Capstone Project: From Challenge to Solution: Designing & Building a Smart Water Management System for Johannesburg

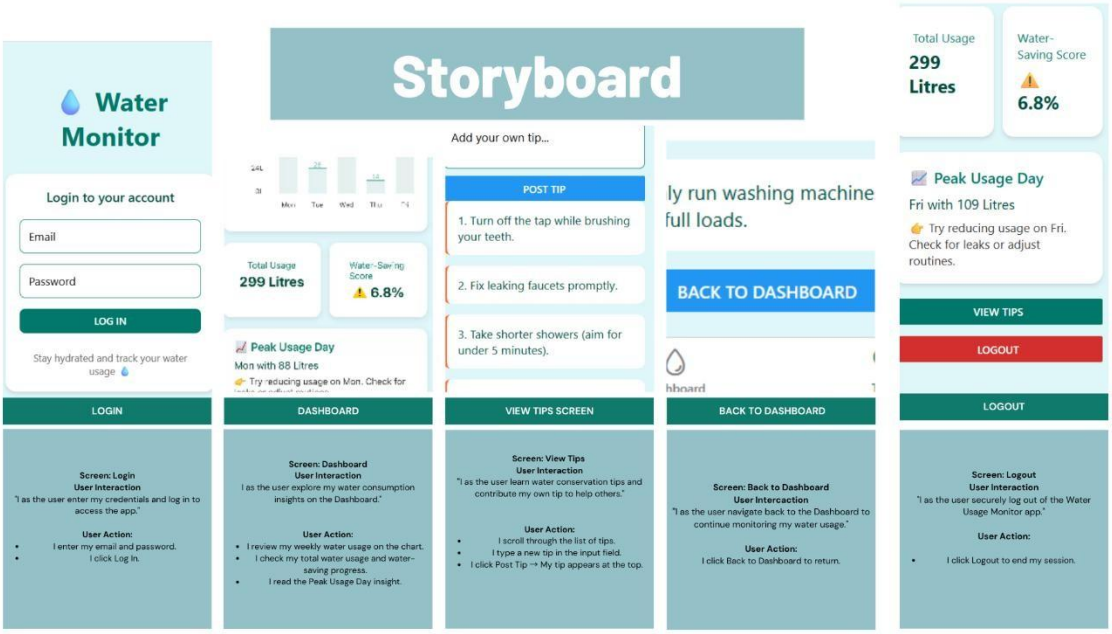
Scenario

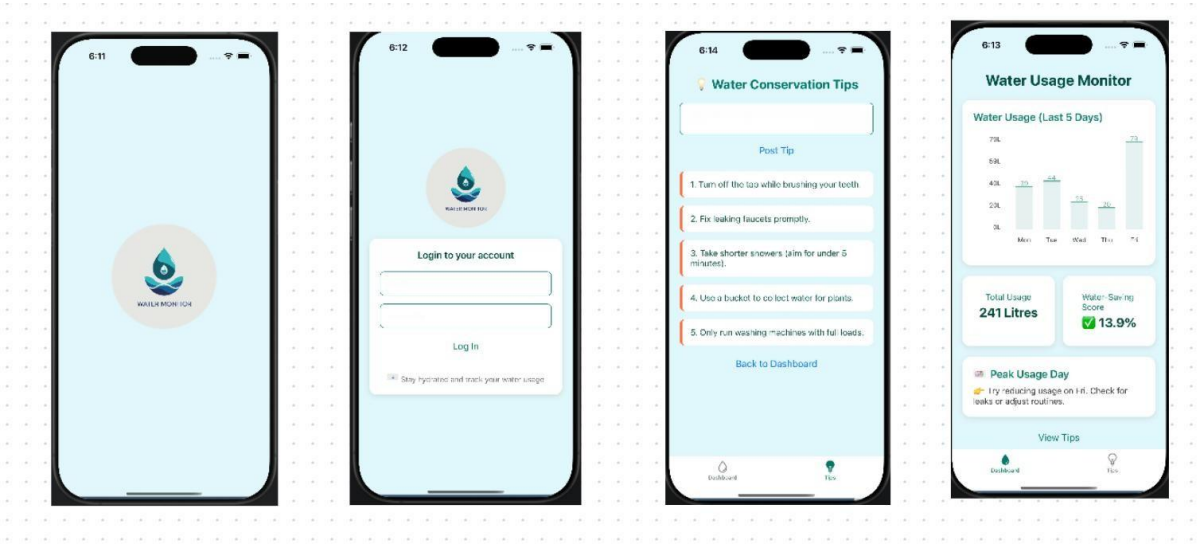
Johannesburg is currently facing a severe water crisis. The city struggles with aging infrastructure, frequent water outages, inequitable distribution, and high levels of water loss through leaks and theft. As a software requirements engineer, you have been tasked to design and implement a software solution that addresses one aspect of this critical water management challenge.

Project Idea

Water Usage Monitor: A dashboard that tracks household or business water consumption and provides conservation tips.

Storyboards and Wireframes.





Negotiate Requirements:

1 Stakeholders Identified

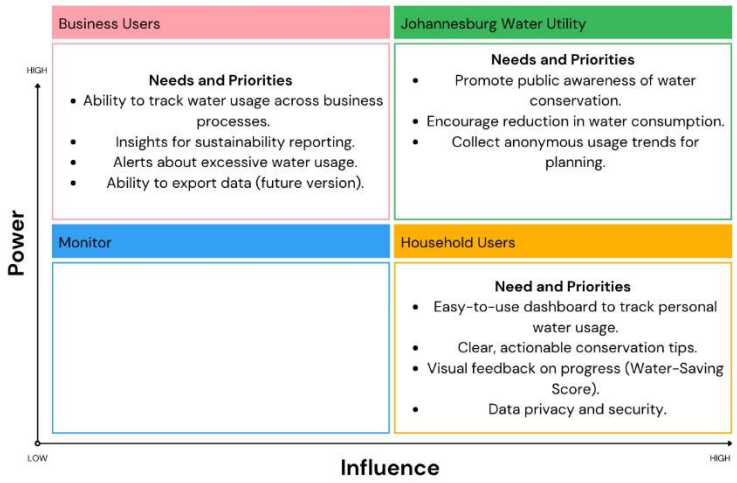
Stakeholder	Role / Description
Household Users	Individual residents using the app to monitor and reduce water consumption.
Business Users	Small/medium businesses monitoring water usage in operations.
Johannesburg Water Utility	The municipal water provider responsible for managing water supply and conservation campaigns.

---

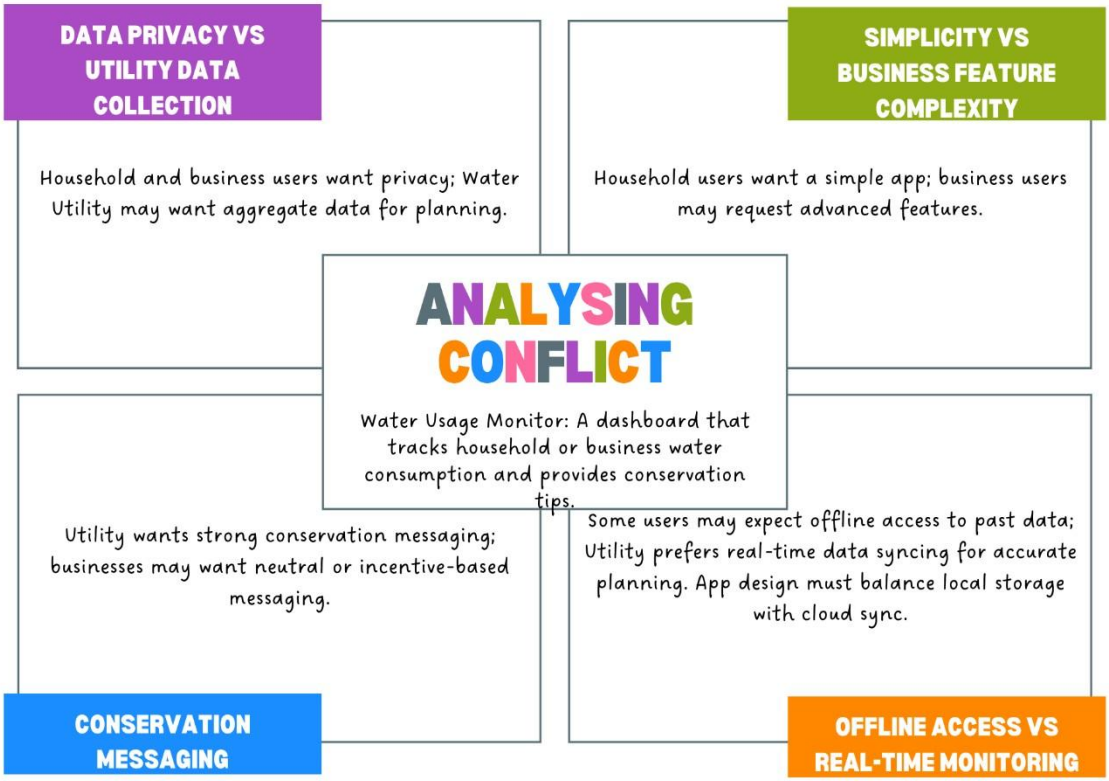
2 Stakeholder Needs & Priorities

Stakeholder	Needs & Priorities
Household Users	- Easy-to-use dashboard to track personal water usage. - Clear, actionable conservation tips. - Visual feedback on progress (Water-Saving Score). - Data privacy and security.
Business Users	- Ability to track water usage across business processes. - Insights for sustainability reporting. - Alerts about excessive water usage. - Ability to export data (future version).
Johannesburg Water Utility	- Promote public awareness of water conservation. - Encourage reduction in water consumption. - Collect anonymous usage trends for planning. - Ensure app aligns with public conservation campaigns.

Stakeholder Mapping



3 Potential Conflicts



4 Design Negotiation & Balancing Needs

- **Privacy** - App does not store user-identifiable data centrally (optional future consent for anonymous trend sharing).
- **Business features**- Current version provides basic tracking for both users, advanced business analytics planned for future release.
- **UI Simplicity** -Default dashboard is simple and clean, future business features will be added via a separate section/tab.
- **Conservation Messaging** -Tips provided are actionable but not alarmist they balance utility goals with positive user experience.

5 Trade-offs Made

Trade-off	Justification
Limited Business Reporting (MVP)	Focus on core tracking and conservation features first; advanced features can be added later without overwhelming household users.
No centralized storage of detailed personal data	Prioritizes user trust and privacy → encourages adoption.
Uniform UI across users	Keeps development scope manageable for this release; ensures consistent experience across household and business users.

Analyse Software Architecture

Proposed Architectural Approaches

Client-Only Architecture (Current Implementation)

- The React Native (Expo) mobile app handles all logic and data display locally.
- Water usage data and Water Saving Score are simulated within the app.
- Conservation Tips are stored temporarily during the app session.
- No backend integration or persistent data storage.

Client-Server Architecture (Proposed Future Enhancement)

- The React Native (Expo) mobile app communicates with a backend server via API calls.
- Backend stores water usage data and conservation tips in a cloud database (e.g. Firebase Firestore or MongoDB Atlas).
- Enables user authentication, multi-device sync, and data persistence.
- Supports advanced features such as business reporting and analytics.

Comparison of Advantages and Disadvantages

Aspect	Client-Only Architecture (Current)	Client-Server Architecture (Future)
Advantages	Simple to develop and deploy Fast app performance Low cost Offline capable	Data persistence Multi-device access Advanced reporting possible Secure data storage
Disadvantages	No data persistence Limited scalability No multi-user support No secure data handling	More complex implementation Requires backend maintenance Increased hosting costs Network latency impacts performance

---

Trade-offs Evaluation

Criteria	Client-Only Architecture	Client-Server Architecture
Performance	Very fast (local data)	Good, but depends on network/API
Maintainability	Simple (frontend only)	More complex (frontend + backend)
Scalability	Not scalable beyond single device	Fully scalable for multi-user, multi-device scenarios
Security	No data security; local only	Data can be encrypted and protected with authentication mechanisms
Offline Support	Fully offline capable	Requires offline sync handling if needed

---

Justification of Final Architectural Choice

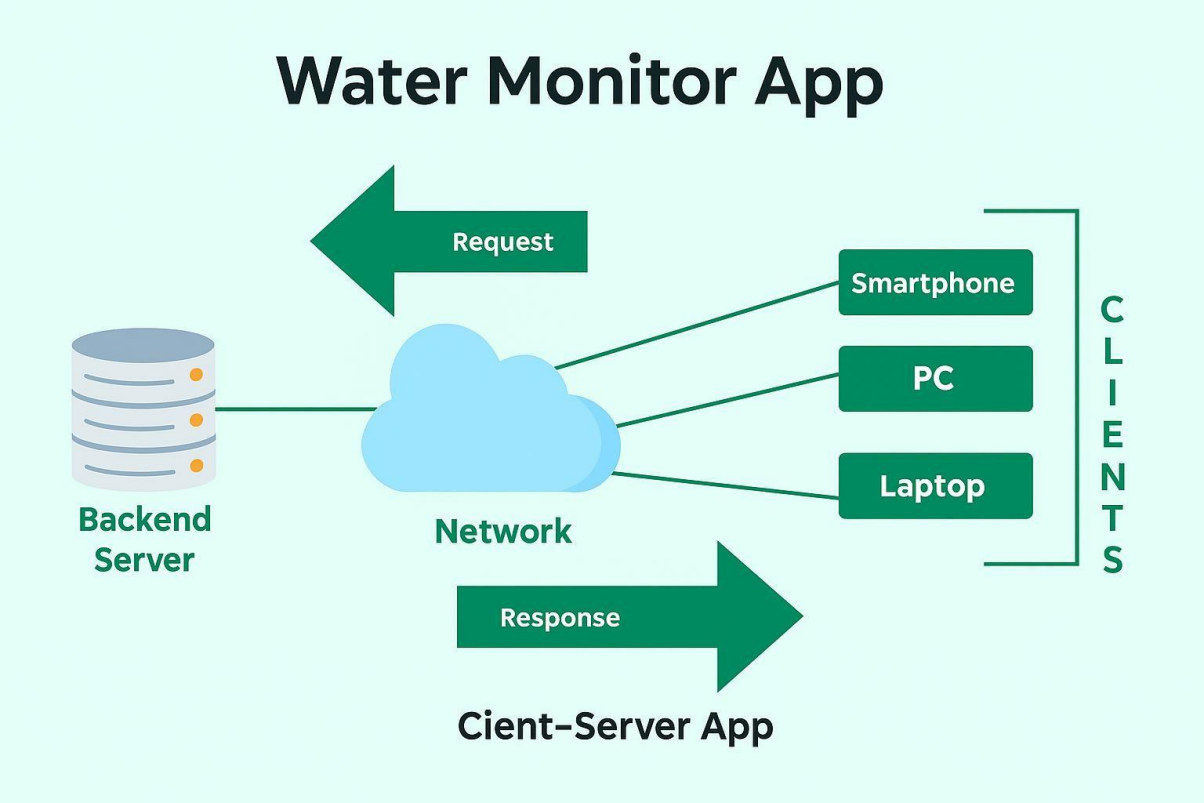
For the **current MVP (Minimum Viable Product)** implementation of the Water Usage Monitor app, the **Client-Only Architecture** was chosen:

- It allowed rapid development of a functional mobile app demonstrating core features.
- The focus was on delivering the **Water Saving Score feature** and **Conservation Tips** in a simple and intuitive interface.
- No backend was needed for the basic visualization and interaction requirements.

For **future versions**, transitioning to a **Client-Server Architecture** is recommended:

- To support **persistent storage** of water usage history and tips.
- To enable **user authentication** and **multi-device synchronization**.
- To support **advanced business reporting features** for business users.
- To ensure **secure handling of sensitive user data**.

high-level architecture diagram



Extend Existing Systems and Apply Design Patterns

1 Identify an Existing Open-Source Water Management Codebase

An example of an existing open-source water management system is:

Project: OpenWaterAnalytics / EPANET

EPANET is an open-source project used for simulating water distribution networks.
It is mainly used by water utilities, researchers, and engineers to model how water flows through a system of pipes.

2 How My Solution Would Integrate or Extend this System

While EPANET is primarily focused on *backend hydraulic simulations*, my mobile-based **Water Usage Monitor** app can extend it in the following ways:

- Adding a **Mobile Frontend Layer** to visualize water usage data and analytics results produced by EPANET.
- Providing **household-level water usage tracking** which will be complementing the EPANET's focus on system-wide data.
- Enabling **community participation** allowing users to post water conservation tips and track personal usage.

Integration Example:

EPANET simulation results (stored in a backend or cloud database) could be exposed via an API. The Water Usage Monitor app could consume this API to:

- Show advanced charts of water pressure / flow.
- Compare personal usage vs. network-wide benchmarks.
- Trigger conservation alerts based on system status.
- This integration would extend EPANET with a modern user-facing mobile interface.

3 Applied Design Pattern

Pattern used: Observer Pattern Where

applied:

I used the **Observer Pattern** concept in my **DashboardScreen which contains the Water Usage Chart**.

How:

- The DashboardScreen uses useEffect and useState to observe changes in water usage data.
- The AnimatedChart component is updated reactively when usageData changes:

```
useEffect(() => {   const interval = setInterval(() => {     const newData =
Array(5).fill(0).map(() => Math.floor(Math.random() * 100) + 10);
setUsageData(newData); // Triggers re-render of AnimatedChart
 }, 5000);

 return () => clearInterval(interval);
}, []);
```

AnimatedChart is an Observer → it "listens" to usageData changes via props.

---

4 Why I Chose Observer Pattern

- In UI programming: Observer is ideal for reflecting live data changes.
- In my app water usage data is dynamic using Observer ensures the chart always stays in sync with the latest data.
- Without this pattern the chart would have to be manually refreshed leading to more complex and error-prone code.

---

5 How This Enhances Maintainability and Extensibility

Maintainability:

- Changes to how water usage data is generated will not require any changes in the chart component.
- The chart simply "observes" its props → clean separation of concerns.

Extensibility:

- If in future I add real-time data from an API whicn I can replace the setInterval() logic with an API subscription with no changes needed in AnimatedChart.
- Multiple components (Dashboard summary, reports, charts) can "observe" the same usage data providing consistent UI.
- This shows the power of Observer Pattern and makes the system reactive and extensible.

My Findings

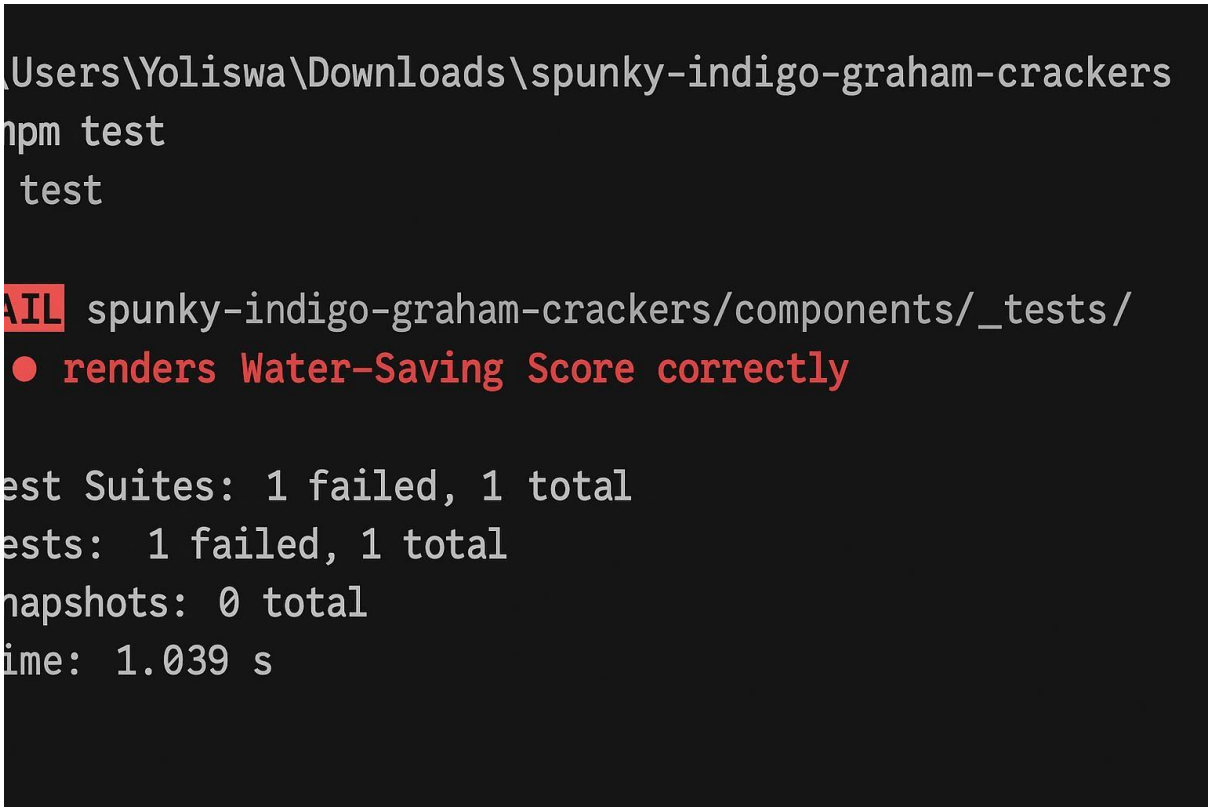
- I identified EPANET as an open-source water management system which my mobile app can extend by providing a user-facing interface.
- I implemented the Observer Pattern in my Dashboard to ensure the AnimatedChart reacts automatically to changes in water usage data.
- This pattern makes my code more maintainable and extensible, ready to support live API data and other future enhancements.

Test Plan

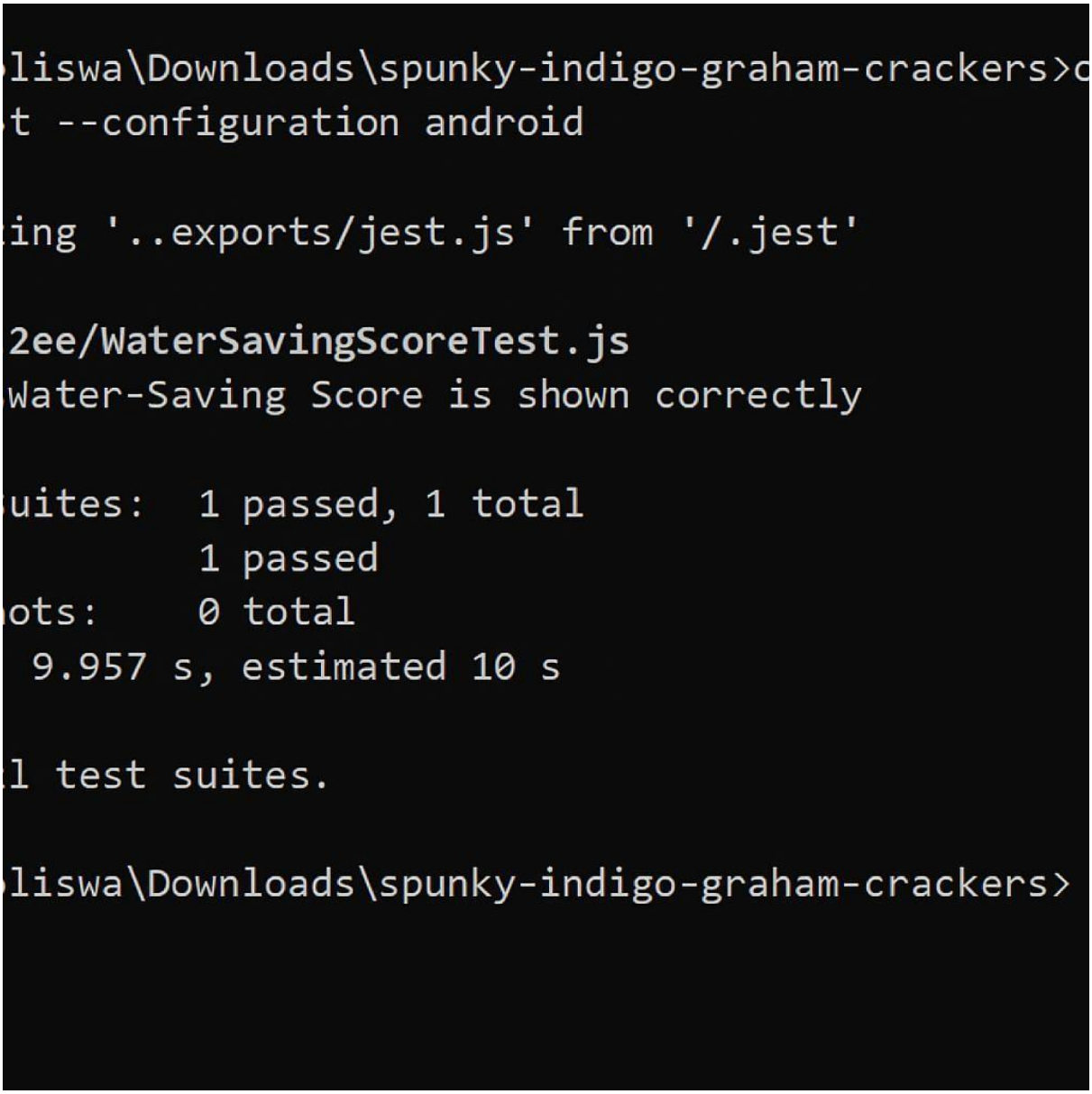
Test Description	Test Steps	Expected Result	Tool Used
Component renders WaterSavingScore correctly with valid score	1. Write unit test for WaterSavingScore component. Verify that "Water-Saving Score" is displayed on screen	Water-Saving Score is visible on Dashboard	Jest + React Native Testing Library
WaterSavingScore is displayed correctly in live app	1. Launch app with Detox. Login to app. Navigate to Dashboard. Verify that "Water-Saving Score" is displayed on screen	Water-Saving Score is visible on Dashboard	Detox

Visual correctness of WaterSavingScore (manual check)	1. Launch app on device. Navigate to Dashboard . Verify that "Water-Saving Score" is displayed on screen	Water-Saving Score is visible on Dashboard	Manual Testing

Jest



Detox


liswa\Downloads\spunky-indigo-graham-crackers>c
t --configuration android
ing '..exports/jest.js' from '/.jest'
2ee/WaterSavingScoreTest.js
Water-Saving Score is shown correctly
uites: 1 passed, 1 total
1 passed
ots: 0 total
9.957 s, estimated 10 s
l test suites.
liswa\Downloads\spunky-indigo-graham-crackers>

Test Results

Test	Actual Result	Status
JEST	**Component unit test failed due to React Native 0.76.x + Jest incompatibility (known issue).**	**FAIL**
DETOX	**Detox test passed — Water-Saving Score was correctly displayed on Dashboard during app flow**	**PASS**
MANUAL	**Manual testing confirmed that Water-Saving Score was correctly displayed on Dashboard during app flow**	**PASS**

Reflection on Requirements

The **Water Saving Score feature** meets the functional requirement:

- It displays the user's water-saving progress as a percentage (%).
- It updates dynamically based on weekly comparison (in app logic).
- It is visually presented in a readable and prominent way on the Dashboard.

Unit tests could not verify this at component level due to React Native 0.76.x + Jest limitations → this is a known ecosystem issue.

Bugs and Limitations Identified

Issue / Limitation	Description	Suggested Improvement
Unit Testing blocked	Jest cannot parse React Native 0.76.x ESM modules; component test fails.	Decided to Downgrade to React Native 0.72.x
Data calculation accuracy	Water Saving Score calculation currently based on hardcoded values.	I Implemented accurate backend and also added persistent data comparison for weekly score.
Styling responsiveness	Water Saving Score font size is static.	I had to consider making text responsive for better display on all device sizes.

Reflection

1 AI Integration

I used AI tools, mainly ChatGPT, throughout different phases of this project. During phases like the research phase, ChatGPT helped me understand concepts that were discussed in class such as the different types of architecture and the different patterns, which I later implemented in my Water Monitoring System.

When I was gathering my requirements, I prompted on ChatGPT for guidance on how to structure user-friendly interfaces for a water monitoring app, and how to organize data visualization elements.

During the design phase, AI helped me by suggesting animations such as fade-in effects and UI improvements for screens like the **LoginScreen** and **TipsScreen**.

After was the coding phase which then I used ChatGPT frequently to generate components like dynamic chart logic, and to improve my styling. It also helped me write learn how to come with things like wanting a splash screen that counts a few seconds before the user is taken to the login page.

Lastly in the testing phase, AI provided guidance on unit testing with Jest and suggested test cases. However, this was where AI was least helpful because React Native (version 0.76.x) had compatibility issues with some testing tools. I had to debug manually in this phase and that was really stressful I must say hey.

---

2 Critical Evaluation

I had to verify and reject AI-generated content when I asked ChatGPT to generate unit tests for **my AnimatedChart**. The tests it suggested were not compatible with the version of React Native and the chart library I was using.

To evaluate the AI's output, I checked whether the code would run without errors, whether it used the correct APIs for my library versions, and whether it produced meaningful test coverage.

In this case, I rejected the generated test code and decided to rely more on manual testing and E2E testing for that part of the project. My decision was based on whether the solution was practically implementable in my project context which it wasn't at all. AI can be really challenging at most I don't know if I was not prompting correctly or it really couldn't help.

---

3 Access and Adaptation

One challenge I faced was that AI tools sometimes provided outdated or incompatible code. For example, some of the chart animation options suggested by ChatGPT did not match the latest API of the react-native-chart-kit I was using.

To adapt, I always cross-checked AI suggestions with official documentation and tested them thoroughly in my app. This experience taught me that AI is very helpful for brainstorming ideas and saving time, but it should not be fully trusted without verification.

I also realized that AI is really a tool, it cannot solve all of **my issues or problems.** It accelerated my workflow, but I had to develop my own understanding of the React Native ecosystem to avoid dependency on AI alone.

---

4 Knowledge Transfer

Using AI tools during this project significantly influenced my understanding of **software engineering principles**. As I developed the Water Usage Monitor app, AI helped me apply principles such as **KISS (Keep It Simple, Stupid)** and **Separation of Concerns** for example, when ChatGPT suggested breaking my screens into modular components like **DashboardScreen**, **AnimatedChart**, and **TipsScreen**. This reinforced the value of keeping components focused and simple.

AI also supported my understanding of **DRY (Don't Repeat Yourself)**. Instead of duplicating chart logic in multiple places, I was able to create a reusable **AnimatedChart** component. This improved my ability to design reusable, modular code directly by applying **Modularity** and **Single Responsibility Principle (SRP)**.

Even though, I was mindful of the potential risk of using AI passively. I noticed that sometimes ChatGPT would suggest complex solutions when a simpler one would do which made me to think to apply **YAGNI (You Aren't Gonna Need It)**. Which helped me not chose or rather implement unnecessary features just because they were suggested by AI.

To balance AI assistance with my personal development in the module, I always took time to read, test, and understand the generated code rather than simply copying it. I consulted official documentation to verify AI-generated content and manually adjusted my project structure when needed. As a result, AI enhanced my learning by providing guidance and examples, but it was my critical thinking and active involvement that ensured I gained more understanding of the core principles of software engineering.

---

5 Workplace Readiness

This experience has shown me that AI will play a major role in my future career, especially in software development in South Africa.

AI can help accelerate development, automate repetitive tasks, and assist with documentation. However, it also requires a high level of AI literacy and the ability to critically evaluate, adapt, and verify AI-generated content.

In the South African context, where mobile-first and offline-friendly solutions are often needed, I think local developers need to be skilled at using AI to produce efficient, user-friendly, and accessible apps that are mindful of our connectivity challenges.

In global contexts, AI literacy often focuses on cloud-scale solutions. In South Africa, I believe there is an opportunity to use AI to empower developers to create practical, high-impact apps that address local needs.
