Software Design Document

Arduino-Based Smart Temperature Monitoring System

Ethan England, John Gahagan
Western Kentucky University

April 25, 2025

Revision History

Name	Date	Version	Reason for Changes
John Gahagan	2/5/2025	1.0.0	Initial Draft of Software Design Document.
Ethan England	2/5/2025	1.0.0	Initial Draft of Software Design Document.
John Gahagan	4/17/2025	1.0.1	Updated project design to remove TailwindCSS/React.js; aligned architecture to final Arduino, WebSocket, Firebase system.
John Gahagan	4/28/2025	1.0.2	Final major revision: corrected APIs used, updated database schema, matched code organization, improved testing plan, added internationalization features.

Table 1: Revision History

Name	Contribution	Version
John Gahagan	Wrote all major sections, created all diagrams, completed all backend/frontend/firmware code integrations, addressed criticism, corrected final document to match system implementation.	1.0.0–1.0.2
Ethan England	Provided edits for front-end documentation sections (2.2, 2.3, 2.4, 3.4) and contributed to minor UI adjustments.	1.0.0

Table 2: Contributor Details

Contents

List of Figures

1 System Architecture

The system follows a structured, layered architecture designed for real-time temperature monitoring, scalability, and modularity. It consists of four main layers: Device Layer, Communication Layer, Backend Layer, and Frontend Layer, each serving a specific function. The Device Layer is responsible for collecting temperature readings from the DHT11 sensor and controlling LED indicators based on predefined thresholds. The Communication Layer facilitates real-time data transmission using WebSockets, ensuring low-latency updates between the Arduino and the backend. The Backend Layer, powered by Firebase Authentication and Firebase Realtime Database, handles user authentication, access control, and data storage, ensuring system integrity. Finally, the Frontend Layer provides an interactive web dashboard, allowing users to view real-time temperature updates, configure threshold settings, and receive alerts when temperature levels exceed defined limits. This architecture ensures efficient real-time data flow, modular system scalability, and secure access management, making it ideal for monitoring applications. **See Figure 1: General Architecture Overview and Figure 2: System Architecture Overview at End of the Document.**

1.1 Architectural Decisions

The system follows a layered architecture to optimize scalability and maintainability. The chosen architecture is event-driven to support real-time communication and minimize resource usage.

1.2 Architecture Pattern

The system consists of:

- **Device Layer**: Arduino Uno R4 WiFi for sensor data collection.
- **Communication Layer**: WebSockets for real-time updates.
- **Backend Layer**: Firebase for authentication and database storage.
- **Frontend Layer**: Web dashboard for user interaction.

2 Overall Code Structure

The system is designed using a modular architecture, ensuring maintainability and scalability. The primary codebase consists of three main sections: hardware (Arduino firmware), web-based frontend, and backend communication.

2.1 Project Structure

The codebase is divided into three main parts:

- **Hardware (Arduino Firmware)**:
  - Written in C++ and executed on the Arduino Uno R4 WiFi.

  - Handles sensor data acquisition and LED control.
  - Communicates with the WebSocket server for real-time data updates.

- **Web Dashboard (Frontend)**:
  - Developed using HTML, CSS, JavaScript, and Firebase SDK.
  - Provides real-time temperature visualization and configuration controls.
  - Communicates with the backend via REST APIs and WebSockets.

- **Backend (Firebase + WebSockets)**:
  - Firebase handles authentication, database storage, and user session tracking.
  - WebSocket server maintains real-time communication with the frontend and Arduino.
  - Processes user-defined threshold settings and updates the device accordingly.

2.2 APIs Used

The system relies on various APIs for device communication, data storage, and real-time synchronization between components.

2.2.1 External APIs

- **Firebase Realtime Database API** (`https://firebase.google.com/docs/database`)
  - Provides a cloud-hosted NoSQL database for storing temperature logs, threshold settings, and session data.
  - Enables real-time synchronization between the backend and all connected clients.

- **Firebase Authentication API** (`https://firebase.google.com/docs/auth`)
  - Manages user login and registration using email/password credentials.
  - Enforces role-based access control (admin vs user) and guards sensitive Firebase paths.

- **WebSockets API** (`https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API`)
  - Establishes persistent, low-latency connections between the Arduino, backend server, and web dashboard.
  - Used for bi-directional data exchange such as temperature logs, LED state, and threshold updates.

2.2.2 Internal API Paths and Events

Instead of traditional REST endpoints, the backend uses structured Firebase paths and WebSocket messages for real-time communication.

- **Firebase Data Paths**
  - `/users/{uid}/thresholds` – Stores user-defined cold and hot threshold values.
  - `/users/{uid}/temperatureLogs` – Logs real-time temperature readings by user ID.
  - `/users/{uid}/ledStatus` – Stores latest LED state sent from Arduino.
  - `/users/{uid}/sessions/{sessionID}` – Tracks active user sessions and online status.
  - `/control/kick/{uid}` – Triggers admin-forced logout of specific users via WebSocket.
- **WebSocket Events**
  - `register-uid` – Maps a connected client's UID to their WebSocket instance.
  - `temperature` – Sent from Arduino to report a temperature reading.
  - `ledStatus` – Sent from Arduino to indicate the current LED state.
  - `threshold_update` – Sent from dashboard to update cold/hot thresholds.
  - `kick` – Sent from backend to notify a user they've been removed from the session.

2.3 Libraries Used

This system uses a set of libraries across the device firmware, backend server, and web dashboard to enable real-time communication, sensor readings, authentication, database storage, and user interface updates.

2.3.1 Arduino Libraries

These libraries are used within the Arduino firmware for temperature sensing, WiFi connectivity, and WebSocket communication.

- **DHT.h** - `https://github.com/adafruit/DHT-sensor-library`
  - Interfaces with the DHT11 sensor for periodic temperature measurements.
  - Handles sensor timing and data conversion internally.
- **WiFiS3.h** - Built into Arduino Core for Uno R4 WiFi
  - Provides WiFi network connectivity to allow real-time WebSocket communication.
- **ArduinoWebsockets.h** - `https://github.com/Links2004/arduinoWebSockets`
  - Manages WebSocket client connections to the backend server for real-time data transmission.

2.3.2 Backend Libraries

These libraries are used in the backend WebSocket server to handle real-time messaging, Firebase database operations, and authentication.

- **firebase-admin** - `https://firebase.google.com/docs/admin/setup`
  - Allows secure server-side access to the Firebase Realtime Database.
  - Handles user session management and data storage.
- **ws** - `https://github.com/websockets/ws`
  - Provides WebSocket server functionality for managing real-time bidirectional communication between Arduino devices, dashboard clients, and the server.
- **https (Node.js built-in module)**
  - Provides HTTPS server support if needed for future extensions, though the main WebSocket server runs on plain WebSocket (ws://).

2.3.3 Frontend Libraries

These libraries are used within the web-based dashboard to handle authentication, real-time data updates, and data visualization.

- **Firebase JavaScript SDK** - `https://firebase.google.com/docs/web/setup`
  - Enables user authentication (registration, login) and database access (thresholds, temperature logs) directly from the web dashboard.
  - Listens for real-time updates from the Firebase Realtime Database.
- **Chart.js** - `https://www.chartjs.org/`
  - Provides dynamic temperature charts displaying historical and real-time temperature trends in an intuitive graphical format.
- **Vanilla JavaScript / HTML / CSS**
  - The dashboard interface and styling are built using standard HTML, CSS, and JavaScript without any frontend frameworks such as React or TailwindCSS.

2.4 Code Organization

The codebase is structured for modularity, maintainability, and ease of debugging. It is divided into three main sections: Arduino firmware, WebSocket server backend, and web dashboard frontend.

- **Arduino Firmware**:
  - `ArduinoTempMonitor.ino` – Single firmware file responsible for:
    - Reading temperature values from the DHT11 sensor.
    - Managing WiFi connectivity using WiFiS3 library.

    - Maintaining WebSocket client connection to the backend.
    - Updating LED states (Red, Green, Blue) based on temperature thresholds.
    - Parsing incoming threshold updates from WebSocket messages.

- **WebSocket Server (Backend)**:
  - `server.js` – Node.js WebSocket server handling:
    - Real-time communication with Arduino and frontend dashboard.
    - Authentication verification and UID registration.
    - Data relaying to Firebase Realtime Database (temperature logs, LED status, sessions).
    - Admin session control logic (kick command handling).
  - `firebase-service-account.json` – Secure service credentials for Firebase Admin SDK.
  - `package.json`, `package-lock.json` – Node.js project dependency management.
  - `tests/` – Folder containing WebSocket and Firebase connectivity test scripts for CI/CD.

- **Web Dashboard (Frontend)**:
  - `index.html` – Live dashboard displaying real-time temperature and LED status.
  - `threshold.html` – Threshold monitor page allowing users to update hot/cold limits.
  - `chart.html` – Historical temperature chart with live and daily view modes.
  - `admin.html` – Admin panel for viewing users, sessions, and forcing user disconnects.
  - `login.html` and `register.html` – Authentication pages for user login and signup.
  - `app.js` – Manages WebSocket client connection, data updates, and frontend interactions.
  - `auth.js` – Handles Firebase Authentication, session tracking, and role-based access control.
  - `firebase-config.js` – Contains Firebase project initialization and configuration keys.
  - `styles.css` – Provides consistent styling across all dashboard pages.
  - `lang/` – Folder containing JSON translation files for English and Italian localization (`en.json`, `it.json`, `i18n.js`).
  - `tests/` – Folder for frontend testing documentation and test scripts.

3 Component Design

3.1 Device Layer

The Device Layer is responsible for collecting temperature data, controlling LED indicators, and ensuring seamless communication with the Communication Layer. **See Figure 3: Device Layer Diagram - Specific Hardware Connections and Components and Figure 4: Device Layer Diagram - Hardware Connections and Components at End of the Document.**

3.1.1 Responsibilities

- Read temperature data from the DHT11 sensor at regular intervals.
- Process the sensor data and determine temperature classification (cold, normal, hot).
- Control LED indicators based on user-defined thresholds.
- Transmit temperature readings to the Communication Layer via WebSockets.
- Receive threshold updates from the backend and modify behavior accordingly.

3.1.2 Hardware Components

- **A**rduino Uno R4 WiFi: Microcontroller responsible for sensor data processing and network communication.
- **D**HT11 Sensor: Measures temperature and provides readings to the Arduino.
- **LED Indicators**:
  - **Red LED**: Signals that the temperature exceeds the hot threshold.
  - **Green LED**: Indicates that the temperature is within the normal range.
  - **Blue LED**: Signals that the temperature is below the cold threshold.
- **Power Source**: USB or external **9V-12V adapter**.

3.1.3 Data Flow

1. The **Arduino Uno R4 WiFi** reads temperature from the DHT11 sensor every 2 seconds.
2. The system processes the temperature value:
   - If temperature **is too high**, turn **Red LED ON**.
   - If temperature **is normal**, turn **Green LED ON**.
   - If temperature **is too low**, turn **Blue LED ON**.
3. The Arduino transmits temperature data to the Communication Layer using WiFi/WebSockets.
4. If a user updates temperature thresholds from the web dashboard, the Arduino receives new threshold values and updates its logic accordingly.

3.2 Communication Layer

The Communication Layer is responsible for establishing real-time bidirectional communication between the Device Layer and the Backend Layer, ensuring seamless data transmission. **See Figure 5: Communication Layer Diagram - Data Flow and WebSockets Integration at End of the Document.**

3.2.1 Responsibilities

- Establish and maintain a WebSocket connection for real-time data exchange.
- Transmit temperature readings from the Device Layer to the Backend Layer.
- Receive threshold adjustments from users and forward them to the Device Layer.
- Ensure reliable communication between the Arduino and the Web Dashboard.
- Handle connection failures and re-establish communication if needed.

3.2.2 Communication Protocol

- **WebSockets**: The system utilizes WebSockets to enable low-latency, real-time communication.
- **WiFi Connectivity**: The Arduino connects to the WebSocket server over a wireless network to ensure fast data transmission.
- **Event-Driven Messaging**: Instead of periodic polling, WebSockets trigger real-time updates only when new data is available.

3.2.3 Data Flow

1. The Arduino Uno R4 WiFi connects to the WebSocket server upon startup.
2. The Arduino reads the temperature from the DHT11 sensor and sends the data to the WebSocket server every 2 seconds.
3. The WebSocket server processes the data and forwards it to the Backend Layer for storage and real-time updates.
4. The Web Dashboard, which is subscribed to WebSocket events, receives updated temperature readings in real-time.
5. If a user modifies the temperature thresholds, the Web Dashboard sends the updated values to the WebSocket server.
6. The WebSocket server transmits the new threshold values to the Arduino, which updates its logic accordingly.

3.3 Backend Layer

The Backend Layer is responsible for managing authentication, data storage, and access control while ensuring secure communication between the system components. **See Figure 6: Backend Layer Diagram - Firebase Authentication and Data Management, Figure 7: Firebase Project Dashboard, Figure 8: Authentication System - Email/Password Enabled, and Figure 9: Realtime Database Configuration at End of the Document.**

3.3.1 Responsibilities

- Authenticate users and enforce role-based access control.
- Store and manage temperature logs, user preferences, and session data.
- Ensure secure and efficient data communication between the Web Dashboard and the Device Layer.
- Process user-defined threshold updates and relay them to the Device Layer.
- Enable administrators to monitor and manage active user sessions.

3.3.2 Authentication and Access Control

- **Firebase Authentication**: Users log in using email and password authentication.
- **Role-Based Access Control** :
  - Regular users can view temperature data and adjust their own thresholds.
  - Admin users have elevated privileges, allowing them to monitor and manage user sessions.
- Authentication tokens are required for all interactions to prevent unauthorized access.

3.3.3 Session Management

- Each active user session is recorded in the backend.
- Administrators can view active sessions and terminate them if needed.
- Users who exceed inactivity limits are automatically logged out for security.

3.3.4 Multi-User Access Control

To prevent conflicts when multiple users attempt to modify the system at the same time, a structured access control mechanism is implemented.

Hybrid Approach Between a Queue and Last Write Wins

- Users request control of the device upon logging in.
- If no other user is actively modifying the system, they gain control immediately.
- If another user is active, they enter a queue and are notified when it is their turn.
- The system applies a *last-write-wins* rule, where settings are updated based on the most recent change.
- The Web Dashboard displays information on the currently active user.
- Administrators have override privileges and can force control if necessary.

This approach ensures that:

- Only one user at a time can modify device settings, preventing conflicting threshold updates.
- Users are aware of their position in the queue and are notified when they gain control.
- The system remains responsive by immediately applying the latest changes while maintaining access fairness.

3.3.5 Data Flow

1. A user logs in via Firebase Authentication.
2. If authentication is successful, Firebase assigns the appropriate role (user/admin).
3. The Web Dashboard retrieves and displays the user's stored threshold values.
4. The Web Dashboard listens for real-time temperature updates from the Backend Layer.
5. If a user modifies their threshold settings, the Web Dashboard updates the database.
6. The WebSocket server pushes updated settings to the Device Layer.

3.4 Frontend Layer

The Frontend Layer includes a web dashboard that provides real-time temperature monitoring and allows users to configure temperature thresholds. **See Figure 10: Frontend Layer Diagram - Web Dashboard UI Flow and Figure 11: Admin Panel at End of the Document.**

3.4.1 Responsibilities

- Display real-time temperature data retrieved from the Backend Layer.
- Allow users to configure hot and cold temperature thresholds.
- Provide authentication and role-based access control for users and administrators.
- Ensure real-time updates using WebSockets for seamless interaction.
- Enable administrators to monitor user sessions and enforce security policies.

3.4.2 User Interface Components

See Corresponding Diagram at End of the Document.

- **Login Page**: The login page has four intractable components. The user must enter username, password with an existing account. If the user clicked forgot password the user will get sent a link to the email of their existing account. If the user clicks signup the user must enter in a valid email, type in a password and then confirm said password. A account will then be created and the user will be able to login using this very account.
  - Users log in using their registered email and password.
  - Authentication is handled via Firebase Authentication.
  - Role-based access control determines available functionalities.
- **Temperature Display Panel**: The display panel will show the current LED color based on the threshold. It also showcases the temperature number, and three buttons. One is to log out when clicked. Another is current temperature which navigates to the current temperature panel. This will not do anything besides refreshed if current in said panel. The threshold monitor will lead to another panel.
  - Shows real-time temperature readings from the Device Layer.
  - Uses color-coded indicators for different temperature ranges.
- **Threshold Configuration Panel**: This panel showcases how you would like to update the threshold. A fill in box along with a slider will be provided for the user to adjust based on their preferences. It is important to now that the logic must follow HOT ¿ Just Right ¿ COLD or an error will be displayed. This panel also has three buttons on top. One to log out, a button current temperature to go back to the current temperature panel, and a threshold monitor which will navigate you to the threshold monitor. If the user is already in said panel it will simply refresh page.
  - Users can input and update their preferred hot and cold thresholds.
  - Changes are sent to the Backend Layer and relayed to the Device Layer.
- **Admin Dashboard**: This panel is only available to ADMIN(s). The panel will showcase users,their sessions,and has access to the historical temperature data collected. It will also have logout button for the purpose of logging out for the admin.

- Displays a list of active user sessions.
- Allows administrators to terminate user sessions if necessary.
- Provides access to historical temperature data for analysis.

3.4.3 Data Flow

1. A user logs into the Web Dashboard using Firebase Authentication.
2. Upon successful login, the dashboard retrieves and displays the user's threshold settings.
3. The dashboard subscribes to real-time temperature updates via WebSockets.
4. Temperature readings are displayed dynamically without requiring page refreshes.
5. If a user updates their threshold settings, the changes are sent to the Backend Layer.
6. The Backend Layer stores the updated values and notifies the Device Layer.
7. The Device Layer adjusts its behavior based on the updated thresholds.

4 Database Design

The Firebase Realtime Database stores user profiles, temperature logs, and system settings, ensuring seamless data retrieval and updates. **See Figure 12: Entity Relationship Diagram at End of the Document. Firebase Related Figures Included as Well (7,8,9)**

4.1 Responsibilities

- Store user authentication credentials and enforce role-based access control.
- Maintain historical temperature logs for analytics and monitoring.
- Store user-defined hot and cold temperature thresholds.
- Track active user sessions for session management and security.
- Manage real-time queue control to prevent concurrent modifications to threshold settings.
- Provide real-time synchronization between the Backend Layer and Frontend Layer.

4.2 Database Technology

- **Firebase Realtime Database**:
  - Ensures real-time data synchronization between all system components.
  - Allows for structured, cloud-hosted NoSQL storage with low latency.
  - Supports automatic updates to all connected clients without polling.

- **Security Rules and Access Control**:
  - Role-based access control ensures that users can only modify their own data.
  - Admins have elevated privileges to monitor and manage user activity.
  - Firebase enforces authentication-based read and write permissions.

4.3 Database Schema

See Corresponding Diagram at End of the Document. The user table structure was revised to include **essential details such as usernames, session tracking, and threshold settings** inside the user object to improve efficiency and ensure seamless data retrieval.

```
{
  "users": {
    "userID_123": {
      "email": "user@example.com",
      "role": "admin",  // Role-based access (admin/user)
      "username": "JohnDoe",
      "createdAt": "2025-02-17T12:00:00",
      "lastLogin": "2025-02-20T09:30:00",
      "activeSession": {
        "isActive": true,
        "sessionStart": "2025-02-20T09:30:00"
      },
      "thresholds": {
        "coldThreshold": 18.0,
        "hotThreshold": 28.0
      }
    }
  },
  "temperatureLogs": {
    "logID_001": {
      "userID": "userID_123",
      "timestamp": "2025-02-17T12:00:00",
      "temperature": 22.5
    }
  },
  "sessions": {
    "sessionID_001": {
      "userID": "userID_123",
      "active": true,
      "lastActivity": "2025-02-17T12:00:00"
    }
  },
  "controlQueue": {
    "activeUserID": "userID_123",  // The user currently modifying thresholds (1-to-1
    "timestamp": "2025-02-17T12:00:00",
```

```
    "waitingList": ["userID_456", "userID_789"]  // Users waiting for access (M:1)
  }
}
```

4.4 Data Flow

1. A user logs into the system, and Firebase retrieves their authentication credentials and role.
2. The Web Dashboard queries the database for the user's stored threshold values.
3. The Web Dashboard listens for real-time temperature logs, updating as new data is received.
4. If a user updates their threshold settings:
   - The system checks if they are the 'activeUserID' in 'controlQueue'.
   - If they are active, their new values are stored in Firebase.
   - If another user is active, they are placed in the 'waitingList' until the active user finishes.
5. Firebase pushes the updated settings to all connected devices, including the Arduino.
6. The Arduino retrieves the new threshold values and updates its internal logic.

4.5 Security and Access Control

- **Role-Based Access Control**:
  - Regular users can view their own data but cannot modify other users' information.
  - Admins can view and manage all users, including modifying temperature thresholds and monitoring sessions.
- **Authentication Enforcement**:
  - Only authenticated users can read or write data.
  - Sessions are tracked, and inactive users are logged out automatically.
- **Secure Data Transmission**:
  - All communication between Firebase and clients is encrypted.
  - Data validation ensures that only legitimate updates are accepted.
- **Concurrency Control in Threshold Updates**:
  - Only the 'activeUserID' in 'controlQueue' can modify temperature settings.
  - If a different user attempts to modify, they are added to 'waitingList'.
  - When the active user finishes, the next user in 'waitingList' is assigned 'activeUserID'.

5 Non-Functional Requirements

This section defines the non-functional requirements that influence the quality, usability, and maintainability of the system.

5.1 Usability

- The Web Dashboard must have an intuitive and user-friendly interface.
- The system should provide clear visual feedback using LED indicators for different temperature thresholds.
- Users should be able to log in, view real-time temperature data, and modify settings with minimal effort.
- The Web Dashboard must be accessible on both desktop and mobile browsers.

5.2 Performance

- The system must process and display temperature updates in real-time with a latency of less than 1 second.
- WebSockets will be used to reduce polling overhead and ensure instant synchronization.
- The backend must support at least 100 concurrent users without performance degradation.
- The Arduino should respond to new threshold updates within 2 seconds.

5.3 Security

- Firebase Authentication must be used to prevent unauthorized access.
- Only authorized users can modify their own settings, while admins have system-wide privileges.
- All communication between the Arduino, backend, and frontend must be encrypted.
- The system should enforce session expiration and automatically log out inactive users.

5.4 Maintainability

- The codebase should be modular, ensuring ease of updates and modifications.
- All components should follow standard programming best practices.
- Documentation should be provided for APIs and system functionality.
- The database schema should allow easy expansion for future features.

5.5 Scalability

- The backend should support future expansion, including additional sensors or devices.
- Firebase should be optimized to handle increasing data loads efficiently.
- The system should be designed to work efficiently with more concurrent users in the future.

6 Testing Plan

To ensure system reliability, security, and performance, a structured testing approach is applied. This plan includes unit tests, integration tests, system-level testing, and automated testing using GitHub Actions.

6.1 Unit Tests

- **Temperature Sensor Accuracy:** Verify that the DHT11 sensor correctly reads and transmits temperature data.
- **Threshold Updates:** Ensure users can set valid hot/cold thresholds, which are properly stored in Firebase.
- **WebSocket Data Handling:** Confirm that WebSocket messages correctly send and receive temperature readings.

6.2 Integration Tests

- **User Authentication & Role-Based Access:**
  - Ensure users can log in/logout and their roles (admin/user) are enforced correctly.
  - Verify that only admins can modify other user settings.
- **Database Interaction:**
  - Validate that user data is securely stored in Firebase.
  - Ensure threshold updates trigger real-time database changes.
- **WebSocket Communication:**
  - Verify real-time updates from the Arduino to the Web Dashboard.
  - Ensure WebSockets remain stable under high load.

6.3 System Tests

- **Concurrent User Sessions:**
  - Simulate multiple users logging in simultaneously and verify the controlQueue logic prevents race conditions.
  - Ensure only the `activeUserID` can modify thresholds.
- **End-to-End Temperature Monitoring:**
  - Test full data flow: sensor data → WebSocket server → Firebase → Web Dashboard.
  - Ensure real-time updates are correctly displayed on the UI.
- **Security & Data Protection:**
  - Ensure unauthenticated users cannot access Firebase data.
  - Verify that role-based security prevents unauthorized modifications.

6.4 Mobile Responsiveness Testing

To ensure the system is accessible across different devices, the web dashboard will be tested for **mobile compatibility** using the following methods:

- **Responsive Web Design (RWD)**: Implementing *flexible layouts* with *CSS media queries* to adjust UI elements dynamically.
- **Cross-Device Testing**: Testing on *smartphones, tablets, and desktops* to ensure consistent performance across different screen sizes.
- **Performance Testing**: Optimizing WebSockets and Firebase interactions for *low-latency mobile connections* to improve data synchronization.
- **Viewport Adjustments**: Ensuring charts and temperature data are displayed correctly on *smaller screens* without horizontal scrolling.

6.5 Automated Testing with GitHub Actions

To ensure continuous integration and automated testing, the system will utilize **GitHub Actions** for CI/CD. The workflow includes:

- **Unit Testing Automation:**
  - Execute Firebase database tests using Jest.
  - Run WebSocket data handling tests to verify real-time communication.
- **Deployment Automation:**
  - Deploy the frontend to Firebase Hosting upon a successful build.
  - Upload firmware updates to Arduino if a new version is detected.

By integrating GitHub Actions, this project ensures **automated testing, continuous monitoring, and rapid deployment**, improving reliability and maintainability.

7 Conclusion

This document provides a comprehensive overview of the system's design, focusing on architecture, component interactions, security, and performance. It ensures that developers have a clear roadmap for implementation and future enhancements.

7.1 Future Considerations and Expansions

- **Humidity Monitoring**: Since the DHT11 sensor measures both temperature and humidity, future updates could incorporate humidity tracking within the application. This would provide users with more comprehensive environmental data.
- **Predictive Mode for Temperature Estimation**: If enough historical data is available, a predictive model could estimate temperature trends based on past readings. This would serve as a fallback in cases where:
  - The DHT11 sensor is malfunctioning or unavailable.
  - A user cannot access the device because another user is actively using it.

  By analyzing previously recorded temperature patterns, the system could predict expected values and notify users whether they are viewing predicted data or real-time readings. The application would display a notification indicating whether it is in **Predict Mode** or **Actual Mode**. One easy way we can put this in the project is by using the a cloud provided service named Firebase ML. **Firebase Machine Learning** (`https://firebase.google.com/docs/ml`)

8 Diagrams

The following diagrams provide a visual representation of the system architecture, database schema, and user interface structure. These diagrams serve to clarify component interactions and data flow within the system.

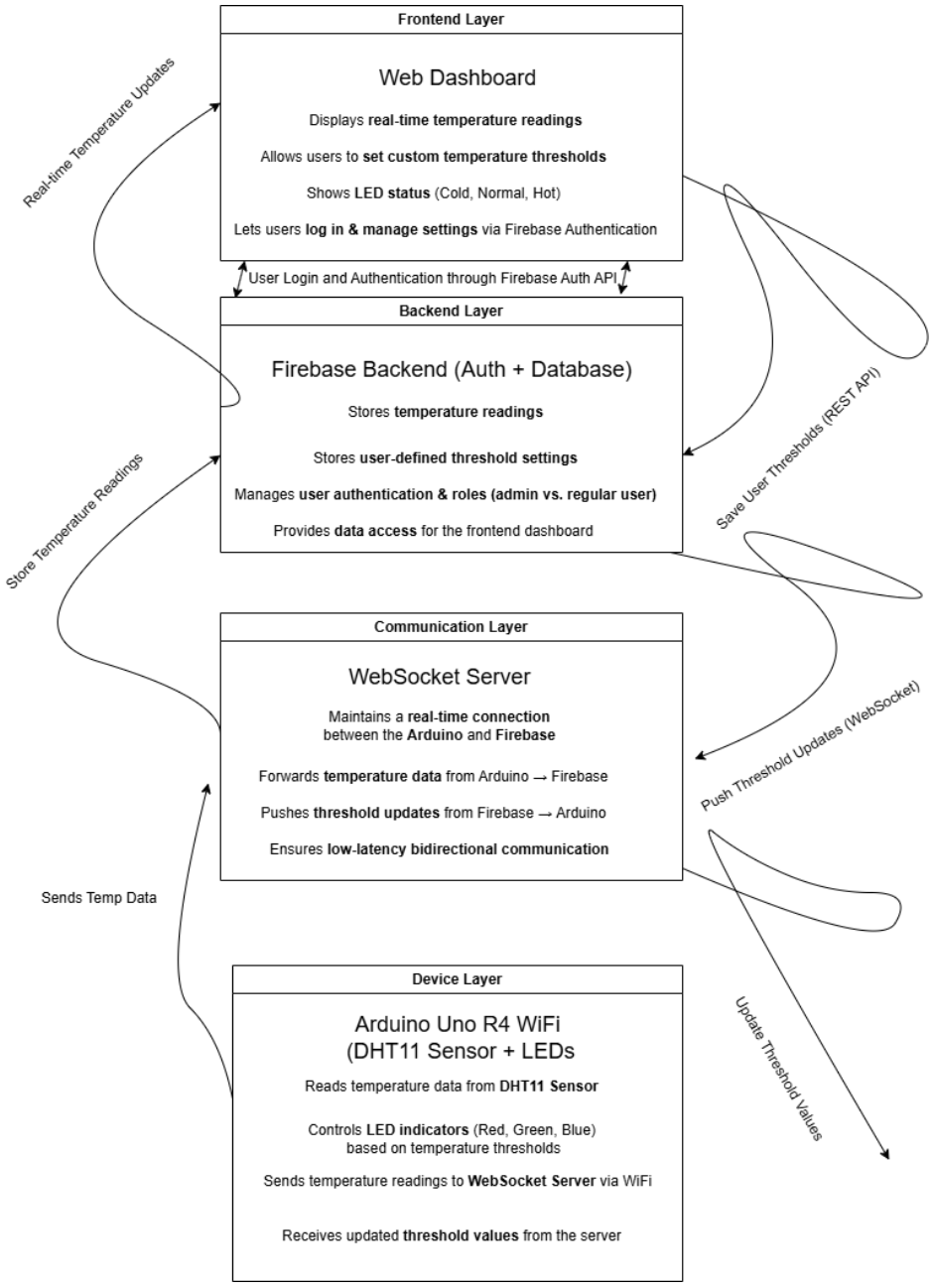

Figure 1: General Architecture Overview

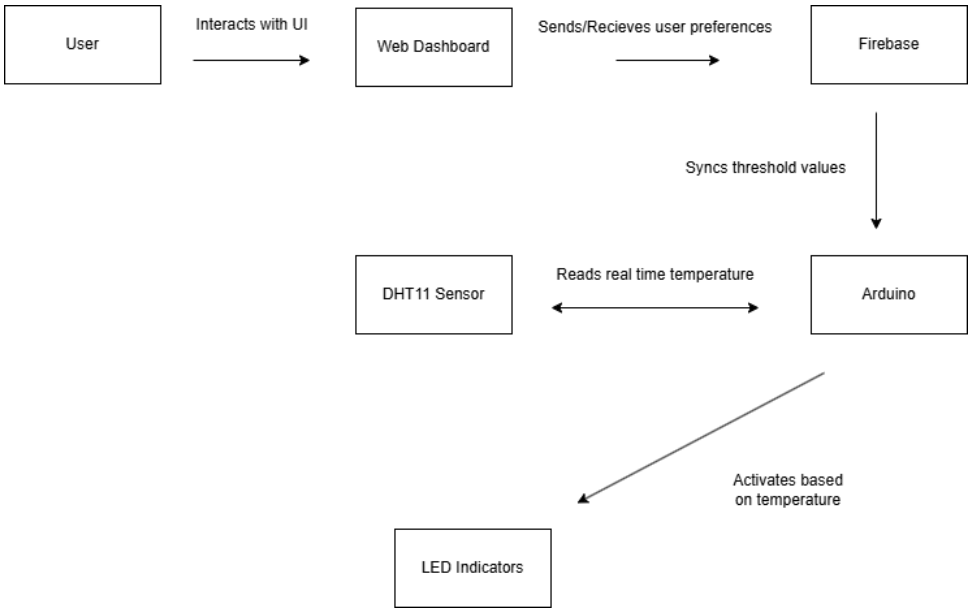

Figure 2: System Architecture Overview

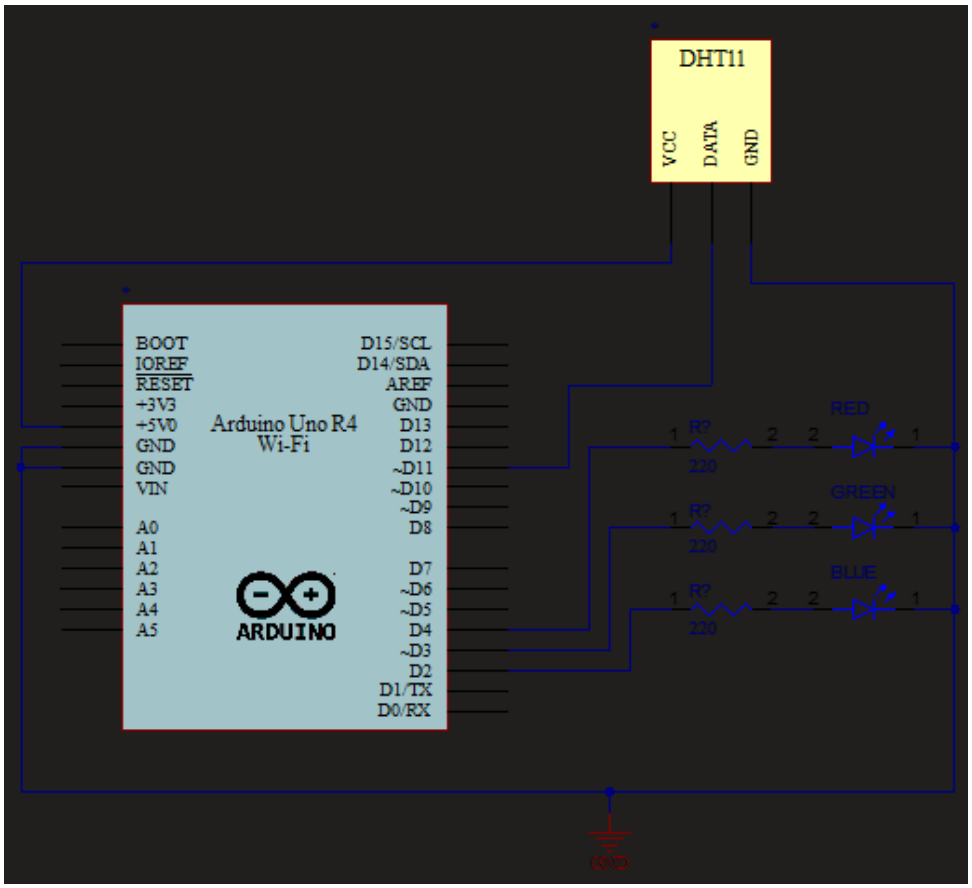

Figure 3: Device Layer Diagram - Specific Hardware Connections and Components

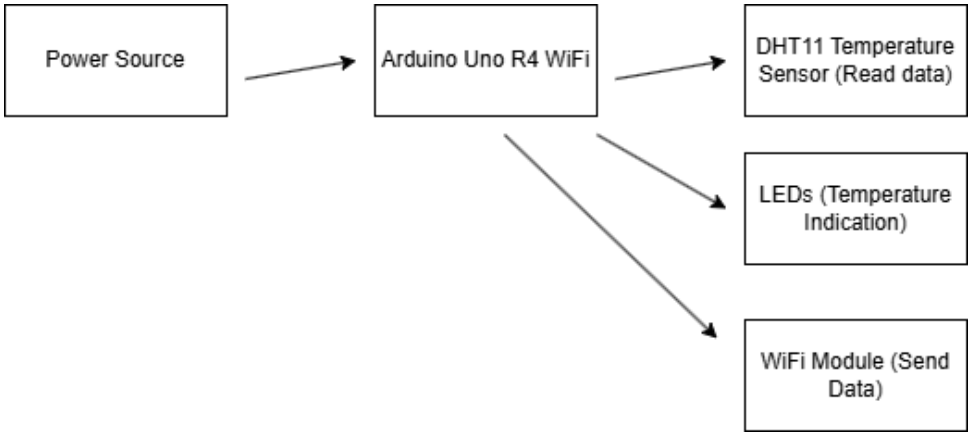

Figure 4: Device Layer Diagram - Hardware Connections and Components

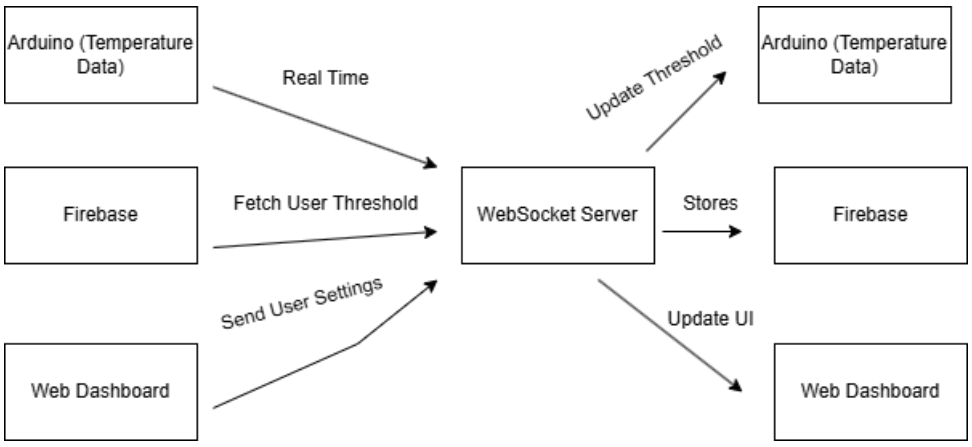

Figure 5: Communication Layer Diagram - Data Flow and WebSockets Integration

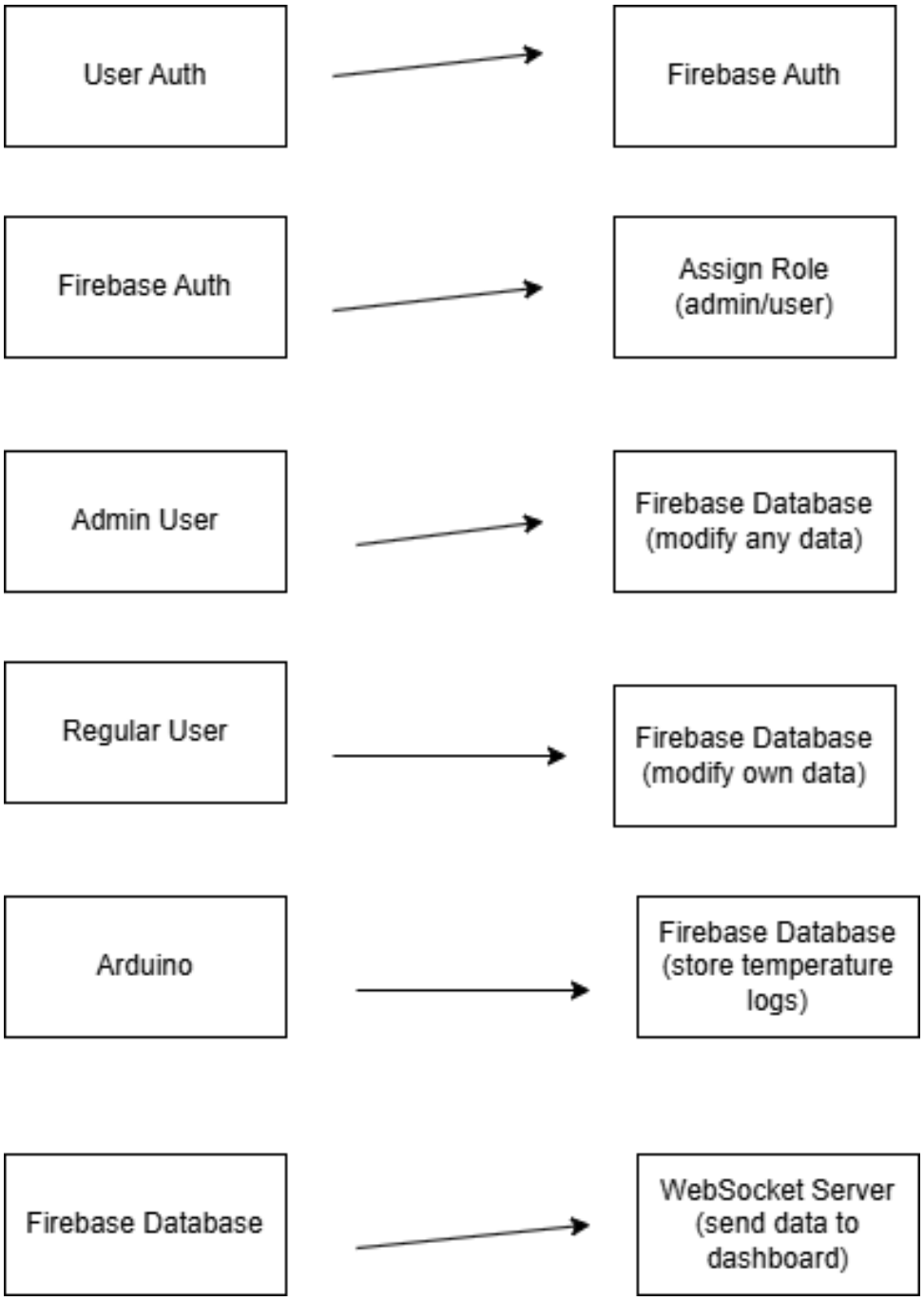

Figure 6: Backend Layer Diagram - Firebase Authentication and Data Management

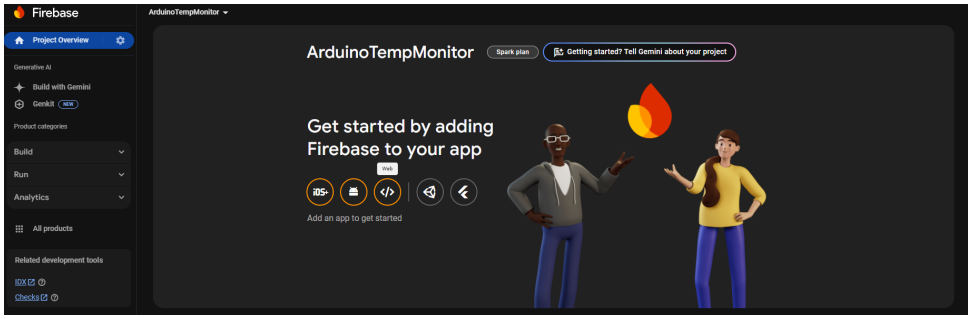

Figure 7: Firebase Project Dashboard

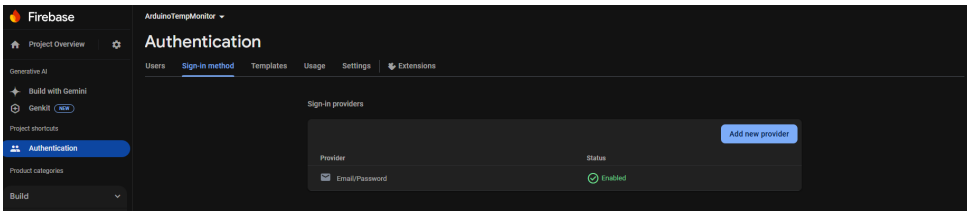

Figure 8: Authentication System - Email/Password Enabled

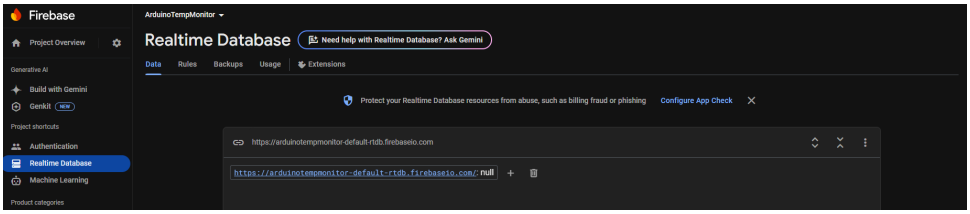

Figure 9: Realtime Database Configuration

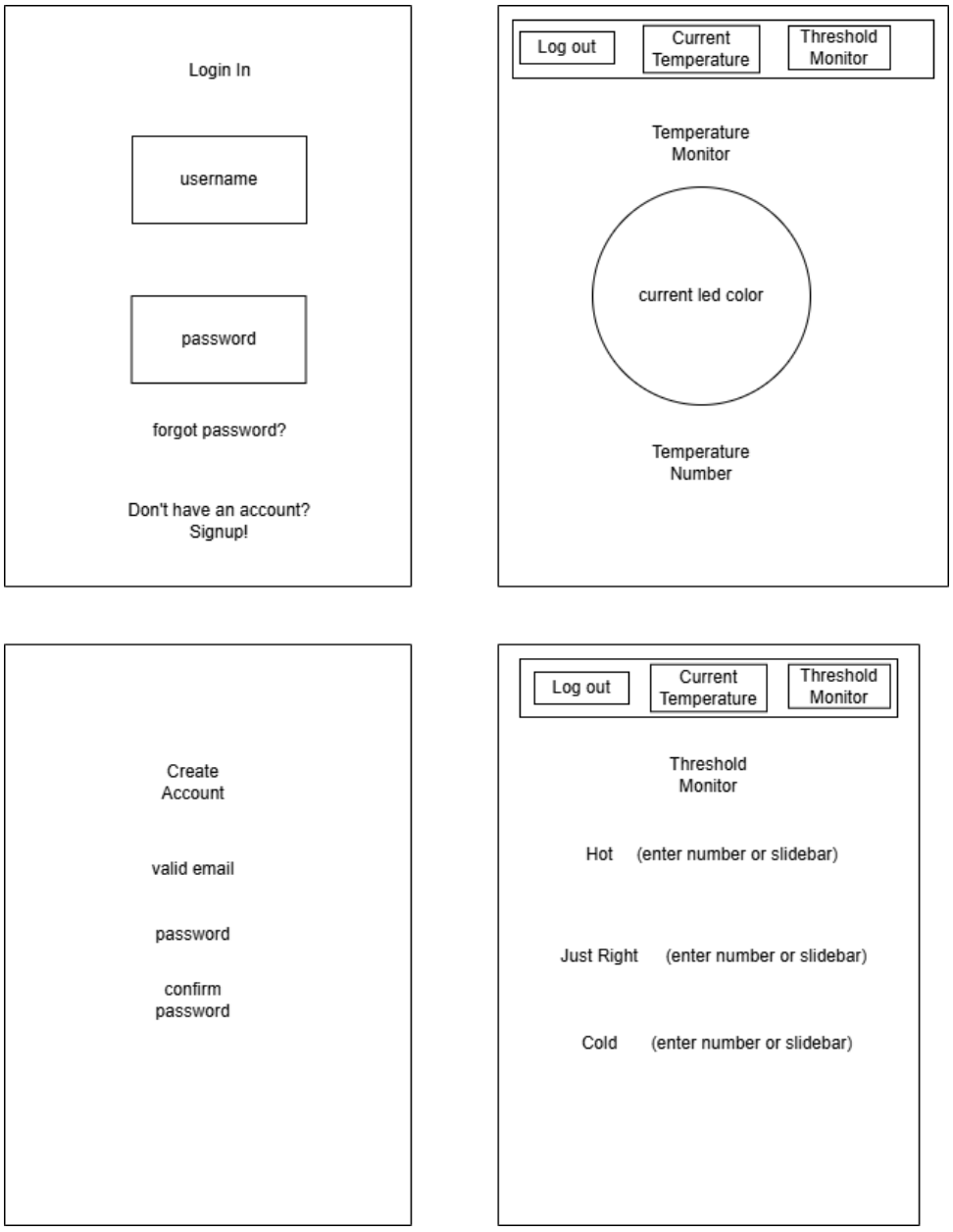

Figure 10: Frontend Layer Diagram - Web Dashboard UI Flow

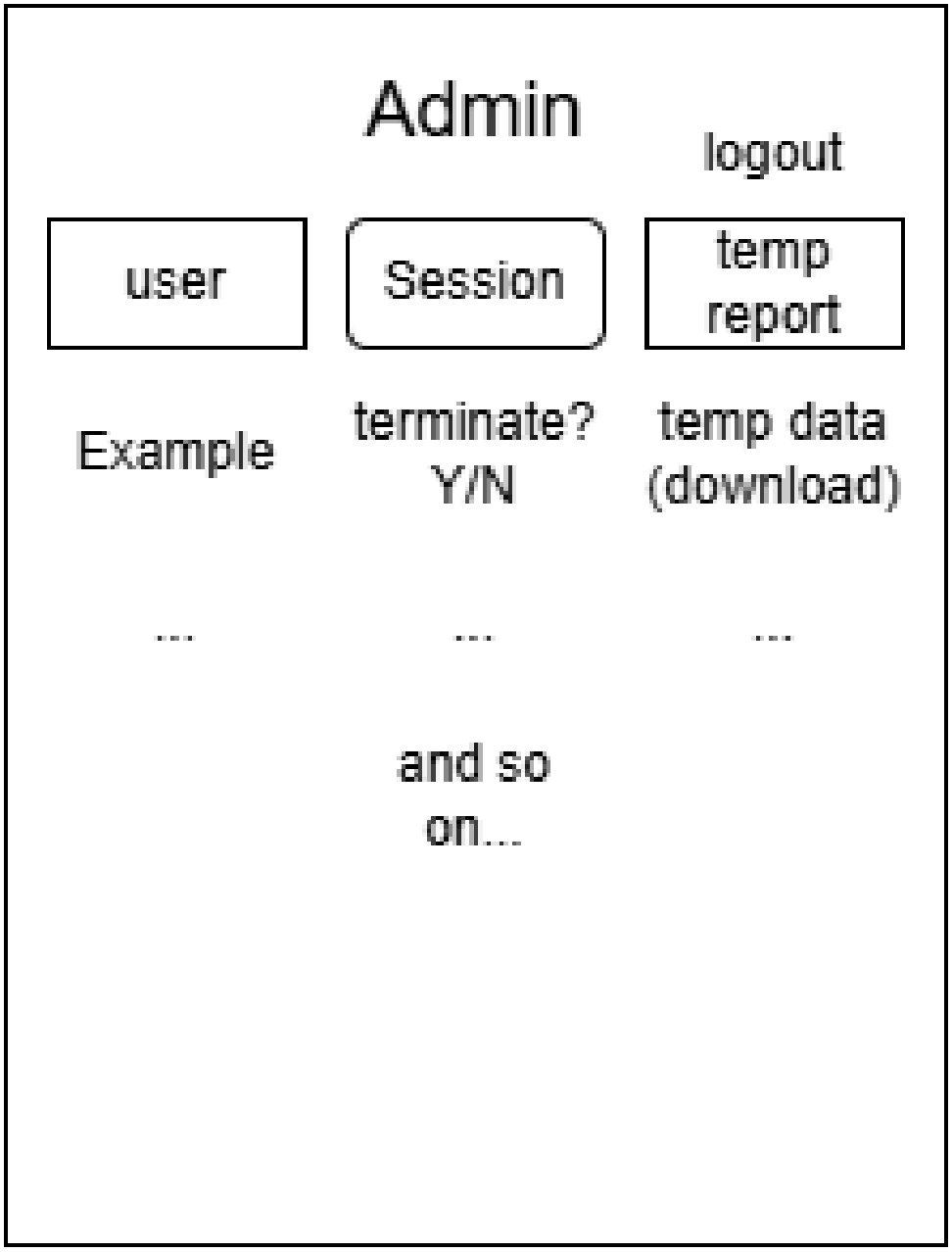

Figure 11: Admin Panel

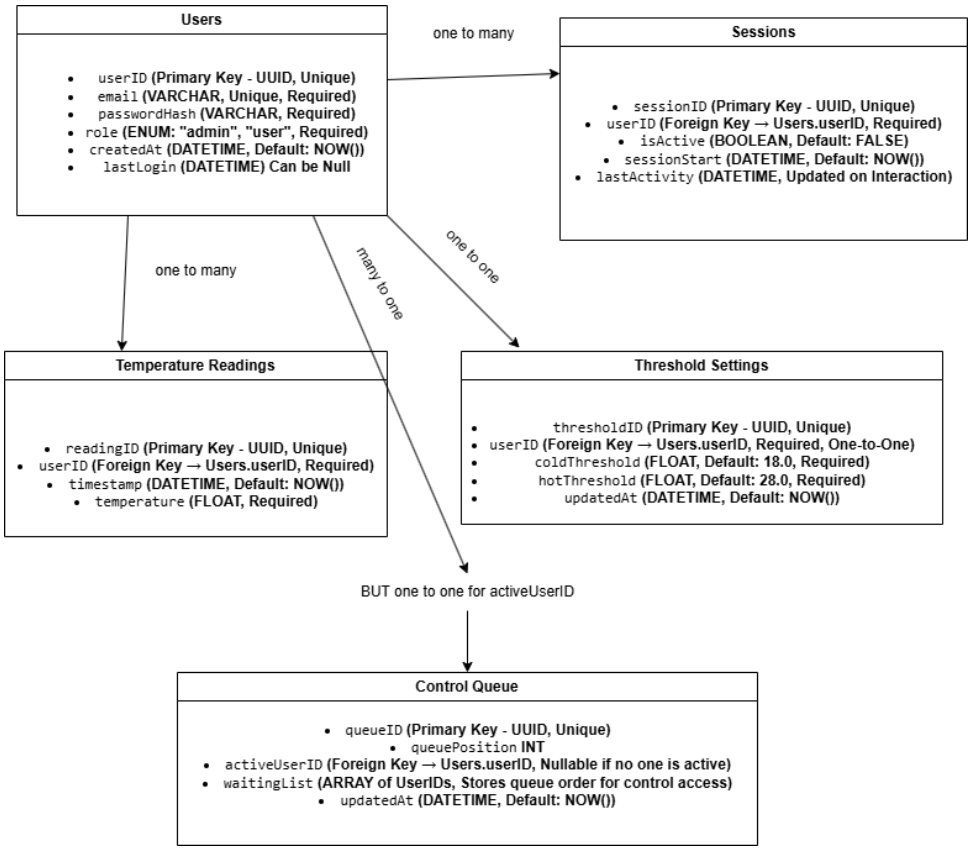

Figure 12: Entity Relationship Diagram