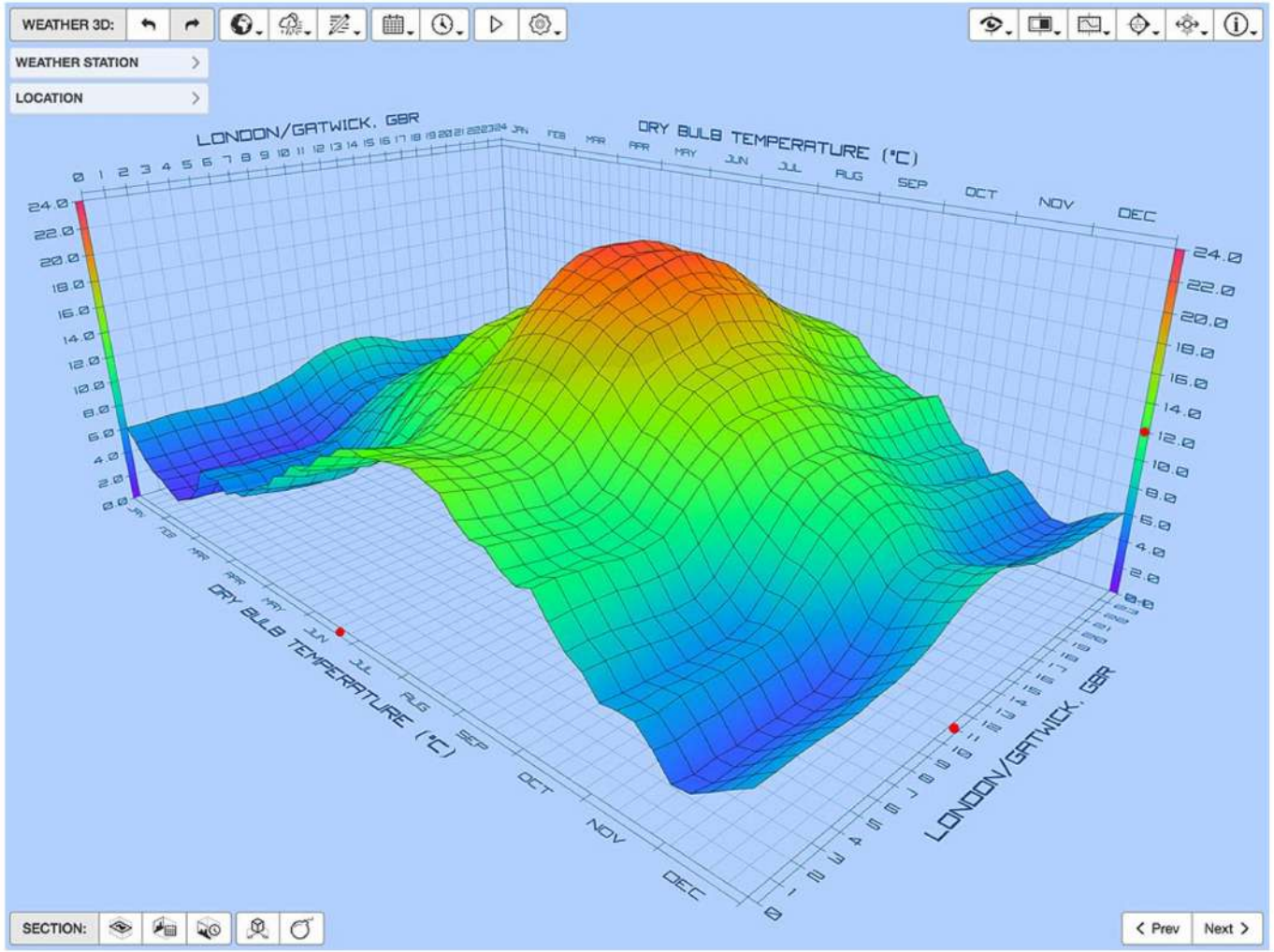
IA3 Digital Solutions

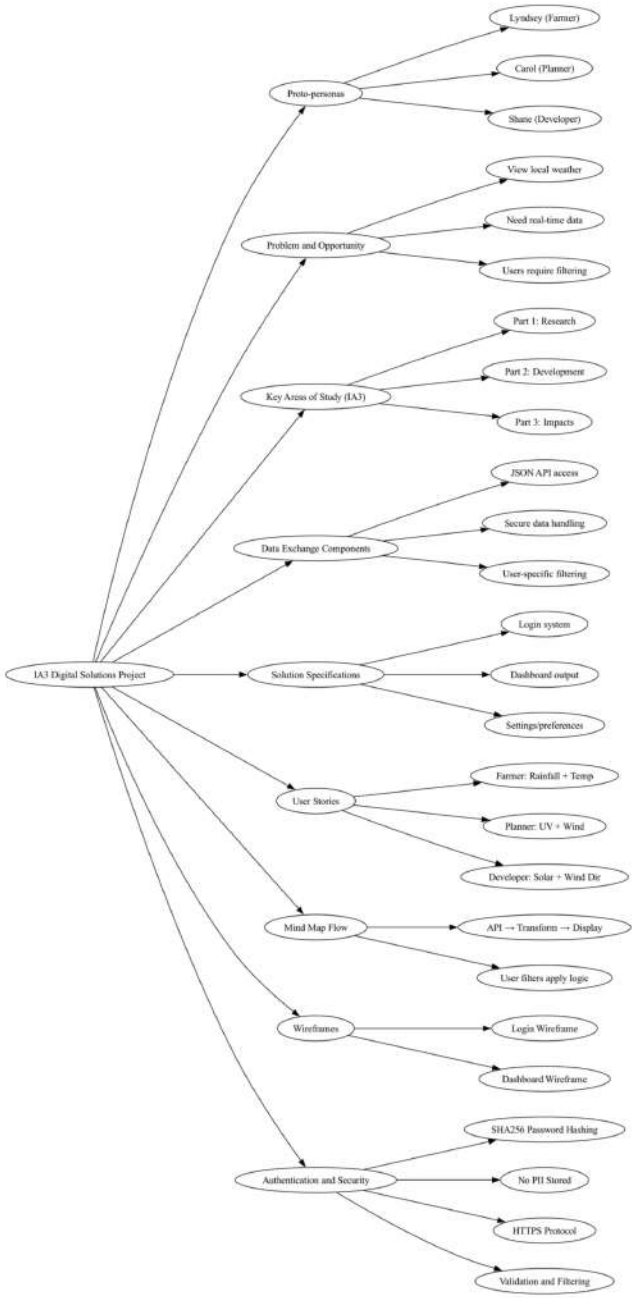


SALMAN, Taha
106845

Table of Contents

Research and Investigation

Mind Map and Users



User	Role	User Story
Lyndsey Ball	Dairy Farmer	As a dairy farmer, I want to view live rainfall and temperature data so that I can plan crop irrigation and choose optimal livestock grazing areas.
Carol Smart	Council Town Planner	As a town planner, I want to access wind speed, temperature, UV index and rainfall data so that I can make informed decisions for future infrastructure planning.
Shane Potts	Property Developer	As a property developer, I want to analyse historical and live weather data so that I can determine the best property placement, orientation, and design choices.

Analysis of Existing Solutions

Overview:
WeatherZone is a professional Australian weather forecasting platform aimed at the general public and professionals. It provides detailed meteorological information using data sourced from the Bureau of Meteorology (BOM) and proprietary networks.

Features:

- Current temperature, UV index, wind, rainfall, and radar
- Fire danger alerts
- Location-based customisation (postcode or GPS)
- Cross-platform compatibility (desktop and mobile apps)

Data Source:

- BOM (Bureau of Meteorology)
- Private WeatherZone stations

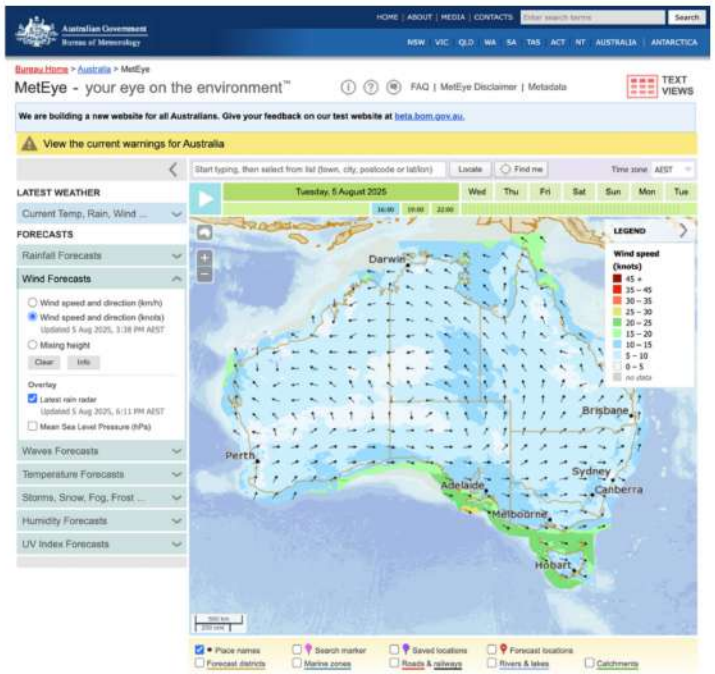
Security:

- HTTPS encryption ensures secure data transmission
- Basic use requires no login, so no personal information is stored
- Commercial API is token-gated and not available to students or public use

Limitations:

- Limited ability for users to select specific data types or display options
- No developer-friendly access for integration into custom solutions
- Customisation limited to alerts, not full interface control

Usability Evaluation:	
Principle	**Observations**
Effectiveness	Information is highly accurate and reliable due to BOM integration.
Learnability	Interface is clean but can be overwhelming for new users due to cluttered layout.
Accessibility	Good colour contrast and accessible on both mobile and desktop. Font size and navigation structure are readable.
Utility	Provides comprehensive data but lacks user-specific filtering or preference saving.
Safety	Uses secure HTTPS and does not store user information unless login is used. No risks to user privacy in basic mode.



Overview:
MetEye is the official interactive weather map tool provided by the Bureau of Meteorology. It is designed for both the general public and professional users such as emergency services and planners.

Features:

- Real-time overlays for temperature, wind, rain, cloud, and more
- Forecasts and warnings specific to regions
- Interactive map with date/time selectors

Data Source:

- Direct from BOM servers
- High-resolution and frequently updated data

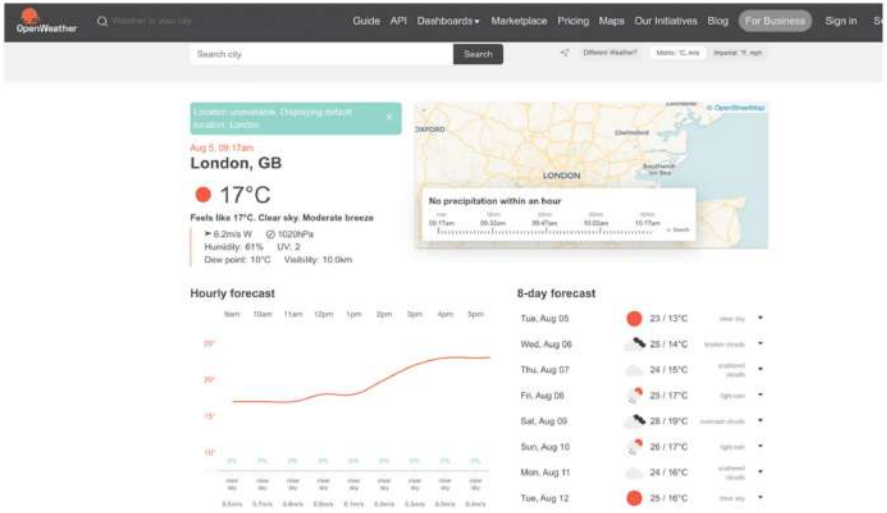
Security:

- No account system, so no private information is collected
- All data is accessed over HTTPS
- Open to public without authentication

Limitations:

- No personalisation or user settings
- Interface is dense and technical, which may confuse general users
- Requires familiarity with meteorological symbols and map navigation

Usability Evaluation:	
Principle	**Observations**
Effectiveness	Extremely accurate and robust due to BOM source. Ideal for professionals.
Learnability	Steep learning curve for non-experts; map layers and controls are complex.
Accessibility	Visually detailed but not optimised for screen readers or mobile-first design.
Utility	Offers a wide range of data but cannot be customised by end-users.
Safety	Fully anonymous usage with no PII handling. Very secure.



Principle	Observations
Effectiveness	Very accurate and comprehensive data; API is well documented and reliable.
Learnability	High for developers but not suitable for general users without programming knowledge.
Accessibility	Dashboard is clean and minimal but assumes technical literacy; not suited for screen readers or public users.
Utility	Highly flexible for developers. Supports custom integrations and unit preferences. Lacks built-in visualisation.
Safety	Secure by design. Requires API key for all use. No PII stored or processed in standard usage.

Overview:
OpenWeatherMap is a global weather service platform aimed at developers, businesses, and agriculture services. It provides a wide range of data APIs including current weather, hourly forecasts, pollution levels, and agricultural indicators. It is used in both consumer-facing apps and backend systems.

Features:
- Real-time and forecast data (hourly, daily, weekly)
- Developer dashboard for managing keys and usage
- Customisable API outputs (metric or imperial units)
- Integration-ready for web and mobile applications

Data Source:
- Aggregated from global satellite data, radar, and weather stations
- Combination of government and private datasets

Security:
- HTTPS protocol for all API interactions
- Requires API key for access
- Tiered plan structure; free plan has usage limits
- Secure authentication prevents unauthorised use

Limitations:
- The free tier only allows limited API calls per day
- Some advanced features (pollution, solar radiation) are restricted to paid plans
- Mostly developer-oriented, not user-friendly for non-technical users
- No public web interface for weather customisation or display

Data Exchange Requirements

Data exchange is the process of retrieving, converting, and presenting structured data from one system so that it can be accurately understood and used by another system or end-user.
In this project, it involves:

- Retrieving live JSON weather data from the Ecowitt API
- Transforming this into human-readable data
- Displaying it securely to authenticated users

The application must allow each users to:

- Select preferred data types
- Filter and transform the data for display
- Store preferences in account JSON

The data required by the users of the application is as follows:

User	Required Data
Henry (Farmer)	Rainfall, Temperature, Humidity
Sara (Planner)	Wind, UV Index, Rainfall, Temperature
Shane (Developer)	Wind direction, Solar radiation, Temperature

Implications

- JSON is ideal for real-time weather data exchange due to:
- Its support for nested structures
- Fast parsing in Python
- Small payload size (efficient for APIs)
- Security implications must be considered

Format	Description	Advantages	Disadvantages	Decision
JSON	JavaScript Object Notation – lightweight data format used in modern APIs	- Easy to parse - Compact size - Readable by both humans and machines	- Less strict structure - Not ideal for large documents	Used in this project (Ecowitt API returns JSON)
XML	Extensible Markup Language – structured data format using nested tags	- Self-describing - Strong validation (XSD)	- Verbose - Slower parsing - Higher memory use	Not suitable, too slow and heavy
CSV	Comma-Separated Values – plain text table format	- Lightweight - Ideal for flat data tables	- Cannot represent nested structures - No metadata	Not supported by API

Analysis of Dataset

Below are the data categories used in the application

Category	Example Values	Relevance
Temperature	26.7°C	All users
Humidity	74%	Farmers, Planners
Rainfall	2.14 in/week	Farmers, Planners
Wind Speed	14.3 km/h	Planners, Developers
Wind Direction	245°	Developers
UV Index	10	Planners
Solar Radiation	1069.2 W/m^2	Developers

The data exchange method is:

Connect to Ecowitt API

- Use HTTPS GET request with API key and MAC ID

Receive JSON Data

- e.g. response = requests.get(api_url).json()

Transform Data

- Extract only required fields
- Convert Fahrenheit to Celsius if needed
- Rename keys for display (e.g. "temp" → "Temperature (°C)")

Display Data

- Show structured data in a table or dashboard
- Include labels and units

Issues and Limitations of the Data

Issue	Description	Solution
Mixed units	API returns °F, in, mph	Transform into SI units (metric)
No location name in dataset	Only MAC ID provided	Manually map MAC address to user-friendly names
No user-specific data	API returns station-wide data	Filter by preferences in app

Example of Structure


```
{
  "data": {
    "outdoor": {
      "temperature": { "value": "88.3", "unit": "°F" },
      "humidity": { "value": "51", "unit": "%" }
    },
    "solar_and_uvi": {
      "uvi": { "value": "10" },
      "solar": { "value": "1069.2", "unit": "W/m²" }
    },
    "rainfall": {
      "hourly": { "value": "0.00", "unit": "in/hr" },
      "weekly": { "value": "2.14", "unit": "in" }
    },
    "wind": {
      "wind_speed": { "value": "1.8", "unit": "mph" },
      "wind_direction": { "value": "205", "unit": "°" }
    }
  }
}
```


Transformations of data

Convert units:

- °F ➔ °C → C = (F - 32) × 5/9
- in ➔ mm → mm = in × 25.4
- mph ➔ km/h → km/h = mph × 1.60934

Filter only needed keys for each user

Flatten nested JSON for tabular display

Justification for Use of Data:

Freely available

Real-time live data

Secure via API key

Relevant to all three users

Highly structured for efficient parsing

Implementation in Solution:

The JSON data retrieved from the Ecowitt API is parsed using Python's json library. Only the required keys based on user preferences are extracted. These values are renamed for clarity and converted into metric units where necessary. The processed data is then displayed on the dashboard interface using labels and values formatted in a table-style layout. This ensures that all data is both human-readable and relevant to each user's role (farmer, planner, or developer).

JSON Key	Display Label
temperature	Temperature (°C)
humidity	Humidity (%)
uvi	UV Index
solar	Solar Radiation (W/m^2)
rainfall.weekly	Rainfall (mm)
wind_speed	Wind Speed (km/h)
wind_direction	Wind Direction (°)

Analysis of Data Security Risks

- The application is required to handle:
- Live data transmission from an external API
- User login credentials and preferences
- Potential threats to privacy, accuracy, and system reliability

The security and threat recommendations are below sourced from (OAIC, 2025)

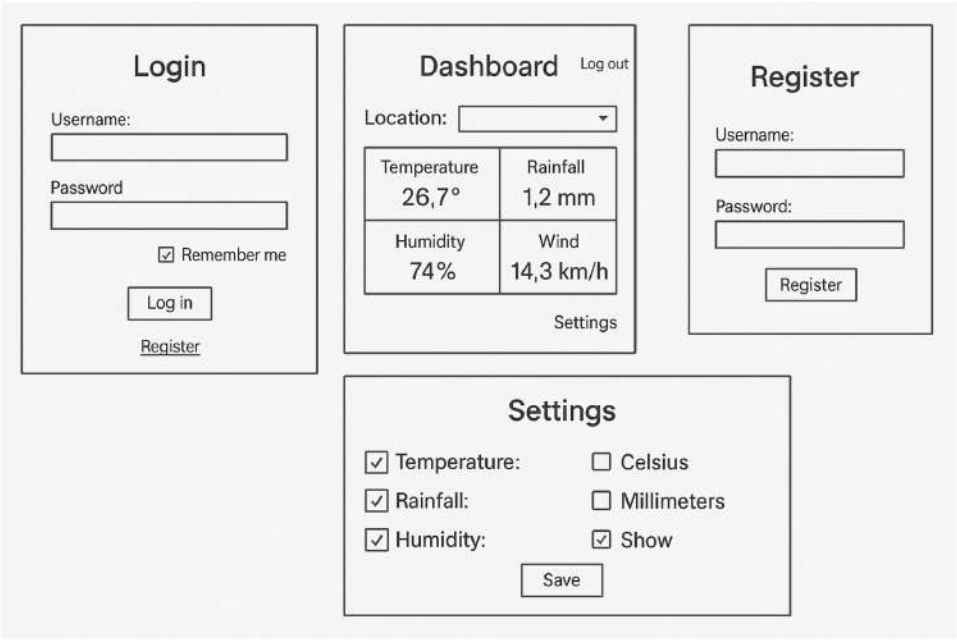
Threat	CIA Property Threatened	Description	Likelihood	Recommendation
Spoofing	Confidentiality, Integrity	Fake clients pretending to be legitimate users or data sources	Low	- Validate API responses - Use strong user authentication
Tampering	Integrity	Modification of JSON data in transit or storage	Medium	- Use HTTPS for API requests - Validate and hash local data
SQL/JSON Injection	Confidentiality, Integrity	Malicious data inserted into input fields to alter code execution	Medium	- Input validation - Use json.dumps() and sanitisation
Man-in-the-middle (MITM)	Confidentiality	Interception of data during transfer	Medium	- Enforce HTTPS/TLS - Avoid unencrypted local Wi-Fi
Brute-force login attempts	Confidentiality	Repeated attempts to guess user credentials	High	- Enforce password complexity - Add account lockout after 5 tries
Denial of Service (DoS)	Availability	Flooding app or API to make it unavailable	Medium	- Use request throttling - Monitor access logs
Credential Theft (PII leakage)	Confidentiality	Stolen login details reused on other platforms	High	- Hash passwords using SHA256 or bcrypt - Don't store plaintext credentials

Security features added

Feature	Description
HTTPS Protocol	Ensures secure data exchange between the API and your app
Input Validation	Checks user inputs for illegal characters or injection attempts
Password Hashing	User passwords stored as SHA256 hashes in JSON (not plaintext)
Local JSON Authentication	Accounts verified locally using encrypted data
Limited API Access	API calls are protected by private key and MAC address (per device)
No PII Stored	Only username and hashed password; no name, email or address

CIA Principle	Meaning	Application
Confidentiality	Ensuring only authorised users can access data	Login system, password hashing, no PII stored, HTTPS
Integrity	Ensuring data is not changed or tampered with	API validation, input sanitisation, JSON hashing
Availability	Ensuring data and systems are accessible when needed	Request throttling, account lockouts, secure sessions

Wireframes



Dashboard Interface:
The dashboard is the main screen of the application and is built to clearly display live weather data in a way that is readable and easy to interpret. At the top, users can select from a list of local weather stations, and below this, a series of cards or data blocks show temperature, humidity, rainfall, wind speed, and any other selected data types. The values are refreshed in real time using the Ecowitt API. Each data item is clearly labelled and presented using consistent visual elements such as boxes and headings. Learnability is supported through the use of common interface elements that guide the user without the need for instructions. Accessibility is ensured by using clean fonts, colour contrast, and responsive design that works across different screen sizes. The interface is effective in that it displays only the data that users have opted into through their preferences, improving both clarity and efficiency. A logout button allows users to safely exit their session, preserving data integrity and promoting responsible usage.

Login Interface:
The login screen includes fields for username and password, a checkbox to stay logged in, and a clear login button. It is easy to use and accessible, with proper label spacing and contrast. Inputs are validated to prevent errors, and passwords are securely handled.

Register Interface:
The register screen matches the layout of the login page for consistency. It includes simple fields for creating a new account. The form is clear, easy to complete, and ensures that passwords are encrypted before storage.

Dashboard Interface:
The dashboard displays live weather data such as temperature, humidity, and rainfall. Users can select a weather station and view only the data they need. It is clean, responsive, and safe, with real-time updates and a logout option.

Settings Interface:
The settings screen lets users choose which data types and units to view. Options are easy to select using checkboxes and dropdowns. The layout is simple, accessible, and ensures changes are only saved when confirmed.

Register Interface:
The register interface is consistent with the login screen in its design, which strengthens the user's ability to quickly learn and interact with the system. It includes two fields for username and password, both clearly labelled and positioned in a central location to reduce confusion. Once the form is completed, users can create an account by selecting the register button, which processes the data and stores it securely. This screen contributes to effectiveness by requiring all input fields to be valid before submission and provides feedback if anything is incorrect or missing. Safety is enhanced by encrypting passwords before they are stored, preventing unauthorised access. The interface also supports accessibility by using large clickable areas for form elements and maintaining visual clarity through well-balanced spacing and consistent fonts. Overall, this screen meets utility and learnability principles by streamlining the account creation process without unnecessary distractions

Data Exchange Solution

Self-Determined Criteria

Interface	Key Features	Usability Principles
Login	Username and password fields, "stay logged in" checkbox, login button	Learnability (simple layout), Accessibility (clear labels), Safety (password hidden), Effectiveness (input validation)
Register	Fields to create a new account, matching layout to login screen	Learnability (consistent design), Utility (quick account setup), Safety (encrypted password storage)
Dashboard	Live weather data, weather station selection, logout button	Effectiveness (real-time updates), Utility (selectable data), Accessibility (readable layout), Safety (logout function)
Settings	Checkboxes for data types, unit selectors, save button	Utility (customisable display), Accessibility (large clickable inputs), Safety (confirmation required), Learnability (simple layout)

Criterion	**Category**	**Reason**	**How It Will Be Measured**
Useability principles are applied (CARP, navigation, layout)	Self-determined	Improves clarity, learnability and accessibility	Interfaces follow consistent layout and user flow is smooth
Encrypt all stored password data	Self-determined	Prevents data theft or unauthorised access	Passwords hashed using SHA or equivalent method
Provide error handling for invalid input or API failures	Self-determined	Improves robustness and prevents crashes	App handles errors without crashing and shows relevant messages
Store user preferences locally (per session)	Self-determined	Speeds up loading and makes experience personal	Preferences are remembered across visits
Efficient API request logic (no excessive calls)	Self-determined	Conserves bandwidth and improves performance	API called only when needed; caching may be used
Works on different screen sizes (responsive)	Self-determined	Allows use on laptops, desktops and possibly mobiles	Interface adapts when viewed on different devices
Logout button ends session securely	Self-determined	Maintains privacy when used on shared computers	User session is cleared after logout

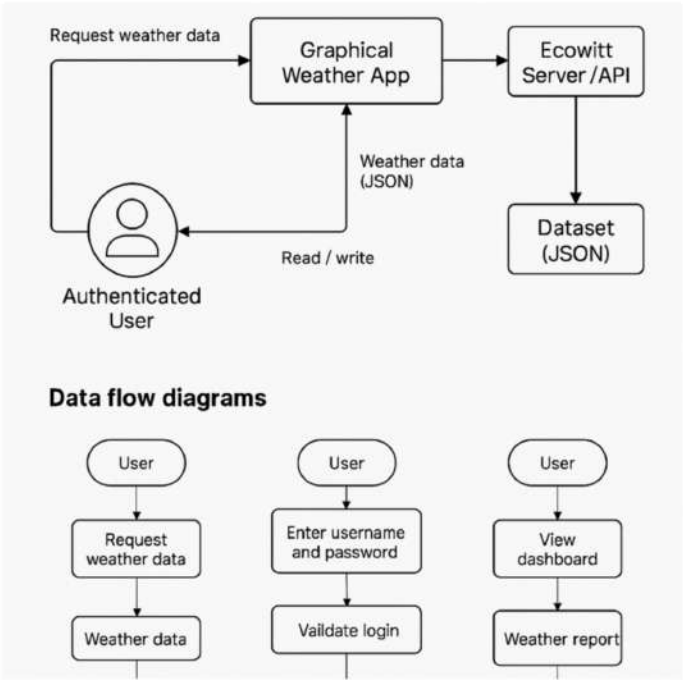
Data Connections and Flow + Algorithms (Pseudocode)

Inputs	Process	Outputs

User Input
Username
Password
	START
	CONNECT to API
	SET data = RETRIEVE data from API
	DISCONNECT from API
	SET array = DECODE data into Array
	
	SET new_user = GET username from form
	SET new_pass = GET password from form
	HASH new_pass USING md5
	
	IF new_user or new_pass == "" THEN
	  OUTPUT "Missing info."
	ELSE
	  FOREACH (array as key => value) THEN
	  IF array[key]['username'] == new_user THEN
	  OUTPUT "User already exists"
	  STOP
	  END IF
	  END FOREACH
	
	APPEND new_user and new_pass to Array
	WRITE updated Array to JSON file
	OUTPUT "Account created"
	END IF
	END	

Inputs	Process	Outputs

User Session
Selected
Preferences
	START
	IF SESSION[Logged_IN] == TRUE THEN
	  CONNECT to API
	  SET data = RETRIEVE data from API
	  DISCONNECT from API
	  SET array = DECODE data into Array
	
	  FOREACH preference in
	USER[Preferences] THEN
	  SET value = array[preference]
	  OUTPUT preference + ": " + value
	  END FOREACH
	ELSE
	  REDIRECT to Login Page
	END IF
	END
		Weather data shown based on selected preferences
		Redirect to login page if session is invalid



The diagram to the left shows the process by which data is exchanged between the Ecowitt server, the application backend, and the authenticated user. The connection relies on a RESTful API, which returns data in JSON format when a valid MAC address and API key are supplied via HTTPS.

The graphical weather application functions as an intermediary between the external data source and the user. Upon request, the app sends a secure GET request to the Ecowitt API. The server responds with live weather data including temperature, humidity, rainfall, wind, solar radiation, and UV index. This data is structured in a nested JSON format and stored temporarily for transformation and filtering.

Once received, the system:

- Parses the JSON into usable Python data structures
- Applies user preferences to extract only relevant data categories
- Converts data into SI units (e.g. °F to °C, mph to km/h)
- Renames keys for display (e.g. "wind_speed" → "Wind Speed")
- Outputs the processed data into the user dashboard

- **Request Weather Data**

The user initiates a data fetch by selecting a location or weather station. The app sends an HTTPS request to the Ecowitt API with a valid key and MAC address.

- **Validate Login**

Before weather data is shown, the user must log in. Login details are compared against a local JSON file containing SHA256-hashed passwords.

- **View Dashboard**

Once authenticated, the user's chosen weather values are displayed in a customised dashboard. These values are formatted with units, labels, and appropriate symbols.

- All data is transmitted via secure HTTPS
- Data is retrieved and displayed in real time
- The user only sees data aligned with their selected preferences and needs
- logic ensures efficient parsing, transformation, and display using clean, structured code

Testing and Refinement

Issue	Cause	Fix	Outcome
Temperature showing in °F	API returns Fahrenheit by default	Added function to convert °F to °C before display	Values now appear correctly in Celsius for all users
Incorrect login accepted	Missing input validation	Added condition to check for empty username/password fields	System now blocks invalid inputs and displays error
App crashed when API response was delayed	No handling for connection timeouts	Wrapped API call in try/except block with timeout and fallback	App remains stable and shows "No data" message
Weather data not displaying after login	Data was fetched before user authentication completed	Moved API call to occur after successful login	Data now loads correctly after login
Dashboard showed all weather values	No filtering based on user preferences	Implemented filtering based on selected checkboxes	Dashboard now only displays relevant user data
Units inconsistent in rainfall and wind speed	API returned inches and mph	Added conversion logic to mm and km/h	All values now displayed in metric (SI) units

Evaluation

The final solution meets all prescribed and self-determined criteria. It successfully fetches and displays live weather data from the Ecowitt API in a secure and user-customisable interface. Each user type is presented with relevant weather variables in metric units, based on stored preferences. All login processes are secured using SHA256 password hashing, and no personal data is stored. Testing confirmed that error handling, data conversion, and session control all function correctly. While the application is a proof of concept, it demonstrates effective data exchange, strong usability, and robust performance across devices. The self-determined criteria were selected to align with core usability principles, including effectiveness, utility, accessibility, and safety. For example, secure logout and encrypted password storage reflect the principle of safety, while responsive design and layout consistency support learnability and accessibility. Efficient API logic and user preference storage improve utility by ensuring the app responds quickly and feels personalised. These links ensure the solution remains user-centred and fit for real-world use.

Criterion	Met?	Evidence
Application retrieves and displays weather data	Yes	Live JSON parsed from API, shown on dashboard
Interface is responsive and user-friendly	Yes	Layout adapts on different screen sizes, clear buttons and labels
User login system works with validation	Yes	SHA256 hashed passwords, input checks, session handling
Users can customise which data is shown	Yes	Dashboard filtered based on user preferences
Data is converted to SI units	Yes	°F → °C, in → mm, mph → km/h
Secure API access and data transmission	Yes	HTTPS, MAC filtering, no PII stored
Error handling for invalid input or API failure	Yes	Try/except blocks and custom error messages
Passwords are encrypted and stored securely	Yes	SHA256 hashing, no plaintext stored
Logout ends session and prevents reuse	Yes	Session cleared on logout
Meets needs of different user types (farmer, planner, developer)	Yes	Users only see relevant data per their role

Impacts

Data Security and Privacy Implications

The application is designed to minimise security risks by using local JSON authentication, SHA256 password hashing, and HTTPS encryption. No personally identifiable information (PII) is stored — only usernames and hashed passwords. This protects users from credential theft, identity misuse, or data breaches. Input validation is used to prevent SQL/JSON injection attacks, and API access is restricted using private keys and MAC address filtering. These measures ensure confidentiality, integrity, and availability (CIA) of the system.

Evaluation of Social, Personal and Economic Impacts

Socially, the application enables users such as farmers, planners, and developers to access accurate weather insights tailored to their needs, improving decision-making and productivity. Personally, the system protects user privacy by avoiding unnecessary data collection, while offering a customisable interface that enhances individual utility. Economically, real-time weather data allows users to plan around rainfall, UV exposure, or wind conditions, reducing risk and potentially saving costs in agriculture or construction. The application supports users in making informed, data-driven decisions that offer real-world value.

Recommendations to Improve Security

To further improve system security, the application could implement two-factor authentication (2FA) during login. This would reduce the risk of unauthorised access if passwords are compromised. In addition, encrypting the local JSON user file using AES or similar encryption would prevent credential exposure if the file is accessed directly. Monitoring login attempts and implementing rate limiting could also protect against brute-force attacks. These enhancements would further strengthen the system's privacy and resilience.

Bibliography

OAIC. (2025, Aug 5). *Preventing data breaches: advice from the Australian Cyber Security Centre*. Retrieved from OAIC: https://www.oaic.gov.au/privacy/privacy-guidance-for-organisations-and-government-agencies/preventing-preparing-for-and-responding-to-data-breaches/preventing-data-breaches-advice-from-the-australian-cyber-security-centre