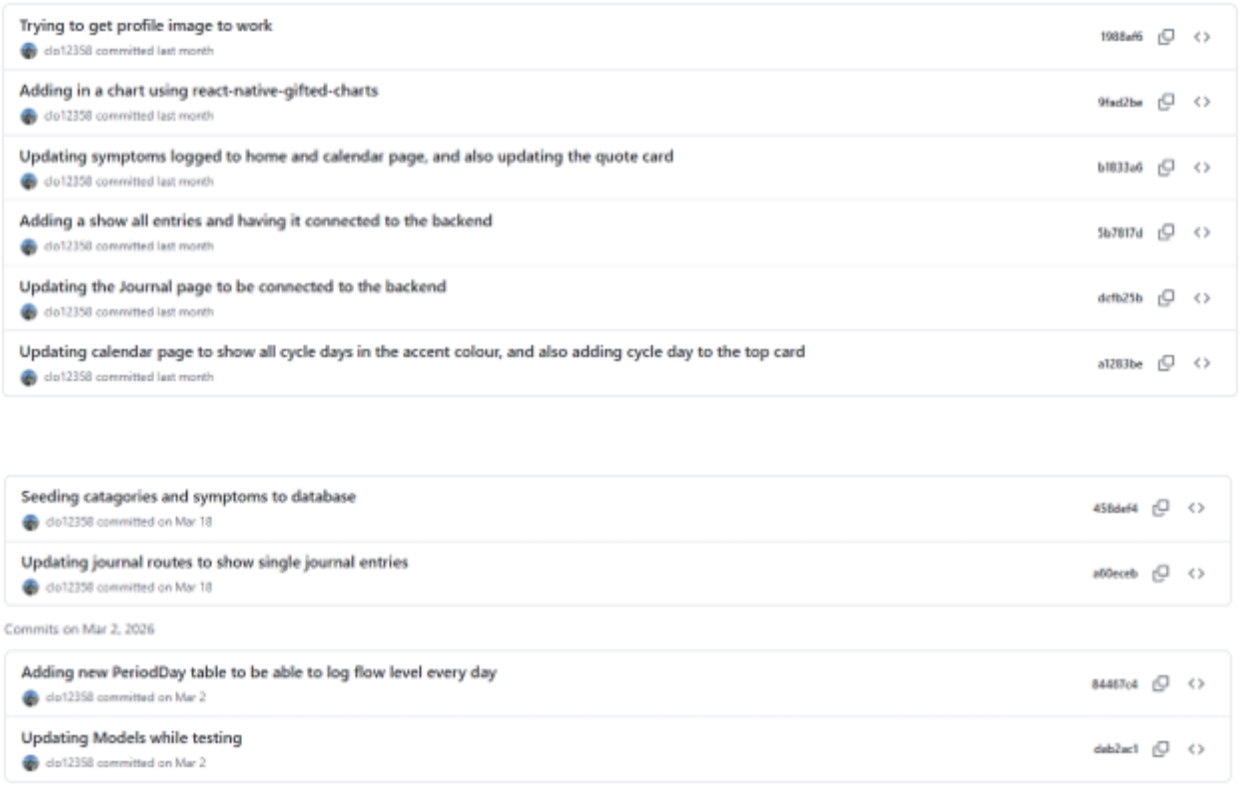


Figure 24 - Screenshot showing GitHub commits

AI tools were also used to support the development phase, particularly during the more challenging parts of the project. The two tools used were GitHub Copilot and Claude

GitHub Copilot is a built-in feature added to Visual Studio Code that offers real-time suggestions as you type, predicting what you might type next based on the code around it. This was very useful at the start of most files, where a large amount of setup code is required, particularly the utility functions. Writing these from scratch was one of the more challenging parts of this project, and Copilot helped by giving suggestions. It also supported the debugging process, since it had full access to the codebase.

Claude was used alongside Copilot as a planning and verification tool. Claude was useful in the planning stage of writing functions. The idea of each function would be given to Claude in plain terms. Claude would explain the best approach to take, and why, before any code was written. When Copilot generated any code, it was given to Claude and asked to explain exactly what it

was doing and why. This helped to understand the code. This two-step process meant that AI assistance supported the learning process.

4.2 Project Plan

To ensure the project would be completed on time and within scope, the workload was divided into a structured weekly plan with specific tasks assigned to each week. Rather than approaching the project as a single large task, breaking it down into smaller, manageable weekly goals made it much easier to maintain steady, consistent progress. This approach had several advantages; having a clear weekly structure meant that at any point in development, it was possible to see what had been completed, what was in progress, and what still needed to be done. It also helped to prevent leaving a large portion of work until the very end of the project. The project development was planned across thirteen weeks, each with a specific focus:

1. **Week 1**: Mid-fidelity prototypes will be created in Figma, initial user testing will be conducted, and the front-end and back-end repositories will be set up.

2. **Week 2**: High-fidelity prototypes will be created, and interactions added, the UI will be finalised, the API plan will be confirmed, and Laravel Sanctum will be implemented.

3. **Week 3**: Core back-end functionality will be developed, the basic front-end navigation and base pages will be started, and testing will be done on the back-end.

4. **Week 4**: Data visualisation will be tested, the backend will be finished, and the API will be connected to the front-end on the register and login page.

5. **Week 5**: Three of the main pages will be built and connected to the back-end, informal testing will be done, and refinements will be made.

6. **Week 6**: Data visualization will be added, and simple testing on the graph will be conducted to ensure it's what the users want.

7. **Week 7**: The front-end will be completed, user testing will be conducted, feedback will be gathered, and any refinements will be made.

8. **Week 8**: The front-end, back-end and data visualisation will be completed, and any issues found in user testing will be resolved.

9. **Week 9**: A final round of user testing will be conducted, feedback will be gathered, and a draft section of the user testing chapter will be written up.

10. **Week 10**: The writing of the thesis will start.

11. **Week 11**: Thesis writing will continue, Miro, Figma, and Notion pages will be cleaned up.

12. **Week 12**: The thesis will be refined based on supervisor feedback, the thesis will be proofread, and comments will be made on what needs to be changed.

13. **Week 13**: A final review of the application and thesis will be completed, submission materials will be prepared, and everything will be submitted.

The sprint plan served as a guide rather than a rigid structure. In practice, development did not follow it exactly; some tasks took longer than expected, priorities shifted as the project progressed, and the order of certain work changed along the way. Refer to Appendix F to see the full sprint information.

A weekly diary was also maintained in Notion throughout the project to record each week's completed work, issues encountered, and how they were resolved. This served as both a personal accountability tool and a useful reference point when writing up the implementation chapter. The screenshot below shows the weekly diary.

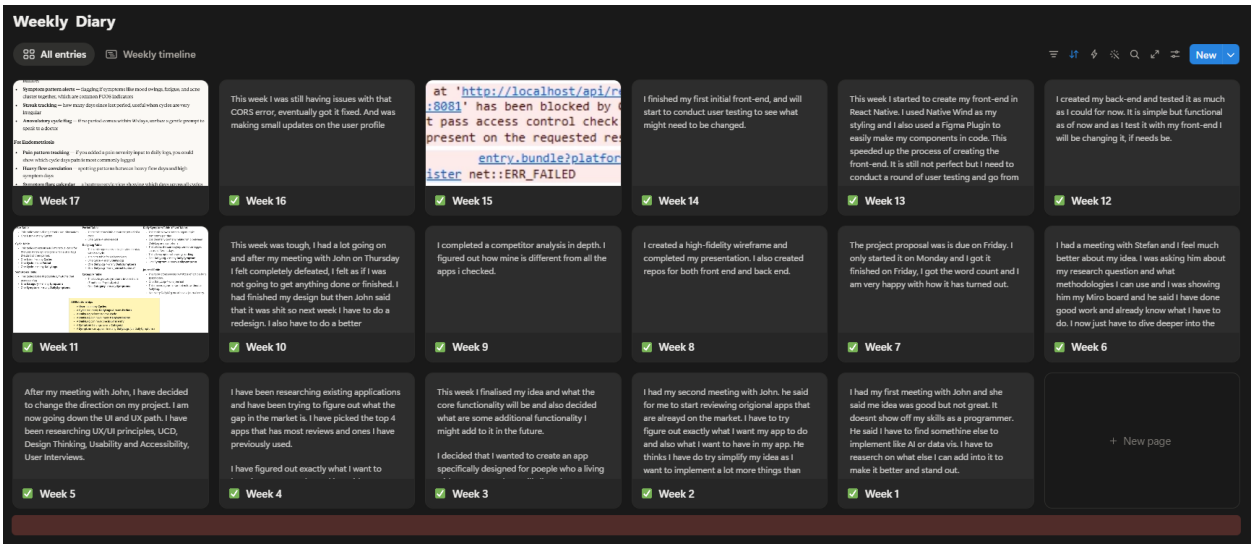


Figure 25 - Weekly diary in Notion

5.0 Design

This chapter covers the design decisions made throughout the development process. It documents the thinking behind the application's structure and appearance, from its architecture to its prototypes. Refer to Appendix A to see all the design chapter diagrams.

5.1 Application Architecture

According to Ravi Sarawathi (2020), application architecture describes the patterns and techniques used to design an application, and the decisions made at this stage significantly impact the performance, security, and maintainability of the final application.

For this project, a client-server architecture was used, which is one of the most common approaches in modern mobile application development. In this model, the client sends a request to the controller, which sends a request to the model. The model then sends a response to the controller, and from there, the controller sends the data to the view, which sends the response to the client.

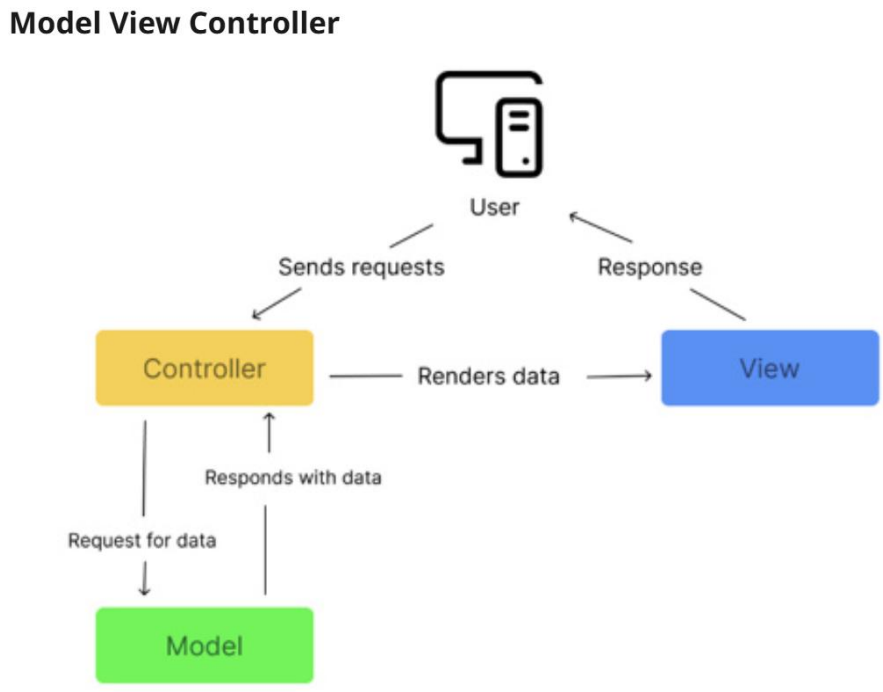


Figure 26 - Model View Controller (MVC)

As shown in Figure 27, the Mira application architecture comprises four main components. The client/user interacts with the application through the front-end, built with React Native and NativeWind for styling. The front-end communicates with the back-end via a REST API, where Laravel handles all server-side logic. Laravel Sanctum manages secure token-based authentication, ensuring that only verified users can access their data. Finally, the data is stored and managed in a MySQL relational database, which handles everything from user accounts to logged symptoms and cycle data.

This architecture was chosen because each technology integrates well with the others, and the combination provides a secure, scalable, and manageable foundation for this application.

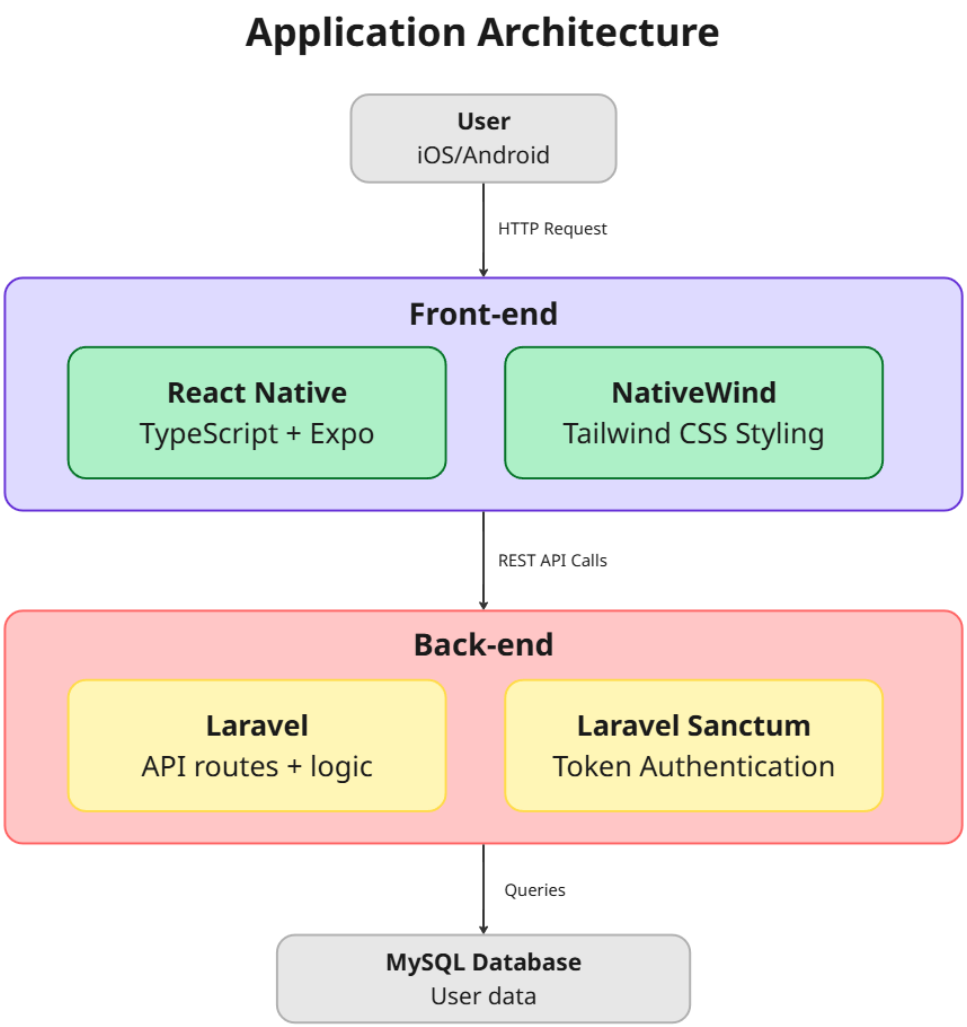


Figure 27 - Application architecture diagram

5.2 Application Design

5.2.1 Technologies

The technologies chosen for this project were selected based on how well they work together, the application's needs, and existing familiarity with most of them. While React Native was a new technology to learn for this project, having prior experience with React made the transition manageable, as both share the same core concepts. This meant the learning curve was easier, so

more time could be devoted to user research, design quality, and building a full-stack application.

The front-end is built with React Native, Expo, and TypeScript, styled with NativeWind. The back-end is built with Laravel and Laravel Sanctum for token-based authentication. GitHub was used as a management tool for both the front-end and back-end. For the database, MySQL is used to store data, with TablePlus as the management tool.

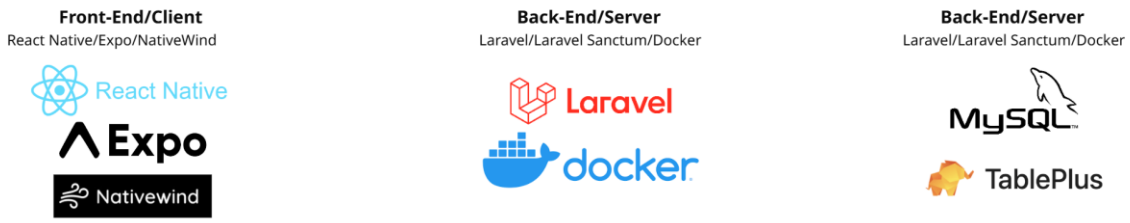


Figure 28 - Application Technology Stack

5.3 Interface Design

5.3.1 Wireframes

A wireframe is the skeleton of an application, a simple visual guide and content placement rather than colours, fonts, or visual appeal. Wireframes help clarify structure, communicate ideas early, and save time identifying problems before development starts (What Is a Wireframe? A Guide for Non-Designers, 2025).

For this project, the wireframing process began with a low-fidelity hand-drawn sketch, which allowed for quick exploration of layout ideas and screen structure. Low-fidelity wireframes use basic shapes like boxes and lines to represent elements such as buttons, text, and images. As shown in Figure 29, the sketches covered the key screens of the application. These were then developed into low-fidelity and then to mid-fidelity digital wireframes using Figma, a platform for creating websites, interfaces, and prototypes (Figma, 2025). This gave a clearer picture of the layout and content placement before moving on to create prototypes and the final design.

Figure 29 - Hand-Drawn Sketches of Initial Screen Concepts

Figure 30 - Low-Fidelity Wireframes for Key Application Pages

5.3.2 Prototypes

A prototype is a working model of a product, not the final version, but an evolving one designed to help understand how the application will work in the real world. Prototypes allow teams to test concepts, identify patterns, and refine the design before full development begins (What Is a Prototype? Definitions and Benefits of Prototyping, 2024). For this project, two prototypes were created, with interactions set up between screens so that navigation and layout could be tested as if it were a real application. Refer to Appendix G to see all prototypes in Figma.

The first, shown in Figure 31, established the core screen structure and layout. At this stage, the branding and visual identity were still being explored. The navigation bar icons were simpler, and the overall layout was sparser. Interactions were linked between key screens, allowing for early testing of the user journey and overall flow.

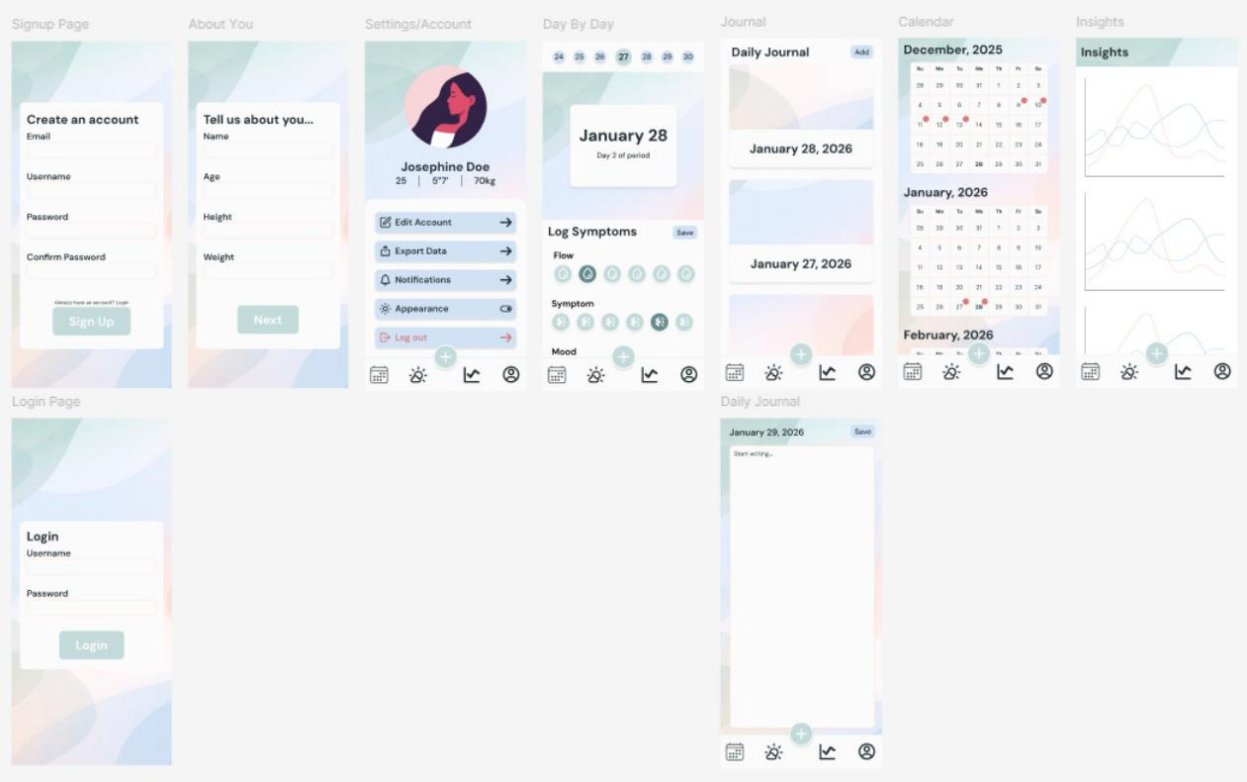


Figure 31 - Prototype One: initial screen design and layout

While no formal user testing was conducted on the prototypes, informal feedback was gathered from family and friends who navigated through the interactions on Figma. This gave a useful

early indication of whether the layout and navigation felt intuitive, and the feedback received informed the changes made between the first and second prototype.

The second, shown in Figures 32 and 33, refined the design significantly based on feedback from the first prototype. The Míra branding was consistently introduced across all screens, including the logo and refined colour palette. The home screen was improved with clearer sections, and better visual hierarchy. The profile page was redesigned to feel cleaner and more organised, and the insights page was updated to include more detailed data visualisation.

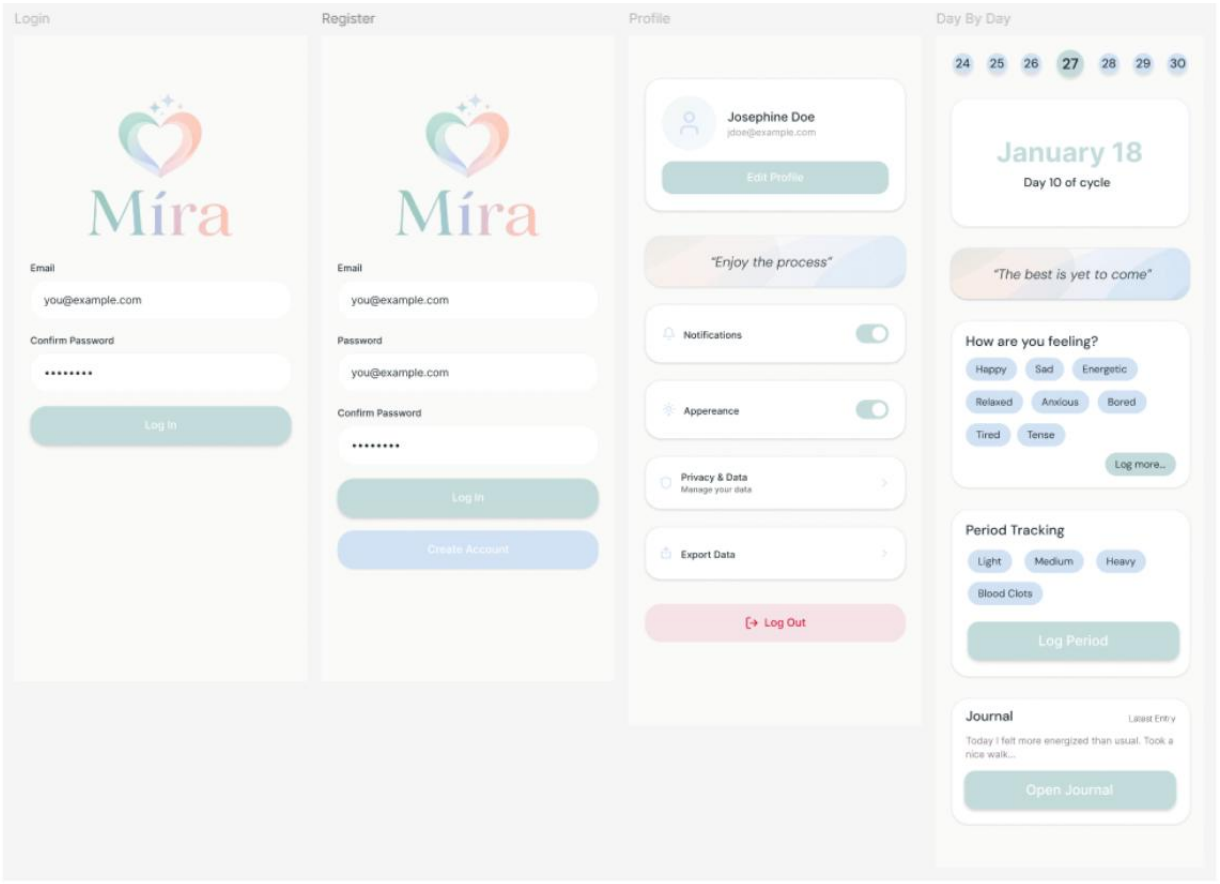


Figure 32 - First 4 pages

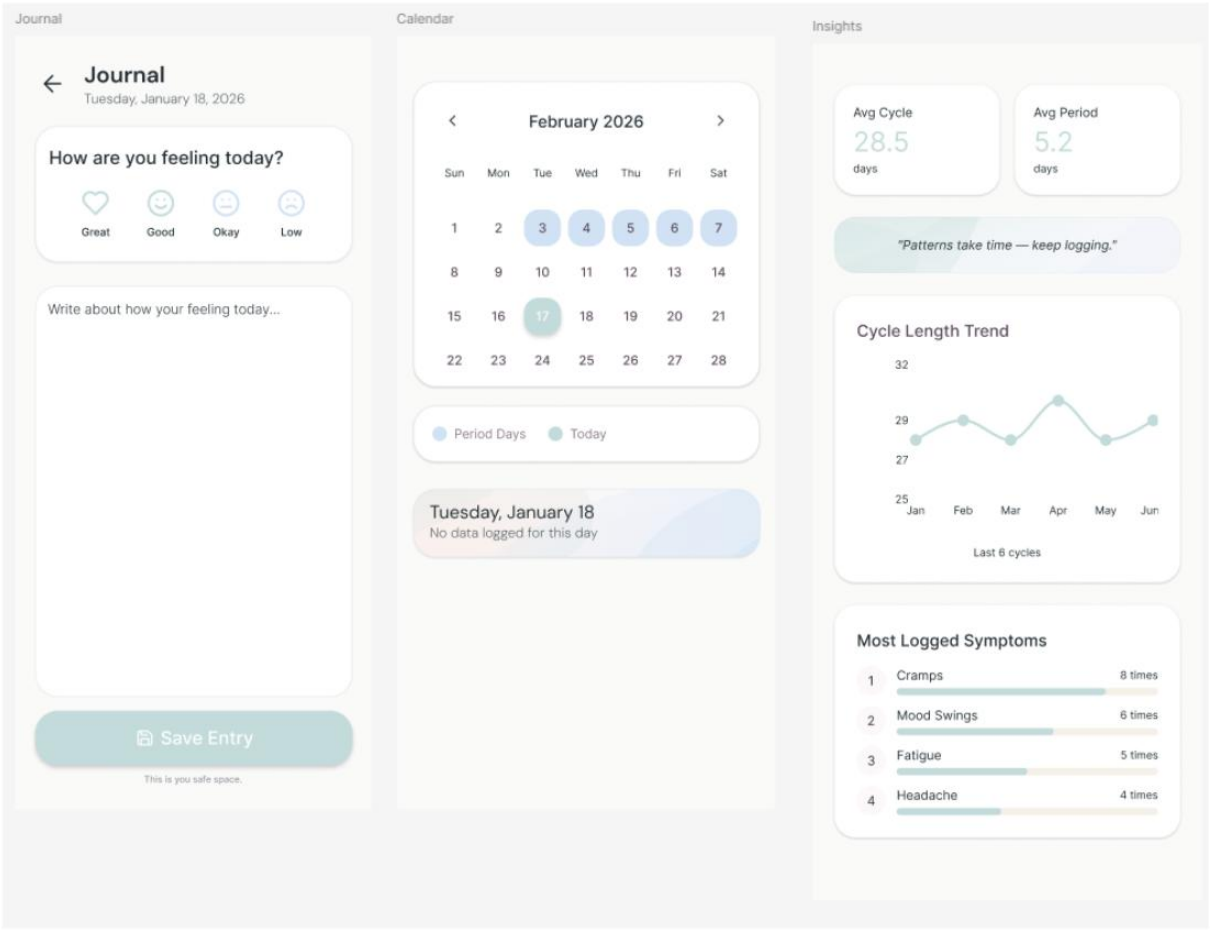


Figure 33 - Following 3 pages

5.3.3 Colour Palette

Colour choices for Míra were guided by both user research and colour psychology. A recurring finding from the survey and interviews was that existing applications feel “too pink” and overly gendered, which directly influenced the decision to take a different approach.

Figure 34 - Colour psychology (Olesen, 2013)

Green is associated with harmony and health, making users feel safe, secure and balanced, making sage green a natural primary colour for this application. Blue carries associations of trust, loyalty, and confidence, making the soft blue the secondary colour to reinforce the sense of empowerment. Pink is the colour of compassion and warmth, making this a great choice for the accent colour, allowing for a touch of femininity to be included in the application without making it overpowering. The dark navy ensures a strong contrast for text and accessibility, while the off-white for the background keeps the overall feel of the application light and clean.

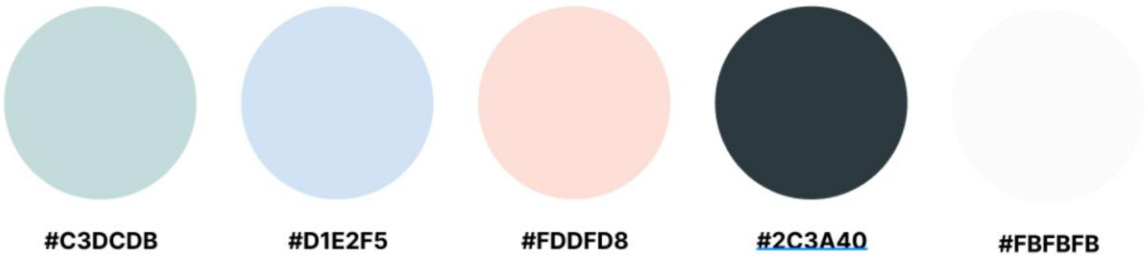


Figure 35 - Application colour palette

5.3.4 Typography

The typography used for this application was selected with clarity and readability as the main priority. Rather than importing a custom font, the application uses platform-native fonts. On iOS, this uses the default system UI font, San Francisco, while Android falls back to the standard system font. For the web version, custom CSS font variables are used to maintain consistency.

5.4 Process Design

5.4.1 Sequence Diagrams

Sequence diagrams are interaction diagrams that show how operations are carried out, showing the order of interactions between objects and what messages are sent and when (Visual Paradigm, 2019). For this project, sequence diagrams were used to plan out key user flows such as registration, login, period/cycle logging, and symptom logging before development began.

Figure 36 shows the key user flows within the application, illustrating how the user, front-end, back-end, and database interact. Each flow follows a similar pattern: the user triggers an action, the front-end sends a request to the back-end, data is stored or retrieved from the database, and the result is returned to the user. This diagram was used to plan the communications between layers of the application before the development began.

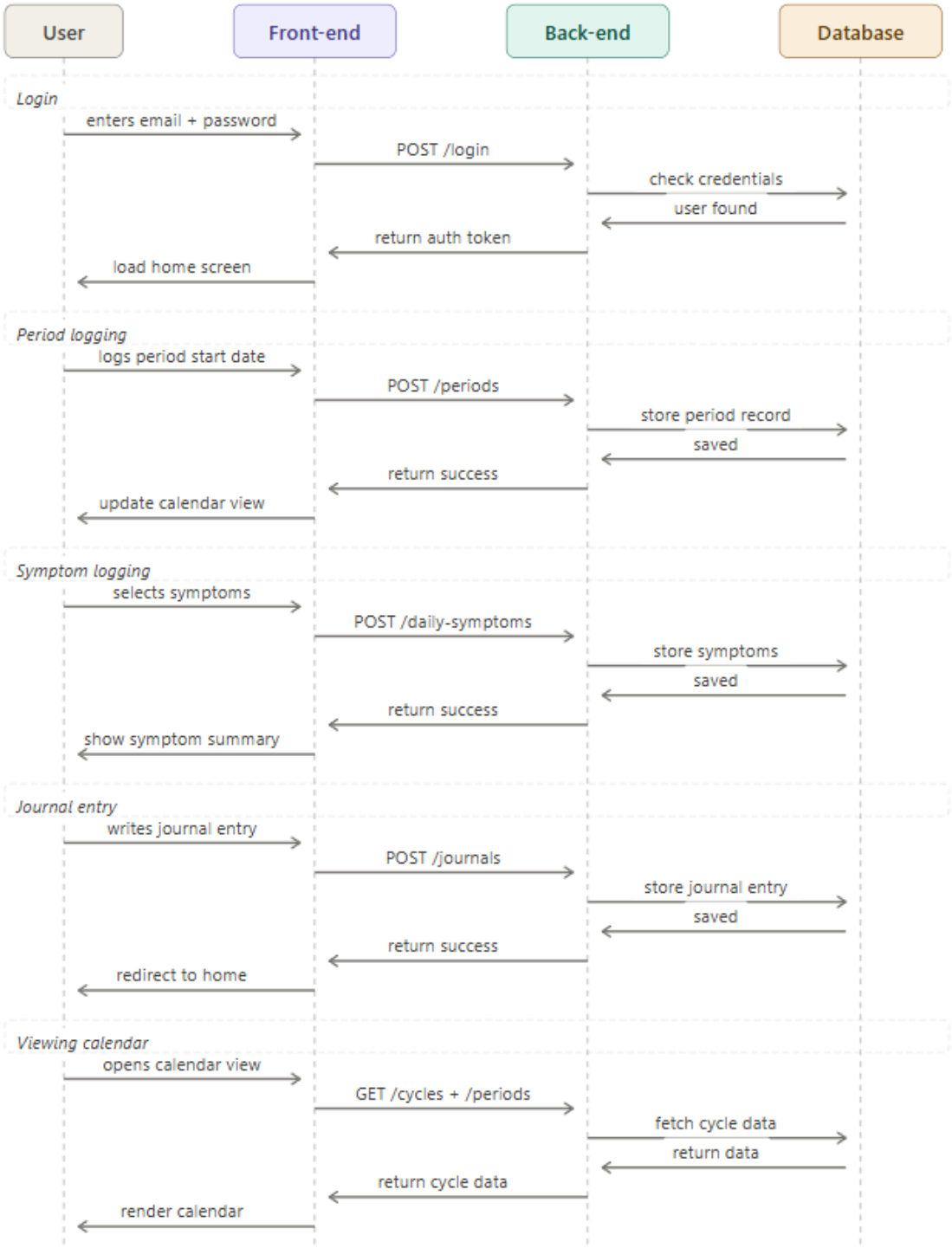


Figure 36 - Sequence Diagram showing key user interactions

5.4.2 Flow Diagrams

Flow diagrams are a simple way of creating ideas and establishing a common understanding of a complex process (Flow Diagram Guide with Examples - Pencil & Paper, 2024). For this project, a user flow diagram was created to map out the full structure of this application, showing how a user moves from registration and login through to the core screen, such as the home page, calendar page, insights page, and profile page. This gave a clear visual overview of how all the screens connect before any implementation work began.

As shown in Figure 37, the user flow begins with registration and login, which then directs to the home page (referred to as Day-by-Day in the diagram). From here users can log their period, track their mood and symptoms and access the journal. The calendar page gives users a monthly overview of their cycle, and the ability to view logged data for any day. The insights page displays trends and patterns based on the user's data, including cycle and period length trends, and most frequently logged symptoms. Finally, the profile page allows users to view and edit their information, manage settings and logout. Each screen connects logically to the next ensuring the application is easy to navigate and that all key features are accessible.

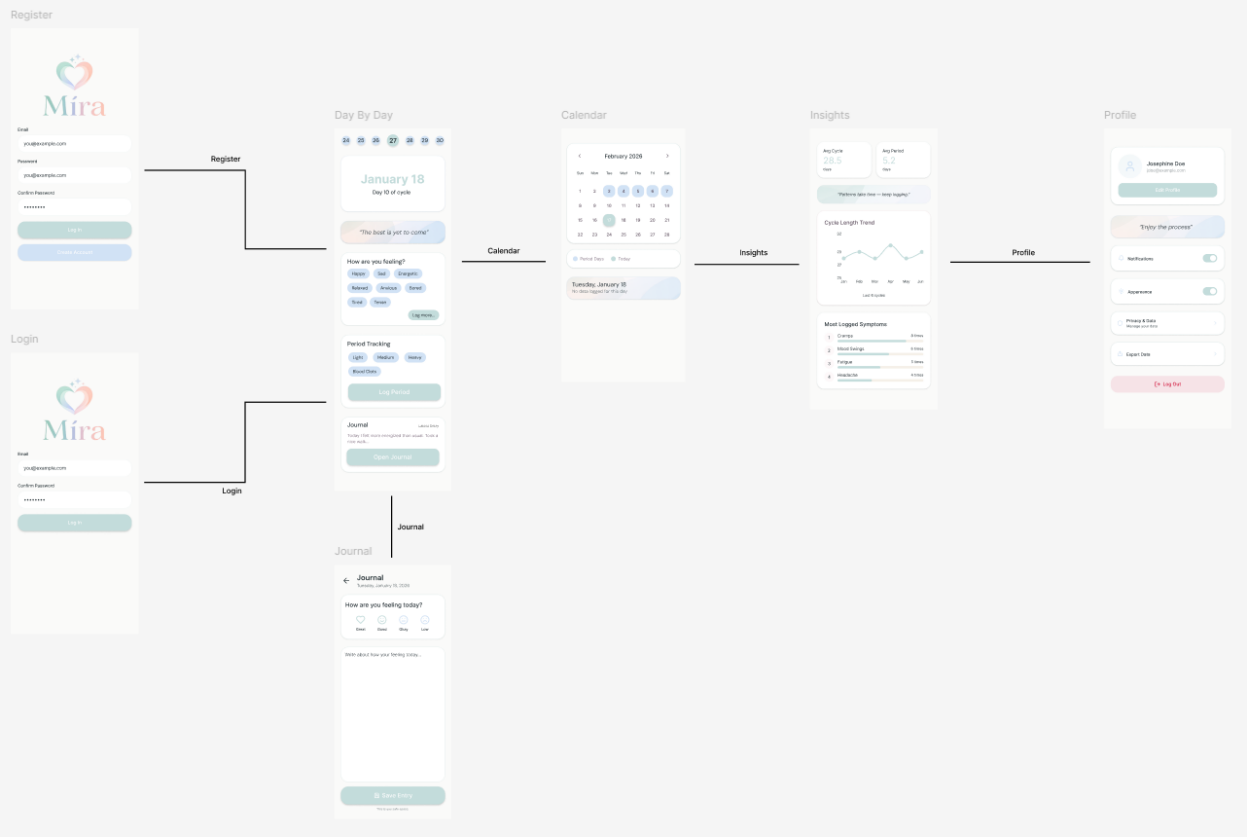


Figure 37 - Flow Diagram showing the structure and navigation

5.4.3 Use Case Diagram

A use case diagram is a type of behavioural diagram that provides a high-level visual overview of how users interact with a system. It captures the functional requirements of an application by showing the relationships between actors (users or external systems involved) and use cases, which represent the actions or goals those actors can perform within the system (Visual Paradigm, 2019b).

Figure 38 shows the use case diagram of this application. There are two actors: the User, who interacts directly with the application, and the System, which represents the back-end processes that run in response to user actions. The user-initiated use cases, shown in green, represent everything a user can do within the application. Several of these use cases include system-level behaviours (shown in purple) that are triggered automatically during the interaction. The

‘includes’ relationships indicate that the system behaviour is required to complete the user action.

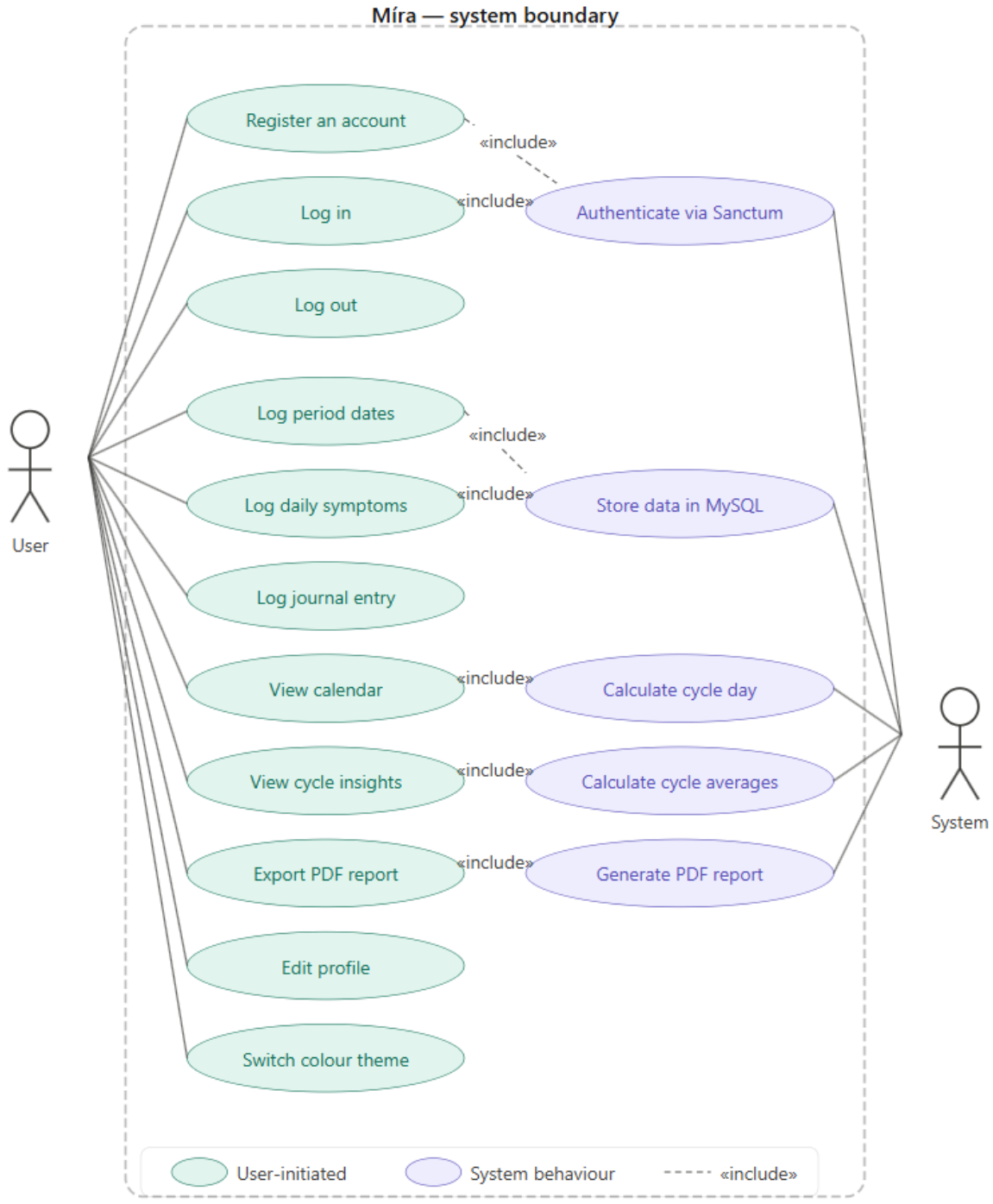


Figure 38 - Use Case Diagram

5.5 Database Design

A database is a digital repository for storing, managing, and securing organised collections of data. Relational databases store data in formatted tables of rows and columns, making them well suited for structured data (Kosinski, 2024). For this project, MySQL is used as the relational database management system to store all user data including accounts, cycle logs, and tracked symptoms.

To plan and visualise the structure of the database before development, and Entity Relationship Diagram (ERD) was created. An ERD is a visual representation of the data and the relationships between different entities within a system (LucidChart, 2024). Each entity represents a table in the database, and the lines between them show how the tables relate to one another. ERDs help ensure that the database is logical, and well organised.

As shown in Figure 39, the database consists of 9 tables. The users table, which stores account information, with each user linking to many cycles. Each cycle can have many periods and many daily logs. The period table tracks the start and end dates, with each period linking to the period_days table that stores more detailed information about days, including flow and whether clots are present. Each daily log can have many symptoms logged, with symptoms categorised for easy management. Each daily log can also have one journal entry, storing a note and an overall feeling for the day. This structure ensures data is organised logically, avoids duplication, and allows the application to retrieve and display data efficiently.

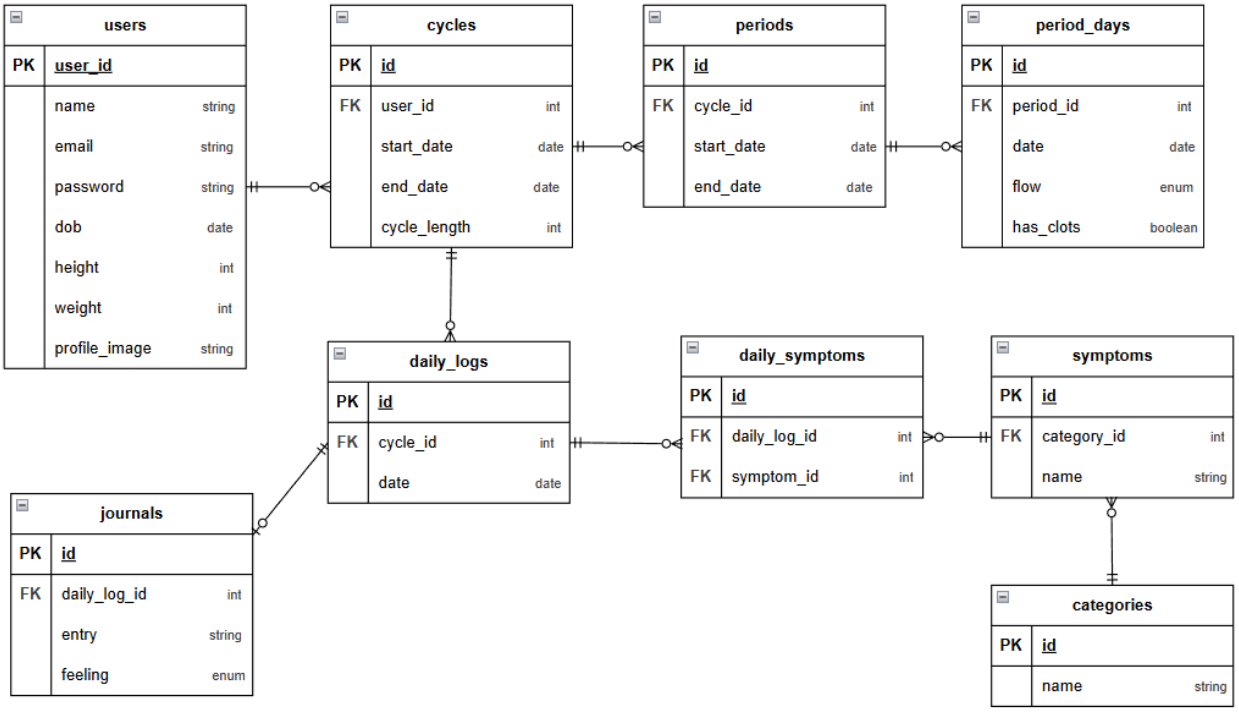


Figure 39 - Database ERD

6.0 Implementation

6.1 Back-End Development

The back-end for this application was built using Laravel, with Laravel Sanctum handling authentication and MySQL as the database. Development progressed through a series of stages, starting with setting up the database and migrations, building the models and relationships, creating API controllers, and progressively adding and refining functionality as the project progressed. Please refer to Appendix E to see the full back-end code.

6.1.1 Database Setup and Migrations

Setting up the database structure was the first phase of back-end development. Migrations were created for all nine tables: users, cycles, periods, period_days, daily_logs, daily_symptoms, journals, symptoms, and categories to make up the ERD. Figure 40 below shows all the migrations created. Each migration file defines the columns and data types for each table. Once

the migrations were run and the tables were created, the categories and symptoms tables were seeded. Without pre-populating the database, users would be unable to log symptoms. Figure 40 shows that the category seeder must be run first to seed symptoms to certain categories.

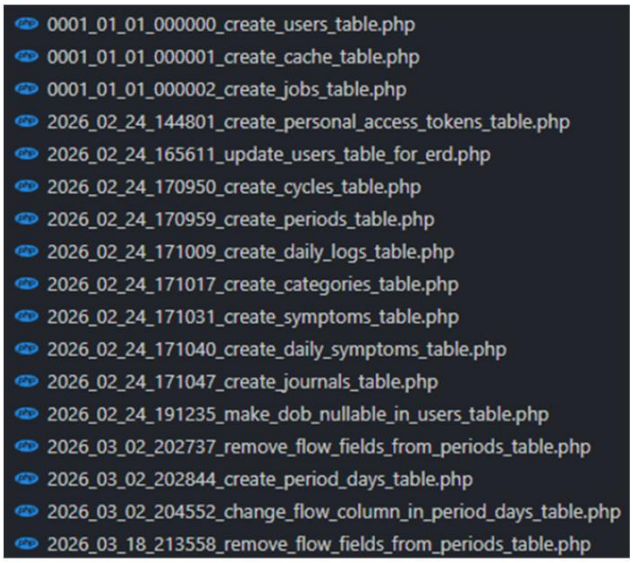


Figure 40 - Image showing migrations created and database seeder

Laravel's Eloquent ORM provides a simple ActiveRecord implementation for working with the database, where each table has a corresponding model for interacting with it (Eloquent ORM - Laravel - the PHP Framework for Web Artisans, 2011). Relationships between models were defined using Eloquent's built-in relationship methods; hasMany to represent one-to-many relationships, hasOne for one-to-one relationships and belongsTo to define the reverse of these relationships. Figure 41 shows the relationships between each table.

User.php

```php
public function cycles()
{
    return $this->hasMany(Cycle::class);
}
```

Symptom.php

```php
public function category()
{
    return $this->belongsTo(Category::class);
}
```

PeriodDay.php

```php
public function period()
{
    return $this->belongsTo(Period::class);
}
```

Period.php

```php
public function cycle()
{
    return $this->belongsTo(Cycle::class);
}

public function days()
{
    return $this->hasMany(PeriodDay::class);
}
```

Journal.php

```php
public function dailyLog()
{
    return $this->belongsTo(DailyLog::class);
}
```

Category.php

```php
public function symptoms()
{
    return $this->hasMany(Symptom::class);
}
```

DailySymptom.php

```php
public function dailyLog()
{
    return $this->belongsTo(DailyLog::class);
}

public function symptom()
{
    return $this->belongsTo(Symptom::class);
}
```

DailyLog.php

```php
public function cycle()
{
    return $this->belongsTo(Cycle::class);
}

public function dailySymptoms()
{
    return $this->hasMany(DailySymptom::class);
}

public function journal()
{
    return $this->hasOne(Journal::class);
}
```

Cycle.php

```php
public function user()
{
    return $this->belongsTo(User::class);
}

public function periods()
{
    return $this->hasMany(Period::class);
}

public function dailyLogs()
{
    return $this->hasMany(DailyLog::class);
}
```

Figure 41 - Database Relationships

6.1.2 Authentication with Laravel Sanctum

Laravel Sanctum provides a lightweight authentication system for mobile applications and token-based APIs, storing users API tokens in a single database table and authenticating incoming HTTP requests via the 'Authorization' header (Laravel Sanctum | Laravel 13.x - the Clean Stack for Artisans and Agents, 2026). Setting up the authentication was aided by a step-by-step tutorial by Noumcpe (2024) on Medium, which covered installing Sanctum, configuring the User model with the 'HasApiTokens' trait, and defining the register and login API routes. Following this allowed the authentication to be put in place quickly and with few issues.

When a user register or logs in, the back-end generates a unique authentication token which is returned to the front-end and stored locally on the device using 'AsyncStorage'. The token is then automatically attached to the header of every subsequent API request; this ensures that only authorised users can access their own data. Protected routes are wrapped in Laravel Sanctums's 'auth:sanctum' middleware. This middleware intercepts every incoming request to a protected endpoint and checks that a valid token is present before allowing the request to proceed. If no token is provided, or the token is invalid or expired, the request is rejected.

This approach to authentication is well-suited to this health-related application, as the data being stored is sensitive and personal.

6.1.3 API Routes and Controllers

Controllers were created for each table, with REST API routes defined in the api.php file. Each controller handles the relevant HTTP requests (GET, POST, PUT, DELETE) and returns JSON responses to the front-end. Protected routes are wrapped in 'auth:sanctum' middleware to ensure that unauthenticated requests are rejected. Below shows all API routes in the application.

```
// Public routes
Route::post('/register', [AuthController::class, 'register']);
Route::post('/login', [AuthController::class, 'login']);

// Protected routes
Route::middleware('auth:sanctum')->group(function () {

    // Auth
```

```
    Route::post('/logout', [AuthController::class, 'logout']);

    // Profile
    Route::get('/me', [ProfileController::class, 'show']);
    Route::put('/me', [ProfileController::class, 'update']);

    // Categories & Symptoms
    Route::apiResource('categories', CategoryController::class);
    Route::apiResource('symptoms', SymptomController::class);

    // Cycles
    Route::get('/cycles', [CycleController::class, 'index']);
    Route::get('/cycles/{cycle}', [CycleController::class, 'show']);

    // Periods
    Route::get('/periods', [PeriodController::class, 'index']);
    Route::post('/periods', [PeriodController::class, 'store']);
    Route::get('/periods/{period}', [PeriodController::class, 'show']);
    Route::put('/periods/{period}', [PeriodController::class, 'update']);
    Route::delete('/periods/{period}', [PeriodController::class, 'destroy']);

    // View all days for a period
    Route::get('/periods/{period}/days', [PeriodDayController::class, 'index']);

    // Add or update flow/clots for a specific date
    Route::put('/periods/{period}/days', [PeriodDayController::class, 'upsert']);

    // Daily Logs
    Route::get('/daily-logs', [DailyLogController::class, 'index']);
    Route::post('/daily-logs', [DailyLogController::class, 'store']);
    Route::get('/daily-logs/{dailyLog}', [DailyLogController::class, 'show']);
    Route::put('/daily-logs/{dailyLog}', [DailyLogController::class, 'update']);
          Route::delete('/daily-logs/{dailyLog}',    [DailyLogController::class,
'destroy']);

    // Daily Symptoms
                                  Route::post('/daily-logs/{dailyLog}/symptoms',
[DailySymptomController::class, 'store']);
                Route::delete('/daily-logs/{dailyLog}/symptoms/{dailySymptom}',
[DailySymptomController::class, 'destroy']);

    // Journal
    Route::get('/journals', [JournalController::class, 'index']);
       Route::get('/daily-logs/{dailyLog}/journal',   [JournalController::class,
'show']);   // single entry
       Route::put('/daily-logs/{dailyLog}/journal',   [JournalController::class,
'upsert']);

});
```

6.1.4 Testing and Iteration

The back-end was tested throughout development using Insomnia, an API testing tool that allows the sending of HTTP requests directly to the API endpoints and inspects the response without the need for a front-end. This was very useful in the early stages of development, as it made it much easier to identify and fix issues quickly. As shown in Figure 42, all the application's endpoints were organised into collections within Insomnia, grouped by each database table. Each endpoint was tested individually to verify that it returned the correct data, handled errors, and responded with the correct HTTP status codes.

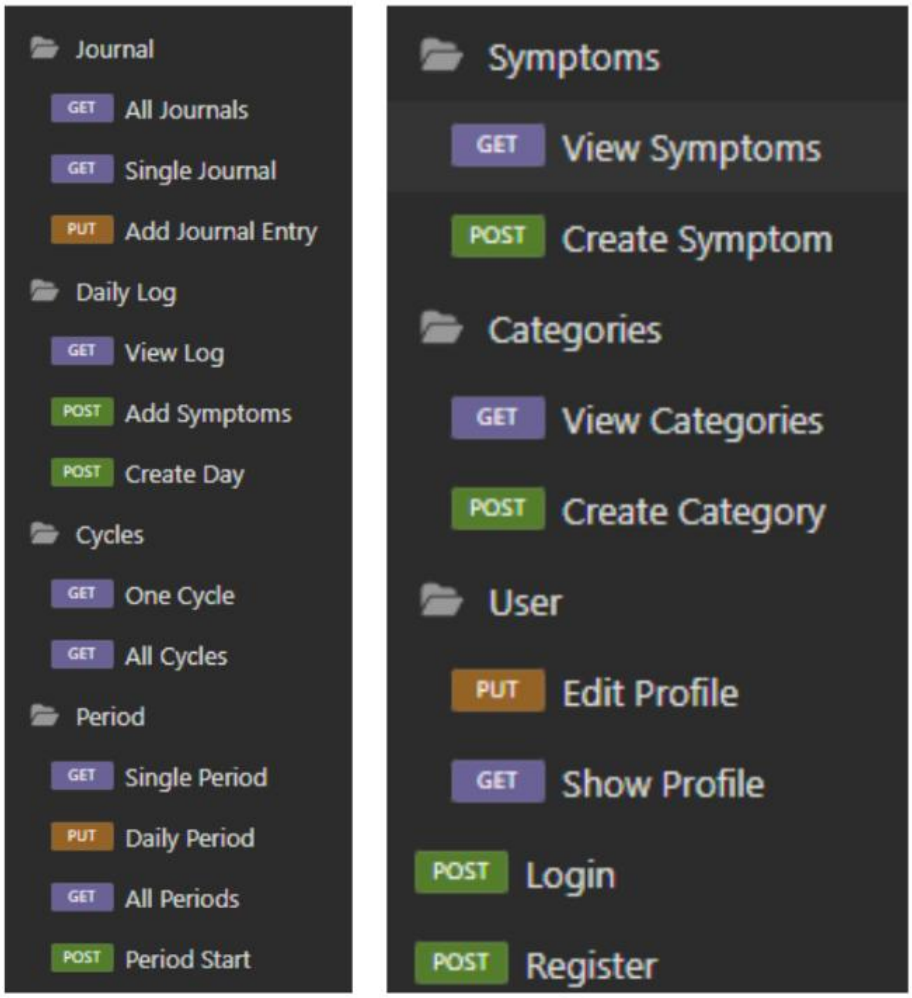


Figure 42 - Insomnia Testing

The back-end was initially kept simple and functional at this stage, with the focus on getting all endpoints working before introducing additional complexity. Once the front-end connection was established, a small amount of refinements were needed.

6.1.5 Deployment

Deploying the back-end was one of the more challenging parts of development, and it took a few attempts to find an approach. The initial plan was to use Laravel Cloud, as it's built specifically for Laravel applications. However, when importing the repository, Laravel Cloud gave an 'unsupported framework' error. The issue was that the Laravel project was inside a subfolder called 'server-api', since the repository was set up this way for Docker use.

Railway was chosen instead, as it allows a custom root directory to be set. After connecting the GitHub repository and setting the root directory to 'server-api', a series of build failures occurred. Railway defaulted to PHP 8.2, but the project required PHP 8.4. This was resolved by adding a 'nixpacks.toml' file. Railway was also detecting the Laravel Sail 'compose.yaml' and trying to use it as the build configuration, so a '. railwayignore' file was added to exclude it, and a custom 'Dockerfile' was created to take control of the build process.

The application kept crashing at runtime due to the empty database connection variables. A MySQL database service and its credentials were added. The 'pdo_mysql' PHP extension was also added to the 'Dockerfile', as it was required for Laravel to communicate with the MySQL. The 'Dockerfile' was also updated to run migrations and seeders automatically on each development environment.

6.2 Front-End Development

The application's front end was built with React Native, Expo, TypeScript, and NativeWind. It began with the initial project setup and hardcoding, then moved on to connecting the back-end to make everything functional. Please refer to Appendix E to see the full front-end code.

6.2.1 Initial Setup

Development started by setting up React Native with Expo and NativeWind, creating the basic login and registration pages, and building the navigation bar. Expo was chosen to simplify development and testing, but in the end, it didn't work properly, so a different route was chosen. NativeWind was configured to bring Tailwind CSS utility-class styling to the mobile environment.

A custom theme constraints file was then created to initially handle light and dark mode variants for the application; in the end, it was used for different colour themes. A global theme system was implemented to manage the application theme.

6.2.2 ThemeContext and Colour Themes

A custom ThemeContext was built using React's Context API. The Context API enables sharing values such as states, functions, and other data across the component tree without manually passing props at every level. Theme management is one of the most common use cases for the Context API, as storing theme state in a context allows any component to access and update it without having to pass props through multiple levels (Matéu.sh, 2024). This made it ideal for this application.

Rather than simple light/dark modes, the theme system was expanded to support six themes. Each theme is defined as a full-colour palette in the 'Colour' object, with the four additional colour themes generated by AI. Below is a screenshot from the app showing the themes:

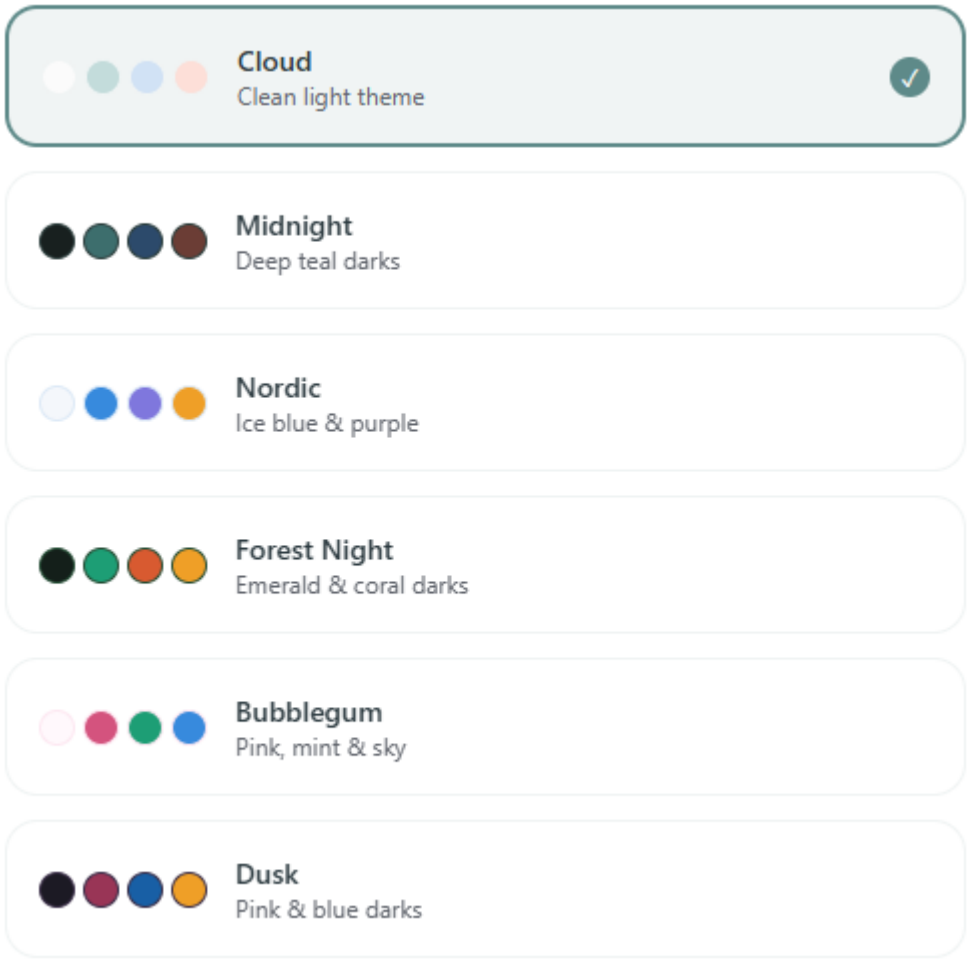


Figure 43 - Screenshot from the application showing the colour themes

The 'ThemeProvider' initialises with 'light' as the default. On startup, a 'useEffect' hook checks 'AsyncStorage' for a previously saved preference and applies it if one exists, overriding the default:

```
useEffect(() => {
  AsyncStorage.getItem("theme_preference").then((saved) => {
    if (saved && saved in require("../constants/theme").Colors) {
      setThemeName(saved as ThemeName);
    }
  });
}, []);
```

When the user selects a new theme, the 'setTheme' function updates the state and writes the chosen theme name to 'AsyncStorage' so it stays across sessions:

```
  function setTheme(name: ThemeName) {
    setThemeName(name);
    AsyncStorage.setItem("theme_preference", name);
  }
```

The context exposes two values ‘themeName’ and ‘setTheme’. Any page or component access these by calling the custom ‘useTheme’ hook:

```
  const { themeName, setTheme } = useTheme();
  const theme = Colors[themeName];
```

The previous appearance toggle switch was replaced with a button that opens a bottom-sheet modal. The currently active theme is highlighted with a border and a checkmark. Selecting a theme calls the ‘setTheme’ function, closes the modal, and the application re-renders immediately with the new theme.

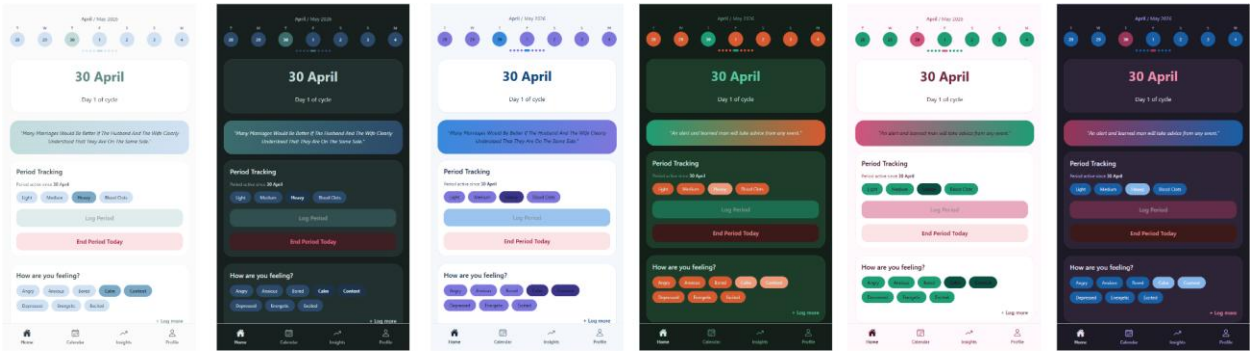


Figure 44 - The six available themes on the application

6.2.3 Axios Configuration

Rather than writing the base URL and headers on every individual API request, a centralised Axios configuration file was created. This file creates a single Axios instance with the back-end URL and default headers pre-configured:

```
const api = axios.create({
  baseURL: "https://majorprojectserver-production.up.railway.app/api",
  headers: {
    "Content-Type": "application/json",
    Accept: "application/json",
  },
});
```

The most important part of this setup is how each request is handled before being sent. Before every outgoing request is sent, it automatically reads the stored ‘auth_token’ from ‘AsyncStorage’ and attaches it to the Authorization header as a Bearer token:

```
api.interceptors.request.use(async (config) => {
  const token = await AsyncStorage.getItem("auth_token");
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});
export default api;
```

This means that once the user logs in, every API call is automatically authenticated without any additional code needed in each screen. Without this file, the base URL, headers, and token would need to be manually included with every request.

6.2.4 Core Components

Before connecting to the back-end, the core pages were first built by hardcoding placeholder data. This allowed the layout and structure of each page to be established, making it easier to spot design issues early on.

Once the page layouts were in place, the custom components were built. Rather than duplicating UI elements across multiple pages, components were created. This meant that a single, well-built component could be used anywhere in the application. If a component needed to look or behave differently depending on where it was used, props were used to pass different values and states rather than creating a separate component.

As shown in Figure 45, a range of components was created. These components form the layout of every page in the application.

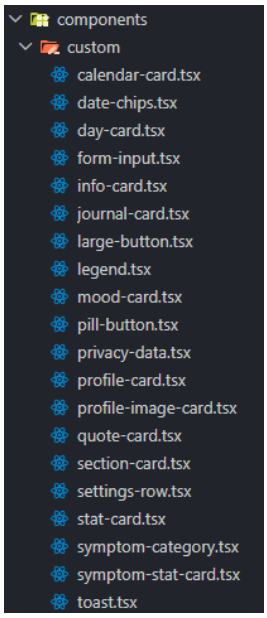


Figure 45 - Components in Front-End

6.2.5 Connecting to the Back-End

Once all the pages were hardcoded, the next phase involved connecting each page to the back-end API. Each page was connected one at a time, starting with the login and register pages, since authentication had to work before any other endpoint could be accessed. From there, the profile and profile edit pages were connected, followed by the journal page, and finally the home, calendar, and insights pages, which were the most complex to connect due to the volume of data they needed to fetch and display.

This incremental approach made it much easier to debug issues, since any problem that appeared after connecting a page could be traced back to that page's API calls rather than having to search for them.

6.2.6 Registration and Login

With the Axios setup in place, the login and registration pages were connected to the back-end. Both pages follow the same overall structure: a form with input fields, validations, error handling, and a submit form button. The registration pages collects three fields: name, email, and password, whereas the login page only requires email and password. A useMemo hook watches all input fields and only enables the submit button when none are empty, preventing incomplete form submission:

```
const canSubmit = useMemo(() => {
  return (
    name.trim().length > 0 &&
    email.trim().length > 0 &&
    password.trim().length > 0
  );
}, [name, email, password]);
```

The request is then sent to the endpoint, the returned token is stored in AsyncStorage, and the user is redirected to the home page:

```
    const response = await api.post("/register", {
      name,
      email,
      password,
    });

    const token = response.data.token;
    await AsyncStorage.setItem("auth_token", token);

    router.replace("/(tabs)/home");
```

A decision made on the login page was to keep the error messages vague. Rather than telling the user exactly that their email wasn't found in the database, or their password is wrong, a single generic message will appear. This protects against a technique called user enumeration, in which a malicious person uses brute-force methods to guess or confirm valid users in a system (Laverty, 2017). Since this application stores sensitive health data, this felt like an important

consideration. Keeping the error messages generic is a standard practice across most major applications.

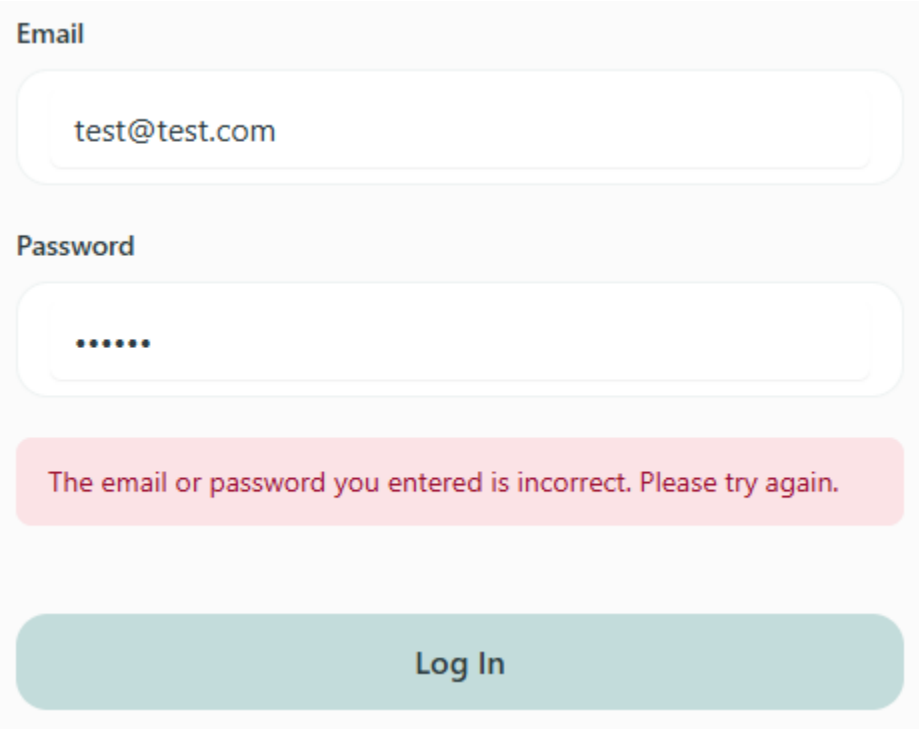


Figure 46 - Login error

The registration page handles errors differently; if Laravel's validation rejects a field, the error message appears beneath the relevant input field. This covers cases such as an email address already in use or a password that does not meet the length requirement.

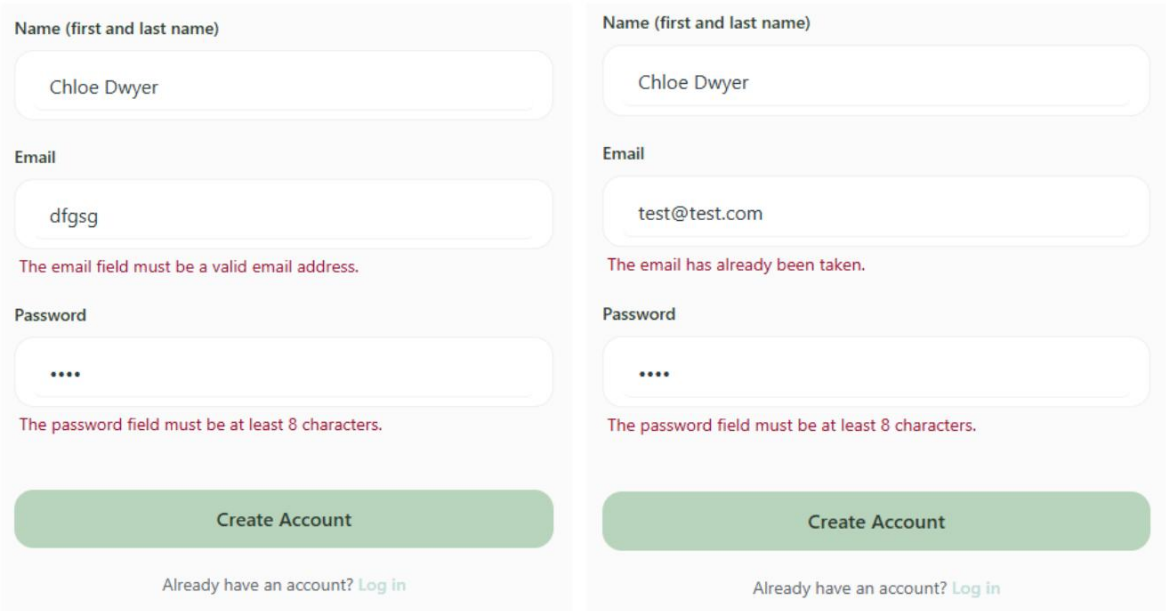


Figure 47 - Register errors

6.2.7 Calendar Page

The calendar page was one of the more complex screens to implement, as it required combining three separate endpoints and displaying them together. On load, it makes the API calls using a 'Promise.all', fetching cycles, periods, and daily logs simultaneously to reduce waiting time rather than fetching them sequentially.

```
async function fetchAllData() {
  try {
    await Promise.all([
      fetchCycles(),
      fetchCalendarData(),
      fetchAllDailyLogs(),
    ]);
  } catch (error) {
    console.error("Failed to fetch data:", error);
  }
}
```

Period dates are calculated by taking each period's start and end date and generating every date in between using a 'getDatesInRange' helper function. This produces an array of every date that should be highlighted on the calendar as a period day:

```
function getDatesInRange(startDate: string, endDate: string): string[] {
```

```
    const dates: string[] = [];
    const current = new Date(`${startDate}T00:00:00`);
    const end = new Date(`${endDate}T00:00:00`);
    while (current <= end) {
      dates.push(formatDateString(current));
      current.setDate(current.getDate() + 1);
    }
    return dates;
  }
```

Alongside this, each individual period is also fetched to retrieve its level logs: flow intensity and if a clot is present. These are stored in a 'periodLogsBydate' object keyed by a date string, so any date can be viewed when a user taps on it.

The useMemo hook is used throughout to avoid recalculating values on every render. The cycle day for the selected date is computed by finding the cycle whose range contains the selected date, then calculating the difference in days from the start day:

```
  const cycleDay = useMemo(() => {
    if (cycles.length === 0) return null;

    const cycle = cycles.find((c) => {
      const afterStart = displayDate >= c.start_date;
      const beforeEnd = c.end_date === null || displayDate <= c.end_date;
      return afterStart && beforeEnd;
    });

    if (!cycle) return null;

    const day = getDayDifference(cycle.start_date, displayDate) + 1;
    return day > 0 ? day : null;
  }, [cycles, displayDate]);
```

Sex and sex drive dates are also collected using UseMemo by filtering daily logs for any symptoms belonging to that category. The resulting array of dates is passed down to the calendar component to be marked with a heart on the calendar:

```
  const sexDates = useMemo(() => {
    return Object.entries(dailyLogsByDate)
      .filter(([_, log]) =>
        log.daily_symptoms?.some(
          (ds) => ds.symptom?.category?.name === SEX_CATEGORY,
        ),
      )
      .map(([date]) => date);
  }, [dailyLogsByDate]);
```

The calendar itself uses a custom ‘dayComponent’ to take full control of how each day cell renders, rather than relying on the default library styling. Each cell checks whether the date is today, selected, a period day, or a sex day, and applies the styling accordingly. Period days render with a droplet icon and sex days render with a small heart badge in the top right corner of the cell:

```
{isPeriodDay && (
  <Ionicons
    name="water"
    size={34}
    color={theme.accent}
    style={{ position: "absolute" }}
  />
)}
```

```
{isSexDay && (
  <View
    style={{
      position: "absolute",
      top: -3,
      right: -3,
      width: 14,
      height: 14,
      borderRadius: 7,
      backgroundColor: theme.backgroundElement,
      alignItems: "center",
      justifyContent: "center",
    }}
  >
    <Ionicons name="heart" size={10} color={theme.secondary} />
  </View>
)}
```

Figure 48 - Calendar component

A legend is then displayed under the calendar card to explain the colour coding used for today, period days, sex and sex drive days, using the application's theme colours so it adapts correctly to the custom themes.

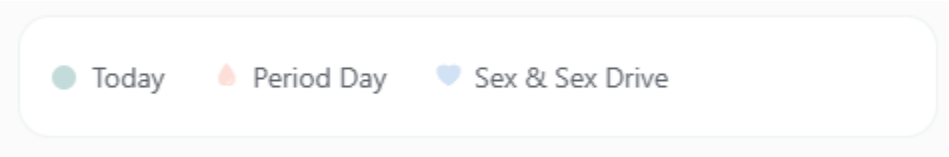


Figure 49 - Legend Component

When a user clicks on a date, the page updates to show an info card for that date. The subtitle displayed is calculated based on what data exists for that date, showing flow heaviness and whether clots were present if a period day log exists, a generic period message if the date falls within a period range, or a prompt to log more if nothing has been recorded:

```
function getPeriodSubtitle(): string {
  if (hasAnyData === false) {
    return "Start logging your cycle on the home page to see your data here.";
  }
  if (dateHasNoData) {
    return "Nothing logged for this date yet.";
  }
  if (selectedDayLog) {
    const { flow, has_clots } = selectedDayLog;
    const capitalised = flow.charAt(0).toUpperCase() + flow.slice(1);
    return has_clots
      ? `${capitalised} flow with blood clots`
      : `${capitalised} flow`;
  }
  if (isPeriodDate) return "Period recorded for this date";
  return "Nothing logged for this date yet.";
}
```

6.2.8 Insights Page

The insights page was built to give users a meaningful overview of their cycle data. On load, four separate API calls are made simultaneously to fetch cycle data, journal entries, daily logs, and a random inspirational quote from the DummyJSON public API (*Authentication API Documentation | DummyJSON*, 2026).

If the user has no information logged, a modal is shown explaining that insights will become available once they start logging data. This check happens inside the ‘fetchCycleStats’ and triggers immediately if the returned cycles array is empty, rather than returning an empty page with no context, which would be confusing for the user.

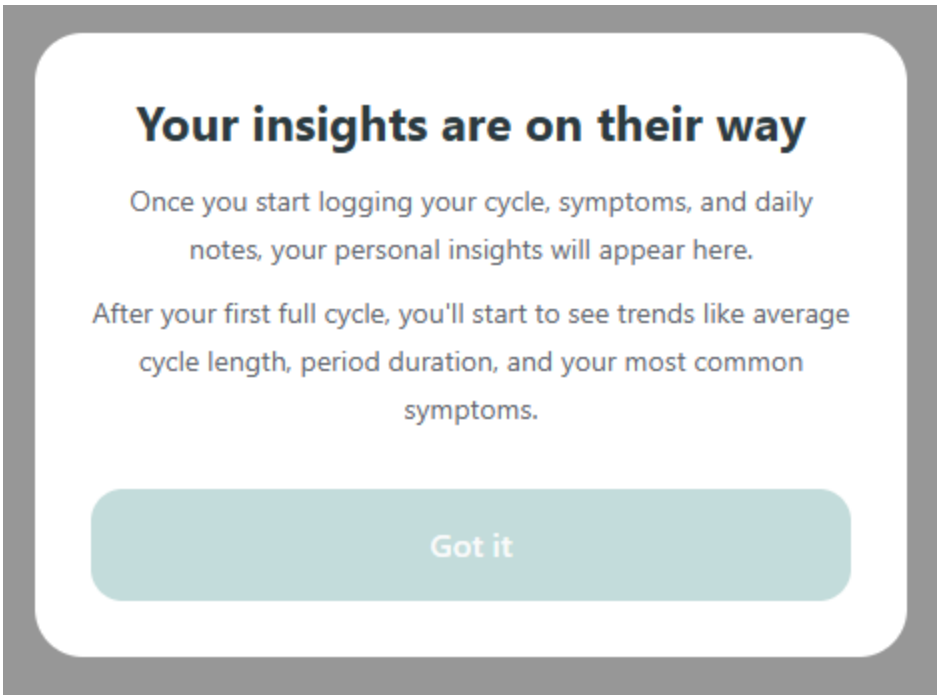


Figure 50 - Modal component

The average cycle length is calculated by filtering for cycles with recorded cycle lengths and averaging them. Period lengths are calculated by finding the difference in days between each period's start and end date. Both averages are then displayed in stat card components at the top of the page.

```
    for (const cycle of cycles) {
      for (const period of cycle.periods) {
        if (period.start_date && period.end_date) {
          const start = new Date(period.start_date);
          const end = new Date(period.end_date);
          const days =
            Math.round(
              (end.getTime() - start.getTime()) / (1000 * 60 * 60 * 24),
            ) + 1;
          if (days > 0) {
            periodLengths.push(days);
            periodChartPoints.push({
              value: days,
              label: start.toLocaleDateString("en-GB", { month: "short" }),
            });
          }
        }
      }
    }
```

```
    if (periodLengths.length > 0) {
      const total = periodLengths.reduce((sum, d) => sum + d, 0);
      setAvgPeriodLength((total / periodLengths.length).toFixed(1));
    }
```

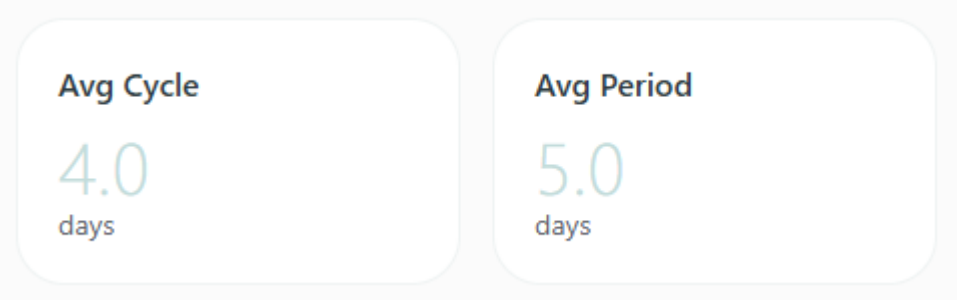


Figure 51 - Insights stats components

Three charts are built and displayed in a scrollable carousel: a cycle-length trend-line chart, a period-length trend-line chart, and a mood-over-time bar chart. Rather than rendering all three unconditionally, the charts are defined as an array of objects and filtered to include only those with data.

```
const charts = [
  {
    title: "Cycle Length Trend",
    subtitle: `Last ${cycleLengthData.length} cycles (days)`,
    data: cycleLengthData,
    type: "line" as const,
  },
  {
    title: "Period Length Trend",
    subtitle: `Last ${periodLengthData.length} periods (days)`,
    data: periodLengthData,
    type: "line" as const,
  },
  {
    title: "Mood Over Time",
    subtitle: "How you've been feeling",
    data: moodData,
    type: "bar" as const,
  },
].filter((c) => c.data.length > 0);
```

The carousel uses a horizontal ‘ScrolView’ with ‘pagingEnabled’ so it snaps between charts. A scroll event listener tracks the current index and updates the row of dot indicators beneath the carousel, with the active dot expanding in width to indicate the position:

```
function handleCarouselScroll(e: NativeSyntheticEvent<NativeScrollEvent>) {
```

```
    const index = Math.round(e.nativeEvent.contentOffset.x / SCREEN_WIDTH);
    setActiveChart(index);
  }
```

The charts are built using the 'react-native-gifted-charts' library (Abhinandan-Kushwaha, 2024). The line charts use a dynamic min and max value calculated from the data to ensure the chart scales correctly, regardless of the values:

```
  function getLineBounds(data: ChartPoint[]) {
    const values = data.map((d) => d.value);
    return {
      max: Math.max(...values) + 4,
      min: Math.max(0, Math.min(...values) - 4),
    };
  }
```

The mood chart maps journal entry feelings to a numeric score from 1 to 5 and plots them over time, giving users a visual representation of their mood throughout the month. The bar chart colours change based on the mood score, green for positive, blue for neutral, and pink for low moods. This gives an immediate visual read on how the users have been feeling.

```
                              frontColor:
                                d.value >= 4
                                  ? theme.primary
                                  : d.value === 3
                                    ? theme.secondary
                                    : theme.accent,
```

Finally, the most logged symptoms are calculated by iterating over all the daily logs and building a count map of how many times each symptom name appears. The results are sorted in descending order, and the top three are displayed on the page. A proportional bar width is then calculated relative to the most frequently logged symptom:

```
      const maxCount = sorted[0][1];
      const items: SymptomItem[] = sorted.map(([name, count], index) => ({
        rank: index + 1,
        name,
        count,
        width: `${Math.round((count / maxCount) * 100)}%`,
      }));
```

Then it is displayed in the SymptomsCard component as a ranked card with a colour-coded badge. Each rank is assigned a different colour, allowing users to easily visually read the data. The count is shown as a simple multiplier alongside the symptom name, keeping the card compact while still conveying frequency at a glance.

6.2.9 Home Page

The home page is the central hub of the application and one of its most feature-rich pages. It brings together period tracking, symptom logging, previous journal previews, cycle day tracking, and a random daily quote, all in a scrollable view.

At the top of the page, a scrollable date chip row is generated dynamically, creating a 63-day window centered around today. The array is built by offsetting from the current date, starting 30 days in the past and ending 32 days ahead:

```
const days = Array.from({ length: 63 }, (_, index) => {
  const date = new Date(today);
  date.setDate(today.getDate() - 30 + index);
  return {
    day: date.getDate().toString(),
    fullDate: date,
    active: formatDate(date) === selectedDate,
  };
});
```

The ‘DateChips’ component breaks this array into weeks and renders as paginated pages inside a horizontal ‘ScrollView’. Dot indicators beneath the chip track, which week is currently visible, with the active dot expanding in width, the same as the chart carousel. The component is built using ‘forwardRef’ and ‘useImperativeHandle’ to expose a ‘scrollToToday’ method, which allows the parent page to snap the scroll position back to the current week when the user clicks the ‘Today’ banner:

```
  useImperativeHandle(ref, () => ({
    scrollToToday() {
      const index = todayWeekIndex !== -1 ? todayWeekIndex : totalWeeks - 1;
      scrollRef.current?.scrollTo({
        x: index * SCREEN_WIDTH,
        animated: true,
      });
      setCurrentWeek(index);
    },
  }));
```

When a user is viewing a past or future date, a banner appears below the date chips showing the currently selected date and a button to return to today. This is conditionally rendered based on whether ‘selectedDate’ matches ‘todatString’.

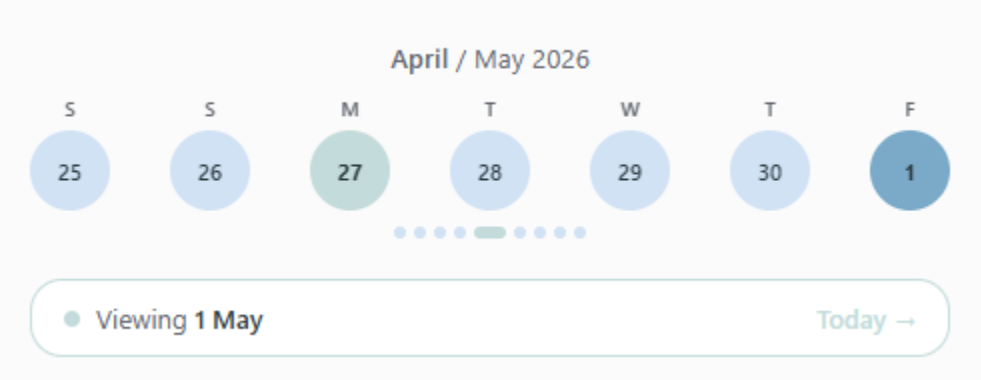


Figure 52 - Day Chips Component

Every time the selected date changes, the page fetches or creates a daily log for that date. The 'fetchOrCreateLogForDate' function sends a POST request to the '/daily-logs' endpoint, and if a log already exists, the back-end returns the existing one, and if not, a new one is created:

```
async function fetchOrCreateLogForDate(date: string): Promise<number | null> {
  try {
    const response = await api.post("/daily-logs", { date });
    const logId = response.data.id;
    setTodayLogId(logId);
    return logId;
  } catch (error: any) {
    const message = error.response?.data?.message;
    Alert.alert("Error", message ?? "Could not create log for this date.");
    return null;
  }
}
```

Once the log ID is retrieved, the full log is fetched to populate the journal preview and any previously saved symptoms. The journal entry is limited to 70 characters for the preview, so it doesn't take up too much space on the home page. The 'useFocusEffect' hook is also used alongside the date change effect to re-fetch the journal preview every time the screen comes back into focus, ensuring the preview always shows the up-to-date journal entry.

The current cycle day is calculated in a 'useEffect' that runs whenever the selected date or cycles array changes. It finds the cycle whose date range contains the selected date and calculates the difference in days from its start date:

```
    const day =
      Math.floor((end.getTime() - start.getTime()) / (1000 * 60 * 60 * 24)) + 1;
```

```
    setCycleDay(day > 0 ? day : null);
```

The symptom logging section shows the first eight symptoms from the Mood category as quick-tap pill buttons for fast logging. The “Log More” link opens a bottom-sheet modal that displays all symptoms grouped by category. Symptoms are toggled on and off using a toggle function that either adds or removes the symptom ID from the selected array:

```
function toggleSymptom(symptomId: number) {
  setSelectedSymptomIds((prev) =>
    prev.includes(symptomId)
      ? prev.filter((id) => id !== symptomId)
      : [...prev, symptomId],
  );
}
```

When the user saves, a POST request is sent to ‘/daily-logs/{id}/symptoms’ with all selected symptom IDs. Saves symptoms are then displayed as pill tags on the home page, each with a small icon sourced from the ‘CATEGORY_ICONS’ map that matches the category. Then the component changes to show all the saved symptoms that the user has logged for easy viewing of logged symptoms.

Figure 53 - Symptom card showing selected symptoms

Condition-specific tracking was addressed through the symptom logging system. Rather than building separate condition-specific pages, they symptom categories were designed to cover the symptoms most commonly associated with these conditions. Users with these conditions can log and track their relevant symptoms through the same interface as other users, keeping the application inclusive without creating a separate experience for those with menstrual health conditions.

The period tracking section allows users to log their period for the selected date. A ‘flowMap’ object maps the user-facing options to the values expected by the back-end:

```
const flowMap: Record<string, { flow: string; has_clots: boolean }> = {
  Light: { flow: "light", has_clots: false },
  Medium: { flow: "medium", has_clots: false },
  Heavy: { flow: "heavy", has_clots: false },
  "Blood Clots": { flow: "heavy", has_clots: true },
};
```

When a user logs their period, the application checks whether there is an active period. If there is, the existing period ID is reused. If not, a new period is created with a POST to ‘/periods’ before the day is logged. If a period is currently active, an “End Period Today” button appears, allowing users to end their period on a specific date. The logged flow option for the selected date is also tracked in a ‘periodLogsBtDate’ lookup object, so the correct pill visually appears as already logged when the user navigates back to a previously logged date.

6.2.10 Data Export and PDF Report Generator

This feature was implemented in a dedicated ‘generateReport.ts’ utility file and is accessible from the profile page. When the export is triggered through the profile page, four API calls are made to gather all the data needed for the report. This fetches the user’s information, full cycle history, all daily logs, and all journal entries, ensuring the report contains all the user’s health data.

The report is generated as an HTML string using a ‘buildHtml’ function that takes the fetched data and formats it into a structured, print-ready document. Claude was used to generate the HTML code by telling it exactly what data to display. The HTML includes several sections: user details, a cycle summary with key statistics, a full cycle log table, a list of the most logged symptoms, and a mood summary. Many helper functions are used to format the data cleanly for the report.

The export method had to handle two environments, web and native mobile, because the application was tested in the web environment during development. The initial approach of using ‘expo-print’ and ‘expo-sharing’ alone wouldn’t work in the web environment, as these are

native-only libraries. To fix this, a platform check was added so that the correct export method was used depending on where the application was running:

```
if (Platform.OS === "web") {
  const blob = new Blob([html], { type: "text/html" });
  const url = URL.createObjectURL(blob);
  const a = document.createElement("a");
  a.href = url;
  a.target = "_blank";
  a.click();
  URL.revokeObjectURL(url);
} else {
  const { uri } = await Print.printToFileAsync({ html });
  await Sharing.shareAsync(uri, {
    mimeType: "application/pdf",
    dialogTitle: "Share Cycle Report",
    UTI: "com.adobe.pdf",
  });
}
```

On the web, a generated HTML report opens in a new browser tab, where the user can use the built-in print functionality to save it as a PDF. A print hint banner is included at the top of the report page to guide users through the process and improve usability. On native mobile, ‘expo-print’ converts HTML to a PDF and ‘expo-sharing’ opens the native sharing sheet, allowing the user to save or send the report.

While this feature works as intended on the hosted web version, it does not function correctly on the mobile version. This is because the application is deployed as a web app, rather than a native application, due to issues with Expo Go.

On the profile page, the export is triggered through the settings row. A loading state is used to disable the button and update the label while the report is generated, preventing the user from triggering the export more than once:

```
async function handleExportReport() {
  setExportingReport(true);
  try {
    await generateAndShareReport();
  } catch (error) {
    console.error("Failed to export report:", error);
    Alert.alert(
      "Export Failed",
      "Something went wrong generating your report. Please try again.",
```

```
      );
    } finally {
      setExportingReport(false);
    }
  }
            <SettingsRow
              title={exportingReport ? "Generating Report..." : "Export Data"}
              subtitle="Download a PDF report for your doctor"
              icon={exportingReport ? "hourglass-outline" : "download-outline"}
              type="link"
              onPress={exportingReport ? undefined : handleExportReport}
            />
```

To save as PDF: use **File → Print → Save as PDF** in your browser, or click the button. Save as PDF

CYCLE HEALTH REPORT

Date of Report: 28 April 2026 · For Medical Reference Only

PATIENT DETAILS

Name: Test Test
DOB: 26/02/2004
Age: 22 years
Weight: 80 kg
Height: 170 cm

CYCLE SUMMARY

CYCLES TRACKED	AVG CYCLE	AVG PERIOD	SHORTEST	LONGEST
2	4.0 days	5.0 days	4 days	4 days

Regularity: Insufficient data

CYCLE LOG

#	CYCLE START	CYCLE END	CYCLE LENGTH	PERIOD START	PERIOD END	PERIOD DURATION
1	25/04/2026	N/A	Ongoing	25/04/2026	N/A	Ongoing
2	21/04/2026	24/04/2026	4 days	21/04/2026	25/04/2026	5 days

MOST LOGGED SYMPTOMS

1. Excited	2×
2. Cramps	2×
3. Normal	2×
4. Neutral Libido	1×
5. Dizziness	1×
6. Hot Flushes	1×
7. Itching	1×
8. Sour	1×
9. Unprotected	1×
10. Cheese	1×

MOOD SUMMARY

Great	1×
Good	1×
Okay	1×
Low	1×
Awful	1×

Figure 54 - Data PDF

6.2.11 Deployment

Deploying the front-end required an approach different from the one originally planned. Expo Go was not functioning correctly, so the application was run and tested using the ‘npm run web’ command, which serves it as a web app in the browser.

Vercel was chosen due to its straightforward GitHub integration and support for React-based web applications. After connecting the repository, the root directory was set, and the build command was set to ‘npx expo export –platform web’, which compiles the application into static files for the web.

After initial deployment, two issues had to be resolved. First, the API base URL was still pointing to the local development of the backend and needed to be updated to use the hosted version. Second, navigating to any route was returning a 404 error, which was fixed by adding a ‘vercel.json’ rewrite rule to handle client-side routing correctly. Refer to Appendix H to view the deployed front-end.

6.3 Challenges and Solutions

Throughout the development of this application, several technical challenges were encountered. This section documents the most significant issues and how they were resolved.

6.3.1 CORS Error

One of the most prominent issues encountered during development was a Cross-Origin Resource Sharing (CORS) error. CORS is a security mechanism built into browsers that blocks cross-origin requests unless the server permits them. Because the application was being developed and tested on the web rather than a physical device, the front-end and back-end were running on different addresses, which caused the browser to block API requests. This was resolved by configuring CORS on the back-end to allow requests from the front-end.

The initial attempt to resolve this issue was to configure CORS on the back-end by updating the CORS configuration file to allow requests from the front-end. However, this didn’t work as

expected. The second attempt was to change the URL in the back-end .env file. Once the address was corrected, the front-end and back-end could communicate properly, and development could continue.

6.3.2 Profile Image Upload

A second CORS-related error was encountered when attempting to implement profile image uploading. The intention was to allow users to upload a photo from their device, store it, and use it as the profile image. This functionality worked in the back end during testing, but due to the complexities of handling file uploads across the back end and front end, this approach kept running into issues that were difficult to resolve within the timeframe given.

Instead of trying to find a solution in the short time left, the decision was made to take a different approach and integrate an avatar API. This allowed users to still personalise their profile by selected and image from a set of generated avatars. The Loreli avatar collection, created by Lisa Wischofsky (*DiceBear*, 2026), was chosen. The selected avatar URL is stored locally using ‘AsyncStorage’, linked to the user’s ID so that each user’s avatar persists across sessions.

6.3.3 Period Logging Issue

During the design phase, an initial ERD was created to depict the database structure, as shown in Figure 55. The Period table included ‘flow_level’ and ‘has_clots’ fields directly on the period, alongside the start and end dates. This worked as an initial design, but it became clear during development that storing flow information at this level was not going to work, a user’s flow changed from day to day throughout a period, so storing a single flow value for the entire period would not accurately reflect this.

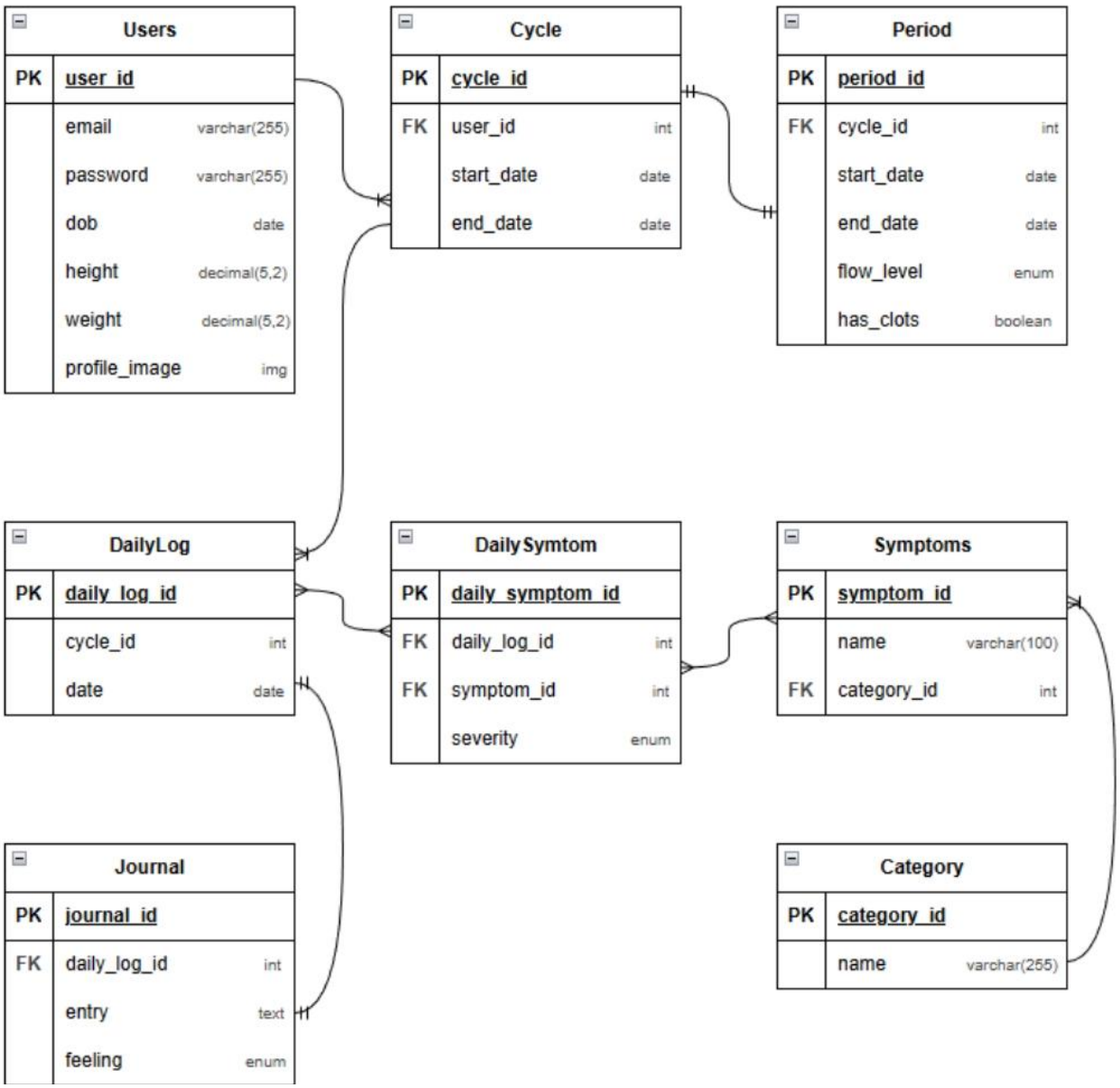


Figure 55 - Initial ERD

A new table was made to move these fields out of the period table and into a new table called ‘period_days’. Rather than storing one flow level per period, the updated structure allows a separate record to be stored for each day of the period, which also has a ‘has_clots’ boolean.

This change required reverting the ERD, creating a new migration for the ‘period_days’ table, and updating the period model to include a hasMany relationship with the period_day model and a belongsTo relationship from the period_day model to the period model. Once this change was made, the periods worked as expected.

7.0 Testing & Evaluation

7.1 End-to-End Testing

End-to-End (E2E) testing is a testing methodology that simulates real user interactions with an application from start to finish. Rather than testing individual components, E2E testing verifies that entire user flows work correctly. The goal is to replicate what a real user would do and confirm the application behaves as expected (Powell & Smalley, 2025).

7.1.1 Tools Used

Cypress, an open-source browser-based testing framework, was used for E2E testing. It was chosen because it integrates well with this application and provides a visual test runner that displays each test step in real time, making it easy to observe and debug (Cypress, n.d.). Tests were written in JavaScript spec files, with each file covering a different section of the application.

7.1.2 How tests were carried out

Five spec files were created, each targeting a distinct area of the application:

- ‘register.cy.js’: this tested the registration flow, verifying that the form cannot be submitted when fields are empty, and that a new user can successfully create an account and be redirected to the home page.

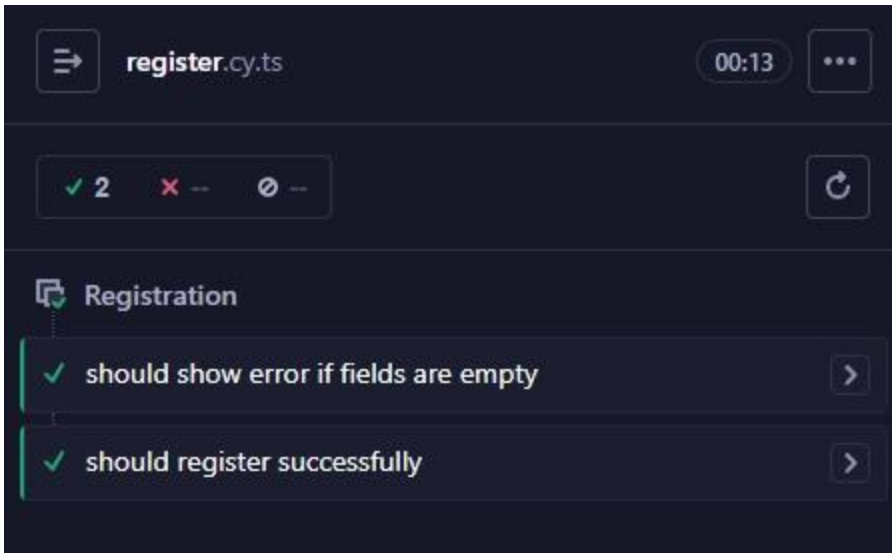


Figure 56 - Registration test results

- 'login.cy.js': this tested the authentication flow, covering successful login with valid credentials, incorrect credentials, and successful logout from the profile page.

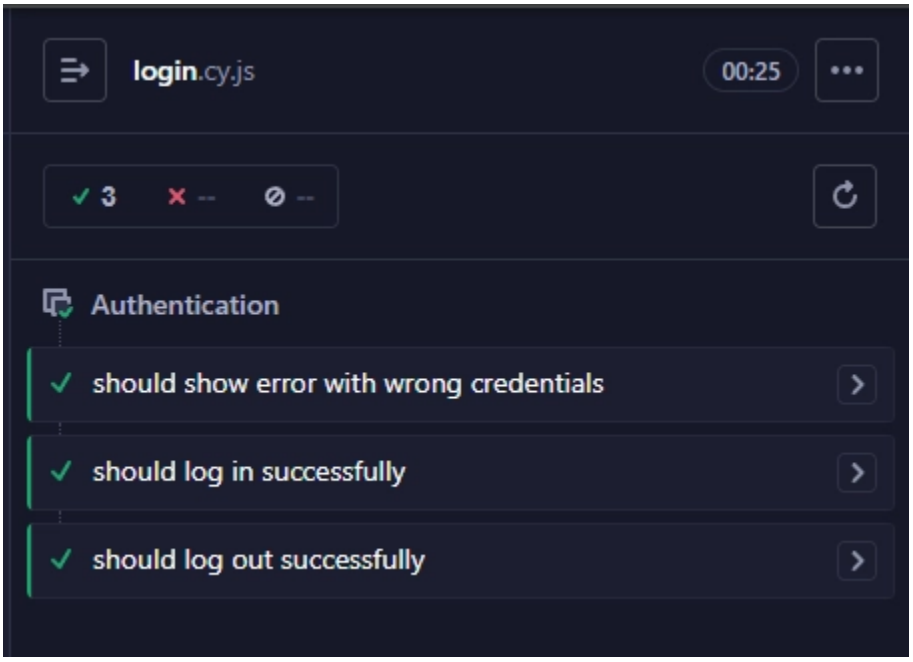


Figure 57 - Login test results

- 'home.cy.js': this tested the core functionality of the home page, including page load verification, date navigation using the date chips, logging a period, opening the

symptoms modal and logging three symptoms, and finally opening the journal and creating a journal entry.

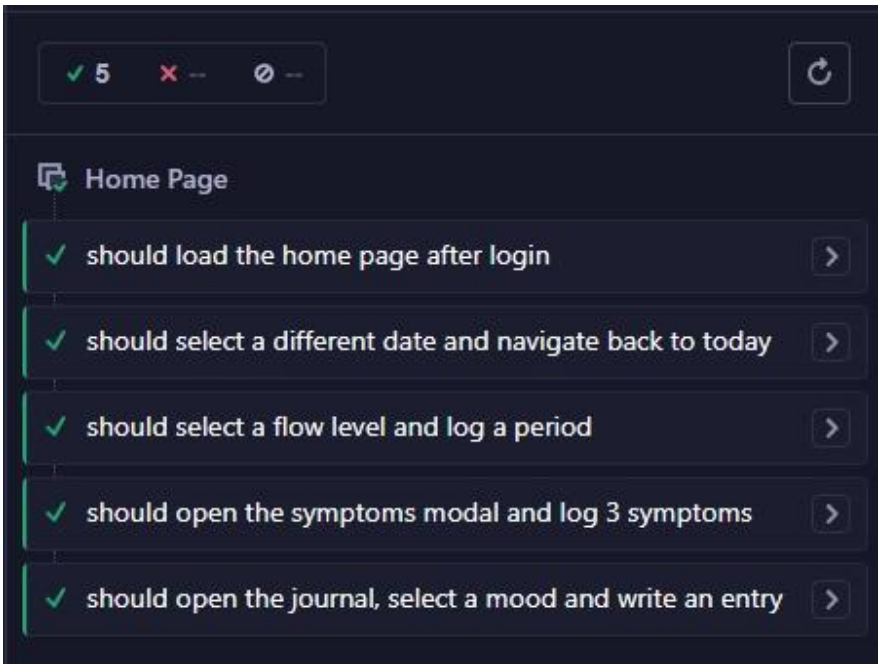


Figure 58 - Home page test results

- 'calendar.cy.js': this tested the calendar page, confirming that the page loads correctly with the legend and cycle information, that clicking the date updates the info card, that period data is shown for logged dates, and that future dates with no data display the appropriate message.

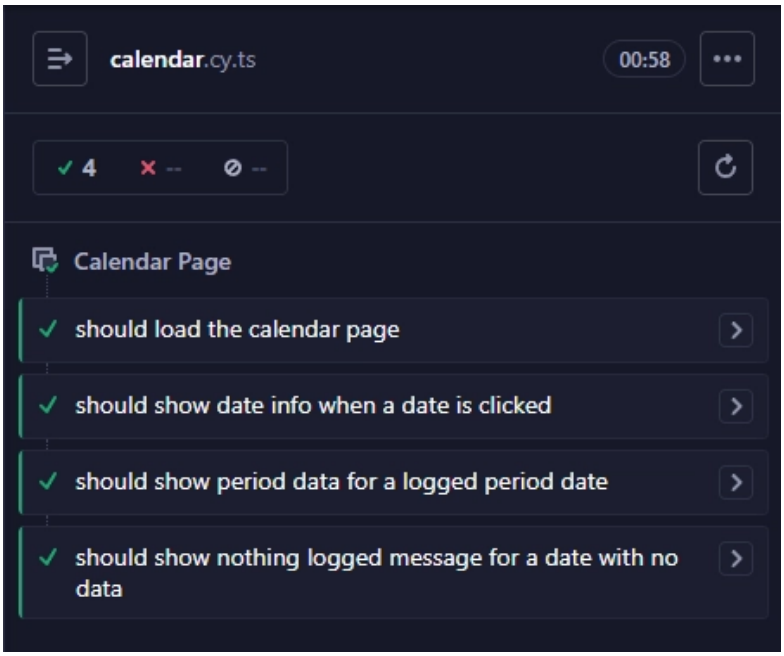


Figure 59 - Calendar page test results

- 'profile.cy.js': this tested the profile page, covering user information displaying, the appearance toggle for switching between light and dark mode, editing profile details, confirming the save with a success toast, selecting an avatar from the avatar picker and logging out.

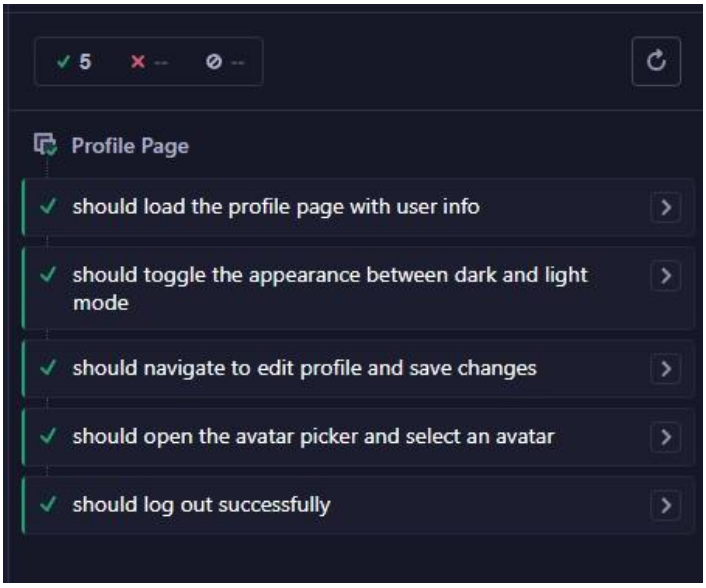


Figure 60 - Profile page test results

7.2 Test Plan for User Testing

User testing was conducted during the Deliver phase of the Double Diamond Methodology to evaluate the application's usability with real users. Unmoderated usability testing is a form of qualitative research in which users complete predetermined activities using a design or interface (*UserTesting Quick Start Guide – Knowledge Base Home*, 2026). This format was chosen because it allows participants to complete tasks at their own pace, resulting in more natural and honest feedback. The decision to conduct multiple rounds of user testing rather than a single session aligns with UCD best practices. Testing iteratively meant that the application was continuously improved based on real feedback rather than assumptions. This approach ensured that by the final round of testing, the majority of bugs and usability issues were already identified and resolved, resulting in a much more polished and reliable user experience for later participants.

7.2.1 Core Objectives

The test was designed around three core objectives:

- Can users successfully log the start of a period without guidance or instruction?
- Can users log a symptom or mood entry and find it again afterwards?
- Can users understand how to navigate between past and future months?

7.2.2 Test Tasks

The test consisted of nine tasks conveying the key areas of the application. Participants began with a first impression task, to explore the application freely for one minute and sharing their thoughts aloud. From there, participants were asked to log a period, log a symptom or mood, interact with the calendar view, add a journal entry, and navigate to and update their profile. Each task was followed by a short follow-up question asking them to reflect on their experience, whether anything was confusing, whether the options matched their expectations, and if anything

was missing. The session closed with an overall usability rating out of five and an open question about how Mira compares to how they currently track their cycle.

7.2.3 Closing Survey

At the end of the testing, participants were asked to complete a short post-survey to gather any additional feedback. The combination of task-based observations and survey responses was designed to provide both qualitative insights into user experience and quantitative data on overall satisfaction.

7.3 User Testing Results

Overall, the testing sessions produced very positive feedback, with the majority of participants rating the application as extremely easy to use and indicating they would be very likely to download it if the application were available today. This is strong validation of the user-centered design approach taken throughout the project, showing that when design decisions are driven by real users from the very start, the end result is what users actually want. Refer to Appendix I to view the full results in the post-testing survey.

The results from the sessions are discussed in the following sections, with findings organised by theme to provide a clearer and more cohesive picture of the overall user experience.

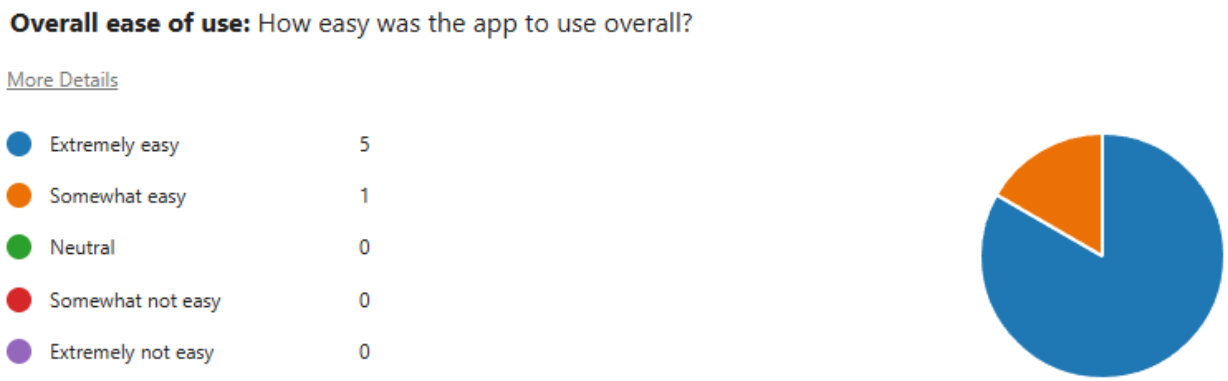


Figure 61 - Survey results showing the overall ease of use rating

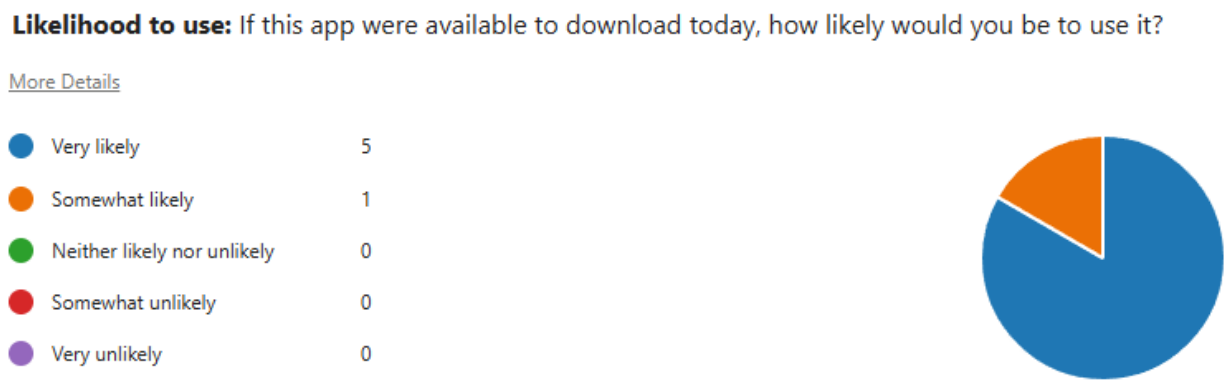


Figure 62 - Survey results showing the likelihood of participants downloading the application

7.3.1 Ease of Use and First Impression

The application as consistently rated as extremely easy to use across all sessions. When asked about their first impressions, participants said the application's purpose was either mostly clear or very clear, suggesting that the overall concept and purpose is communicated well from the beginning. This aligns with one of the core principles of UCD, that the product should be immediately understandable to the people it is designed for, without requiring instruction or explanation.

In the first testing session, the participant was able to identify the purpose of the application within the first few seconds of exploration, correctly describing it as a period tracking tool. However, they noted some initial uncertainty about which elements on the screen were interactive. This feedback was taken on board, and the contrast and styling of interactive elements was refined ahead of the second round of testing.

By the second and third round of sessions, participants navigated the application with more confidence from the beginning, with first impressions described as very clear. This improvement reflected the iterative refinements made between each session and demonstrated the value of testing in multiple rounds rather than relying on a single session.

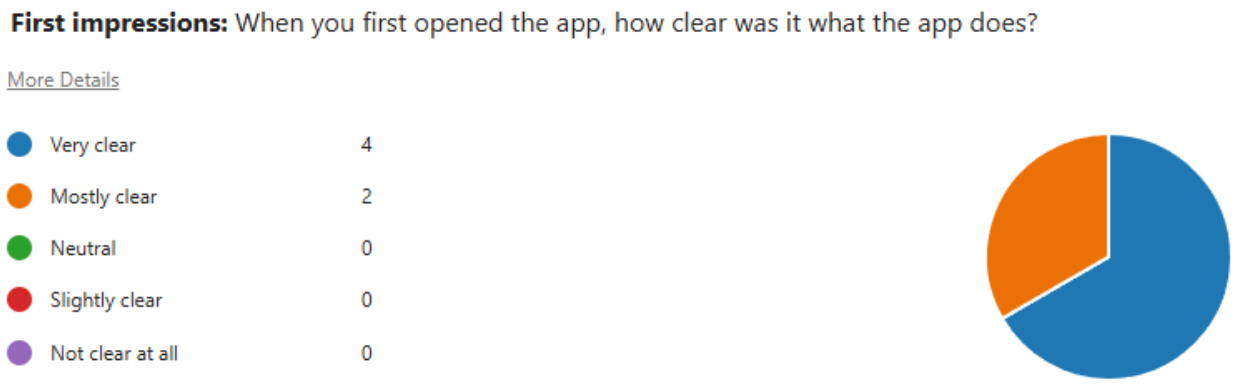


Figure 63 - Survey results showing how clearly participants understood the application on first impression

7.3.2 Core Features

Period logging, symptom logging, and calendar viewing were the most consistently completed tasks across all sessions. Journal entries and profile updating were also completed successfully. However, a number of usability issues and suggestions emerged across the sessions that led to meaningful improvements.

In the first session, the participant noted difficulty finding where to log a period, this highlights the importance of making the primary logging function as prominent as possible. As a result of this feedback, the home page layout was updated to move period tracking above symptom logging, ensuring the most important feature was the first thing users saw.

During the second session, a further issue was identified with the pill buttons, the participant found it difficult to tell which symptoms had already been selected. The selected state was updated to use a darker, more distinct colour to make it more clear which symptoms have already been chosen. In the same session, the participant also noted that when browsing past or future dates on the home page using the day chips, there was no quick way to get back to today's date. A 'Back to Today' button was added based on this comment.

In the third session, the participant noted a similar navigation issue on the calendar page, a 'Back to Today' button was also added to return to the current day and month. The participant also suggested that the day chips on the home page would benefit from showing the month above the chips so users can tell which month they are viewing whilst scrolling, this was also implemented.

7.3.3 Design and Appearance

The visual design of the application was rated highly throughout all testing sessions, with an average rating of 4.83 out of 5. Participant praised the minimalist design and the overall appearance. The insight page and its charts was highlighted as a particular strength, with one participant describing it as “Very well done”. The positive feedback of the visual design validated the decisions made during the design and development stages, especially the choice to move away from an overly pink, stereotypical feminine aesthetic which was directly informed by the user research findings.

In the first session, it was noted that the calendars colour coding was not immediately intuitive without referring to the legend. It was suggested that the period days should be marked with a droplet icon rather than relying on colour alone. This was added in response to the feedback. During the second session, a further suggestion was made to add a small heart icon to the days where sexual activity was logged. This gave users a discreet and recognisable visual indicator within their cycle overview and made the calendar more informative at a glance. This was implemented ahead of the third session and received positive feedback.

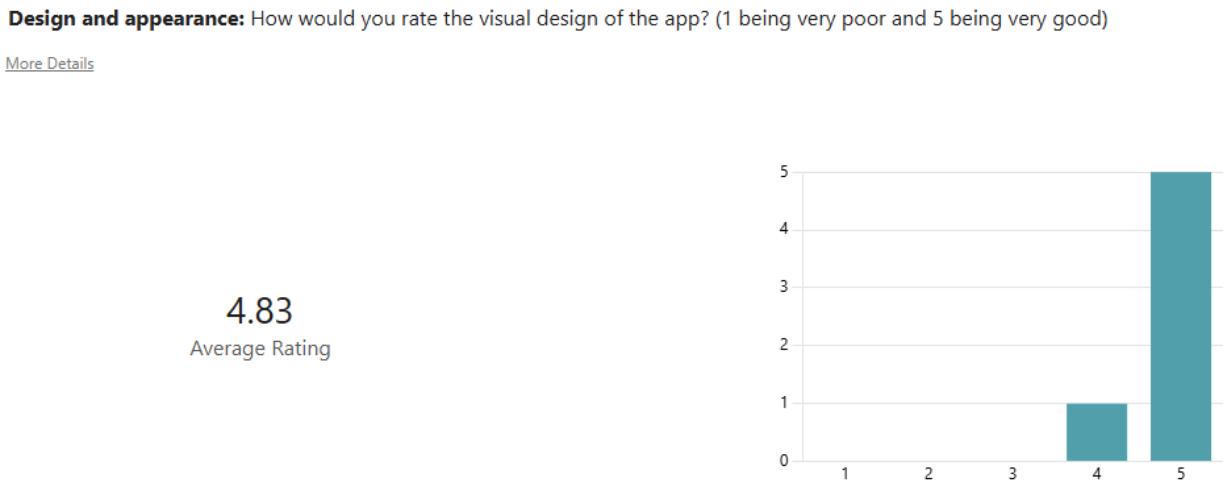


Figure 64 - Survey results showing ratings of the applications' visual design and appearance

7.3.4 Overall Feedback

The most consistent piece of feedback across all sessions was positive; participants appreciated how clean, simple, and easy to navigate the application was. One participant compared it to their

current tracking application. Flo, noting the layout is much better. The fact that this was raised unprompted during testing suggests that the design decisions made in response to early user research successfully addressed user needs.

The iterative nature of this testing process was key to the application's outcome. Each session provided new feedback that led to great improvement. The application that participants experienced in the final round of testing was noticeably more polished and refined than the initial application; this is a result of listening to users and acting on what they suggested.

7.4 WCAG Testing

To evaluate the application against WCAG 2.1 AA standards, accessibility testing was conducted using two tools: the axe DevTools extension and Google Lighthouse.

The axe DevTools scan identified four categories of issues. Colour contrast failures were the most common, appearing across all four pages. These were fixed by darkening the active tab colour, updating the date header to use a darker shade of the primary colour, deepening the danger button colour, adjusting the ‘InfoCard’ subtitle colour, and redesigning the BMI badge. A missing document title was resolved by adding a title field to ‘app.json’. Missing ARIA attributes within the react-native-calendar library could not be fully resolved.

Following these fixes, a Lighthouse audit returned the following accessibility scores:

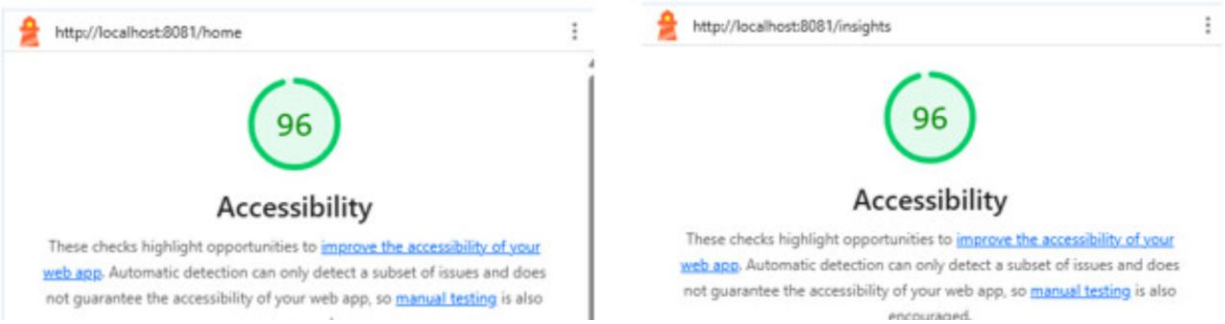


Figure 65 - Home and insights page results

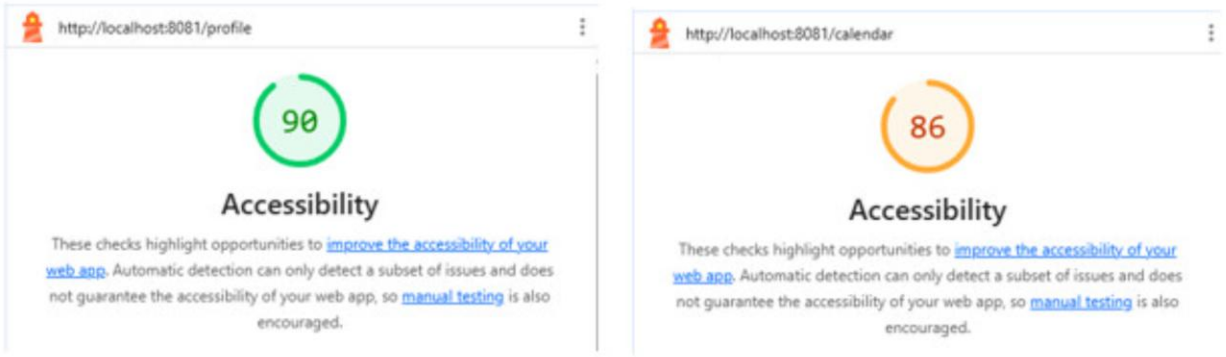


Figure 66 - Profile and calendar page results

Three pages scored 90 or higher, with the calendar page scoring lower due to limitations in the react-native-calendar library. Overall, these results demonstrate strong WCAG 2.1 AA compliance across the application.

7.5 Performance Testing

Performance testing was carried out using Google Lighthouse, auditing the home page under slow 4G throttling on an emulated Moto G Power device to replicate realistic conditions. The application scored 53 out of 100 for mobile performance. Claude was used to help interpret the Lighthouse results, explaining what each metric meant and why certain scores were lower, which helped the understanding of the results. Refer to Appendix J to see this conversation.

The Largest Contentful Paint and Total Blocking Time are the main reasons for the lower score. This was expected, as the home page fires off several API calls on load to pull cycle data, daily logs, and the daily quote. The back-end is hosted on Railway's free tier, which introduces cold-start delays when the server has been sitting idle. The Cumulative Layout Shift score is 0.045, which is strong, indicating the layout is stable once it loads and nothing jumps around on the screen. This matters a lot for day-to-day usability rather than raw load speed.

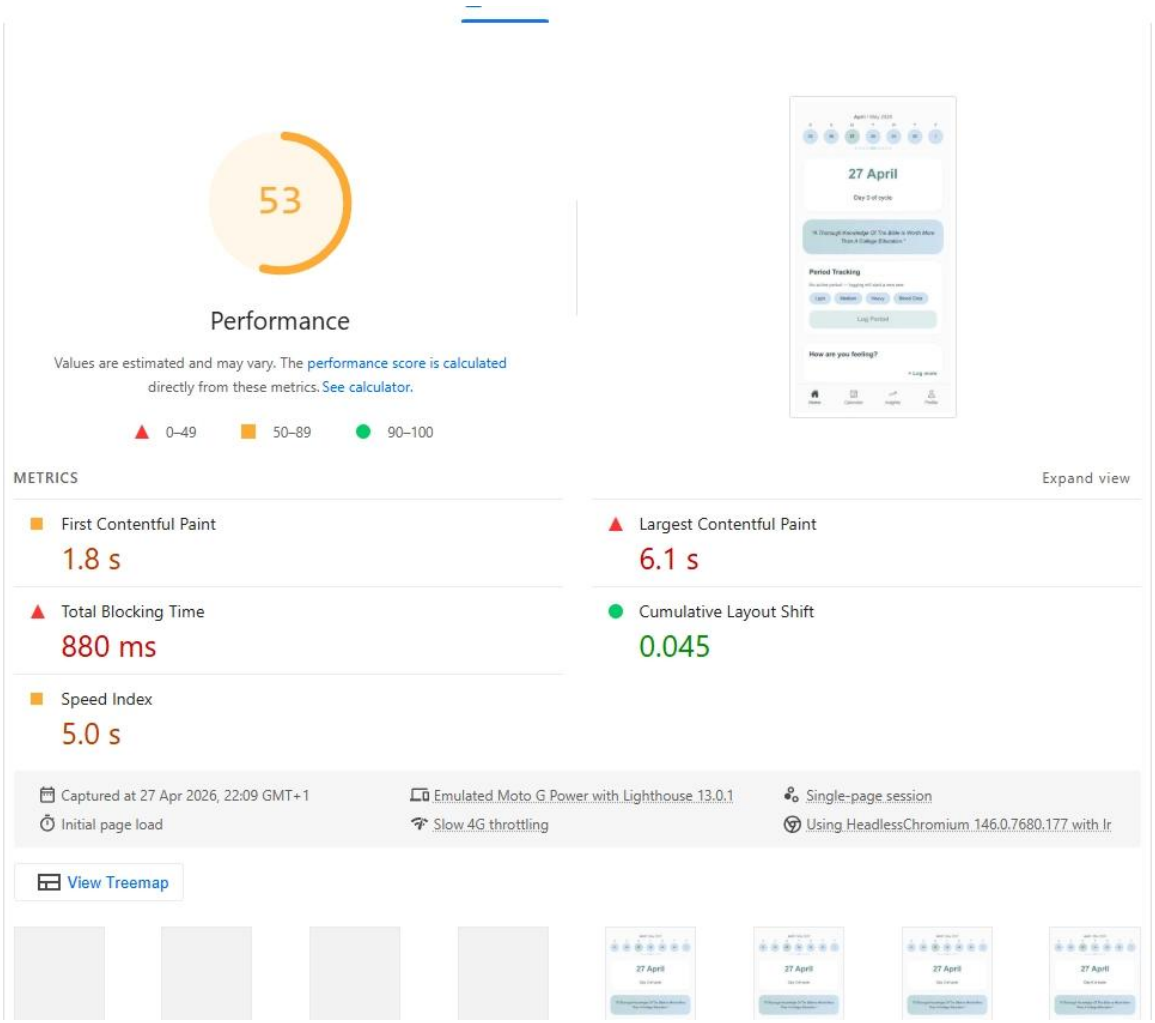


Figure 67 - Google Lighthouse performance audit results for the home page

7.6 Conclusion

The following table evaluates the application against the success criteria defined at the start of this project.

No.	Success Criteria	Outcome	Evidence
1	All high-priority functional requirements are implemented and functional	TRUE	All core features — period logging, symptom tracking, calendar, journal, insights, and PDF export

			— were implemented and verified through Cypress E2E testing
2	The application gets an average rating of 4 out of 5 in usability testing	TRUE	Participants rated the application 4.83 out of 5 on average across all three user testing sessions
3	Users can successfully complete all tasks	TRUE	All participants completed core tasks across all three sessions; iterative improvements between sessions resolved the issues that arose
4	The back-end API handles authentication securely	TRUE	Laravel Sanctum token-based authentication was tested via Insomnia; login, logout, and protected route behaviour all confirmed working
5	Users report that the application is visually clear and easy to navigate	TRUE	Visual design averaged 4.83 out of 5; first impression clarity was rated as mostly or very clear by all participants

Table 5 - Evaluation against success criteria

Overall, all five success criteria were met. The strong usability scores, accessibility results, and positive user feedback across all rounds of testing demonstrate that Mira successfully meets its core aim of delivering a user-centered, accessible, and usable cycle-tracking application.

8.0 Conclusion & Future Work

8.1 Conclusion

This project set out to design a user-centered cycle-tracking application that reflects users' needs rather than assumptions about what they want. The title claims that user-centered design can produce something more inclusive and useful than what already exists on the market. Based on the research, design, and testing carried out throughout this project, that claim holds up.

The research chapter laid the foundation for everything that followed. The survey of 29 respondents revealed that users were frustrated with existing applications. The interviews backed this up, with all participants noting paywalls as a major pain point and consistently asking for a simpler, more organised symptom-logging system. Analysis of current market applications confirmed these gaps; each application has clear strengths, but none fully addresses the combination of simplicity, inclusivity, and condition-aware tracking that users seek. These findings shaped the scope and feature set of Míra.

Every design decision was grounded by user research. The colour palette moves away from stereotypical pink, using green and blue based on colour psychology. The typography prioritises readability. The symptom logging was designed to be fast, as the survey and interviews showed that easy logging was the top factor in keeping users engaged. The Double Diamond methodology provided the process structure, ensuring that research fed into design and that design decisions were evaluated before moving to development.

The implementation brought the design to life. The front-end was built with React Native and TypeScript, styled with NativeWind, and deployed through Vercel. The back end runs on Laravel with Laravel Sanctum for authentication and MySQL for data storage and is deployed through Railway. Core features, including period logging, symptom tracking, cycle insights, a journal, a calendar view, and a PDF export report, were all successfully implemented. There were many challenges faced throughout, including CORS errors, deployment issues, and a structural rethink of the period logging database; in the end, all were resolved.

User testing conducted confirmed that the application met its goals. Participants rated it as extremely easy to use, navigated core tasks without guidance, and responded positively to its clean, minimal design. WCAG 2.1 AA accessibility testing returned strong scores across three of the four main pages, with the lower score on the calendar page due to a known limitation. Overall, the testing validated both the design decisions and the application's usability.

Míra does what is set out to do. It is focused, inclusive, and a usable cycle tracking application built from the ground up with what real users want. The user-centered approach was not just a methodology, it was the reason the application turned out the way it did.

8.2 Future Work and Reflection

With the core features of the application successfully implemented within the scope of this project, there are several areas that could be explored and expanded in future development.

Notifications were included in the UI but were not implemented within the project timeframe. A future version would integrate push notifications to remind users to log in, addressing the finding that existing applications are not doing enough to encourage consistent engagement. A contraception-tracking feature could also be added, as mentioned in some user interviews. An optional period prediction feature could also be introduced, giving users who want to see upcoming cycle predictions the option without making it available to everyone.

Due to issues with Expo during development, the application was deployed as a web application rather than a fully native mobile application. Resolving this would be a priority in future development, and it would also fix the PDF export feature. Finally, a dedicated condition-specific mode could be developed, offering users with menstrual health conditions tailored symptom suggestions and condition-aware insights, building on the existing symptom-tracking system.

Looking back on this project as a whole, there is a lot that I feel went well. The user research phase was probably the strongest part of the project by conducting surveys and interviews, and then being able to turn those findings into design and feature decisions, felt like genuine UCD in

practice. The iterative user testing was also something I am proud of; running three rounds of testing and acting on the feedback each time made a noticeable difference to the final application, and the results reflected that. I am also genuinely proud of the development work itself. AI tools were used throughout to support the process by helping to plan functions, explain generated code, and work through problems, but every decision about what to build, how to build it, and why was mine. Building a full-stack mobile application from scratch, is something I did not know I was capable of at the start of this project, and that is probably what I am most proud of.

Appendix

Appendix A – Project Miro Board

This appendix contains the Miro Board used for project planning, diagram creation, and research

Link: Appendix A

Appendix B – Initial Survey

This appendix contains the full results from the Microsoft Forms survey conducted as initial research.

Link: Appendix B

Appendix C – Requirement Iterations

This appendix contains the first and second iterations of functional and non-functional requirements

Link: Appendix C

Appendix D – Notion Web Page

This appendix contains the notion web page used for project management.

Link: Appendix D

Appendix E – GitHub Repository

This appendix contains the link to access the GitHub repository.

Link: Appendix E

Appendix F – Sprint PDF

This appendix contains the link to a PFD showing the full project sprint information.

Link: Appendix F

Appendix G – Figma Board

This appendix contains the Figma board with all prototype designs.

Link: Appendix G

Appendix H – Deployed Front-End

This appendix contains the link to the deployed front-end.

Link: Appendix H

Appendix I – Post-Test Survey

This appendix contains the results from the post-test survey.

Link: Appendix I

Appendix J – Claude conversation for understanding performance testing results

This appendix contains the conversation with Claude to understand the performance testing results.

Link: Appendix J

References

Abhinandan-Kushwaha. (2024, September 17). GitHub - Abhinandan-Kushwaha/react-native-gifted-charts: The most complete library for Bar, Line, Area, Pie, Donut, Stacked Bar and Population Pyramid charts in React Native. Allows 2D, 3D, gradient, animations and live data updates. GitHub. https://github.com/Abhinandan-Kushwaha/react-native-gifted-charts

Abras, C., Maloney-Krichmar, D., & Preece, J. (2004). User-Centered Design. Semantic Scholar. https://www.semanticscholar.org/paper/User-Centered-Design-Abras-Maloney-Krichmar/b0bc70a8c835e570d6b1f6e9b3cb4cbde40d78de

Authentication API Documentation | DummyJSON. (2026). DummyJSON. https://dummyjson.com/docs/quotes

Chappal, M. S. (2021, May 25). The 6 key principles of ui design. Maze. https://maze.co/collections/ux-ui-design/ui-design-principles/

Clue. (2019). Clue: Period and Ovulation Tracker for iPhone and Android. Helloclue.com. https://helloclue.com/

Cypress. (n.d.). JavaScript End to End Testing Framework. JavaScript End to End Testing Framework | Cypress.io. https://www.cypress.io/

Design Council. (2025). The Double Diamond. Www.designcouncil.org.uk; Design Council. https://www.designcouncil.org.uk/our-resources/the-double-diamond/

DiceBear. (2026). DiceBear. https://www.dicebear.com/styles/lorelei/

Eloquent ORM - Laravel - The PHP Framework For Web Artisans. (2011). Laravel.com. https://laravel.com/docs/5.0/eloquent

Figma. (2025). Figma: the Collaborative Interface Design tool. Figma. https://www.figma.com/

Flo. (2023). Flo - ovulation calendar, period tracker, and pregnancy app. Flo.health - #1 Mobile Product for Women's Health. https://flo.health/

Flow Diagram Guide with Examples - Pencil & Paper. (2024). Pencilandpaper.io. https://www.pencilandpaper.io/articles/flow-diagram-guide

GeeksforGeeks. (2020, April 28). Functional vs. Non Functional Requirements. GeeksforGeeks. https://www.geeksforgeeks.org/software-engineering/functional-vs-non-functional-requirements/

Gibbons, S. (2018, January 14). Empathy mapping: the first step in design thinking. Nielsen Norman Group. https://www.nngroup.com/articles/empathy-mapping/

Harley, A. (2015, February 16). Personas Make Users Memorable for Product Team Members. Nielsen Norman Group. https://www.nngroup.com/articles/persona/

Justinmind. (2020, July 15). User-centered design: a beginner's guide. Medium. https://uxplanet.org/user-centered-design-a-beginners-guide-32a58846627d

Kosinski, M. (2024, September 30). Database. Ibm.com. https://www.ibm.com/think/topics/database

Laravel Sanctum | Laravel 13.x - The clean stack for Artisans and agents. (2026). Laravel. https://laravel.com/docs/13.x/sanctum

Laverty, P. (2017, June 15). User Enumeration Explained: Techniques and Prevention Tips | Rapid7 Blog. Rapid7. https://www.rapid7.com/blog/post/2017/06/15/about-user-enumeration/

LucidChart. (2024). What is an Entity Relationship Diagram (ERD)? Lucidchart. https://www.lucidchart.com/pages/er-diagrams

Matéu.sh. (2024, May 30). React Context API Explained with Examples. FreeCodeCamp.org. https://www.freecodecamp.org/news/react-context-api-explained-with-examples/

Maze. (2024, April 17). How to conduct user interviews: Step by step + best practices. Maze; Maze. https://maze.co/guides/user-interviews/conduct/

Miro. (2024). Team collaboration software & online whiteboard for teams | Miro. Https://Miro.com/; Miro. https://miro.com/

Notion. (2019). Notion – The all-in-one workspace for your notes, tasks, wikis, and databases. Notion. https://www.notion.so/

Noumcpe. (2024, June 4). Laravel 11 REST API Authentication using Sanctum Tutorial. Medium. https://medium.com/@noumcpe0007/laravel-11-rest-api-authentication-using-sanctum-tutorial-12231b02354b

Olesen, J. (2013). Color Meanings - All About Colors and Symbolism. Color Meanings. https://www.color-meanings.com/

Pernice, K., & Rosala, M. (2023, September 17). User Interviews: How, When, and Why to Conduct Them. Nielsen Norman Group. https://www.nngroup.com/articles/user-interviews/

Powell, P., & Smalley, I. (2025, August). End-to-end (E2E) testing. Ibm.com. https://www.ibm.com/think/topics/end-to-end-testing

Ravi Saraswathi. (2020, January 6). Application architecture types. Ibm.com. https://www.ibm.com/think/topics/application-architecture-types?mhsrc=ibmsearch_a&mhq=application%20architecture

React Native. (2020, March 27). Wikipedia. https://en.wikipedia.org/wiki/React_Native

React Native. (2024). React Native · A framework for building native apps using React. Reactnative.dev. https://reactnative.dev/

Schneider, J. (2015, February 3). The Double Diamond: Strategy + Execution of the Right Solution. ThoughtWorks. https://www.thoughtworks.com/insights/blog/double-diamond

Sobrinho, R. (2023, July 13). UX Process: The Double Diamond - Hi Interactive - We deliver applications with amazing user experience. Hi Interactive - We Deliver Applications with Amazing User Experience - We Deliver Applications with Amazing User Experience. https://hi-interactive.com/blog/ux-process-the-double-diamond/

Stardust. (n.d.). Stardust. Stardust.app. https://stardust.app/

Stevens, E. (2023, May 5). How to design effective user surveys for UX research. Www.uxdesigninstitute.com. https://www.uxdesigninstitute.com/blog/user-surveys-for-ux-research/

Stevens, E. (2024, September 10). 7 fundamental UX design principles in 2026 (with examples). UX Design Institute. https://www.uxdesigninstitute.com/blog/ux-design-principles-2026/

UserTesting Quick Start Guide – Knowledge Base Home. (2026). Usertesting.com. https://help.usertesting.com/hc/en-us/sections/12573123482397-UserTesting-Quick-Start-Guide

Visual Paradigm. (2019a). What is Sequence Diagram? Visual-Paradigm.com. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/

Visual Paradigm. (2019b). What is Use Case Diagram? Visual-Paradigm.com. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/

W3C. (2016, May 6). Accessibility, Usability, and Inclusion. Web Accessibility Initiative (WAI). https://www.w3.org/WAI/fundamentals/accessibility-usability-inclusion/

W3C. (2024, December 12). Web Content Accessibility Guidelines (WCAG) Overview. Web Accessibility Initiative (WAI). https://www.w3.org/WAI/standards-guidelines/wcag/

What is a Prototype? Definitions and Benefits of Prototyping. (2024). Https://Miro.com/. https://miro.com/prototyping/what-is-a-prototype/

What is a wireframe? A guide for non-designers. (2025). Balsamiq.com. https://balsamiq.com/blog/what-are-wireframes/

What is User Centered Design (UCD)? (2016, June 5). IxDF - Interaction Design Foundation. https://ixdf.org/literature/topics/user-centered-design#ucd_is_an_iterative_process-0