Snowboarding Wearable SDD

https://github.com/Egank2/Snowboard-Wearable.gi

April 19, 2025

Contents

1 Introduction

1.1 Document Purpose

This document provides a comprehensive software design for the Snowboarding Wearable project. It details the architecture, system components, and design decisions that satisfy the system requirements defined in the SRS. The purpose is to serve as a guide for implementation and future maintenance while ensuring alignment with user needs and technical constraints.

1.2 Product Overview

The Snowboarding Wearable is a wearable IoT device for snowboarders that attaches to the board and connects to a phone via Bluetooth. It collects data from sensors like a gyroscope, accelerometer, GPS, and altimeter, providing post-ride analysis such as speed, airtime, technique analysis, and trick recognition. It can analyze snowboard-specific dynamics like carving angles, board flex, and snow conditions. Ride visualization, performance grading, and tips to help riders improve and tailor their experience to different terrains, making it an essential tool for snowboarders of all levels.

1.2.1 **Problem Statement:** Snowboarders lack a dedicated tool for analyzing ride performance and improving techniques.

1.2.2 **Proposed Solution:** A wearable IoT device coupled with a mobile app that provides real-time feedback and post-ride analytics.

1.2.3 **Novelty:** Focus on snowboard-specific dynamics (carving angles, edge angles, posture) with advanced trick recognition and performance grading.

1.3 Product Functionality

1.3.1 **Real-Time Data Collection:** Continuously capture sensor data (gyroscope, accelerometer, GPS, altimeter).

1.3.2 **Trick Recognition:** Identify snowboard tricks using predefined movement signatures.

1.3.3 **Performance Grading:** Analyze parameters such as carving efficiency, airtime, and landing stability.

1.3.4 **Data Visualization:** Present 3D ride visualizations and detailed performance graphs on the mobile app.

1.3.5 **Historical Data Analytics:** Store ride history for long-term progress tracking.

1.3.6 **Wireless Communication:** Ensure seamless Bluetooth connectivity between the wearable and mobile app.

1.4 Definitions

1.4.1 **IoT:** Internet of Things

1.4.2 **BLE:** Bluetooth Low Energy

1.4.3 **GPS:** Global Positioning System

1.4.4 **ESP32:** A low-power microcontroller with integrated Bluetooth/Wi-Fi

1.4.5 **IMU:** Inertial Measurement Unit (e.g BNO086)

1.5 Acronyms and Abbreviations

1.5.1 **SDD:** Software Design Document

1.5.2 **SRS:** Software Requirements Specification

1.5.3 **UI:** User Interface

1.5.4 **API:** Application Programming Interface

1.5.5 **MVC:** Model-View-Controller

1.5.6 **PCB:** Printed Circuit Board

2 System Requirements

2.1 Functional Requirements

Based on the SRS and architectural design, the system must:

2.1.1 **Bluetooth Connectivity:** Enable a stable Bluetooth connection between the wearable device and the mobile app.

2.1.2 **Sensor Data Collection:** Gather data from gyroscope, accelerometer, GPS, and altimeter sensors at a sampling rate of 200Hz.

2.1.3 **Trick Recognition:** Process movement data using machine learning algorithms to identify and score tricks.

2.1.4 **Real-Time Performance Grading:** Provide immediate feedback on ride metrics such as carving angles and landing stability.

2.1.5 **User Interface for Data Visualization:** Display real-time and historical ride data through an intuitive mobile app interface.

2.1.6 **Cloud Integration:** Support data storage and long-term trend analysis.

2.2 Non-functional Requirements

2.2.1 **Performance:**

- Data transmission latency $\leq$ 50ms.
- Sensor sampling rate of 200Hz.

2.2.2 **Reliability:** Device must function in extreme cold (as low as –20°C) and snowy conditions.

2.2.3 **Battery Life:** Minimum operational time of 4–8 hours per charge.

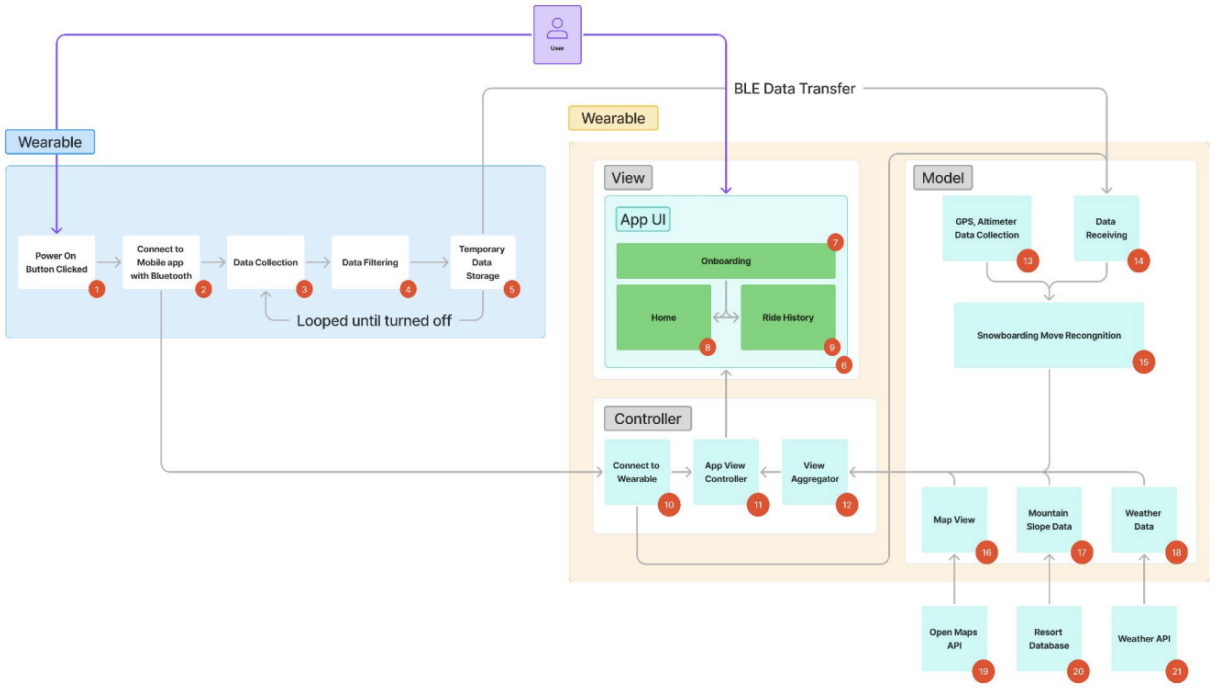
2.2.4 **Packaging:** Compact and durable design, ensuring minimal interference with rider movement.

2.2.5 **Security:** Encrypted Bluetooth data transmission and secure data storage.

2.2.6 **Platform Compatibility:** Mobile app to support Android 10+ and iOS 14+.

3 System Architecture

3.1 Overall Architecture



The system follows a combination of a **Client-Server** model, an **MVC** and a **pipe and filter consisting** of two primary components:

IoT Wearable (Client): Utilizes a Pipe & Filter pattern to collect and pre-process sensor data. The core hardware includes an ESP32 for Bluetooth communication and sensors (e.g., BNO086) for motion tracking.

Mobile App (Server): Implements a Model-View-Controller (MVC) design to ensure a clear separation between the user interface, data processing (using TensorFlow Lite for trick recognition), and control logic. The app manages data visualization, performance grading, and cloud integration.

3.2 Components Mapping

The functional and non-functional requirements are mapped to specific system components as follows:

Functional Requirements Mapping

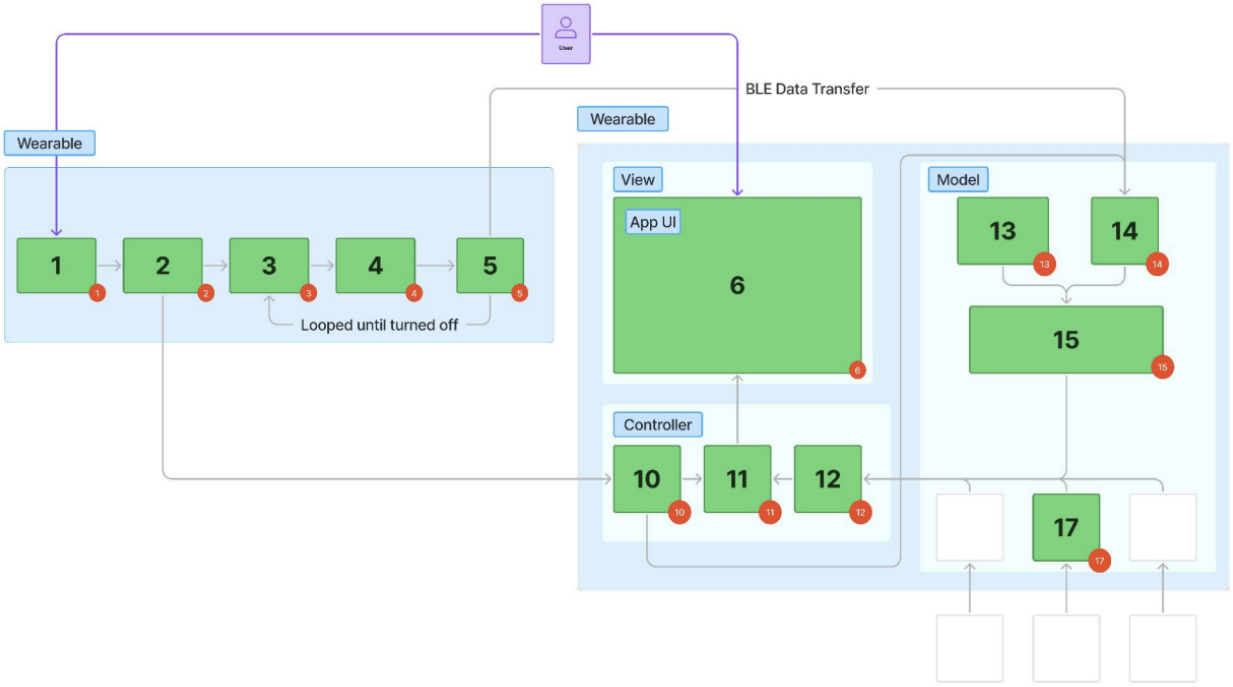
3.2.1.1 **Bluetooth Connectivity:** (2, 10) Handled by the ESP32 and Flutter Blue library on the mobile app.

3.2.1.2 **Sensor Data Collection:** (1, 2, 3, 4, 5, 13) Managed by the wearable's sensor module and data filtering algorithms.

3.2.1.3 **Trick Recognition:** (13, 14, 15) Executed on the mobile app using TensorFlow Lite along with predefined movement signatures.

3.2.1.4 **Performance Grading:** (13, 14, 15) Integrated within the mobile app's analytics module.

3.2.1.5 **Data Visualization:** (15, 17, 12, 11, 6) Provided by the mobile app's UI components, including 3D ride visualization.



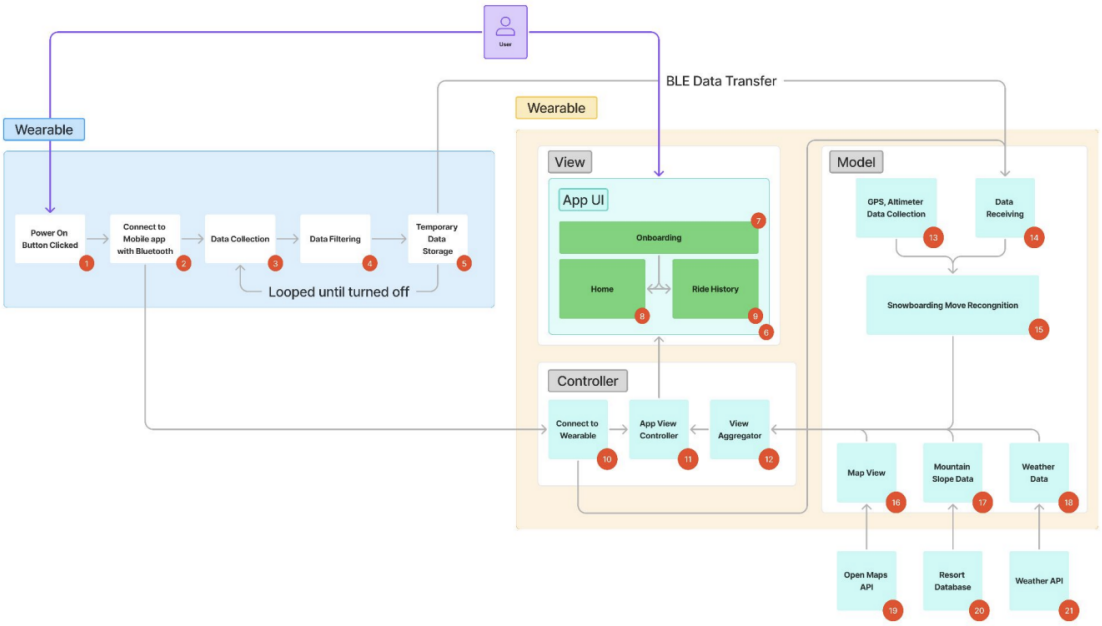
Non-functional Requirements Mapping

3.2.2.1 **Data Transmission Latency:** Addressed via optimized Bluetooth communication protocols.

3.2.2.2 **200Hz Sensor Sampling:** Achieved through efficient I2C sensor communication on the wearable.

3.2.2.3 **Cold Environment Operation:** Ensured by selecting components rated for low temperatures and robust wearable housing.

3.2.2.4 **Battery Life:** Supported by a 1,000–1,250mAh Lithium-Ion battery with efficient power management.



3.3 Technology Stack Selection

3.3.1 Mobile App:

3.3.1.1 **Flutter:** For cross-platform UI development.

3.3.1.2 **Flutter Blue:** To manage Bluetooth Low Energy (BLE) communication.

3.3.1.3 **TensorFlow Lite:** For executing on-device machine learning models for trick recognition.

3.3.2 APIs:

3.3.2.1 **Open Maps API:** To fetch slope and mapping data.

3.3.2.2 **Weather API:** To deliver real-time snow and weather conditions.

3.3.3 Wearable Hardware:

3.3.3.1 **ESP32:** For Bluetooth and onboard processing.

3.3.3.2 **Sensors:** BNO086 for motion and orientation data.

3.3.3.3 **Battery Pack:** 3.7V, 1,000–1,250mAh Lithium-Ion battery ensuring power efficiency in cold climates.

3.3.3.4 **Enclosure:** IP67/IP68-rated housing for durability.

4 System Design

4.1 User Interface (UI)

The mobile app UI is designed with the following screens:

– **Splash Screen:** Displays the app logo for 3–5 seconds upon launch.

– **Onboarding Screen:** Guides first-time users through device pairing and setup.

– **Home Screen:**

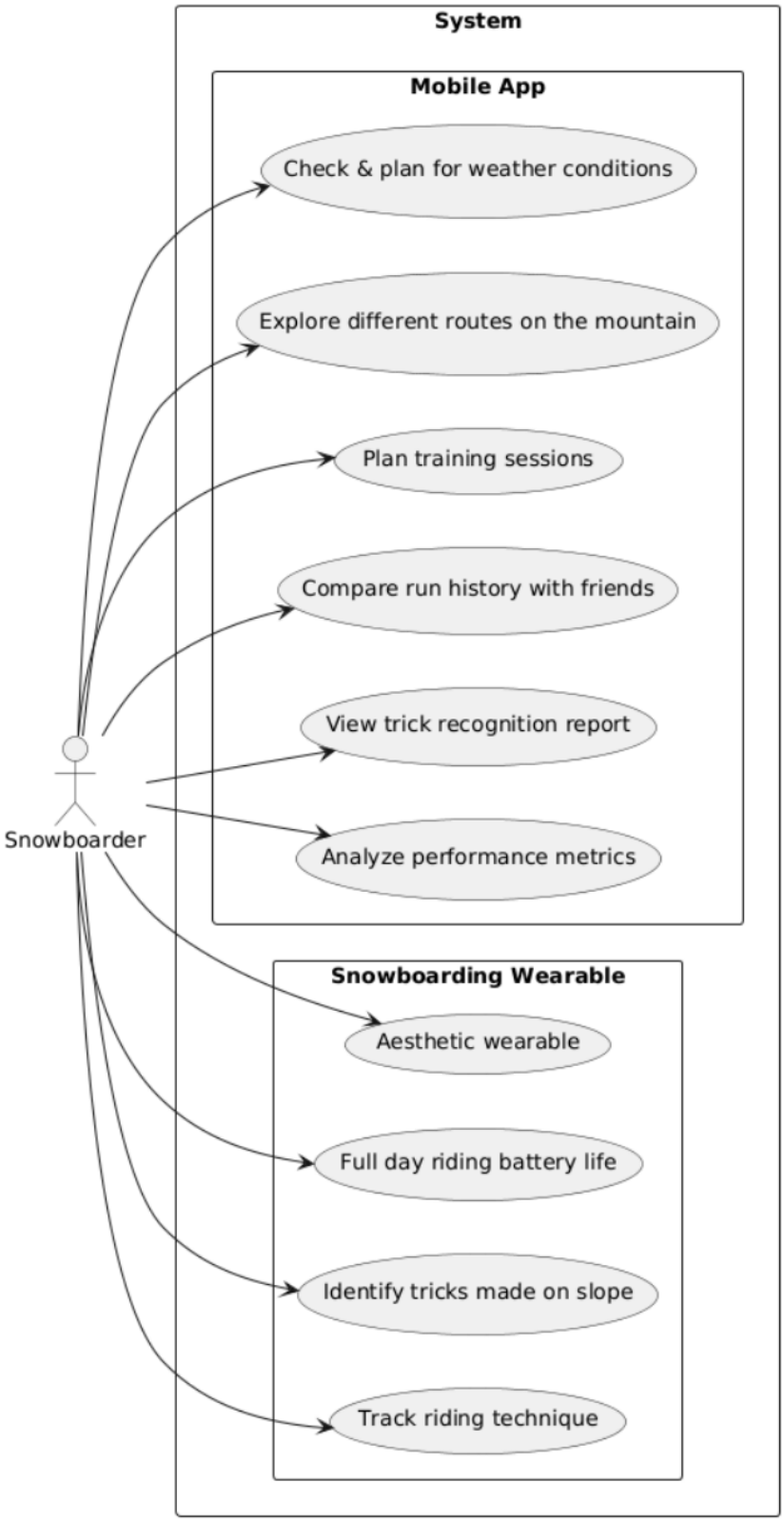
- Real-time 3D visualization of snowboard movements.
- Map view displaying current ride path and GPS tracking.
- Performance scoring and feedback section.

– **History Log Page:**

- List of previous rides with metadata (date, time, score, etc.).
- Detailed view for individual ride analytics with graphs and 3D replay.

– **Navigation Bar:** Persistent access to key sections of the app.

4.2 Use Case Diagrams and Tables



– **Device Pairing:**

  - **Actors:** User, Wearable Device, Mobile App.
  - **Description:** The user initiates Bluetooth pairing; the mobile app establishes a connection and confirms device readiness.

– **Real-Time Data Visualization:**

  - **Actors:** User, Mobile App.
  - **Description:** Sensor data is received and processed, then visualized on the home screen.

– **Trick Recognition and Feedback:**

  - **Actors:** User, Mobile App.
  - **Description:** The mobile app analyzes sensor inputs, recognizes tricks, and provides performance grading.

Detailed use case tables are maintained in the project repository.

4.3 Class Diagram

The class diagram includes (but is not limited to) the following classes:

– **MobileApp:**

  - **Attributes:** userProfile, rideHistory, currentSession.
  - **Methods:** displayUI(), initiatePairing(), updateData().

– **BluetoothManager:**

  - **Attributes:** connectionStatus, deviceList.
  - **Methods:** connect(), disconnect(), sendData(), receiveData().

– **SensorDataManager:**

  - **Attributes:** sensorReadings, filteredData.
  - **Methods:** collectData(), filterNoise(), triggerTrickRecognition().

– **TrickRecognitionEngine:**

  - **Attributes:** model, trickDatabase.
  - **Methods:** analyzeData(), classifyTrick(), provideFeedback().

– **DataAnalyticsModule:**

  - **Attributes:** performanceMetrics, historicalData.
  - **Methods:** calculateScore(), generateReports(), updateHistory().

4.4 Sequence/Activity Diagram

Device Pairing and Data Transmission Sequence:

1. **User Action:** Launch mobile app and initiate pairing.
2. **BluetoothManager:** Scans for the wearable and establishes a connection.
3. **SensorDataManager (Wearable):** Begins collecting and filtering sensor data.
4. **Data Transfer:** Processed sensor data is transmitted to the mobile app with latency $\leq$ 50ms.

5. **TrickRecognitionEngine:** Receives data, analyzes movement patterns, and classifies tricks.

6. **DataAnalyticsModule:** Computes performance grading and updates the UI.

7. **UI Update:** Mobile app displays real-time analytics and visualizations.

Detailed activity diagrams are maintained as supplementary design files.

4.5 Database Design

Cloud Storage: Stores historical ride data including sensor readings, performance scores, and user profiles.

Local Database (on Mobile App):

– **Tables:**

- **Rides:** rideID, dateTime, score, GPS data.
- **UserProfile:** userID, name, preferences.
- **SensorData:** sensorID, rideID, dataPoints.

– **Data Model:** Uses a relational schema ensuring data integrity and efficient querying.

5 Hardware Architecture & Electrical Design

The Snowboard Wearable device is built around a compact, low-power embedded system designed for extreme cold weather conditions and real-time sensor data acquisition. The electrical design was implemented using KiCad 7.0. It includes a full schematic diagram, PCB layout, and 3D board rendering to validate component placement and mechanical fit.

5.1 Overview

The hardware system integrates:

- **ESP32-WROVER Module (U1):** Central microcontroller responsible for Bluetooth communication and control logic. It operates at 3.3V and interfaces with sensors and the mobile application.
- **IMU Sensor – BNO086 (U5):** Provides 9-axis motion tracking via I^2C for trick recognition and ride analysis.
- **Battery Management Circuit (U4):** Uses an MCP73831T-2ACI/OT Li-ion charger IC for regulated charging via the USB port.
- **Power Supply Circuit (U3):** Converts 5V input to 3.3V using the SPX3819 LDO regulator to power the ESP32 and sensors.
- **USB–UART Bridge (U2):** CP2102N for serial communication and flashing firmware over USB.
- **Boot and Reset Buttons (SW1, SW2):** Manual control for entering programming mode and restarting the ESP32.

5.2 Schematic Design

The schematic includes carefully designed protection and power management:

- Schottky diode (MBR120) to prevent reverse current flow from battery to USB.
- Test points for debugging critical pins (e.g., I^2C lines, RX/TX, EN, BOOT).
- Pull-up/down resistors for proper GPIO initialization.

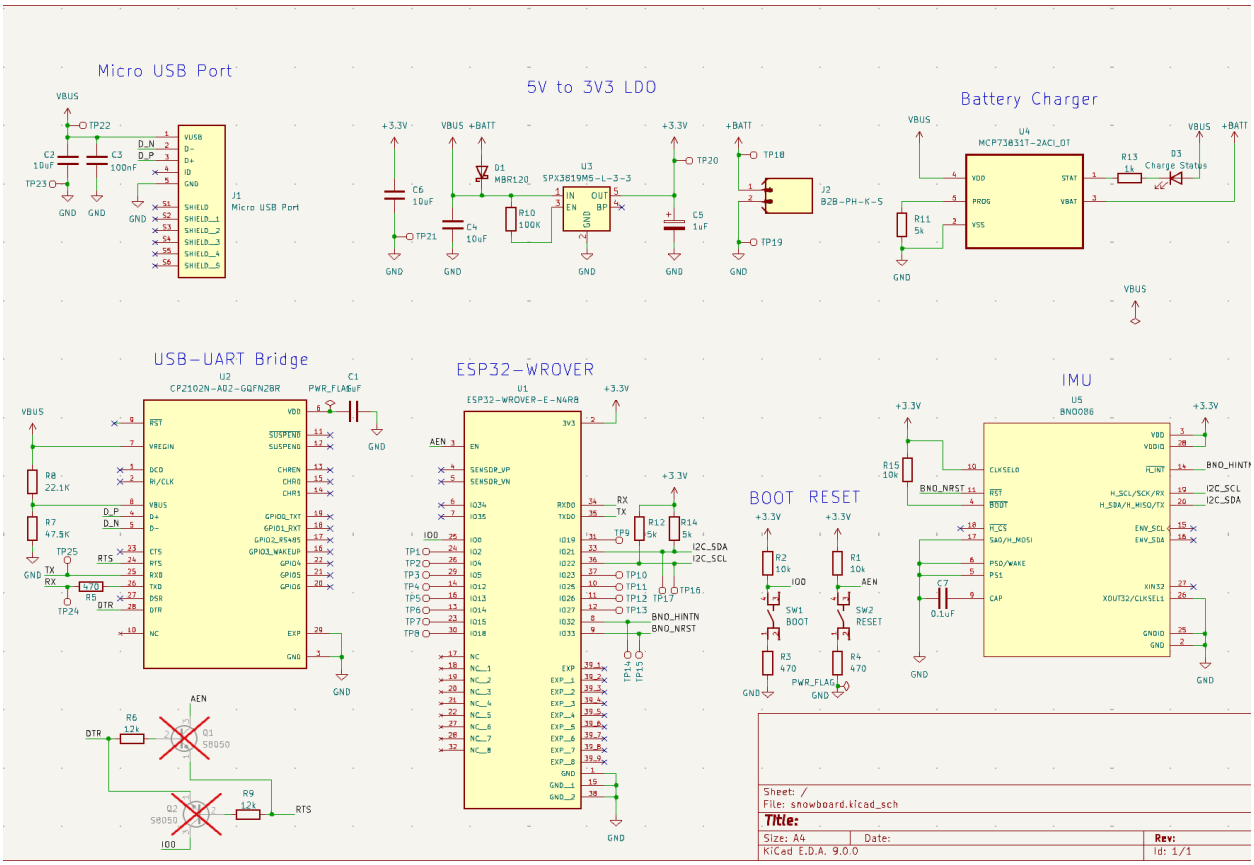


Figure 1: Complete Electrical Schematic of the Snowboard Wearable

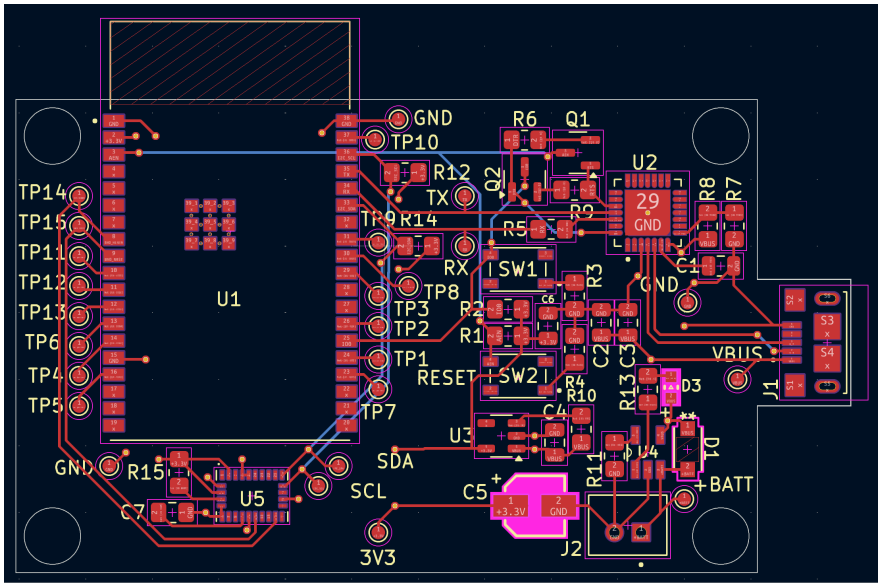


Figure 2: Top Layer PCB Layout (KiCad)

5.3 PCB Layout

The board layout is optimized for:

- Compact size with mounting holes at all four corners.
- Minimized trace length for I^2C communication to reduce noise.
- Clearly labeled test points and headers for rapid prototyping and field debugging.
- Ground plane continuity and proper decoupling capacitor placement for noise reduction.

5.4 3D Model

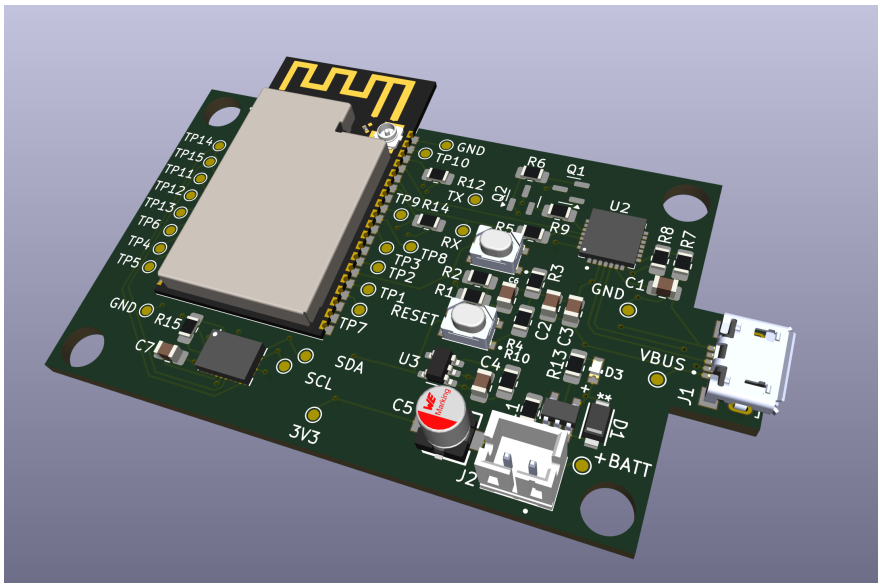


Figure 3: 3D PCB Rendering

The 3D rendering validates:

- Mechanical alignment of USB port, battery connector, and push buttons.
- Proper spacing for the ESP32 module antenna for optimal wireless performance.
- Component placement consistency with electrical and thermal design rules.

5.5 Design Considerations

- The board supports cold-weather operation ($-20°C$) with components rated for low temperatures.
- LDO regulator and power rail design minimizes power loss, supporting up to 5 hours of continuous use.
- IP68-compatible mechanical design under consideration for final enclosure integration.

This hardware implementation represents the foundation for the wearable system's real-time data acquisition capabilities and seamless mobile integration. The design files are maintained in the project repository for future enhancements and manufacturing.

6 Mobile Application UI Design

The mobile application for the Snowboarding Wearable device serves as the primary interface for users to interact with their ride data, track progress, and receive feedback. Designed in Flutter, the UI focuses on clarity, accessibility, and gamified engagement. Below is an overview of the four main screens currently designed and implemented.

6.1 Home Screen

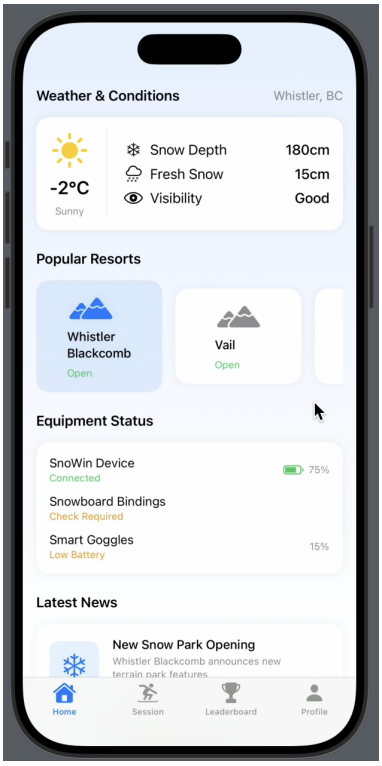


Figure 4: Home Screen — Real-Time Weather and Device Status

The home screen delivers high-level contextual and environmental data to prepare users for snowboarding sessions:

- **Weather Conditions:** Real-time weather, snow depth, fresh snow, and visibility at the selected resort (e.g., Whistler).
- **Popular Resorts:** A horizontal scroll of available resorts and their current status (Open/Closed).
- **Equipment Status:** Displays the connectivity and battery levels of key gear including the SnowWin device, snowboard bindings, and smart goggles.
- **Latest News:** Dynamic content on resort announcements and feature rollouts.

6.2 Leaderboard Screen

The leaderboard integrates social competition to drive user engagement and improvement:

- **Top Performers:** Ranks weekly, daily, monthly, and all-time snowboarders based on experience points (XP).
- **Category Highlights:** Highlights the top user in key metrics such as highest jump, top speed, and most tricks.
- **Friends Ranking:** A personalized leaderboard comparing XP with friends.

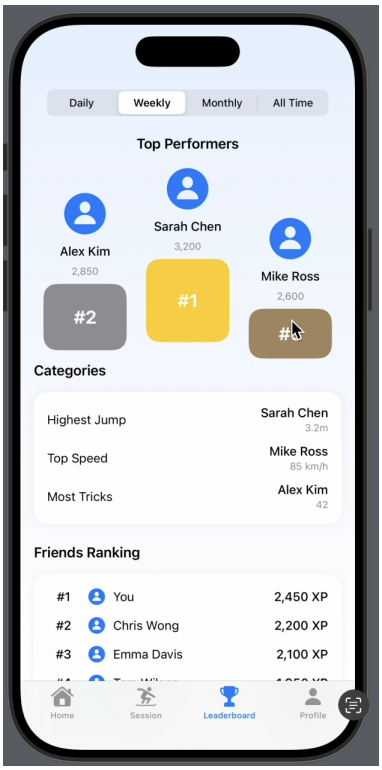


Figure 5: Leaderboard Screen — Gamification and Social Ranking

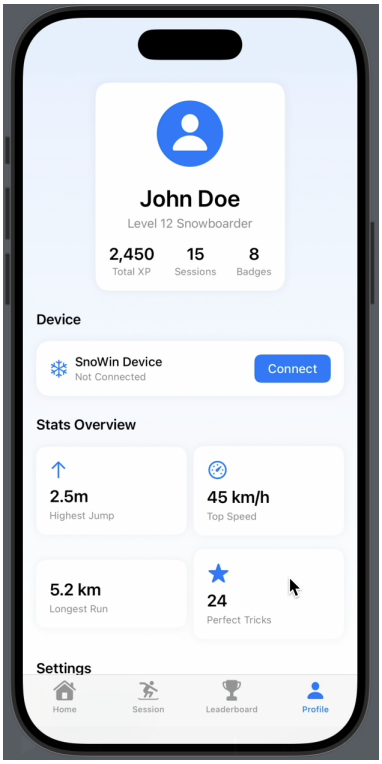


Figure 6: Profile Screen — User Summary and Stats Overview

6.3 Profile Screen

The profile screen helps users track their long-term progress and device connectivity:

- **User Identity:** Name, XP level, number of sessions, and earned badges.
- **Device Status:** Allows users to connect/disconnect their SnowWin device.
- **Stats Overview:** Summarizes personal performance metrics: highest jump, top speed, longest run, and perfect tricks.

6.4 Session Playback Screen

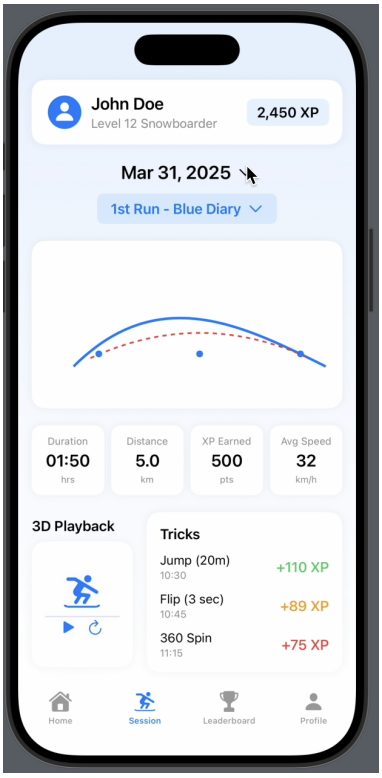


Figure 7: Session Screen — Performance Analytics for Each Run

This screen provides post-session analysis, breaking down an individual snowboarding session:

- **Run Summary:** Displays XP earned, ride duration, average speed, and distance.
- **3D Playback:** Preview of trajectory and motion using real-time sensor data.
- **Trick List:** Timestamped list of recognized tricks, XP earned per trick, and execution quality.

6.5 Design Considerations

- The UI follows a **bottom-tab navigation structure** for seamless access to Home, Session, Leaderboard, and Profile sections.
- A **gamified XP system** and leaderboard foster engagement and motivation for skill improvement.
- UI components are designed with accessibility, clarity, and winter-sport usability in mind — including large tap targets and legible fonts.

The current interface is optimized for iOS 14+ and Android 10+ devices and will continue to evolve as new features are implemented and user testing insights are gathered.

7 Test Plan

7.1 Test Plan Overview

Testing is structured around verifying both functional and non-functional requirements. The table below outlines the key test cases:

No.	Test Case Description	User Input/Action	Pass Criteria
1	Device Pairing	User initiates pairing via mobile app	Successful Bluetooth connection with the wearable established
2	Sensor Data Transmission	Wearable sends sensor data continuously	Data received with latency $\leq$ 50ms; sensor sampling at 100Hz
3	Trick Recognition Accuracy	Perform known trick maneuvers	System correctly identifies and scores the trick with $>$ 70% accuracy
4	UI Data Visualization	User navigates to home screen	3D ride visualization and performance data are accurately displayed
5	Battery Life Test	Continuous operation over 5 hours	Device maintains operation for minimum required duration

8 References

– Architectural Design Presentation – Group 6, COMP4960 – Software Engineering, Feb. 18, 2025.

– SRS Document – WIT_COMP4960_SP25_Group06 (Version 1.2, January 2025).

– SDD Sample Document – Wentworth Institute of Technology Format.

Additional references:

- Espressif ESP32 Datasheet
- Ceva BNO08X Datasheet
- OpenStreetMap License
- Open-Meteo License
- Flutter and Flutter Blue documentation

9 Conclusion & Status Summary

This Software Design Document outlines the architecture, component design, and system functionality for the Snowboarding Wearable project, aimed at delivering a data-driven performance analysis tool for snowboarders. It documents all major design decisions, technology choices, and the system's modular breakdown across hardware, mobile software, and analytics.

As of April 14, 2025, the project has completed key phases including:

- A functional design of the mobile app UI with data visualization components.
- Configuration and partial testing of the wearable hardware system using the ESP32 and BNO086 IMU.
- Development of a modular software architecture accommodating sensor integration, Bluetooth communication, and analytics pipelines.
- Semi-functional simple trick recognition with decent accuracy

However, full system integration—specifically the real-time connection between the mobile application, the hardware device, and the trick recognition algorithm remain incomplete. A few simple trick recognition algorithms were able to be programmed, however more advanced learning-based trick recognition features could not be implemented within the current project timeline and is marked for future development.

Despite these limitations, the current version still lays a solid technical foundation for further iterations. Future work should focus on completing the hardware-software integration, training and embedding the trick recognition model, and performing field testing to refine user experience and system performance.

This document serves as a comprehensive guide for future teams or contributors who wish to continue or expand upon this project.